mathPAD

Band 9, Ausgabe 1, März 1999

In diesem Heft

Redaktion:
Benno Fuchssteiner,
Frank Postel, Andreas Sorgatz
Universität-GH Paderborn
D-33095 Paderborn
email: `benno@mupad.de`

ISSN 0941-9187
V.i.S.d.P.: Benno Fuchssteiner
Nachdruck gegen Belegexemplar erlaubt

Die Grafik der Titelseite stammt im Original von: Werbeagentur Künkel + Lopka, Heidelberg; Quelle: Springer Verlag Berlin · Heidelberg · New York, ISBN 3-540-65043-1

Liebe Leserin, lieber Leser,

beim Durchblättern der vorliegenden mathPAD werden Sie sich vielleicht über eine Anzeige der Firma SciFace Software wundern. Sie sollten daraus nicht schließen, daß die *MuPAD* Gruppe nun noch mehr auf die Kommerzialisierung ihrer Entwicklungsergebnisse setzt. Die Notwendigkeit, die mathPAD mit kommerzieller Werbung zu verbinden, besteht für uns, da die Universität-GH nicht mehr bereit ist, die Druckkosten für die mathPAD in der bisherigen Weise zu tragen (und dies auch wegen der Haushaltsenge und der Druckkosten für eine Vielzahl beachtlicher Publikationen wohl nicht mehr kann).

Wir müssen deshalb die Druckosten entweder durch Zweckentfremdung von Mitteln aus Forschungsprojekten tragen oder aber diese Mittel durch Anzeigen selbst erwirtschaften. Leider haben wir in der kurzen Zeit für die vorliegende Ausgabe nur einen Anzeigenkunden gewinnen können. Für die nächsten Ausgaben hoffen wir, den Kreis der Inserenten auf Verlage und Mitbewerber der Firma SciFace Software ausweiten zu können.

Lassen Sie sich durch diese weitere Hinwendung zur kommerziellen Finanzierung, die wir selbst am meisten bedauern, nicht in Ihrem Lesevergnügen stören. Denn wir glauben, auch mit diesem Heft wieder interessante Neuheiten über MuPAD und Berichte aus zahlreichen Gebieten der Computer Algebra zusammengetragen zu haben. Für das Gelingen dieses Heftes möchten wir insbesondere den auswärtigen Autoren herzlich danken!

B.F. & F.P. & A.S.

Kurz notiert

◇ EIN NETTES KOMPLIMENT für die *MuPAD*-Entwickler findet sich in der Neujahrsbotschaft der französischen Vereinigung der TEX-Nutzer ASTEX (Association Tex, `www.univ-orleans.fr/EXT/ASTEX`).

Dort heißt es im Schlußabsatz der Botschaft von Michel Lavaud unter dem Titel *Pourqoi MuPAD*:

> *Pour toutes ces raisons, il me parât inéluctable que MuPAD remplacera les logiciels commerciaux á plus ou moins longue échéance. Ainsi, le calcul formel re- deviendra une science ce qu'il était à ses débuts, il y a une vingtaine d'années, avant que le dieu Ploutos ne le fasse éclater en une multitude de langues pour mieux asseoir sa puissance...*

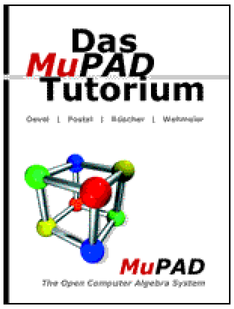
Der vollständige Artikel findet sich unter `www.univ-orleans.fr/EXT/ASTEX/astex/doc/fr/web/html/whymup.htm`.

◇ ALS STANDARD-SOFTWAREUMGEBUNGEN wurden bei dem diesjährigen *Examen de Agrégation* in Frankreich (landesweite Prüfung für die Befähigung zum höheren Lehramt) die Computeralgebrasysteme Maple und *MuPAD* zugelassen und bereitgestellt. Siehe `www.math.u-psud.fr/~agreg/Agreg/installation.html`.

◇ AUF DER PFINGSTTAGUNG *Neues Lernen mit Neuen Medien – Mathematikunterricht in der Zukunft* in Münster (25.–28. Mai 1999) bietet die *MuPAD* Research Group den Workshop „Alternative Wege in der Analysis“ an. Dieser Workshop richtet sich insbesondere an *MuPAD* Einsteiger. Kenntnisse werden im Umgang mit Windows 95/98 oder Windows NT vorausgesetzt.

In dem Workshop wird die deutschsprachige *MuPAD* Pro 1.4.1 Version eingesetzt. Die Arbeiten lassen sich aber direkt auf Mac*MuPAD* und den Unixversionen von *MuPAD* (u.a. Linux) übertragen. Weitere Informationen finden sich unter `www.mupad.de/EDU/`.

◇ DEN ERSTEN PROTOTYP einer parallelen Version von *MuPAD* hat Ende Juli vergangenen Jahres die *MuPAD* Research Group freigegeben. Weitere Informationen finden Sie unter `www.mupad.de/PRESS/parallel.shtml`.



Das MuPAD Tutorium

W. Oevel, F. Postel, G. Rüscher, S. Wehmeier

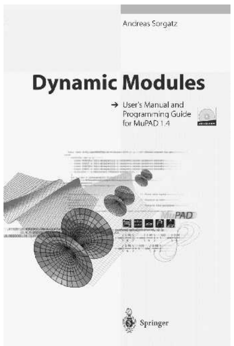
September 1998, SciFace Software GmbH & Co.KG, Paderborn, viii, 299 Seiten, Softcover DM 48,-, ISBN 3-933764-00-9

Das *MuPAD Tutorium* stellt eine elementare Einführung in den Umgang mit dem Computer-Algebra-System *MuPAD* dar. Es richtet sich an Mathematiker, Ingenieure, Informatiker, Naturwissenschaftler, allgemein an alle, die mathematische Berechnungen für ihren Beruf oder in der Ausbildung benötigen und die Fõhigkeiten eines modernen Computeralgebra-Systems für sich nutzbar machen möchte.

In einfachen Schritten werden die wichtigsten Bausteine des Systems vorgestellt und an Anwendungsbeispielen demonstriert. Die Benutzung von Systemfunktionen, der Graphik sowie die Programmierung *MuPAD*s wird anhand zahlreicher Beispiele eingeübt.

Die Anwendungsbeispiele wurden bewußt elementar gehalten, so daß auch Schülern ein einfacher Einstieg in die Computeralgebra geboten wird. Dies eröffnet Lehrern die Möglichkeit, das Computeralgebra-System in den Unterricht zu integrieren.

Dynamic Modules – User's Manual and Programming Guide for MuPAD 1.4



Andreas Sorgatz

October 1998, Springer Verlag, Berlin · Heidelberg · New York, 234 pages, with an accompanying CD-ROM, ISBN 3-540-65043-1.
`http://www.mupad.de/~andi/Pub/Intro/sor98.shtml`

The concept of dynamic modules enables users to extend the *MuPAD* kernel with user-defined C/C++ routines and to integrate external software packages in an easy, flexible and very efficient way. This technology was introduced in mathPAD Vol.7/No.1 page 16. Applications of dynamic modules were demonstrated in mathPAD Vol.8/No.1 with articles about interacting with the *Singular* system using *MP* (see page 8) and implementing the *Macro Parallelism* in *MuPAD* 1.4.1 using *PVM* (see page 33).

The book *Dynamic Modules — User's Manual and Programming Guide for MuPAD 1.4* is addressed to users and developers of dynamic modules in *MuPAD*. It introduces the general concept and describes how to use, write, create and document modules. Although most information is independent from the operating system, the book emphasizes UNIX platforms, namely Solaris and Linux.

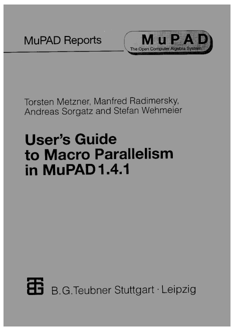
As background information for module developers, the book describes in brief the general structure of the *MuPAD* system and the *module generator* (mmg), which makes creating executable module binaries easy. Also important internals of the *MuPAD* kernel and its data structures are discussed. The central part of the book is a detailed description of the *MuPAD Application Programming Interface* (MAPI) which defines a C/C++ interface to the kernel and provides routines for data conversion, creating and manipulating *MuPAD* data structures, a *MuPAD* interpreter interface, service routines and more. The appendix contains installing instructions, short references and several demonstrations, e.g. using the NAG-C and IMSL libraries, the GMP arithmetic package and the number theory packages MAGNUM and NTL from within *MuPAD*, interacting with GB and RealSolving, interacting with the Singular system using MP, a pre-release of *MuPAD* macro parallelism based on PVM and more.

The accompanying CD-ROM contains trial versions of *MuPAD* 1.4.1 for Solaris and Linux including hypertext documentation. It provides a life CD-ROM system with an HTML guide, for running *MuPAD* directly from the CD-ROM. Besides the module binaries and their source code, it includes the following contributions: ASAP and MP for communication mathematical data, Singular for polynomial computations emphasizing the needs of commutative algebra, algebraic geometry and singularity theory, GB/RealSolving for computing Gröbner bases and information about real roots of polynomials, GMP, MAGNUM and NTL for doing number theory as well as the PVM package for distributed computing.

The book is available at Springer Verlag *Computer Science* (`http://www.springer.de`). Additional information about dynamic modules are available at `http://www.mupad.de/PAPER/MODULES/`. At this site, also future updates and extensions will be available.

Note, that this book is also relevant for coming *MuPAD* versions. Extensions, with respect to the Dynamic Modules Manual, will be available as online documentation.

User's Guide to Macro Parallelism in MuPAD 1.4



T. Metzner, M. Radimersky, A. Sorgatz, S. Wehmeier

Jan 1999, B.G. Teubner Verlag, Stuttgart · Leipzig, 134 pages, with an accompanying CD-ROM, ISBN 3-519-02199-4.
`http://www.mupad.de/~andi/Pub/Intro/mpb98.shtml`

MuPAD macro parallelism supports users in doing distributed computations in heterogeneous computer networks in a very easy and convenient way. This technology was introduced in mathPAD Vol.8/No.1 page 33.

The book *User's Guide to Macro Parallelism in MuPAD 1.4* is addressed to users and developers of parallel algorithms in *MuPAD*. It describes *MuPAD* macro parallelism which is based on the concepts of message passing, global variables and work groups. In combination with the concept of *Domains* and *Dynamic Modules*, *Macro Parallelism* is the third pile for using *MuPAD* as an open parallel problem solving environment for mathematical applications. Although most information is independent from the operating system, the book emphasizes UNIX platforms, namely Solaris and Linux.

Besides a detailed description of the parallel programming interface and programming guidelines and the demonstration of parallel algorithms, like factoring integers using elliptic curves or quadratic sieves and multiplying polynomials using Karatsuba's method, the book discusses the design and implementation of macro parallelism as a dynamic module. The technical information given in this report enables users to extend and/or change *MuPAD* macro parallelism according to their need. The appendix contains installing instructions and short references.

The accompanying CD-ROM contains trial versions of *MuPAD* 1.4.1 for Solaris 2.5/2.6 and Linux 2.0 operating systems including a hypertext version of the book. It also provides a life CD-ROM system, including an HTML guide, for running *MuPAD* directly from the CD-ROM.

Furthermore, the module binary and the complete C/C++ source code of macro parallelism as well as the PVM package are available on the CD-ROM.

The book is available at B.G. Teubner Verlag. For details refer to Teubner web site: `http://www.teubner.de`

The CDROM is also available as a separate product (ISBN 3-933764-01-7) at SciFace Software. For details refer to SciFace web site: `http://www.sciface.com`

Additional information about parallelism in *MuPAD* are available at `http://www.mupad.de/PAPER/PARALLEL/`. At this site, also future updates and extensions will be available.

Vereinfachen von komplexen Integralen reeller Funktionen

Winfried Fakler

Universität-GH Paderborn, fakler@mupad.de
www.mupad.de/~fakler

Automatische Vereinfachung komplexwertiger Funktionen in Computeralgebra-Systemen birgt viele Gefahren. Man bekommt zwar häufig mathematisch korrekte Ergebnisse, jedoch können diese für die Praxis völlig unbrauchbar sein, weil dabei beispielsweise eine ungeeignete Verzweigung gewählt wurde. Wie man diesem Problem beim Vereinfachen von komplexen Integralen reeller Funktionen begegnen kann, wird in diesem Artikel beschrieben.

Bei der Integration von elementaren Funktionen können komplexwertige Resultate entstehen. Jedoch befinden sich darunter gelegentlich auch nur scheinbar komplexwertige Funktionen, also Funktionen die *MuPAD* nicht automatisch in ihre reelle Darstellung konvertieren kann. Diese treten vor allem bei Benutzen des Risch-Algorithmus auf. Ein Grund dafür ist, daß a priori alle auftretenden Unbestimmten in *MuPAD* als komplexwertig angenommen werden und im Komplexen viele Vereinfachungen, die im Rellen erlaubt sind, nicht durchgeführt werden dürfen.
Im folgenden wird gezeigt, wie man dennoch häufig scheinbar komplexwertige Integrationsergebnisse vereinfachen kann. Die weiter unten vorgestellten Funktionen findet man in der Datei `simpCInt141.mu` auf der Internetseite des Autors.

Das Problem

Beim Integrieren in Computeralgebra-Systemen wird häufig davon ausgegangen, daß die Integrationsvariable reell ist. In speziellen Unterprogrammen wird gelegentlich – stillschweigend – sogar zusätzlich angenommen sie habe ein positives Vorzeichen. Das kann zu folgendem überraschendem Verhalten führen:

```
>> int(sin(ln(x+1)),x);

   sin(ln(x + 1))   cos(ln(x + 1))   x cos(ln(x + 1))   x sin(ln(x + 1))
   -------------- - -------------- - ---------------- + ----------------
         2                2                 2                  2
```

Wegen der Annahme $x > 0$ konnte das Integral vereinfacht werden. Dagegen kann aus demselben Grunde

```
>> int(sin(ln(x-1)),x);

                                1 - I                        1 + I
         - (1/4 - 1/4 I) (x - 1)      - (1/4 + 1/4 I) (x - 1)
```

nicht mehr vereinfacht werden, denn hier könnte das Argument des Logarithmus ja trotz dieser Annahme noch negativ sein. Es kann aber auch umgekehrt das Nicht-Vereinfachen von (scheinbar) komplexwertigen Ergebnissen sehr problematisch sein:

```
>> f:=(2*x^3+x^2+3)/(x^4-x^3+x+1);

                                2      3
                               x  + 2 x  + 3
                              --------------
                                   3    4
                              x - x  + x  + 1
>> F:=int(f,x);

  /    2                  \              /                 1/2 \
  | 7 x    /  2   x      \2 |      1/2     |  2   x         x 7    |
ln| ---- + | x  - - - 1 |  |   5 7    atan| x  - - - 1, - ------ |
  \  4     \      2     /  /              \      2          2    /
--------------------------- + ---------------------------------- -
             2                                14

            /                1/2 \
     1/2    |  2   x        x 7    |
   5 7    atan| x  - - - 1, ------ |
            \      2         2    /
   --------------------------------
                  14
```

Zwar wird die unbestimmte Integration der (reellen) rationalen Funktion `f` laut Definition formal korrekt durchgeführt, denn die Ableitung ihrer Stammfunktion `F`

```
>> normal(diff(F,x));

                                2      3
                               x  + 2 x  + 3
                              --------------
                                   3    4
                              x - x  + x  + 1
```

ist wieder `f`. Man kann nun leicht zeigen, daß `f` im Intervall $x \in [0, 2]$ stetig und positiv ist und daher das bestimmte Integral über diesem Intervall eine positive reelle Zahl sein muss. Wendet man aber den Hauptsatz der Differential- und Integralrechnung auf die Stammfunktion `F` an, so bekommt man

```
>> float(eval(subs(F,x=2)-subs(F,x=0)));

                               -0.5463250105
```

einen falschen negativen Wert heraus. Dieses falsche Ergebnis kommt daher, daß der Nenner von `f` in diesem Intervall zwei konjugiert komplexe Nullstellen enthält und die Auswahl der komplexen Arcustangens-Zweige eine unstetige Stammfunktion nach sich ziehen.

Eine gute Idee ist also, stets das Ergebnis numerisch mittels Gauss-Quadratur zu überprüfen:

```
>> numeric::quadrature(f,x=0..2);

                              5.390727048
```

Wie lassen sich nun komplexe Integrale für die Praxis tauglich vereinfachen? Im ersten Schritt muß man klären, unter welchen Voraussetzungen Vereinfachungen zulässig sind. Hier wird von Integralen reeller Funktionen ausgegangen. Unter dieser Annahme sind alle weiteren Umformungen gültig.

Umformungen via Eulerscher Formel

Man kann bei der Integration ohne Beschränkung von einer reellen Integrationsvariablen x ausgehen, denn selbst die Integration komplexwertiger Funktionen wird auf reelle Integrale zurückgeführt. Sind also u und v reelle Funktionen in x und $z = u + iv$, dann gilt die Identität

$$\exp(u + iv) = \exp(u)\left(\cos(v) + i\sin(v)\right), \tag{1}$$

bzw.

$$\exp(iz) = \cos(z) + i\sin(z), \tag{2}$$

welche man oft auch als Eulersche Formel bezeichnet. Mit dieser Formel läßt sich die komplexe Einheit i aus dem Exponenten der Exponentialfunktion entfernen. Für eine beliebige Basis $a \neq e$ ist definiert

$$a^{u+iv} = a^u \exp(\ln(a)iv). \tag{3}$$

Allerdings ist hier große Vorsicht geboten, denn diese Definition greift auf die Logarithmusfunktion zurück, welche im Komplexen nicht eindeutig ist, genauer, sie hängt von der Wahl des Zweiges des Logarithmus ab. Um Fehler zu vermeiden, läßt man den Logarithmus unverändert und arbeitet mit Gleichung (2) weiter:

$$a^{u+iv} = a^u\left(\cos(\ln(a)v) + i\sin(\ln(a)v)\right), \tag{4}$$

bzw.

$$a^z = \cos\left(\ln(a)\frac{z}{i}\right) + i\sin\left(\ln(a)\frac{z}{i}\right). \tag{5}$$

In der Funktion `intlib::applyEulerIdentity` werden – aufgrund der speziellen Form des Ergebnisses aus dem Risch-Algorithmus – die Formeln (4) und (1) mit Hilfe der Funktion `maprec` rekursiv auf die gegebene Funktion angewandt, wobei jeweils noch erst das Argument auf die Form $u + iv$ gebracht wird. Mit ihr kann man also die komplexe Einheit i aus den Exponenten einer Funktion eliminieren und damit beispielsweise obiges Integral wie folgt umformen:

```
>> int(sin(ln(x-1)),x): intlib::applyEulerIdentity(%,x);

   cos(ln(x - 1))    sin(ln(x - 1))    x cos(ln(x - 1))    x sin(ln(x - 1))
   -------------- - -------------- - ---------------- + ----------------
         2                2                  2                   2
```

Der Algorithmus von Rioboo

Um bei der Vereinfachung komplexer Logarithmen jeweils den Zweig zu bestimmen, der notwendig ist, um eine stetige formale Stammfunktion zu erhalten, kann man auf einen Satz von Rioboo zurückgreifen:

Satz 1 *([1, Theorem 2.8.1])*
Es seien A und B zwei von Null verschiedene reelle Polynome in x, so daß $A^2 + B^2 \neq 0$. Dann gilt

$$\frac{d}{dx}\ln\left(\frac{A+iB}{A-iB}\right) = \frac{d}{dx}\ln\left(\frac{-B+iA}{-B-iA}\right)$$

und für beliebige reelle Polynome C und D mit $BD - AC = \gcd(A,B)$, $C \neq 0$ und $C^2 + D^2 \neq 0$

$$i\frac{d}{dx}\ln\left(\frac{A+iB}{A-iB}\right) = 2\frac{d}{dx}\arctan\left(\frac{AD+BC}{\gcd(A,B)}\right) + i\frac{d}{dx}\ln\left(\frac{D+iC}{D-iC}\right),$$

wobei $i^2 = -1$ ist.

Hierbei ist es wichtig, daß in den Polynomen A, B, C und D die komplexe Einheit $i = \sqrt{-1}$ nicht auftreten kann. Dieser Satz liefert einen rekursiven Algorithmus zur Ersetzung einer Summe von Logarithmen obiger Form in eine Summe von Arcustangens. Was noch bleibt, ist die Logarithmen aus der zu vereinfachenden Funktion herauszufiltern und in eine Form zu bringen, daß man Satz 1 anwenden kann.

In der Funktion `intlib::complexLnConversion` werden die Terme der zu vereinfachenden Funktion nach Logarithmen mit komplexen Argumenten durchsucht und die Terme mit derartigen Logarithmen weiter unterteilt in Logarithmen mit komplexen und mit reellen Vorfaktoren. Bei letzteren wird versucht die Logarithmen mit konjugiert komplexen Argumenten zu vereinfachen und bei ersteren der Algorithmus von Rioboo angewandt.

Will man nun das eingangs berechnete Integral `int(f,x)` mit dieser Funktion vereinfachen, muß man zuerst den internen Vereinfachungsmechanismus von `int` abstellen. Da dieser nur auf reelle Integranden angewandt wird, kann man das leicht folgendermaßen erreichen:

```
>> F:=int(f+I,x)-int(I,x);

                1/2    /  2   x            1/2     \
(1/2 - 5/14 I 7    ) ln| x  - - - 1/2 I x 7    - 1 | +
                       \      2                    /

            1/2           /  2   x            1/2     \
   (5/14 I 7    + 1/2) ln| x  - - + 1/2 I x 7    - 1 |
                          \      2                    /
```

Mit dem Algorithmus von Rioboo bekommt man daraus die Stammfunktion

```
>> F:=intlib::complexLnConversion(F,x): expand(%);

                                   /      1/2    1/2 \
                          1/2      | 2 x 7      7    |
        3    4         5 7    atan| -------- - ---- |
ln(x - x  + x  + 1)               \    7        7   /
------------------ + ------------------------------ +
         2                          7

               /  1/2         1/2       2  1/2       3  1/2 \
      1/2      | 7       2 x 7      2 x  7      2 x  7     |
   5 7    atan| ---- + -------- - --------- + --------- |
               \  7        7           7           7     /
   ------------------------------------------------------
                             7
```

mit der man nun das bestimmte Integral im Intervall $[0,2]$ korrekt berechnen kann:

```
>> float(eval(subs(F,x=2)-subs(F,x=0)));

                                    5.390727048
```

Die Funktion `intlib::simplifyComplexIntegral`

Kombiniert sind diese und weitere Vereinfachungsprozeduren in der Funktion `intlib::simplifyComplexIntegral`. Genauer, es wird darin versucht das komplexe Integral erst durch die Funktionen `combine` und `rewrite` zu vereinfachen. Falls dies mißlingt, wird `intlib::applyEulerIdentity`, `intlib::complexLnConversion` und schließlich noch `rectform` verwendet. Man beachte, diese Funktion wird nur aktiv, falls wie oben beschrieben die interne Vereinfachung von `int` abgeschaltet oder unwirksam ist.

```
>> read("simpCInt141.mu"):
>> int(1/(x^3-1),x);

ln(x - 1)              1/2                        1/2
--------- + (- 1/6 I 3    - 1/6) ln(x + 1/2 I 3    + 1/2) +
    3

           1/2                        1/2
   (1/6 I 3    - 1/6) ln(x - 1/2 I 3    + 1/2)

>> intlib::simplifyComplexIntegral(%,x);
                                                  /    1/2             \
                                          1/2     | 2 3    (x + 1/2) |
                               2         3    atan| ---------------- |
     ln(x - 1)   ln((x + 1/2)  + 3/4)             \        3         /
     --------- - -------------------- - -----------------------------
         3                6                           3
```

Es sei noch einmal erwähnt, daß die Datei mit den hier beschriebenen Funktionen im Internet unter

`www.mupad.de/~fakler/OdeFiles/simpCInt141.mu`

zur Verfügung steht.

References

[1] Bronstein, M. (1997). *Symbolic Integration I.* Berlin, Heidelberg, New York: Springer.

[2] Fischer, W., Lieb, I. (1988). *Funktionentheorie.* 5. Auflage; Braunschweig: Vieweg.

[3] Freitag, E., Busam, R. (1995). *Funktionentheorie.* 2. Auflage; Berlin, Heidelberg, New York: Springer.

Flexible Mathematical GUI Controls for Mathematical Electronical Documents

Frank Postel, Ralf Hillebrand
frank@mupad.de, tonner@mupad.de

In cooperation with SciFace Software[1] we develop open and flexible mathematical user interfaces which can be integrated into open software environments such as presentation tools, authoring tools or internet browsers.

The main goal of our project is to provide authors of interactive educational documents with a variety of flexible tools which ease the author's burden of dealing with the technical issues of the software environment.

This article gives an overview about the project, and following articles are planned and will cover technical details and implementation issues of the Controls.

Introduction

In the last years electronical tools entered the education of mathematics in different ways. Beside graphical pocket calculators computer algebra systems such as Derive, Maple, Mathematica or MuPAD as well as special mathematical software like Euclid or Mathview are nowadays used in schools and universities for educational purposes.

However, none of these systems provides solutions to *all* the different requirements in teaching mathematics in schools and universities. Presently teachers and students are forced to handle more than one tool simultaneously which is a serious technical challenge for both authors as well as readers.

Another disadvantage of the existing systems is their limited suitability for writing interactive electronical mathematical documents. They do not offer ways to adapt the user interface or to adapt the mathematical functionality of the mathematical software according to didactical needs. E.g., if the problem of inverting a matrix is given as an exercise to the students, the author might want to switch off the built-in function `inverse` of the underlying algebra system and to lead the student through the various steps to be performed by the corresponding algorithm. Further, existing authoring tools provide only limited communication between different controls of the electronical document. E.g., when an algebraic expression is visualized by a graphical tool, the author might want to let the corresponding graph be redrawn *automatically* upon changes of the algebraic input.

[1] SciFace Software GmbH&Co. KG, Germany, `http://www.sciface.com`.

Presentation tools such as Microsoft PowerPoint, Internet Browsers or professional authoring tools such as Toolbook[2], Director or Authorware[3] are presently used for writing educational mathematical documents. Unfortunately, none of the mathematical software packages mentioned above can be integrated into these tools in a simple way. This forces the author either to develop his or her own controls (a quite expensive and inefficient way to go) or ultimately to do their work without these mathematical tools.

A flexible combination of powerful authoring tools and powerful mathematical controls designed for educational purposes is presently missing.

The Mathematical GUI Controls

Presently Windows 95 and Windows NT are becoming more and more important in schools and universities. Powerful authoring tools such as Authorware or Toolbook are supported by these systems. The ActiveX technology under Windows 95 and Windows NT [Ar97] provides a technical concept for the development of flexible controls which can be integrated into these authoring tools or into software packages like internet browsers, typesetting software like Microsoft Word, and so on. Each of these Controls defines an interface of functions which can be used from outside. For example, a Control for visualization could offer a function for drawing a graph for a given function, or offer functions for manipulations such as rotating or zooming.

The author uses Java, C++, Visual Basic or the script language of the authoring tool to access the interface functions and to implement the communication between these Controls.

Using this powerful technology we are developing three ActiveX Controls under Windows 95 and Windows NT:

1. a Control for input and output of mathematical expressions,
2. a Control for mathematical computations based on a special kernel of the computer algebra system MuPAD [Fu96, OePoRuWe98],
3. a graphical Control for visualizing and manipulating three-dimensional objects.

The mathematical Control is based on special library packages developed in MuPAD. The integration of such a powerful computer algebra system into authoring software will provide tools for writing highly flexible and interactive electronical mathematical documents.

Based on these tools interactive electronical documents for teaching Linear Algebra at schools and universities will be developed by Prof. Dr. Wolfgang Fraunholz et al.[4] in cooperation with SciFace Software and the Cornelsen Verlag[5].

Writing about the ideas and the development of this course on Linear Algebra would be beyond the scope of this article. We focus on the specifications of the three ActiveX Controls which are described in the following subsections.

A Control for Input and Output of Mathematical Expressions

As part of the project, an editor for entering and modifying mathematical expression is developed. On the first sight it resembles well-known math editors like the one included in the Microsoft Office distribution. By using

[2] Toolbook is a product of Asymetric Inc.: `http://www.asymetrix.com/`.
[3] Director and Authorware are products of Macromedia Inc.: `http://www.macromedia.com/`.
[4] Mathematisches Institut und Institut für Mediendidaktik, University of Koblenz, Germany.
[5] Cornelsen Verlag Berlin, Germany, `http://www.cornelsen.de`.

keyboard shortcuts, menus and the mouse, arbitrary mathematical expressions like for example fractions, sums, integrals or matrices can be entered easily. While typing, the new (sub-)expressions are entered by applying the so-called overlay concept [So91] (in contrast to the template concept used in many other math editors). The following example demonstrates how the overlay concept allows to type a mathematical expression in a natural, left to right manner:

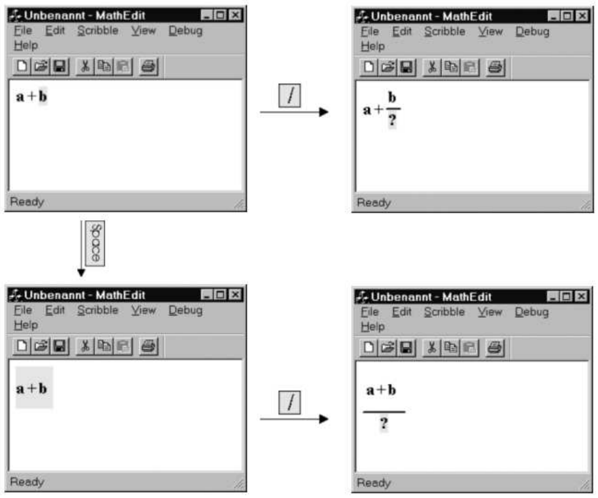


Figure 1: How to create the terms $a + \frac{b}{c}$ and $\frac{a+b}{c}$.

The most important feature of the math editor is however, that it is not a simple *layout* editor but an editor for entering *syntactically correct* mathematical *structures*. This means that the editor "knows" about the mathematical meaning of what is typed. It is not possible to enter syntactically wrong expressions like for example an expression with missing closing brackets or a definite integral with only an upper limit. In fact, the user does *not* enter mathematical *notation* while typing, but mathematical *content*. The editor only decides which layout is appropriate for which mathematical (sub-)expression and then draws the whole expression accordingly. Which layout is used for which mathematical "function"[6] is decided by looking up so called "templates". A template specifies how a particular mathematical expression has to be drawn. For example a fraction (_divide) is drawn by two vertical boxes which contain the denominator and the numerator and a line which adjusts its length depending on the size of the arguments of the whole fraction. The following table shows some mathematical functions and the corresponding templates.

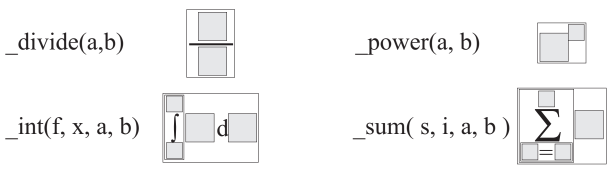


Figure 2: Layout templates for some mathematical functions.

The templates used for the various operators and functions are not hard coded into the editor. In fact a lot of standard templates are compiled in to allow the editor to run standalone. But in combination with a MuPAD kernel it is possible to configure the editor. New templates may be defined and the existing compiled in templates may be overridden by user defined templates.

[6]Here, the term "function" means anything that could be written in functional notation. The operator +, for example, could be written as _plus(x_1,x_2,...,x_n). In fact, most computer algebra systems represent all mathematical operators as functions.

So the user can change any template to suit his or her needs. A simple example is the notation of simple mappings. In school, mappings are usually written in a notation like $f(x) = x^2$. However someone might prefer a notation like $f : x \mapsto x^2$. In our math editor new templates can be easily introduced. Once the editor knows about the new template, all occurences of the according mathematical object are drawn using the new template.

Figure 3: notation of functions: $f : x \mapsto f(x)$ versus $y = f(x)$.

In addition to simply drawing mathematical expresssions, the math editor “knows” about how and where parantheses have to be set. All operators (functions) have priorities and the templates specify where parentheses need to be set. An example is the expression A^B where A and B may be any subexpressions. If the base A is an expression with *lower* priority as the exponentiation operator, it is automatically nested in parantheses. For example, if A equals the subexpression $x + y$, the whole expression is automatically drawn as $(x + y)^B$. The exponent B however is usually not enclosed in parantheses. For example, if B equals $x + y$ the editor draws the whole term as A^{x+y}. The math editor sets parantheses automatically where they are needed. Of course, the user may add own parantheses if he/she wishes to.

Another important feature of the editor is importing and exporting mathematical expressions to other formats like for example MathML code. An important issue for future uses of the math editor is the integration of mathematical expressions in web-documents. The recommendation of the World-Wide-Web Consortiums for displaying math on the internet is the Mathematical Markup Language MathML [W3C] which can be seen as an extension to the well known HTML standard. MathML allows mathematical expression to be specified either by content markup or as presentation markup. Content markup preserves the mathematical meaning of the expression, leaving the concrete layout to the web browser while presentation markup discards the mathematical structure and specifies exactly how the expression looks like. For example, the expression $\frac{1}{b+c}$ written in content markup looks like:

```
<apply>                 // Apply the following tag as a function
   <divide/>            // Function: divide
   <mn>1</mn>           // First argument is the number 1
   <apply>              // Second argument is another function
      <plus/>           // Subfunction is plus
      <mi>b</mi>        // 1st argument of subfunction is identifier b
      <mi>c</mi>        // 2nd argument of subfunction is identifier b
   </apply>             // End of subfunction
</apply>                // End of function
```

Written in presentation markup, the same expression looks like:

```
<mfrac>                  // Fraction of two subexpressions
   <mn>1</mn>            // Upper subexpression: Number: 1
   <mrow>                // Lower subexpression: row of subexpressions
      <mi>b</mi>         // First element in the row: identifier b
      <mo>+</mo>         // Second element in the row: operator +
      <mi>c</mi>         // Third element in the row: identifier c
   </mrow>               // Closing tag for row
</mfrac>                 // Closing tag for fraction
```

The math editor developed in our project can import and export both MathML content and presentation. Based on the editor it should be easily possible to develop plugins for the usual web browsers which can be used to create and display mathematical expressions on the web.

As a conclusion, the math editor developed is suitable for entering and modifying mathematical expression while preserving their meaning (therefore, the expression may be further manipulated by a computer algebra system). How mathematical expressions are drawn is fully user definable by using so called layout templates. The editor maintains syntactically correct expressions and sets paranthesis automatically where needed. Import and export to and from MathML allow future uses for web display.

A Control for Mathematical Computations

The Control for mathematical computations integrates the Control for the input and output of mathematical expressions (see above) and communicates with a special kernel and with libraries of the computer algebra system MuPAD.

The Control consists of the following elements:

- an editor for the input and output of mathematical expressions,
- operations being applied to expressions or selected (sub-)expressions,
- results from operations,
- assignments of expressions or subexpressions to variables,
- text representing user information, comments, warnings or error messages.

The reader chooses operations using templates or textual commands. Templates can be organized in menus or palettes designed by the author. The Control itself does not provide such input elements. This allows the author to choose the operations to be made available to the reader for each instance of this Control.

The reader can select an expression or a certain subexpression and apply operations to the selected argument. The selection states the *main argument* of an operation. The meaning of the main argument is defined by the operation itself. In addition to this main argument possible further arguments are given by the reader by textual input or by copying existing (sub-)expressions of the current computation.

Operations act as *copy and replace* operations. This means that the result of a computation step is a copy of the expression where the selected subexpression of the original expression was substituted by the result of the operation. The resulting expession is displayed under the original expression. The result of the computation step is not being simplified.

Starting a computation the results of following copy and replace operations can be organized in a right sided tree. Following a branch of this tree each node is a valid consequence of the operations being applied before. If an input (e.g., a mathematical expression or a statement for a definition of a variable) was changed, subsequent results will be marked as (possibly) invalid as long as they were re-evaluated in the new context.

Wants the reader use an (sub-)expression to start a sub-computation he or she has to copy the (sub-)expression to a region below the current computation.

Each operation is defined as a sequence of MuPAD functions. The author defines the name and output of an operation, defines its main argument and possibly further arguments, and build a template of this operation which helps the reader to select the operation. The reader usually does not see the corresponding MuPAD functions behind a certain operation.

The MuPAD functions can be organized in special MuPAD libraries offering functions for step-by-step computations, for checking the readers results, for giving information about computational steps and so on. Hence the flexibility and power of the mathematical features of the electronical document mainly depends on the underlying MuPAD libraries. The MuPAD system gives the author a powerful mathematical programming environment which provides various mathematical data structures (such as numbers, strings, lists, sets, arrays, tables, matrices, ...) and a variety of powerful mathematical functions such as gcd computations, factorization of polynomials, differentiation and integration of functions, solving of arbitrary equations and inequalities, etc.

The reader and author can give names to (sub-)expressions or regions. Each instance of the Control defines a new *namespace* for names of (sub-)expressions. Hence names are kept uniquely for each instance of the Control.

The interface of the Control offers a function where the author can ask for the value of such a name. This allows the author for example to use this expression on following pages of the electronical document, to check the correctness of the readers computation, or to plot a graph of the expression (see the following subsection).

A Control for Three-Dimensional Graphics

The third Control developed in the project is a graphical tool for the visualization and manipulation of three-dimensional (3D) objects.

The Control offers graphical primitives for points, vectors, plains, spheres, cones, function plots and so on. These objects can be collected in graphical scenes, they can be selected and manipulated, rotated and zoomed. The reader can ask for the 3D-coordinates of a point, the distance between two points, he can change the coordinate system and much more.

The graphical objects are also defined in MuPAD libraries (see above) and can be manipulated in an algebraic way. Some graphical operations will be mapped to their corresponding algebraic operations in the computer algebra system. For example, the operation “rotate an object around the axes” can be mapped to a matrix representing the geometrical operation.

The interface of the Control will offer functions for drawing graphical primitives, performing operations on them, setting options and default values and so on. This enables the author to adapt each instance of the Control to his or her needs.

The graphical Control uses the MAM/VRS (Modeling and Animation Machine/ Virtual Rendering System) toolkit [DoHi98]. MAM/VRS is a toolbox providing an efficient high-level interface to 3D rendering and modeling features. As the underlying rendering system OpenGL is used. This ensures high visual quality for the graphical rendering of the 3D objects.

Summary and Prospects

We develop three ActiveX Controls under Windows 95/Windows NT for the development of electronical mathematical documents: An editor for input and output of mathematical expressions, a Control for mathematical computations based on a special kernel of the computer algebra system MuPAD and a graphics tool for visualization and manipulation of three-dimensional objects.

These Controls can be integrated into presentation tools such as Microsoft PowerPoint, Internet Browsers or professional authoring tools such as Toolbook, Director or Authorware.

The Controls define interfaces which enable the author, using a programming language like Java, C++, Visual Basic or the scripting language of an authoring tool, to adapt *each instance* of such a Control to his or her needs.

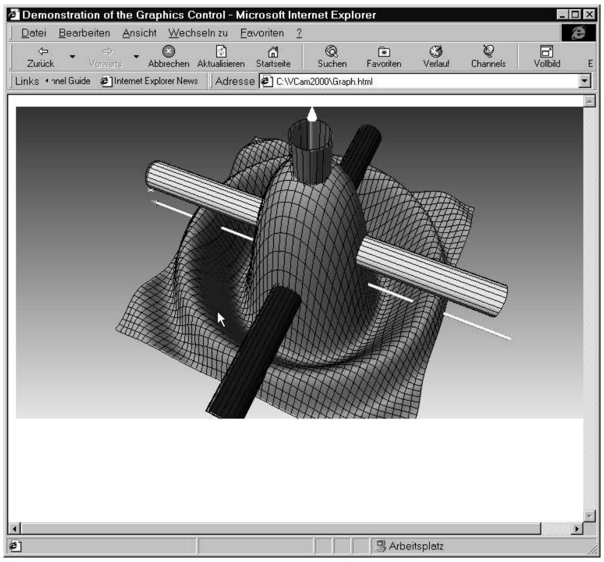


Figure 4: The graphical Control integrated into the Microsoft Internet Explorer.

The author can, for example,

- put together palettes of templates for the input of mathematical expressions which should be available to the reader,
- define the mathematical operations the reader should use for the mathematical computations in a certain context,
- ask for a result from the reader and, for example, check whether the result is correct or not,
- visualize the result of a computation,
- compute and display a possible solution of a computation to the reader,
- set options and default values for a Control,

and more.

We note that the Control for mathematical computations combines the power of the underlying computer algebra system with the demand for special user interfaces for educational purposes.

By replacing or adding new Controls (e.g., a Control for two-dimensional geometry or for visualizing solutions of ordinary differential equations) further electronical documents can be written without having to develop each Control from scratch: standard tools such as the mathematical Control or the Control for input and output of mathematical expressions can be re-used and will be part of most documents. Also existing documents can be updated or extended by modifying or extending the underlying MuPAD library. This significantly reduces the amount of time and money for the production of electronical documents.

The target group of these Controls are authors and publishers of electronical mathematical documents. However, in order to profit from the flexibility of the Controls the author must be familiar with the use of ActiveX Controls and the scripting language of the chosen presentation software or authoring tool. This, of course, is a high demand on the technical knowledge of the author. In addition, to profit from the power of the computer algebra system he or she also has to be familiar with its use and its programming language.

Default palettes, default instances of Controls and special MuPAD libraries will be provided to ease the author's burden in using and defining instances of the Controls.

Future development could include the design of an easy-to-use authoring tool which does not have the power of professional authoring tools but is specially designed for authors with no or limited experience with Visual Basic, Java, C++ or an authoring scripting language. This would give teachers in schools and universities the chance to write their own electronical documents using the powerful mathematical Controls developed in our project.

References

[Ar97] TOM ARMSTRONG *Designing and Using ActiveX Controls* M&T Books, New York, 1997. ISBN 1-55851-503-8

[DoHi98] JÜRGEN DÖLLNER AND KLAUS HINRICHS *Introduction to MAM/VRS* University of Münster, `http://wwwmath.uni-muenster.de /math/inst/info/u/mam/tutorial/`, 1998.

[Fu96] THE MUPAD GROUP, BENNO FUCHSSTEINER ET AL. *MuPAD User's Manual - MuPAD Version 1.2.2* John Wiley and sons, Chichester, New York, 1996

[OePoRuWe98] W. OEVEL AND F. POSTEL AND G. RÜSCHER AND S. WEHMEIER *Das MuPAD–Tutorium* SciFace Software, 1998. ISBN 3-933764-00-9

[So91] NEIL MORRELL SOIFFER *The Design of a User Interface for Computer Algebra Systems* pp. 118–125, Ph.D. thesis, University of California, Berkeley, 1991

[W3C] W3C *Mathematical Markup Language (MathML)* W3C's Math Home Page, `http://www.w3.org/Math/`. 1998

`DETools`: A Library for Differential Equations

Werner M. Seiler
Lehrstuhl I für Mathematik
Universität Mannheim, 68131 Mannheim, Germany
`seiler@euler.math.uni-mannheim.de`
`http://iaks-www.ira.uka.de/iaks-calmet/werner/werner.html`

We present a preview on a new MuPAD library called `detools` containing a number of routines for symbolically (and numerically) treating differential equations. This includes especially a package for Lie symmetry analysis.

Introduction

The next release of *MuPAD* will contain a number of new features for differential equations. Some of them are already contained in the current version 1.4.1, although only in an undocumented and experimental form. As an "official" documentation in the usual *MuPAD* format is still missing, this article tries to give at least a brief introduction.

The *MuPAD* library is extended on two levels. The first one consists of a new library `detools` containing a number of routines for treating differential equations. This includes support for the graphical presentation of the output of the numerical routines in *MuPAD*, some methods for analysing or generating differential equations and also routines for solving some classes of partial differential equations. The use of this new library will be described in this article.

The second level is somewhat more advanced and requires a certain familiarity with the object-oriented `domains` package of *MuPAD* [5]. Thus it addresses mainly programmers who want to implement sophisticated algorithms, but "ordinary" *MuPAD* users can profit of it, too. This level contains categories and domains providing unified data structures for differential equations. Superficially seen, this results just in more concise and comfortable in- and output. However, one of the most important aspects of providing such a programming environment is that it facilitates the simultaneous usage of different packages written by different authors.

There exist many computer algebra packages for differential equations implementing algorithms for many different tasks. As a simple example we may take the Lie symmetry analysis of a physical theory. Thus our starting point is a Lagrangian and the first step is the derivation of the corresponding Euler-Lagrange equations. The second step consists of setting up the determining system for the symmetry generators of these equations. As determining systems are usually rather large and overdetermined, one will probably need some automatic simplification and completion of this system. As fourth step one might try to automatically solve the determining system. In the fifth step one would like to use the found symmetry generators for symmetry reduction, preferably to some ordinary differential equations which should finally be solved automatically, too.

For each of these six steps there exist computer algebra packages (several are actually implemented in `detools`). However, they have usually been written by different authors. Furthermore, these authors may have had different tasks in their minds and not necessarily precisely the sequence of steps we described above. In any case, chances are high that each of the authors implemented his/her own data structures for differential equations, as no computer algebra system provides standardised ones.[1] If one wants to take the output of one package as input for another one, one must usually first write some conversion routines between these different data structures. This problem can be almost completely avoided by using the object oriented approach of the `domains` package of *MuPAD*.

The more advanced parts of the `detools` library are actually implemented using the categories and domains of the second level. Some of the library routines represent only interfaces taking input in the standard notation of *MuPAD* (i. e. `diff` or `D` for representing derivatives) and converting it to a suitable domain. Then the real computations take place within this domain and the output is finally converted back into the standard notation. This strategy allows users who have no experiences with domains to use the more advanced algorithms. But obviously the conversions need some time and thus the user can save computation time, if (s)he directly uses the appropriate domains for differential equations.

Details of the categories and domains implemented in the second level will be described elsewhere. This article concentrates on the methods contained in the library `detools` and their standard usage. Users familiar with the domains of the second level can however easily guess the correct arguments: whereas in the standard usage it is necessary to explicitly state the names of the independent and dependent variables, this information is already encoded in the domains, if one enters corresponding objects. Thus one may omit these arguments.

The `detools` Library

The methods of the `detools` library can roughly be grouped into the following five categories: (a) *solving* partial differential equations, (b) *generation*, *manipulation* and *analysis* of differential equations, (c) *visualisation* of numerical results, (d) support for *numerical analysis*, and (e) *symmetry analysis*. With the exception of the last one each of these groups will now be described in a separate subsection. The symmetry package is discussed in the next section.

In the sequel we will always assume that the methods of the `detools` library have been exported with the command `export(detools)`, i. e. we will write `method` instead of `detools::method`. As usual a list of all the methods included in the library can be obtained with `info(detools)`.

Solving Partial Differential Equations

The method `pdesolve` can determine the solution of some classes of partial differential equations. In this first version of the `detools` library it can handle essentially only *linear* equations, but a number of extensions are already planned. This includes especially general first order equations.

The solution of (the initial value problem for) linear first order equations is comparatively easy, as it can be reduced to solving a system of ordinary differential equations, the so-called *characteristic system*. As this system is in general non-linear, it may of course still be a formidable task to solve it. This also implies that the ability of `pdesolve` to find the solution depends decisively on the power of the *MuPAD* solver for ordinary differential equations (incorporated in the standard `solve` command). `pdesolve` tries to return an explicit expression for the solution. However, the method of characteristics yields the solution in a parametrised form and it might not always be possible to eliminate the parameters or to solve for the dependent variables in terms of the independent ones. In this case, `pdesolve` returns an implicit solution.

In the following *MuPAD* session the constant coefficients equation $2u_x + u_y + 3u_z - 2u = 0$ where $u = u(x, y, z)$ is solved subject to the initial condition that on the plane parametrically described by $x(s,t) = 2s$, $y(s,t) = 3t$,

[1] This is only reasonable, as for different tasks different data structures are optimal.

$z(s,t) = s+t$ the unknown function takes the value $u(s,t) = s-t$. `pdesolve` takes here five arguments. The first one is the differential equation; the second one consists of a list containing the names of the independent variables; the third one is a similar list for the dependent variables; the fourth argument is a set containing the Cauchy surface in parametric form and the fifth one is a set with the names of the parameters.

```
MuPAD
>> detools::pdesolve(2*u([x])+u([y])+3*u([z])-2*u,[x,y,z],[u],
>>                   {x=2*s,y=3*t,z=s+t,u=s-t},{s,t});
Output
                   / 7 x   y   2 z \    /   3 x   2 y   6 z  \
               u = | --- - - - --- | exp| - --- - --- + ---  |
                   \ 10    5    5  /    \    5     5     5   /
```

In this example we indirectly used our domains for differential equations in order to simplify the input. Namely, we simply entered `u([x])` instead of the much more cumbersome `diff(u(x,y,z),x)`. We will continue to use this simplified input, as it is so much easier and does not require any knowledge of the domains involved. The rules are simple: for the dependent variables there is no need to enter the arguments (but it is not harmful, if one does it); for derivatives just give a list of the independent variables with respect to which one differentiates as argument of the dependent variable.

Mathematical details on the method of characteristics can be found in any standard textbook on partial differential equations. Characteristics are also of importance for the classification of linear partial differential equations. Therefore the method `characteristics` is provided with which it is possible to work directly with the characteristics. Its arguments are very similar to `pdesolve`. The first three ones are identical; the fourth one gives the name of an identifier used to parametrise the characteristic curves; the fifth one gives again the initial conditions in parametrised form.

The following command determines the characteristic curves for the variable coefficients equation $2u_x + xu_y + 3u_z - u = 0$ with the same initial conditions as above and `r` as curve parameter.

```
MuPAD
>> detools::characteristic(2*u([x])+x*u([y])+3*u([z])-u,[x,y,z],[u],r,
>>                         {x=2*s,y=3*t,z=s+t,u=s-t});
Output
                            2
{{y(r) = 3 t + 2 r s + r , u(r) = exp(r) (s - t), x(r) = 2 r + 2 s,

   z(r) = 3 r + s + t}}
```

It is also possible to use `characteristics` to set up the characteristic system without trying to solve it. In this case no initial conditions should be given; instead the option `Solve=FALSE` must be specified. The characteristic system is returned as an object of type `ode` (see the online documentation).

```
MuPAD
>> detools::characteristic(2*u([x])+x*u([y])+3*u([z])-u,[x,y,z],[u],r,Solve=FALSE);
Output

ode({diff(u(r), r) = u(r), diff(y(r), r) = x(r), diff(x(r), r) = 2,
```

```
  diff(z(r), r) = 3}, {u(r), x(r), y(r), z(r)})
```

Currently `characteristics` can handle only single first order equations. But an extension to linear systems and higher order equations is under development.

With the option `Polynomial=[N,str]` `pdesolve` is instructed to look only for polynomial solutions. Currently, this is only implemented for linear systems with constant coefficients. `pdesolve` further assumes that the given system is complete or involutive, i. e. that it contains already all its non-trivial integrability conditions (that is an issue only for over-determined systems). The integer `N` prescribes the maximal degree of the polynomials; the string `str` is used for the names of the coefficients of the polynomials. `str` can be omitted; in this case `ä"` is used as default value and the option is simplified to `Polynomial=N`.

In the following example polynomial solutions of maximal degree 3 of the heat equation $u_t = u_{xx}$ are determined.

```
MuPAD
>> detools::pdesolve(u([t])-u([x,x]),[x,t],[u],Polynomial=[3,c"]);
Output
          / / c5 \  3   / c6 \  2                                     \
      poly| | -- | x  + | -- | x  + c5 x t + c3 x + c6 t + c1, [x, t] |
          \ \ 6  /      \ 2  /                                        /
```

Note that `pdesolve` returns here an object of type `DOM_POLY` (with coefficient ring `Expr`). The unknowns are just the independent variables as passed to `pdesolve`. In our example the solution contains four arbitrary parameters `c1`, `c3`, `c5`, `c6`.

Generating, Manipulating and Analysing PDEs

The method `euler` implements the *Euler operator* for Lagrangians of arbitrary order and of an arbitrary number of independent and dependent variables. Thus it allows to determine the *Euler-Lagrange equations* for variational problems. This is for example of great importance in physics where the equations of motion of almost everything from a simple mechanical system to a gauge theory describing the interaction of elementary particles are derived via variational principles.

Let

$$I[\phi_\alpha] = \int \mathcal{L}\left(x_i, \phi_\alpha(x_i), \frac{\partial^{|\mu|}\phi_\alpha}{\partial x^\mu}(x_i)\right) dx \tag{6}$$

be a functional of the functions $\phi_\alpha(x_i)$ where $\alpha = 1, \ldots, m$ and $i = 1, \ldots, n$ defined by the Lagrangian $\mathcal{L}$ depending on the independent variables x_i, the functions ϕ_α and their partial derivatives up to some order q. $\mu = [\mu_1, \ldots, \mu_n]$ represents a multi-index of length $|\mu| = \sum \mu_i$ in the usual notation of multivariate calculus. Then a necessary condition for the functions ϕ_α to yield an extremum of the functional I is that they satisfy the Euler-Lagrange equations:

$$\sum_{a=0}^{q}(-1)^a \sum_{|\mu|=a} \frac{\partial^a}{\partial x^\mu}\left(\frac{\partial \mathcal{L}}{\partial \frac{\partial^a \phi_\alpha}{\partial x^\mu}}\right) = 0\,. \tag{7}$$

The left hand side of (7) defines the component α of the Euler operator $E[\mathcal{L}]$. Generally the Euler-Lagrange equations form a system of m equations of order $2q$; however in special cases the order of at least some equations may be lower.

The method `euler` takes as arguments the Lagrangian $\mathcal{L}$, the list of the independent variables x_i and the list of the dependent variables or fields ϕ_α. The output consists of a list with the components of the Euler operator as defined in (7). As a first example we derive the equations of motion for a particle moving in a central force field. Its coordinates are given by (x, y) and the time variable is denoted by t. As potential we simply take $V(x, y) = 1/r$ with $r^2 = x^2 + y^2$. Thus we need the Euler-Lagrange equations for the Lagrangian $L = (\dot{x}^2 + \dot{y}^2)/2 - 1/r$.

```
MuPAD
>> detools::euler((x([t])^2+y([t])^2)/2-1/sqrt(x^2+y^2),[t],[x,y],Expr=NoDiff);
Output
   --                 x                            y        --
   |  x([t, t]) - ------------, y([t, t]) - ------------   |
   |                2    2 3/2                2    2 3/2   |
   --             (x  + y )                 (x  + y )      --
```

Here we used again our simplified notation for the input. Alternatively, we could have written `diff(x(t),t)` instead of `x([t])`. The option `Expr` specifies that we want the output in the basic type `DOM_EXPR`. By using `Expr=NoDiff` instead of `Expr=TRUE` we furthermore required the output in our simplified notation instead of the standard notation using `diff`. As an example for a field theory we take the ϕ^4-theory in $(1 + 1)$ dimensions (this time without `NoDiff`).

```
MuPAD
>> detools::euler((Phi([t])^2-Phi([x])^2)/2-Phi^4/4,[t,x],[Phi]);
Output
                                                                3
   diff(Phi(t, x), t, t) - diff(Phi(t, x), x, x) + Phi(t, x)
```

The two methods `hasHamiltonian` and `hasPotential` check whether a given (autonomous) system of first order ordinary differential equations (or better: a given vector field) has certain properties, namely in the first case whether it is a Hamiltonian system and in the second case whether it is a gradient system, i. e. a system derivable from a potential. In both cases the call returns a list of integrability conditions which must be satisfied for the property in question.

In a first example we check whether the harmonic oscillator is a Hamiltonian system. Thus we take the differential system $\dot{q} = p$, $\dot{p} = -q$ where q denotes the space and p the momentum coordinate. `hasHamiltonian` requires as arguments three lists: the first one contains the components of the vector field, i. e. the right hand sides of the differential equations; the second one gives the names of the space coordinates and the third one the names of the momentum coordinates.

```
MuPAD
>> detools::hasHamiltonian([p,-q],[q],[p]);
Output
                              []
```

As an empty list is returned, the system is unconditionally Hamiltonian. Next we study a system with two degrees of freedom, namely $\dot{x} = p_x$, $\dot{y} = p_y$, $\dot{p}_x = F(x, y)$ and $\dot{p}_y = G(x, y)$ where F and G are yet arbitrary functions of their arguments.

```
MuPAD
>> detools::hasHamiltonian([px,py,F(x,y),G(x,y)],[x,y],[px,py]);
  Output
                  [diff(F(x, y), y) - diff(G(x, y), x)]
```

Now the result indicates a condition on the functions F and G, if the system is to be Hamiltonian. This condition $F_y = G_x$ just says that the force vector $(F, G)^t$ must be the gradient of a potential. Thus it is not surprising that in the following example we obtain exactly the same results.

```
MuPAD
>> detools::hasPotential([F(x,y),G(x,y)],[x,y]);
  Output
                  [diff(F(x, y), y) - diff(G(x, y), x)]
```

Note that `hasPotential` requires only two arguments, as we have here no distinction into space and momentum coordinates. One should furthermore remark that `hasHamiltonian` checks only whether the system is Hamiltonian with respect to the canonical symplectic structure of phase space.

Visualisation

The two methods `odeplot` and `phaseplot` both take the output of the routine `numeric::odesolve` with the option `Alldata` (for a detailed description of this and other numerical routines see the online documentation) for numerically solving first order systems of ordinary differential equations and represents it graphically using the `VCam`, *MuPAD*'s graphics tool.

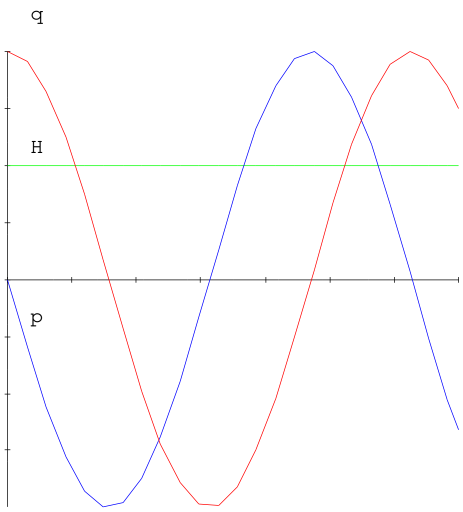
`odeplot` plots components of the solution vector over the independent variable. The components are given as a list of indices; their number can be arbitrary. Additional graphics options may be given in order to obtain a nicer picture. As third argument a list with object options may be given; as fourth argument a list with scene options. The list for the object options should actually be a list of lists; each sublist containing the object options for one of the generated curves.

As a simple example we take again the harmonic oscillator. We first integrate its equation of motion with a Runge-Kutta-Fehlberg method of order 8, then we plot position and momentum. The option `Plot` given as last argument to `odeplot` specifies that the plot should actually be rendered. Without it only a list of graphic objects would be returned which can be combined with other ones for more complex diagrams (see below for an example).

```
MuPAD
>> F := proc(t,y) begin [y[2],-y[1]] end_proc:
>> H := proc(t,y) begin 0.5*(y[1]^2+y[2]^2) end_proc:
>> nsol := numeric::odesolve(0..7,F,[0,1],RKF78,Alldata):
>> detools::odeplot(nsol,[1,2,H],
>>         [[Title="q",TitlePosition=[2.5,0.6],Color=[Flat,[1,0,0]]],
```

```
>>          [Title=p",TitlePosition=[2.5,5.6],Color=[Flat,[0,0,1]]],
>>          [Title="H",TitlePosition=[2.5,2.8],Color=[Flat,[0,1,0]]]],
>>         [Axes=Origin,Ticks=7,Title="Harmonic Oscillator",TitlePosition=Below,
>>          Scaling=UnConstrained,FontSize=15,ForeGround=[0,0,0],BackGround=[1,1,1]],
>>         Plot):
```

Graphics



Harmonic Oscillator

odeplot is also able to plot functions of the solution vector. For this a procedure is passed instead of an index. This procedure must take as input one float and one float vector of the dimension of the solution vector and return as output a simple float. In our example above we used this to check visually that the numerical integration preserved the energy. We defined a procedure H computing the energy at a given point of the trajectory and indeed the corresponding curve is a horizontal line.

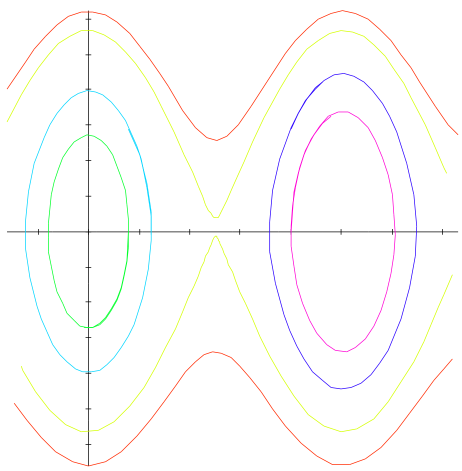
phaseplot generates phase diagrams, i. e. it plots one or two components of the solution over another component (in the case of two components a 3D plot is generated). Again the components are specified as indices and like for odeplot it is also possible to pass a procedure instead of an index. The specification of plot options is similar to odeplot with the sole difference that only one list of object options can be passed. We will now use a sequence of calls of numeric::odesolve and phaseplot (without the option Plot) for various initial conditions to generate the phase portrait of a pendulum.

MuPAD

```
>> F := proc(t,y) begin [0.5*y[2],-sin(y[1])] end_proc:
>> ICs := [[10,[-2,2]],    [16,[-2,1.54]],    [12,[1,0]],    [12,[1,1.45]],
>>         [12,[5,1.45]],    [12,[5,0]],    [8.5,[9,-1.8]],    [19,[9,-0.6]]]:
>> nsol := map(ICs,
>>     proc(l,f) begin numeric::odesolve(0..l[1],f,l[2],RKF78,Alldata) end_proc,
>>     F):
>> out := map(nsol,detools::phaseplot,[1,2]):
>> plot2d(Axes=Origin,Title=Pendulum Phase Portrait",TitlePosition=Below,
```

```
>>      FontSize=15,ForeGround=[0,0,0],BackGround=[1,1,1],Scaling=UnConstrained,
>>      op(out)):
```

Graphics



Pendulum Phase Portrait

Note that `odeplot` and `phaseplot` are not very sophisticated methods. They will simply plot the points as computed by `numeric::odesolve`. No attempt is made to smoothen the resulting curves. It is assumed that in regions where the solution rapidly changes `numeric::odesolve` has taken sufficiently small steps to produce a reasonable output. Thus the smoothing is basically "controlled" by the choice of the numerical method and by the setting of the variable `Digits`.

Numerical Analysis

As already mentioned above `numeric::odesolve` implements a number of methods for the numerical solution of systems of ordinary differential equations. The new library `detools` contains a procedure `modode` which supports the analysis of such methods.

`modode` constructs a so-called *modified* equation for a given numerical method for solving ordinary differential equations. Modified equations represent a powerful tool for the backward error analysis of a numerical methods; a detailed introduction can be found in [7]. Loosely spoken, given a differential equation and an approximate solution of it computed with some numerical method, a modified equation is another differential equation whose exact solution is closer to the approximate solution than the exact solution of the original equation.

Thus the modified equation may be considered as a continuous model for the application of the numerical method to the original equation. Besides other possibilities it allows to study whether the numerical method preserves the qualitative features of the original equation. For example, given a Hamiltonian system and a numerical method to integrate it, one can check whether the modified equation is also a Hamiltonian system. In general, this will only be the case, if a symplectic integrator is used.

modode computes modified equations in the form of a Taylor polynomial in the step size h of the numerical method. Its implementation is based on the MAPLE program present in [1]; however, it contains some minor improvements which make it more efficient. The algorithm uses a brute force Taylor series expansion and is rather straight-forward. We consider in this article only explicit one step methods which can be written in the form

$$\vec{y}_{n+1} = \vec{y}_n + h\vec{\Psi}(h, t, \vec{y}_n) \tag{8}$$

where $\vec{y}$ denotes the vector of the unknown functions, t the independent variable and h the step size. For the analysis of implicit methods or for the incorporation of variable step sizes see [1].

There are two ways to use modode. The generic way is to pass the following arguments: (i) a list containing the components of $\vec{\Psi}$, (ii) a list with the dependent variables, i. e. the components of $\vec{y}$, (iii) the independent variable t, (iv) the step size h and (v) the order N up to which the series for the modified equation should be evaluated. The following example computes the modified equation for the Euler method $y_{n+1} = y_n + hf(t, y_n)$ applied to the simple equation $\dot{y} = f(t, y) = y$ up to order 4. Note that although we are treating a single equation here, we must pass it in form of a list like for a system. Similarly, the output is always in form of a list.

MuPAD

```
>> detools::modode([y],[y],t,h,4);
```

Output

```
--               2       3       4    --
|       h y   h  y    h  y    h  y    |
|   y - --- + ---- - ---- + ----     |
--       2      3       4       5     --
```

The second possibility makes use of the symbolic option of numeric::odesolve. Thus modode let *MuPAD* derive the function Ψ appearing in (8). One passes now as arguments a procedure encoding the differential equation as it is expected by numeric::odesolve and a string with the name of the numerical method one wishes to analyse. Obviously, this approach is restricted to the methods implemented in the numerics library.[2] Thus the following call produces exactly the same result as the previous one.

MuPAD

```
>> F := proc(t,z) begin [z[1]] end_proc:
>> detools::modode(F,EULER1,[y],t,h,4);
```

Note here that one can give in the call of modode new names for the dependent variables independent of those used in the procedure F. While this procedure always requires a list as second argument (z in our example), we can later on give individual names to the components of the vector (y for z[1]).

Finally, we want to combine modode with odeplot in order to show that the modified equation is indeed a better approximation to the numerical solution than the original equation. As in [1] we take as example the equation $\dot{y} = y^2$. For the initial condition $y(0) = 1$ its exact solution is $y(t) = 1/(1-t)$, as one can easily compute with solve. With the following *MuPAD* commands we generate a plot comparing the exact solution, the numerical solution obtained by the Euler method with step size $h = 0.1$, and a high accuracy numerical solution of the fourth order modified equation for this problem.

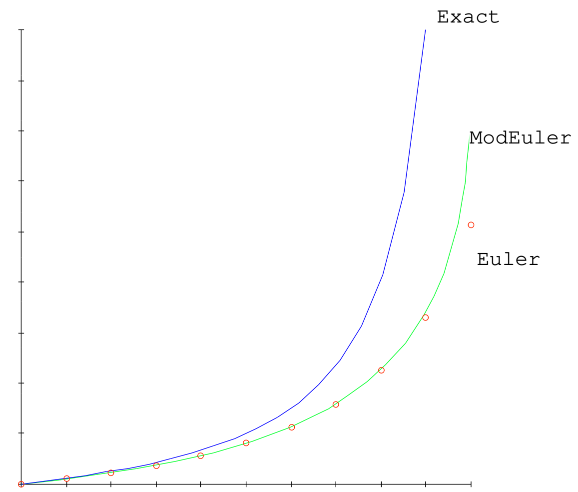
MuPAD

```
>> F := proc(t,y) begin [y[1]^2] end_proc:
```

[2]For further details and for the methods available see its online documentation.

```
>> eulersol := numeric::odesolve(0..1,F,[1],EULER1,Stepsize=0.1,Alldata):
>> euler := [Mode=List,map(eulersol,((l)->point(l[1],l[2][1]))),
>>           PointStyle=Circles,PointWidth=40,Title=Ëuler",TitlePosition=[8.5,5]]:
>> exsol := op(solve(ode({y'(t)=y(t)^2,y(0)=1},y(t))),1):
>> exact := [Mode=Curve,[t,exsol],t=[0,0.9],
>>           Title=Ëxact",TitlePosition=[8,1],Color=[Flat,[0,0,1]]]:
>> mde := detools::modode(F,EULER1,[z],t,h,4)[1]:
>> ModF := proc(t,y) begin [subs(mde,h=0.1,z=y[1])] end_proc:
>> modsol := numeric::odesolve(0..0.95,ModF,[1],RKF78,Alldata):
>> mod4 := detools::odeplot(modsol,[1],[[Title=ModEuler",TitlePosition=[9,3]]],[]):
>> plot2d(Title=Modified Equation for Euler Method",TitlePosition=Below,
>>     FontSize=15,ForeGround=[0,0,0],BackGround=[1,1,1],Scaling=UnConstrained,
>>     euler,exact,mod4[1]):
```

Graphics



Modified Equation for Euler method

The first line defines our differential equation in the form required by numeric::odesolve. Then we compute a numerical solution with the Euler method with fixed step size $h = 0.1$. The third command converts the output into a graphic object. The next two lines generate a curve for the exact solution. The following four commands generate a curve for the numerical solution of the fourth order modified equation: firstly, the modified equation mde is determined; then it is converted into a procedure ModF as required by numeric::odesolve; the final step generates a corresponding graphic object. Finally, plot2d is called (note that mod4 is a *list* of objects, hence we must use mod4[1]).

The output clearly shows that the solution of the modified equation almost interpolates between the points obtained by the Euler method. In contrast the exact solution of the original differential equation grows much more rapidly (the Euler method completely misses the singularity at $t = 1$). This shows that it indeed makes sense to consider modified equations as a continuous model for numerical methods.

A Lie Symmetry Package

Some Background

Sophus Lie's symmetry theory represents one of the most powerful approaches to differential equations. Especially for nonlinear equations it often provides the only possibility for finding closed form solutions. But symmetries have also many other applications like the construction of conservation laws, separation ansätze or the classification of differential equations. It is beyond the scope of this article to give a detailed introduction into Lie theory, but there exists by now a number of excellent textbooks [4, 10, 18] where the reader can find in depth information both on theoretical and practical aspects.

After Lie's death symmetry theory was almost forgotten for a long time. One reason for this was definitely that it is computationally very demanding. Determining systems containing several hundred of equations are not exceptional. Thus it is not surprising that already very early in the history of computer algebra the first packages for the symmetry analysis of differential equations emerged. Today it is almost impossible to maintain an overview on the many existing packages; comparatively recent surveys can be found in [8, 9].

MuPAD has been the only one among the major general purpose computer algebra system without a symmetry package. Therefore the first larger component of the `detools` library is such a package. In this first version it contains basically only methods for setting up determining systems; later versions will also provides methods supporting their automatic solution and the subsequent use of the found symmetries.

Our package can handle not only classical point symmetries but also *generalised* (or *Lie-Bäcklund*) symmetries, as this represents a fairly straightforward extension of the basic algorithm for setting up determining systems. However, we do not think that this represents a very useful approach to generalised symmetries and thus not much time has been invested in the corresponding methods. For alternative approaches and their implementation in *MuPAD* we refer to the recent work of Fuchssteiner et al. [6].

On the other hand we provide probably one of the most general implementation for the so-called *non-classical method* going beyond the usual Bluman-Cole approach. In principle, this allows us to determine every solution of a given system of differential equations via a symmetry reduction. However, in practice one is usually not able to obtain all solutions of the arising determining systems.

Classical Symmetries

As already said we cannot give an introduction to symmetry theory. For this we refer again to the above mentioned textbooks. Here we only sketch the basic concepts. The most general definition of a symmetry of a differential equation is that of a transformation mapping solutions into solutions. In the case of classical symmetries one considers only point transformations, i. e. transformations of the independent and dependent variables that do not depend on derivatives.

A more geometric picture can be given as follows. If we consider the derivatives in a differential equation as further unknowns, we may say that the equation defines a hypersurface in an extended space. A symmetry is now a transformation that leaves this hypersurface invariant. In order to relate this geometric idea with the definition above, we must extend the point transformations to transformations in this extended space. But this is a straightforward exercise in using the chain rule.

In general, it is very difficult to find such point transformations that leave a differential equations invariant, as this leads to very complicated non-linear equations. The decisive insight of Lie was that without (well, almost) loss of generality one can linearise the problem by considering only infinitesimal transformations; in a more abstract language one can pass from the Lie group to the Lie algebra. He furthermore provided an effective criterion for deciding whether or not a given vector field generates a symmetry group.

The main steps of determining the symmetries of a given differential equations are: (a) making an ansatz for the symmetry generators, (b) prolonging this ansatz to the order of the equation (this means finding the generator in the extended space), (c) evaluating the symmetry criterion. The last step yields the so-called *determining system* for the coefficients of the ansatz. It is an in general very over-determined linear system of partial differential equations.

Steps (b) and (c) are very tedious and for them the detools library provides support in form of the method detSys. Step (b) will usually be invisible for the user, as it is performed in a domain for vector fields. As a concrete example we consider the heat equation $u_t = u_{xx}$. The basic call for users who do not want to have anything to do with domains is:

```
MuPAD
>> detools::detSys(u([t])-u([x,x]),[t,x],[u],Expr=NoDiff);
Output
[XI2([x]), XI2([t]) + (-2) XI1([x]), XI1([u]), XI2([u]),

   PHI1([x, x]) + (-1) PHI1([t]), PHI1([x, u]) + (-1/2) XI1([x, x]) +

   1/2 XI1([t]), PHI1([u, u])]
```

The first three arguments specify as for other methods in the detools library the differential equations, the independent and the dependent variables, respectively. The option Expr specifies again that the output should be of the basic type DOM_EXPR. As we used Expr=NoDiff instead of Expr=TRUE, the output is furthermore in our simplified notation.

As no special ansatz for the symmetry generator has been given to detSys, it has used the most general one $v = \tau(t,x,u)\partial_t + \xi(t,x,u)\partial_x + \eta(t,x,u)\partial_u$. As default the names XI1, XI2, ... are used for the coefficients in direction of the independent variables and PHI1 etc. for the dependent variables. This can be changed by prescribing an ansatz. Thus in the following call we indeed use τ, ξ, η for the coefficients. This requires the setting of two options: Ansatz for the ansatz itself and Coeff for a list of the parameters of the used ansatz. The ansatz is given as a list of lists; each sublist consists of a coefficient and a variable indicating the basis vector (i. e. t for ∂_t and so on).

```
MuPAD
>> detool::detSys(u([t])-u([x,x]),[t,x],[u],
>>                Ansatz=[[tau(t,x,u),t],[xi(t,x,u),x],[eta(t,x,u),u]],
>>                Coeff=[tau,xi,eta],Expr=NoDiff,Unsimplified=TRUE);
Output
[2 tau([u]), 2 tau([x]), tau([u, u]), 2 tau([x, u]) + 2 xi([u]),

   tau([x, x]) + (-1) tau([t]) + 2 xi([x]), xi([u, u]),

   (-1) eta([u, u]) + 2 xi([x, u]), (-2) eta([x, u]) + xi([x, x]) +

   (-1) xi([t]), (-1) eta([x, x]) + eta([t])]
```

Here we set additionally the option Unsimplified. This instructs detSys to return the determining system in “raw” form. By default, `detSys` automatically performs some simplification (a so-called auto reduction) in order to eliminate redundant equations. Comparing the output of the two calls one can clearly see the effect of this option even for this small example. The number of equations has been reduced from 9 to 7 and the number of terms from 16 to 11.

`detSys` suffers from the same restrictions as other symmetry packages. Its result can be incorrect, if it is applied to differential equations which are not polynomial in the derivatives. This will be the case, if the appearing transcendental terms are not algebraically independent. `detSys` must also be able to solve each equation for a derivative; by default it tries to solve for the leading derivative. Alternatively, it is possible to prescribe with the option `Solve` for each equation a derivative for which it can be solved.

Generalised symmetries [2, 10] are not so easy to understand from a geometric point of view. Computationally however, not much changes. The transformations of the independent and dependent variables are now allowed to depend on derivatives, too. This leads to only minor modifications in the algorithm for setting up the determining system. However, generalised symmetries are most important for evolution equations, i. e. differential equations of the form $u_t = K(u, u_x, u_{xx}, \ldots)$. Here one can use completely different techniques for finding them. Some of them have been implemented in *MuPAD* by Fuchssteiner et al. [6]. The `detools` library nevertheless provides a method `genDetSys` for setting up the determining system for generalised symmetries of a prescribed order.

The main difference between `detSys` and `genDetSys` is that the latter one uses by default a vertical ansatz for the symmetry generator, i. e. an ansatz of the form $v = Q(x, t, u, u_t, u_x, \ldots)\partial_u$. The order of the ansatz, i. e. the maximal order of the derivatives appearing in the coefficient Q is specified by the option `GenOrder`.

MuPAD

```
>> detools::genDetSys(u([t])-u([x,x]),[t,x],[u],GenOrder=1,
>>                        Expr=NoDiff,Unsimplified=TRUE);
```

Output

```
[(-1) Q1([u([t]), u([t])]), ((-2 u([t]))) Q1([u([x]), u([t])]) +

   ((-2 u([x]))) Q1([u, u([t])]) + (-2) Q1([x, u([t])]),

               2
   ((- u([t]) )) Q1([u([x]), u([x])]) +

                                                           2
   ((-2 u([t]) u([x]))) Q1([u, u([x])]) + ((- u([x]) )) Q1([u, u]) +

   ((-2 u([t]))) Q1([x, u([x])]) + ((-2 u([x]))) Q1([x, u]) +

   (-1) Q1([x, x]) + Q1([t])]
```

Non-classical Symmetries

One of the most important applications of symmetries is the so-called *symmetry reduction*. Here we must distinguish between ordinary and partial differential equations. If we know a point symmetry of an ordinary differential equation, we can construct a reduced equation whose order is one less than that of the original equation. If we can find the general solution of the reduced equation, the general solution of the original equation can be obtained from it with one quadrature.

For partial differential equations the situation is very different. Here the reduced equation is not of lower order but has less independent variables. Furthermore it is *not* possible to construct the general solution of the original equation from the general solution of the reduced equation. Thus symmetry theory yields here only special, namely *group invariant solutions*.

Only rather recently it was fully realized that group invariant solutions exist also for groups which are *not* symmetry groups of the differential equations in the sense defined above. One started to distinguish between strong and weak symmetry groups [11, 13, 17]. *Strong symmetries* are symmetries in the sense we have used so far; *weak symmetries* are symmetries for which it is possible to perform symmetry reductions. The naming is somewhat misleading, as opposed to what the names seem to indicate not every strong symmetry is also a weak one!

With the introduction of weak symmetries one starts to move away from the original geometric ideas and gets more into the field of solvability of over-determined systems. If a solution is to be invariant under the group generated by $v = \tau\partial_t + \xi\partial_x + \eta\partial_u$, it must satisfy the corresponding *invariant surface condition* $\xi u_t + \tau u_x = \eta$. Hence group invariant solutions must simultaneously solve this equation and the original differential equation.

The determining system arises now from the condition that this over-determined system possesses solutions. In the so-called Bluman-Cole method one requires that the arising integrability conditions are identically satisfied (one says that the over-determined system is *compatible*). But this is a stronger requirement than necessary. It suffices that the over-determined system is *consistent*, i. e. that it has solutions. Thus if the first integrability conditions are not satisfied, we may add them to our system and check when this larger system is compatible. This way we may continue until we find either a compatible or an inconsistent system.

The procedure `ncDetSys` takes this situation into account. The user can prescribe with the option `Steps` at what step (s)he wants the integrability conditions to be satisfied. Thus the Bluman-Cole approach is obtained by specifying `Steps=1` (which is the default value) and we can recover the famous analysis of the heat equation [3] with the following line where we use a simplified ansatz for the vector field assuming that $\tau(t, x, u)$ does not vanish and can therefore be set to 1.

MuPAD

```
>> detools::ncDetSys(u([t])-u([x,x]),[t,x],[u],Expr=NoDiff,Steps=1,
>>                   Ansatz=[[1,t],[xi(t,x,u),x],[eta(t,x,u),u]],Coeff=[xi,eta]);
```

Output

```
--
|  - eta([t]) + eta([x, x]) - 2 eta xi([x]) ,
--

   xi([t])   xi([x, x])
   ------- - ---------- + eta([x, u]) + xi xi([x]) - eta xi([u]),
      2          2

                                                                --
   xi([u, u]), - 2 xi([x, u]) + eta([u, u]) + 2 xi xi([u])   |
                                                                --
```

Non-classical symmetries face two main obstacles. As one can clearly see from the output of our last example, the determining systems become non-linear and thus much harder to solve. Furthermore the higher one sets the value of `Steps` the less and the more complicated equations one gets. In some sense one thus looses the main reasons for the effectivity of Lie theory! Nevertheless non-classical reductions have proven to be a valuable tool for constructing solutions to partial differential equations.

Conclusions and Outlook

This article has given a preview on the new `detools` library which will be contained in the next release of *MuPAD*. It has described the current status of the library which is only at the very beginning of its development. Obviously, with the exception of the symmetry package the `detools` library represents currently a fairly accidental collection of methods which were developed on different occasions for different tasks. Most of them are not very sophisticated but rather small utilities.

A number of extensions are currently under development and will hopefully be included in the near future. The major points we are working at right now are

1. Extending the capabilities of `pdesolve`: (a) generalising the method of characteristics to arbitrary first order equations, (b) some heuristics for the at least partial solution of linear systems as the arise in symmetry analysis, and (c) series solutions for general systems.
2. Improvements of the symmetry package: (a) special routines for polynomial differential equations as these represent the most important case for applications, (b) more efficient prolongation of vector fields, and (c) further analysis of the results like determination of the structure constants of the symmetry algebra, computation of differential invariants etc.
3. Implementation of a completion package on the basis of the formal theory of differential equations [12, 15].
4. As a special application more support for the analysis of mechanical systems: (a) generation of Hamiltonian equations of motion, (b) support for systems with constraints, (c) automatic generation of the equations of motion in a form suitable for numerical integration.

Acknowledgements

Eduard Niess (Universität Karlsruhe) ported large parts of our earlier AXIOM version of a programming environment for differential equations [14, 16] to *MuPAD*. Marcus Hausdorf (Universität Karlsruhe) wrote the Lie symmetry package; Jay Belanger (Northeast Missouri State University) implemented the method of characteristics. Both provided examples for this article and helped to improve our programming environment by busily reporting bugs.

This article was written using the Emacs modes `MuPAD` and `EMuPAD` interfacing *MuPAD* and LaTeX within the Emacs editor. These very useful modes have been developed by Jay Belanger.

This work was partially supported by Deutsche Forschungsgemeinschaft.

References

[1] M.O. Ahmed and R.M. Corless. The method of modified equations in Maple. Technical report, Dept. of Applied Mathematics, University of Western Ontario, London (Canada), 1997.

[2] R.L. Anderson and N.H. Ibragimov. *Lie-Bäcklund Transformations in Applications*. SIAM, Philadelphia, 1979.

[3] G.W. Bluman and J.D. Cole. The general similarity solution of the heat equation. *J. Math. Mech.*, 18:1025–1042, 1969.

[4] G.W. Bluman and S. Kumei. *Symmetries and Differential Equations*. Applied Mathematical Sciences 81. Springer-Verlag, New York, 1989.

[5] K. Drescher. Axioms, categories and domains. Automath Technical Report No. 1, Universität Paderborn, 1996.

[6] B. Fuchssteiner, S. Ivanov, and W. Wiwianka. Algorithmic determination of infinite-dimensional symmetry groups for integrable systems in 1+1 dimensions. *Math. Comp. Model.*, 25:91–100, 1997.

[7] D.F. Griffiths and J.M. Sanz-Serna. On the scope of the method of modified equations. *SIAM J. Sci. Stat. Comp.*, 7:994–1008, 1986.

[8] W. Hereman. Review of symbolic software for the computation of Lie symmetries of differential equations. *Euromath Bull.*, 2:45–82, 1994.

[9] W. Hereman. Symbolic software for Lie symmetry analysis. In N.H. Ibragimov, editor, *CRC Handbook of Lie Group Analysis of Differential Equations, Volume 3: New Trends in Theoretical Development and Computational Methods*, chapter 13. CRC Press, Boca Raton, Florida, 1995.

[10] P.J. Olver. *Applications of Lie Groups to Differential Equations*. Graduate Texts in Mathematics 107. Springer-Verlag, New York, 1986.

[11] P.J. Olver and P. Rosenau. Group-invariant solutions of differential equations. *SIAM J. Appl. Math.*, 47:263–278, 1987.

[12] J.F. Pommaret. *Systems of Partial Differential Equations and Lie Pseudogroups*. Gordon & Breach, London, 1978.

[13] E. Pucci and G. Saccomandi. On the weak symmetry groups of partial differential equations. *J. Math. Ana. Appl.*, 163:588–598, 1992.

[14] J. Schü, W.M. Seiler, and J. Calmet. Algorithmic methods for Lie pseudogroups. In N. Ibragimov, M. Torrisi, and A. Valenti, editors, *Proc. Modern Group Analysis: Advanced Analytical and Computational Methods in Mathematical Physics*, pages 337–344. Kluwer, Dordrecht, 1993.

[15] W.M. Seiler. *Analysis and Application of the Formal Theory of Partial Differential Equations*. PhD thesis, School of Physics and Materials, Lancaster University, 1994.

[16] W.M. Seiler. Applying AXIOM to partial differential equations. Internal Report 95–17, Universität Karlsruhe, Fakultät für Informatik, 1995.

[17] W.M. Seiler. Involution and symmetry reductions. *Math. Comp. Model.*, 25:63–73, 1997.

[18] H. Stephani. *Differential Equations: Their Solution Using Symmetries*. Cambridge University Press, Cambridge, 1989.

Note on Hurwitz' Equation

Winfried Fakler

Universität-GH Paderborn, `fakler@mupad.de`
`www.mupad.de/~fakler`

In this note, we give a new method for computing algebraic solutions of some homogeneous linear differential equations with a primitive unimodular Galois group and we use it to solve the differential equation constructed by Hurwitz [10] in closed form. Furthermore we will describe very briefly the problem where this equation comes from.

Introduction

The differential equation $H(y) = 0$ (see below) due to Hurwitz goes back to the famous work "*Ueber die Transformation siebenter Ordnung der elliptischen Functionen*" of F. Klein [14]. In this work, Klein made an important contribution to the much discussed question of the solvability of algebraic equations by means of elliptic modular functions. In order to investigate the transformation of seventh order of elliptic functions (see also [12], [13]), starting from the simple group G_{168} consisting of 168 linear substitutions, Klein constructed the Riemann surface [14, p. 441]

$$F(\lambda, \mu, \nu) = \lambda^3\mu + \mu^3\nu + \nu^3\lambda = 0. \tag{9}$$

In order to get an explicit solution (parametrization) of the surface (9) Klein built an equation of degree 168 (the Galois resolvent of the modular equation of degree 8) and solved it in terms of elliptic functions (see [14, p. 456]). Refering to this equation he said in the meanwhile frequently cited footnote [14, p. 455]: *"Sie* [die Gleichung 168^{ten} Grades] *muss sich auch durch eine lineare Differentialgleichung dritter Ordnung lösen lassen; wie hat man dieselbe aufzustellen?"*

The first solution of this problem stems from G. Halphen [9]. But he gave the desired differential equation only in implicit form. In [10] A. Hurwitz developed a second solution of this problem. His solution is

$$\begin{aligned} H(y) \quad = \quad & x^2(x-1)^2y''' + (7x-4)x(x-1)y'' \\ & + \left(\frac{72}{7}(x^2-x) - \frac{20}{9}(x-1) + \frac{3}{4}x\right)y' \\ & + \left(\frac{72\cdot 11}{7^3}(x-1) + \frac{5}{8} + \frac{2}{63}\right)y = 0. \end{aligned}$$

Hurwitz, a student of Klein, found this equation mainly by using so-called 1-forms together with relations deduced from Klein. Since $H(y) = 0$ is necessarily of Fuchsian type, he had to apply the Fuchsian relations (see also [22, p. 323], [8, p. 232 and 390]) and to take care that no logarithmic term could occur. He also determined the solutions of $H(y) = 0$ in terms of theta functions ([10, p. 122]). Theta series converge very fast (see e.g. Naas [17]), i.e. one gets a good representation of the solutions by them. However, these solutions are not in closed form.

In the following we give an extended method for computing algebraic solutions of some homogeneous linear differential equations with a primitive unimodular Galois group (see Fakler [4]). We will use it to solve $H(y) = 0$ for the first time in closed form. For that we first give a briefly introduction to differential Galois theory and the concept of invariants.

Galois theory, symmetric powers and invariants

For the exact definitions of the following concepts we refer to Kaplansky [11], Kolchin [15], Singer [19], Singer and Ulmer [20], [21] and Sturmfels [24].

Differential Galois theory.

A *differential field* $(k, ')$ is a field k together with a derivation $'$ in k. The set of all constants $\mathcal{C} = \{a \in k \mid a' = 0\}$ is a subfield of $(k, ')$.
Let $\mathcal{C}$ be algebraically closed and k be of characteristic 0. Consider the following ordinary homogeneous linear differential equation

$$L(y) = y^{(n)} + a_{n-1}y^{(n-1)} + \dots + a_1y' + a_0y = 0 \qquad (a_i \in k) \tag{10}$$

over k with a system $\{y_1, \dots, y_n\}$ of fundamental solutions.
By extending the derivation $'$ to a system of fundamental solutions and by adjunction of these solutions and their derivatives to k in a way the field of constants does not change, one gets $K = k\langle y_1, \dots, y_n\rangle$, the so-called *Picard-Vessiot extension* (PVE) of $L(y) = 0$. With the above assumptions, the PVE of $L(y) = 0$ always exists and is unique up to differential isomorphisms. This extension plays the same role for a differential equation as a splitting field for a polynomial equation.

The set of all automorphisms of K, which fix k elementwise and commute with the derivation in K, is a group, the *differential Galois group* $\mathcal{G}(K/k) = \mathcal{G}(L)$ of $L(y) = 0$. Since the automorphisms must commute with the derivation, they map a solution to a solution. Therefore $\mathcal{G}(L)$ operates on the $\mathcal{C}$-vector space of the fundamental solutions and from that one gets a faithful matrix representation of $\mathcal{G}(L)$, hence $\mathcal{G}(L)$ is isomorphic to a linear subgroup of GL$(n, \mathcal{C})$. Moreover, it is isomorphic to a linear algebraic group. Furthermore, there is a (differential) Galois correspondence between the linear algebraic subgroups of $\mathcal{G}(L)$ and the differential subfields of K/k (see Kaplansky [11], Theorems 5.5 and 5.9).
The choice of another system of fundamental solutions leads to an equivalent representation. Hence, for every differential equation $L(y) = 0$, there is exactly one representation of $\mathcal{G}(L)$ up to equivalence.

A solution of $L(y) = 0$ which lies in k is called a *rational solution*. A solution lying in an algebraic extension of k is called an *algebraic solution*, i.e. an algebraic solution satisfies an irreducible polynomial over k.

Symmetric powers and invariants.

Let V be a finite dimensional $\mathcal{C}$-vector space and G a linear subgroup of GL(V). An *(absolute) invariant* is a polynomial function $f \in \mathcal{C}[V]$ which remains unchanged under the group action, i.e. $f = f \circ g$ for all $g \in G$. The set of all invariants of G forms the *ring of invariants* $\mathcal{C}[V]^G$. For irreducible groups $G \in$ GL(V), the rings of invariants $\mathcal{C}[V]^G$ are finitely generated by Hilbert's finiteness theorem (see e.g. Sturmfels [24]).
For finite groups $G \in$ GL(V) the *Reynolds operator* $R_G(f) = \frac{1}{|G|}\sum_{g\in G} f \circ g$ maps a polynomial function

$f \in \mathcal{C}[V]$ to the invariant $R_G(f) \in \mathcal{C}[V]^G$. With the *Hessian* $H(I_1) = \det\left(\frac{\partial^2 I_1}{\partial v_i \partial v_j}\right)$, the *bordered Hessian*

$$bH(I_1,\, I_2) = \det \begin{pmatrix} \frac{\partial^2 I_1}{\partial v_1 \partial v_1} & \cdots & \frac{\partial^2 I_1}{\partial v_1 \partial v_n} & \frac{\partial I_2}{\partial v_1} \\ \vdots & & \vdots & \vdots \\ \frac{\partial^2 I_1}{\partial v_n \partial v_1} & \cdots & \frac{\partial^2 I_1}{\partial v_n \partial v_n} & \frac{\partial I_2}{\partial v_n} \\ \frac{\partial I_2}{\partial v_1} & \cdots & \frac{\partial I_2}{\partial v_n} & 0 \end{pmatrix}$$

and the *Jacobian* $J(I_1, \dots, I_n) = \det\left(\frac{\partial I_i}{\partial v_j}\right)$ it is possible to generate new invariants from some invariants $I_1(\mathbf{v}), \dots, I_n(\mathbf{v})$ (see e.g. [24]).

Let V be the $\mathcal{C}$-vector space of a system of fundamental solutions of $L(y) = 0$ and let $I(\mathbf{v}) \in \mathcal{C}[V]^{\mathcal{G}(L)}$ be an invariant of $\mathcal{G}(L)$. If one evaluates the invariant $I(\mathbf{v})$ with the fundamental solutions and takes into account that exactly the elements $a \in k$ are invariant under the Galois group $\mathcal{G}(L)$ then $I(y_1, \dots, y_n)$ must be an element of k. An important tool for computing such an element are the symmetric powers of $L(y) = 0$.
The m^{th} *symmetric power* $L^{\circledS m}(y) = 0$ of $L(y) = 0$ is the differential equation whose solution space consists exactly of all m^{th} power products of solutions of $L(y) = 0$. There is an algorithm to construct symmetric powers described e.g. in Singer and Ulmer [20, pp. 20] or Fakler [3, pp. 14]. A more efficient variant one can find in Bronstein, Mulders and Weil [2].

We will denote the rational solution r of the m^{th} symmetric power $L^{\circledS m}(y) = 0$ corresponding to the invariant $I(y_1, \dots, y_n)$ as the *rational invariant* of $I(y_1, \dots, y_n)$. In [25], van Hoeij and Weil give a method for determining rational invariants without computing symmetric powers.

Satz 2 *([21], Corollary 1.4)*
Let $k \subset K$ be a differential field of characteristic 0 and let the common field of constants of k and K be algebraically closed. If $y \in K$ is algebraic over k and y'/y is algebraic of degree m over k, then the minimal polynomial $P(\mathbf{Y}) = 0$ of y over k can be written in the following way

$$P(\mathbf{Y}) = \mathbf{Y}^{d \cdot m} + a_{m-1}\mathbf{Y}^{d\cdot(m-1)} + \ldots + a_0 = \prod_{\sigma \in \mathcal{T}} \left(\mathbf{Y}^d - (\sigma(y))^d\right), \tag{11}$$

where $a_j \in k$, $[k(y) : k(y'/y)] = d = |H/N|$, H/N is cyclic, $H = \mathcal{G}(K/k(y'/y))$ is a 1-reducible subgroup of $G = \mathcal{G}(K/k)$, N is a normal subgroup of H and $\mathcal{T}$ is a set of left coset representatives of H in G of minimal index m.

Note, since in (11) the coefficients a_j are elements in k, they are invariant under the Galois group $\mathcal{G}(K/k)$. This fact will be used to compute the minimal polynomial decomposed into invariants for the Galois group of Hurwitz' equation in the next section.

Minimal polynomial decomposed into invariants

In Singer and Ulmer [22] it is shown, that the Galois group of the equation $H(y) = 0$ is isomorphic to G_{168}. A matrix representation of this group is given by $G_{168} = \langle S, T, R\rangle$ (see [16], [21], p. 54), where

$$S = \begin{pmatrix} \beta & 0 & 0 \\ 0 & \beta^2 & 0 \\ 0 & 0 & \beta^4 \end{pmatrix} \qquad T = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \qquad R = \frac{1}{\sqrt{-7}} \begin{pmatrix} a & b & c \\ b & c & a \\ c & a & b \end{pmatrix}$$

and $\beta^7 = 1$, $\frac{1}{\sqrt{-7}} = \frac{\beta+\beta^2+\beta^4-\beta^6-\beta^5-\beta^3}{7}$, $a = \beta^4 - \beta^3$, $b = \beta^2 - \beta^5$, $c = \beta - \beta^6$. Note, this is up to a small change in matrix S the representation deduced by Klein [14, p. 444]. He gave also the following fundamental invariants of G_{168} (up to the sign of I_6)

$$\begin{aligned}
I_4 &= 3R_{G_{168}}(v_1^3 v_2)(\mathbf{v}) = v_2 {v_3}^3 + {v_1}^3 v_3 + v_1 {v_2}^3 \\
I_6 &= -\frac{H(I_4)}{54} \\
I_{14} &= \frac{bH(I_4, I_6)}{9} \\
I_{21} &= -\frac{J(I_4, I_6, I_{14})}{14}.
\end{aligned}$$

These invariants satisfy the syzygy

$$\begin{aligned}
I_{21}^2 - I_{14}^3 - 88 I_4^2 I_6 I_{14}^2 + \left(-1008 I_4 I_6^4 - 1088 I_4^4 I_6^2 + 256 I_4^7\right) I_{14} + \\
1728 I_6^7 - 60032 I_4^3 I_6^5 + 22016 I_4^6 I_6^3 - 2048 I_4^9 I_6 \quad = \quad 0.
\end{aligned} \tag{12}$$

Using the set of left coset representatives and the eigenvector given in [21, p. 65] together with the algorithms given in [3] one can compute a minimal polynomial decomposed into invariants of a solution of a third order linear differential equation with Galois group G_{168}:

$$\begin{aligned}
&\mathbf{Y}^{42} + 98 I_4 \mathbf{Y}^{38} + 126 I_6 \mathbf{Y}^{36} + 3087 I_4^2 \mathbf{Y}^{34} \\
&+ 6174 I_4 I_6 \mathbf{Y}^{32} + \left(1547 I_6^2 + 33712 I_4^3\right) \mathbf{Y}^{30} + \left(2499 I_{14} - 2940 I_4^2 I_6\right) \mathbf{Y}^{28} \\
&+ \left(42532 I_4 I_6^2 - 2401 I_4^4\right) \mathbf{Y}^{26} + \left(52724 I_4 I_{14} - 21420 I_6^3 - 1908452 I_4^3 I_6\right) \mathbf{Y}^{24} \\
&+ \left(\begin{array}{l} -3332 I_6 I_{14} + 1560258 I_4^2 I_6^2 \\ -3669414 I_4^5 \end{array}\right) \mathbf{Y}^{22} + \left(\begin{array}{l} -33614 I_4^2 I_{14} - 1238916 I_4 I_6^3 \\ +6084134 I_4^4 I_6 \end{array}\right) \mathbf{Y}^{20} \\
&+ \left(\begin{array}{l} 239316 I_4 I_6 I_{14} + 227591 I_6^4 \\ +1222060 I_4^3 I_6^2 + 54125057 I_4^6 \end{array}\right) \mathbf{Y}^{18} + \left(\begin{array}{l} \left(-40278 I_6^2 - 2649332 I_4^3\right) I_{14} \\ -14036932 I_4^2 I_6^3 - 51243514 I_4^5 I_6 \end{array}\right) \mathbf{Y}^{16} \\
&+ \left(1715 I_{14}^2 + 2067604 I_4^2 I_6 I_{14} + 4847178 I_4 I_6^4 + 123928987 I_4^4 I_6^2 - 261257612 I_4^7\right) \mathbf{Y}^{14} \\
&+ \left(\left(-344372 I_4 I_6^2 + 3733555 I_4^4\right) I_{14} - 396018 I_6^5 - 121236780 I_4^3 I_6^3 + 11198264 I_4^6 I_6\right) \mathbf{Y}^{12} \\
&+ \left(\begin{array}{l} -11662 I_4 I_{14}^2 + \left(-7252 I_6^3 - 5025636 I_4^3 I_6\right) I_{14} \\ +36297191 I_4^2 I_6^4 + 342404552 I_4^5 I_6^2 + 706815984 I_4^8 \end{array}\right) \mathbf{Y}^{10} \\
&+ \left(\begin{array}{l} 6174 I_6 I_{14}^2 + \left(2449706 I_4^2 I_6^2 + 8509144 I_4^5\right) I_{14} \\ -3720570 I_4 I_6^5 - 224349440 I_4^4 I_6^3 - 1094318176 I_4^7 I_6 \end{array}\right) \mathbf{Y}^{8} \\
&+ \left(\begin{array}{l} 45619 I_4^2 I_{14}^2 + \left(-569380 I_4 I_6^3 - 10679648 I_4^4 I_6\right) I_{14} + 186445 I_6^6 \\ +50394932 I_4^3 I_6^4 + 641816112 I_4^6 I_6^2 - 214053952 I_4^9 \end{array}\right) \mathbf{Y}^{6} \\
&+ \left(\begin{array}{l} -24010 I_4 I_6 I_{14}^2 + \left(48363 I_6^4 + 3630312 I_4^3 I_6^2 - 2958032 I_4^6\right) I_{14} \\ -4151672 I_4^2 I_6^5 - 157198272 I_4^5 I_6^3 + 169952384 I_4^8 I_6 \end{array}\right) \mathbf{Y}^{4} \\
&+ \left(\begin{array}{l} \left(3087 I_6^2 + 9604 I_4^3\right) I_{14}^2 + \left(-378672 I_4^2 I_6^3 + 460992 I_4^5 I_6\right) I_{14} \\ +63504 I_4 I_6^6 + 14839552 I_4^4 I_6^4 - 31962112 I_4^7 I_6^2 + 17210368 I_4^{10} \end{array}\right) \mathbf{Y}^{2} \\
&+ 49 I_{14}^3 - 4312 I_4^2 I_6 I_{14}^2 + \left(7056 I_4\ I_6^4 + 53312 I_4^4 I_6^2 - 87808 I_4^7\right) I_{14} + 1728 I_6^7 \\
&- 420224 I_4^3 I_6^5 + 1078784 I_4^6 I_6^3 - 702464 I_4^9 I_6.
\end{aligned}$$

We note, to get that "nice" decomposition of the minimal polynomial, it became necessary to multiply I_4 by the factor $\frac{1}{7}(\beta^5 - 4\beta^4 + 6\beta^3 - 4\beta^2 + \beta)$, I_6 with $-\beta^5 - \beta^4 - 3\beta^2 + 2\beta - 3$ and I_{14} with $-\frac{1}{7}(100\beta^5 + 48\beta^4 + 48\beta^3 + 100\beta^2 + 131)$.

Solving Hurwitz' equation

Let $y_1(x), y_2(x)$ and $y_3(x)$ denote three linear independent solutions of a third order linear differential equation $L(y) = 0$ with unimodular primitive Galois group $\mathcal{G}(L)$. It is well-known that there exists a unique ring homomorphism $\Phi_{\mathbf{y}}$

$$\mathcal{C}[v_1, v_2, v_3]^{\mathcal{G}(L)} \longrightarrow \mathcal{C}(x), \qquad I(v_1, v_2, v_3) \longmapsto I(y_1(x), y_2(x), y_3(x)).$$

The problem is now to determine the homomorphism $\Phi_{\mathbf{y}}$ which is fixed by the vector $\mathbf{y} = (y_1(x), y_2(x), y_3(x))$. However, we do not know this solution vector $\mathbf{y}$. On the other hand when the Galois group of the equation is known, it is possible to compute rational invariants, e.g. with the algorithm in [25]. Unfortunately, these rational invariants need no longer to correspond to the same solution vector $\mathbf{y}$. Therefore, one has to choose these rational invariants in such a way that they belong to the same vector $\mathbf{y}$.

Let $\{ I_{d,j} \}$ denote a set of fundamental invariants of $\mathcal{G}(L)$, where d is the degree of invariant $I_{d,j}$ and j labels invariants of same degree. Further, let $\{ r_{d,k} \}$ denote the rational invariants of $\mathcal{G}(L)$. We can now restate the previous problem of determining a homomorphism $\Phi_{\mathbf{y}}$ by

$$\Phi_{\mathbf{y}}(I_{d,j}) \longrightarrow \sum_k c_{d,j,k} \cdot r_{d,k}\,, \tag{13}$$

for which one has to determine the constants $c_{d,j,k}$ in a suitable way. This can be done using the syzygies, if the corresponding system for the constants $\{ c_{d,j,k} \}$ depends only on one parameter, i.e. $c_{d,j,k} = \lambda$ ($\lambda \neq 0$) for one fixed constant and all remaining constants depends only on λ. By the existence and uniqueness of $\Phi_{\mathbf{y}}$ one has, that the images of the fundamental invariants (13) determines a vector $\mathbf{y}$. Obviously this is true for Galois groups whose ring of invariants has only fundamental invariants of distinct degree. Thus we have the following Proposition.

Proposition 3
With the above notation, let $\{ I_{d,j} \}$ be a set of fundamental invariants of $\mathcal{G}(L)$ and let $\{ r_{d,k} \}$ be the rational invariants of it. The syzygies between $\{ I_{d,j} \}$ determine a ring homomorphism $\Phi_{\mathbf{y}}$ given by (13) for a vector $\mathbf{y}$, if the system for the constants $\{ c_{d,j,k} \}$ corresponding to the images of the syzygies under $\Phi_{\mathbf{y}}$ depends only on one free parameter.

Proof. Follows directly from the previous discussion (for arbitrary order equations). □

Proposition 3 allows us to determine the minimal polynomial of a solution of $H(y) = 0$. For that one has to compute the rational invariants of the four invariants I_4, I_6, I_{14} and I_{21}. By the curve (9), the rational invariant of I_4 must vanish, i.e. $I_4 = 0$. The computation of the remaining rational invariants by determining the corresponding symmetric powers and their rational solutions is currently not possible with computer algebra systems (see Fakler [3, pp. 57], Weil [27, p. 66]). One can expect an improvement of this situation by an implementation of van Hoeij and Weil's algorithm [25]. In [27], Weil has developped a method for computing the rational invariants via Darboux polynomials ([27], Théorème 52, Proposition 50). His solution for the four rational invariants for equation $H(y) = 0$ (see [27, p. 67]) leads to the following ansatz

$$\begin{aligned}
I_4 &= c_1 \cdot 0\\
I_6 &= c_2 \cdot \tfrac{1}{x^4(x-1)^3}\\
I_{14} &= c_3 \cdot \tfrac{1}{x^9(x-1)^7}\\
I_{21} &= c_4 \cdot \tfrac{1}{x^{14}(x-1)^{10}}.
\end{aligned}$$

By substituting this ansatz in the syzygy (12), one gets the determining equation

$$\frac{1728c_2^7 - c_4^2 + (c_4^2 - c_3^3)x}{x^{28}(x-1)^{21}} = 0,$$

which needs to hold for all regular points x. This equation yields two conditions for the three constants $c_2,\ c_3,\ c_4$ and thus its solution space depends only on one parameter. One concrete solution is given by

$$\{c_2 = 27,\ c_3 = 26244,\ c_4 = -4251528\}.$$

Recalling the factors at the end of the previous section and substituting

$$\begin{aligned}
I_4 &= 0 \\
I_6 &= \left(-13\beta+22\beta^2-13\beta^3+13\beta^4+13\right)\frac{27}{x^4(x-1)^3} \\
I_{14} &= \left(23896\beta^2+10672\beta^3+10672\beta^4+23896\beta^5+29865\right)\frac{26244}{49x^9(x-1)^7}
\end{aligned}$$

in the minimal polynomial decomposed into invariants, we obtain the minimal polynomial $P(\mathbf{Y}) = 0$ of a solution of $H(y) = 0$:

$$\begin{aligned}
\mathbf{Y}^{42} &+ \left(2\beta-3\beta^2-\beta^4-\beta^5-3\right)\frac{3402}{x^4(x-1)^3}\mathbf{Y}^{36} \\
&+ \left(-13\beta+22\beta^2-13\beta^3+13\beta^4+13\right)\frac{1127763}{x^8(x-1)^6}\mathbf{Y}^{30} \\
&+ \left(-100\beta^2-48\beta^3-48\beta^4-100\beta^5-131\right)\frac{9369108}{x^9(x-1)^7}\mathbf{Y}^{28} \\
&+ \left(-113\beta+96\beta^2-170\beta^3+96\beta^4-113\beta^5\right)\frac{421609860}{x^{12}(x-1)^9}\mathbf{Y}^{24} \\
&+ \left(210\beta-797\beta^2-435\beta^4-435\beta^5-797\right)\frac{337287888}{x^{13}(x-1)^{10}}\mathbf{Y}^{22} \\
&+ \left(-910\beta-1651\beta^3+419\beta^4-1651\beta^5-910\right)\frac{120951188631}{x^{16}(x-1)^{12}}\mathbf{Y}^{18} \\
&+ \left(-2811\beta+5050\beta^2-2811\beta^3+3471\beta^4+3471\right)\frac{110084814504}{x^{17}(x-1)^{13}}\mathbf{Y}^{16} \\
&+ \left(23896\beta^2+10672\beta^3+10672\beta^4+23896\beta^5+29865\right)\frac{24106163760}{x^{18}(x-1)^{14}}\mathbf{Y}^{14} \\
&+ \left(-5791\beta-5791\beta^2-12983\beta^3-16213\beta^5-12983\right)\frac{5682425452326}{x^{20}(x-1)^{15}}\mathbf{Y}^{12} \\
&+ \left(27367\beta-22004\beta^2+39618\beta^3-22004\beta^4+27367\beta^5\right)\frac{535156835472}{x^{21}(x-1)^{16}}\mathbf{Y}^{10} \\
&+ \left(46506\beta-187731\beta^2-104105\beta^4-104105\beta^5-187731\right)\frac{2343119117472}{x^{22}(x-1)^{17}}\mathbf{Y}^{8} \\
&+ \left(-25234\beta-81797\beta^2-81797\beta^3-25234\beta^4-127206\beta^5-127206\right)\frac{72232613071605}{x^{24}(x-1)^{18}}\mathbf{Y}^{6} \\
&+ \left(215098\beta+387713\beta^3-95877\beta^4+387713\beta^5+215098\right)\frac{96360773706036}{x^{25}(x-1)^{19}}\mathbf{Y}^{4} \\
&+ \left(-656205\beta+1182278\beta^2-656205\beta^3+817909\beta^4+817909\right)\frac{31632108085872}{x^{26}(x-1)^{20}}\mathbf{Y}^{2} \\
&+ \left(-800660\beta^2-356384\beta^3-356384\beta^4-800660\beta^5-998509\right)\frac{18075490334784}{x^{28}(x-1)^{20}}.
\end{aligned}$$

We note, since the Galois group of $H(y) = 0$ is not solvable one knows from Galois theory that it is not possible to represent the algebraic solutions of $H(y) = 0$ in terms of radicals. Therefore, the given minimal polynomial $P(\mathbf{Y}(\mathbf{x})) = 0$ is the most explicit way for representing closed form solutions of $H(y) = 0$. The method presented above is an extension of the method for primitive unimodular Galois groups given in Fakler [4, 5].

To verify this result by computation, one has to calculate the first three derivatives of $\mathbf{Y}(\mathbf{x})$ using the determined minimal polynomial $P(\mathbf{Y}(\mathbf{x}))$ and substitute these derivatives into $H(y) = 0$. A method to avoid the (very) large powers of $\mathbf{Y}(\mathbf{x})$ which arise in the total differentiation of the minimal polynomial $P(\mathbf{Y}(\mathbf{x}))$ with respect to x, one can find in Salvy and Zimmermann [18, p. 170]. Nevertheless, using this method MAPLE V.4 was not able to handle the occuring large objects. For testing with *MuPAD* the author did a very fast (half a day) implementation of a domain of function fields, but the test was stopped by the author after 6 days and 671 Mb memory. The system MAGMA V2.3-1 (see Bosma, Cannon und Playoust [1]) checks the result in 243 min on a Sun Ultra Sparc 1 using 33 Mb memory.

The equation $H(y) = 0$ makes clear, that it is much more efficient to use syzygies for determining the constants whenever possible than calculating them by substituting into the decomposed minimal polynomial and determine the coefficients as proposed in Singer and Ulmer [21], section 5. If one has the rational invariants, calculating the minimal polynomial of a solution is possible within 1 minute of computation time.

We further note, that there is a new algorithm for solving nth order homogeneous linear differential equations given in Singer and Ulmer [23] and some improvements given in van Hoeij, Ragot, Ulmer and Weil [26] which makes this algorithm more practical. However, from this algorithm one obtains the Riccati polynomial $R(\mathbf{Y})$ of the equation, i.e. the minimum polynomial of the logarithmic derivative of a solution. Thus, for representing a solution of $H(y) = 0$ in closed form one will have something like $\exp(\int R(\mathbf{Y}))$. But such an expression is less explicit than $P(\mathbf{Y})$ given above.

It remains to mention, that in the primitive case for computing the Riccati polynomial $R(\mathbf{Y})$ no algebraic extension of the constant field of the equation is required, while in determining the minimal polynomial $P(\mathbf{Y})$ it can be necessary.

Comment
This note was already presented on the $6th$ Rhine Workshop on Computer Algebra, held in Sankt Augustin, April 1998. In the meanwhile, Mark van Hoeij suggested a way to present the above given minimal polynomial $P(\mathbf{Y})$ over $\mathbb{Q}(x)$ rather than over $\mathbb{Q}(\beta)(x)$ to get rid of β. After sending him my result, he gave me in return the following nice representation of $P(\mathbf{Y})$:

$$\begin{aligned}
&\mathbf{Y}^{42} - \tfrac{18}{x^4(x-1)^3}\mathbf{Y}^{36} + \tfrac{221}{7x^8(x-1)^6}\mathbf{Y}^{30} + \tfrac{612}{7x^9(x-1)^7}\mathbf{Y}^{28} + \tfrac{3060}{49x^{12}(x-1)^9}\mathbf{Y}^{24} + \tfrac{816}{49x^{13}(x-1)^{10}}\mathbf{Y}^{22} \\
&+ \tfrac{32513}{343x^{16}(x-1)^{12}}\mathbf{Y}^{18} - \tfrac{9864}{343x^{17}(x-1)^{13}}\mathbf{Y}^{16} + \tfrac{720}{343x^{18}(x-1)^{14}}\mathbf{Y}^{14} + \tfrac{8082}{343x^{20}(x-1)^{15}}\mathbf{Y}^{12} \\
&+ \tfrac{1776}{2401x^{21}(x-1)^{16}}\mathbf{Y}^{10} - \tfrac{2592}{2401x^{22}(x-1)^{17}}\mathbf{Y}^{8} + \tfrac{3805}{2401x^{24}(x-1)^{18}}Y^{6} + \tfrac{1692}{2401x^{25}(x-1)^{19}}\mathbf{Y}^{4} \\
&+ \tfrac{1296}{16807x^{26}(x-1)^{20}}\mathbf{Y}^{2} + \tfrac{1728}{823543x^{28}(x-1)^{20}}.
\end{aligned}$$

References

[1] Bosma, W., Cannon, J., Playoust, C. (1997). *The Magma algebra system I: The user language.* J. Symb. Comp. **24**, 235-265.

[2] Bronstein, M., Mulders, T., Weil, J.A. (1997). *On Symmetric Powers of Differential Operators.* In: Prodeedings of ISSAC'97, Maui, Hawaii, 156-163.

[3] Fakler, W. (1994). *Algorithmen zur symbolischen Lösung homogener linearer Differentialgleichungen.* Diplomarbeit. Universität Karlsruhe.

[4] Fakler, W. (1997). *On second order homogeneous linear differential equations with Liouvillian solutions.* Theor. Comp. Science **187** (1-2), 27-48.

[5] Fakler, W. (1999). *Algebraische Algorithmen zur Lösung von linearen Differentialgleichungen.* MuPAD Reports; Stuttgart, Leipzig: Teubner.

[6] Fuchs, L. (1878). *Ueber die linearen Differentialgleichungen zweiter Ordnung, welche algebraische Integrale besitzen. Zweite Abhandlung.* Journal für die reine und angewandte Mathematik **85**, 1-25.

[7] The MuPAD Group (Benno Fuchssteiner et al.) (1996). *MuPAD User's Manual - MuPAD Version 1.2.2.* New York: John Wiley and sons.

[8] Gray, J. (1986). *Linear differential equations and group theory from Riemann to Poincaré.* Boston, Basel, Stuttgart: Birkhäuser.

[9] Halphen, G. (1884). *Sur une équation différentielle linéaire du troisième ordre.* Math. Ann. **24**, 461-464.

[10] Hurwitz, A. (1886). *Ueber einige besondere homogene lineare Differentialgleichungen.* Math. Ann. **26**, 117-126.

[11] Kaplansky, I. (1957). *Introduction to differential algebra.* Paris: Hermann.

[12] Klein, F. (1879). *Ueber die Transformation der elliptischen Functionen und die Auflösung der Gleichungen fünften Grades.* Math. Ann. **14**, 111-172.

[13] Klein, F. (1879). *Ueber die Erniedrigung der Modulargleichungen.* Math. Ann. **14**, 417-427.

[14] Klein, F. (1879). *Ueber die Transformation siebenter Ordnung der elliptischen Functionen.* Math. Ann. **14**, 428-471.

[15] Kolchin, E. R. (1948). *Algebraic matrix groups and the Picard-Vessiot theory of homogeneous ordinary differential equations.* Annals of Math. **49**.

[16] Miller, G.A., Blichfeldt, H.F., Dickson, L.E. (1938). *Theory and Applications of Finite Groups.* New York: G. E. Stechert and Co.

[17] Naas, J., Schmid, H.L. (1961). *Mathematisches Wörterbuch.* Band I und II. Stuttgart: Teubner.

[18] Salvy, B., Zimmermann, P. (1994). *GFUN: A Maple Package for the Manipulation of Generating and Holonomic Functions in One Variable.* Trans. Math. Soft. **20**, No. 2, 163-177.

[19] Singer, M.F. (1990). *An outline of differential Galois theory.* In: Computer Algebra and Differential Equations, Ed. E. Tournier, New York: Academic Press.

[20] Singer, M.F., Ulmer, F. (1993). *Galois Groups of Second and Third Order Linear Differential Equations.* J. Symb. Comp. **16**, 9-36.

[21] Singer, M.F., Ulmer, F. (1993). *Liouvillian and Algebraic Solutions of Second and Third Order Linear Differential Equations.* J. Symb. Comp. **16**, 37-73.

[22] Singer, M.F., Ulmer, F. (1993). *On a Third Order Differential Equation whose Differential Galois Group is the Simple Group of 168 Elements.* In: Proceedings of the 1993 Symposium on Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, LNCS 673, Springer Verlag.

[23] Singer, M.F., Ulmer, F. (1997). *Linear differential equations and products of linear forms.* J. Pure and Applied Algebra **117 & 118**, 549-563.

[24] Sturmfels, B. (1993). *Algorithms in Invariant Theory.* Texts and Monographs in Symbolic Computation; Wien; New York: Springer-Verlag.

[25] van Hoeij, M., Weil, J.A. (1997). *An algorithm for computing invariants of differential Galois groups*. J. Pure and Applied Alg. **117 & 118**, 353-379.

[26] van Hoeij, M., Ragot, J.F., Ulmer, F., Weil, J.A. (1997). *Liouvillian solutions of linear differential equations of order three and higher.* Prépublication 97-21, IRMAR, Rennes.

[27] Weil, J.A. (1995). *Constantes et polynômes de Darboux en algèbre différentielle: applications aux systèmes différentiels linéare.* Thèse, École Polytechnique.

Geometric Integration in *MuPAD*

Walter Oevel, Renate Wagner
SciFace Software, `walter@sciface.com`
`renate_wagner@WestLB.de`

The new algorithm `numeric::odesolveGeometric` *available in the next release of MuPAD is discussed. It is a numerical integrator for ordinary differential equations on homogeneous manifolds which keeps the numerical solution on the manifold. The routine implements the Munthe-Kaas extensions of classical Runge-Kutta schemes to Lie groups. It makes intensive use of the exponential of a matrix. A corresponding fast routine* `expMatrix` *has been added to MuPAD's* `numeric` *library.*

Introduction

The last decade has seen exciting new developments in the numerical analysis of ordinary differential equations [4]. In this rather old and well established area of numerical analysis (the "classical" Runge-Kutta scheme dates back to the turn of the century) the traditional approach aims at computing the actual solution curves as accurately as possible while minimizing the costs via suitable high order schemes. Very few qualitative aspects such as asymptotic stability were considered leading to the concept of A-stable integrators. Generally, however, geometric considerations used to play a marginal role only.

More general geometric aspects of dynamical systems are now in the focus of modern numerical reseach. The phrase "geometric integration" [1] was coined reflecting the fact that the new schemes are capable of preserving certain "geometric" invariants exactly. One special new class of integrators consists of "symplectic" schemes [9] which preserve the symplectic 2-form when solving Hamiltonian systems (but do not preserve scalar invariants such as the Hamiltonian function). Other classes deal with more general features such as Lie group actions.

Indeed, often the geometry of a dynamical system is connected to an action of a Lie group on the system or on the underlying phase space. This often restricts the exact solutions to certain submanifolds of the phase space. Although it is an as yet unsolved problem how to conserve scalar invariants systematically in a numerically stable way, the problem of keeping numerical solutions on submanifolds can be solved in certain situations. In particular, if invariants of the dynamical system are invariants of a Lie group action, then the integrators described in the following can be used to preserve them exactly.

Let us consider a concrete example: the motion of a spinning top is given by the dynamical system

$$\frac{d}{dt}\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} (J_2 - J_3)\, y_2\, y_3 \\ (J_3 - J_1)\, y_3\, y_1 \\ (J_1 - J_2)\, y_1\, y_2 \end{pmatrix} . \tag{14}$$

The constant parameters J_i are related to the moments of inertia of the top. With $J = \mathrm{diag}(J_1, J_2, J_3)$ and the usual cross product $\wedge$ on $\mathbb{R}^3$ one can write this system as $d\vec{y}/dt = \vec{y} \wedge J\vec{y}$. There are two scalar quantities $H_1(\vec{y}) =< \vec{y}, \vec{y} >$ and $H_2(\vec{y}) =< \vec{y}, J\vec{y} >$ which are preserved by the dynamics:

$$\begin{aligned} \frac{d}{dt} H_1(\vec{y}) &= 2 < \vec{y}, \frac{d\vec{y}}{dt} > = 2 < \vec{y}, \vec{y} \wedge J\vec{y} > = 0 \,, \\ \frac{d}{dt} H_2(\vec{y}) &= 2 < J\vec{y}, \frac{d\vec{y}}{dt} > = 2 < J\vec{y}, \vec{y} \wedge J\vec{y} > = 0 \,. \end{aligned}$$

Consequently, the orbit of the system through the point $\vec{y}(t_0) = \vec{y}_0$ is the intersection of the sphere $\{\vec{y};\ H_1(\vec{y}) = H_1(\vec{y}_0)\}$ and the ellipsoid $\{\vec{y};\ H_2(\vec{y}) = H_2(\vec{y}_0)\}$. This fixes the geometric behavior of the system completely up to the actual parametrization of the solution curves by the time t. A simple consequence is that all solution curves are closed (apart from four special orbits, the so-called “separatrices”).

Traditionally, an attempt to solve the system (14) numerically concentrates on computing the exact solution $\vec{y}(t)$ as accurately as possible within reasonable limits given by the costs of the numerical algorithm. I.e., the emphasis is put on computing the actual parametrization of the solution curves. Typically, the geometric nature of the orbits is ignored in this approach. In the example above this means that traditional integrators do not care very much about preserving the conservation laws $H_{1,2}$ numerically[1]. Consequently, with such schemes it would be difficult to detect by a numerical simulation that orbits are closed.

In the following we consider a class of dynamical systems with some Lie group background. This class includes equations of the form

$$\frac{d\vec{y}}{dt} = f(t, \vec{y})\, \vec{y} \,, \quad \vec{y}(t_0) = \vec{y}_0 \,, \quad \vec{y} \in \mathbb{C}^n$$

with a field $f(t, \vec{y}) \in \mathbb{C}^{n\times n}$ of $n \times n$ matrices. There is following result restricting the solution to a submanifold of $\mathbb{C}^n$:

> *Let $f(t, \vec{y})$ take values in some Lie algebra $\subset \mathbb{C}^{n\times n}$, let G be the corresponding matrix Lie group. Then for all times t the solution $\vec{y}(t)$ lies on the manifold $M = \{x\, \vec{y}_0\ ;\ x \in G\}$.*

The challenge of “integrating the equation geometrically” by some numerical algorithm is to guarantee that the numerical solution lies on the manifold $M \subset \mathbb{C}^n$, too. The orbit M of the matrix group acting on the vector $\vec{y}_0$ is an instance of a general concept: M is a so-called homogeneous manifold.

A new breed of numerical algorithms designed in the last years is capable of integrating equations on homogeneous manifolds. They trace the dynamical system on the manifold back to an equivalent dynamical system on the underlying group. This is solved using structural properties specific to Lie groups. A particular algorithm designed by Munthe-Kaas and co-workers [6, 7] is implemented in the routine `numeric::odesolveGeometric` available with the next *MuPAD* release. The code is based on Ref. [11]. The geometric properties of this routine are formulated in the section “The *MuPAD* Implementation” below.

The two conserved quantities of the spinning top introduced above have a geometric origin: there are two Lie groups acting in the background, the solution curves are the intersections of certain homogeneous manifolds. We shall solve this equation numerically by *MuPAD*’s `odesolveGeometric` (cf. Example 2 below). Depending on the formulation this Lie group integrator is capable of preserving either one of the two conservation laws exactly.

[1] It should be mentioned, though, that symplectic integrators of Runge-Kutta type preserve quadratic invariants and will preserve the geometry of this particular example. However, generic symplectic integrators are implicit methods, i.e., expensive.

Some Background: Lie Groups, Lie Algebras and Homogeneous Manifolds

Every finite dimensional Lie group/algebra can be realized as a group/algebra of matrices. We give a brief and simplified review of the basic notions for the matrix case.

A (complex) **Lie group** G consists of invertible (complex) matrices with the group operations given by matrix multiplication and the usual inversion. The term "Lie" refers to the fact that the group is not discrete: it is also a manifold, i.e., the set is "spanned" by a number of free (non-discrete) parameters. Mathematically: for each point in G there is a neighborhood which is homeomorphic to an open set in $\mathbb{C}^d$.

Examples of matrix Lie groups are the " general linear group" of all invertible matrices

$$G = GL(n) = \{ x \in \mathbb{C}^{n \times n} ; \ \det(x) \neq 0 \ \} ,$$

the "special linear group"

$$G = SL(n) = \{ x \in \mathbb{C}^{n \times n} ; \ \det(x) = 1 \ \} ,$$

the group of unitary matrices

$$G = U(n) = \{ x \in \mathbb{C}^{n \times n} ; \ x^T x = \mathbb{1} \ \} ,$$

or the group of invertible upper triangular matrices

$$G = \{ x \in GL(n) ; \ x \text{ is upper triangular} \ \}.$$

The above examples show that typical matrix Lie groups arise by putting constraints on the elements in $GL(n)$: $SL(n)$ is given by those matrices for which the *polynomial function* det has the constant value 1, the unitary group is given by a set of *quadratic* equations $x^T x = \mathbb{1}$, the upper triangular group is given by a set of *linear* constraints (lower triangular components = 0).

There is a **Lie algebra** g associated with a Lie group G:

$$g = \{ \frac{dx(t)}{dt} \, x(t)^{-1} ; \ x(t) \text{ is an arbitrary smooth curve in } G \ \}.$$

For $GL(n)$ the algebra is the space $g = gl(n) = \mathbb{C}^{n \times n}$ of *all* $n \times n$ matrices, the algebra associated with $SL(n)$ is given by the traceless matrices

$$g = sl(n) = \{ u \in \mathbb{C}^{n \times n} ; \ \mathrm{tr}(u) = 0 \ \},$$

the algebra associated with $U(n)$ consists of all skew hermitean matrices

$$g = su(n) = \{ u \in \mathbb{C}^{n \times n} ; \ u^T + u = 0 \ \},$$

the algebra associated with the upper triangular group consists of all upper triangular matrices.

Similar to the realization of matrix groups as subgroups of the general linear group $GL(n)$ we may think of these Lie algebras as embedded in the general matrix algebra $gl(n) = \mathbb{C}^{n \times n}$ via constraints. However, for the Lie groups the constraints are in general *non-linear*, whereas for the Lie algebras the constraints are always given by *linear* equations: the Lie algebra is a vector space.

The following two important facts are key issues for the numerical considerations:

1) the matrix commutator $[u, v] := u\,v - v\,u$ of two Lie algebra elements is again an element of the Lie algebra,

2) the exponential function $\exp:\ g \mapsto G$ given by

$$\exp(u) = \mathbb{I} + u + \frac{1}{2!}u^2 + \cdots$$

always maps algebra elements to elements of the corresponding group. In fact, $\exp$ is a homeomorphism between a neighbourhood of the origin in g and a neighbourhood of the unit element $\mathbb{I}$ in G.

A **Lie group action** $\Lambda\ :\ G \times M\ \mapsto\ M$ of G on a manifold M is a smooth function compatible with group multiplication:

$$\Lambda(\mathbb{I}, y) = y\ , \quad \Lambda(x_2 x_1, y) = \Lambda(x_2, \Lambda(x_1, y))\ , \quad x_1, x_2 \in G\ , \quad y \in M\ .$$

A typical example of an action (Λ, G, M) is the space $M = \mathbb{C}^{n\times p}$ of $n \times p$ matrices, on which $G = GL(n)$ or one of its subgroups acts via matrix multiplication $\Lambda(x, y) = x\, y$. Another important example is the space $M = \mathbb{C}^{n\times n}$ of $n \times n$ matrices with $GL(n)$ or one of its subgroups acting by conjugation $\Lambda(x, y) = x\, y\, x^{-1}$.

A manifold M with a group action Λ is called **homogeneous**, if each pair of points $y_{1,2} \in M$ can be connected via a group action: $y_2 = \Lambda(x, y_1)$ with some $x \in G$. Starting with an action Λ of G on the matrix space $\mathbb{C}^{n\times p}$, we may think of the homogeneous manifolds embedded in $\mathbb{C}^{n\times p}$ as the group orbits through points $y_0 \in \mathbb{C}^{n\times p}$:

$$M_{y_0} = \{\Lambda(x, y_0)\ ;\ x \in G\}\ .$$

E.g., for the unitary group $G = U(n)$ acting on $\mathbb{C}^n = \mathbb{C}^{n\times 1}$ via matrix-vector multiplication $\Lambda(x, \vec{y}) = x\, \vec{y}$ the homogeneous manifolds are spheres

$$M_{\vec{y}_0} = \{x\, \vec{y}_0\ ;\ x \in U(n)\} = \{\vec{y}\ ;\ <\vec{y}, \vec{y}> = <\vec{y}_0, \vec{y}_0>\}\ .$$

Other interesting examples include "isospectral" manifolds of $n \times n$ matrices. Starting with the action $\Lambda(x, y) = x\, y\, x^{-1}$ of $G = GL(n)$ on the space $\mathbb{C}^{n\times n}$ one may consider the homogeneous manifolds

$$M_{y_0} = \{x\, y_0\, x^{-1}\ ;\ x \in GL(n)\ \}\ , \quad y_0 \in \mathbb{C}^{n\times n}$$

consisting of matrices sharing the same eigenvalues.

Of course, any Lie group acts on itself via group multiplication, i.e., Lie groups may be regarded as homogeneous manifolds.

The homogeneous manifolds mentioned above may be regarded as submanifolds of the ambient linear space $\mathbb{C}^{n\times p}$ defined by constraints: spheres are given by the quadratic equation $<\vec{y}, \vec{y}>= const$, isospectral manifolds are defined by the (much more complicated) constraints *eigenvalues*$(y) = const$.

Lie Group Integrators of Munthe-Kaas Type

We consider matrix differential equations of the type $dx/dt = f(t, x)\, x$, where x is a curve in a matrix Lie group G and $f(t, x)$ takes values in the corresponding Lie algebra g.

The goal of Lie group integrators for this equation is to solve this system numerically, while keeping the numerical solution $x(t)$ on the Lie group. As pointed out before, most matrix Lie groups may be regarded as $GL(n)$ subject to certain *non-linear* constraints. Unfortunately, traditional numerical integrators such as the classical Runge-Kutta schemes preserve only linear constraints, whence after a few steps one obtains approximations $x(t) \in \mathbb{C}^{n\times n}$ which are not on the manifold G any more.

The following simple idea helps to take advantage of the classical schemes while keeping the numerical solution on the group: locally, the function $\exp$ is a chart from a linear space (the Lie algebra g) to a neighborhood of the unit element $\mathbb{I}$ of the group. One uses $\exp$ to transform the initial value problem

$$\frac{dx}{dt} = f(t,x)\,x\ , \quad x(t_0) = x_0 \tag{15}$$

on G to a corresponding differential equation on g by putting $x(t) = \exp(u(t))\,x_0$. One solves the equation for $u(t) \in g$ by any classical method. Here it is important to note that classical methods preserve the linear constraints which define the actual Lie algebra g as a subalgebra of $gl(n)$. Consequently, also the numerical solution $u(t)$ lies in g and $x(t) = \exp(u(t))\,x_0$ will automatically lie in G, if the matrix exponential is evaluated up to numerical machine precision, i.e., up to roundoff effects.

One problem in the implementation of this idea is that the equation on the group does in general not give rise to a system on g in closed form. However, there is a representation by an infinite series, which can be used: the ansatz $x(t) = \exp(u(t))\,x_0$ transforms the Lie group equation (15) to the initial value problem

$$\frac{du}{dt} = \text{dexpinv}(u, f(t, \exp(u)\,x_0))\ , \quad u(t_0) = 0 \tag{16}$$

on g, where

$$\text{dexpinv}(u,f) = \sum_{j=0}^{\infty} \frac{B_j}{j!}\,(ad_u)^j f\ , \quad (ad_u)^j f = \underbrace{[u,[u,[..,[u}_{j},f]..]]]$$

and B_j are the usual Bernoulli numbers. The funny name dexpinv is well motivated: the differential (linearization) of $\exp$ at a point $u \in g$ is a linear map $d\exp\ : g \mapsto g$ and $\text{dexpinv} = (d\exp)^{-1}$. We note that the series representation of dexpinv has a nontrivial radius of convergence: the infinite sum exists, if the spectral radius of the linear map $ad_u : f \mapsto [u,f]$ is less than $2\,\pi$. Since for a matrix $u \in gl(n)$ the eigenvalues of ad_u are given by the differences of pairs of eigenvalues of u, one can guarantee the convergence for all matrices u satisfying

$$\|u\| < \pi\ . \tag{17}$$

Following Munthe-Kaas [6] one proceeds to integrate (16) with any classical numerical scheme to obtain a numerical approximation of $u(t_0+h)$, where h is a suitable (small) step size. Here the problem arises, how the infinite series defining dexpinv should be evaluated. Noting

$$u(t_0+h) = h\,f(t_0,x_0) + O(h^2) = O(h)\ , \quad f(t_0+h, \exp(u(t_0+h))\,x_0)) = f(t_0,x_0) + O(h)$$

one finds $(ad_u)^j f = O(h^{j+1})$ for $j \geq 1$. Hence one may truncate the series and replace dexpinv by the finite sum

$$\text{dexpinv}_q(u,f) = \sum_{j=0}^{q-2} \frac{B_j}{j!}\,(ad_u)^j f = \text{dexpinv}(u,f) + O(h^q)\ , \tag{18}$$

where q is the order of the classical integrator. It is easy to verify that the numerical solution of the approximate problem

$$\frac{du}{dt} = \text{dexpinv}_q(u, f(t, \exp(u)\,x_0))\ , \quad u(t_0) = 0$$

obtained by this integrator still approximates the exact solution of the exact equation (16) with the order p.

Suppose that an acceptable step size h and a corresponding numerical approximation $u(t_0+h)$ is found. Since the algebra version (16) of (15) is valid only in some neighbourhood of $u = 0$, one does not proceed to integrate (16) any further. Instead, the accepted solution $u(t_0+h)$ is mapped back to the group:

$$t_1 = t_0 + h\ , \quad x_1 := \exp(u(t_1))\,x_0 \approx x(t_1)\ .$$

Now the process starts again: assuming a numerical approximation $x_n \approx x(t_n)$ the ansatz $x(t) = \exp(u(t))\, x_n$ is used to obtain the next approximation x_{n+1} by solving

$$\frac{dx}{dt} \;=\; f(t,x)\,x\;, \quad x(t_n) = x_n$$

via the approximate Lie algebra version

$$\frac{du}{dt} \;=\; \text{dexpinv}_q(u, f(t, \exp(u)\,x_n))\;, \quad u(t_n) = 0 \tag{19}$$

etc. Thus, the numerical solution of (15) is a mesh x_N defined by Lie group products

$$x_N \;=\; \exp(u_{N-1})\;\cdots\exp(u_1)\;\exp(u_0)\,x_0 \tag{20}$$

with solutions $u_n \in g$ of (19). This ensures that the numerical solution points x_N stay on the manifold G.

The *MuPAD* Implementation: `odesolveGeometric`

We consider the ambient linear space of $\mathbb{C}^{n\times p}$ of $n \times p$ matrices which is to contain the homogeneous submanifold M under consideration. Points $y \in M$ are represented by $n \times p$ *MuPAD* arrays. Points in $\mathbb{C}^n \equiv \mathbb{C}^{n\times 1}$ may also be entered as *MuPAD* lists or as 1-dimensional arrays. The group G under consideration is regarded as a subgroup of $GL(n)$, its elements $x \in G$ are represented by $n \times n$ arrays.

Let `LAMBDA:= proc(x,y) .. end_proc` be a user defined procedure representing the Lie group action Λ of $x \in GL(n) \subset \mathbb{C}^{n\times n}$ on $y \in \mathbb{C}^{n\times p}$. The return value of the call `LAMBDA(x,y)` must be an $n \times p$ array or a corresponding matrix of domain type `Dom::Matrix(Dom::Numerical)` representing a new point in M.

Let `f:= proc(t,y) .. end_proc` be a user defined procedure mapping the scalar time parameter t and $y \in \mathbb{C}^{n\times p}$ to an $n \times n$-array or a corresponding matrix of domain type `Dom::Matrix(Dom::Numerical)`. This function should be regarded as a mapping $f : \mathbb{R} \times M \mapsto g$, where g is the Lie algebra of G.

There is no need to specify the manifold M, the Lie group G or its algebra g. Only the dimensional parameters n and p of the ambient spaces $G \subset \mathbb{C}^{n\times n}$ and $M \subset \mathbb{C}^{n\times p}$ must be entered into the geometric integrator `odesolveGeometric`. These parameters are encoded in the initial condition $y_0 \in M$, which must be an $n \times p$ array, a corresponding matrix of type `Dom::Matrix(Dom::Numerical)` or a list or 1-dimensional array (implying $p = 1$).

The call

```
>> y:= numeric::odesolveGeometric(t0..t, f, y0, LAMBDA, Options);
```

solves the Lie group equation

$$\frac{dx}{dt} \;=\; f(t, \Lambda(x, y_0))\,x\;, \quad x(t_0) = \mathbb{1} \tag{21}$$

on the $n \times n$ matrices and returns the $n \times p$ array $y = \Lambda(x(t), y_0)$. The argument `LAMBDA` is optional, the default action is the usual matrix multiplication $\Lambda(x, y) = x\,y$. For this setting the result $y(t) = x(t)\,y_0$ returned by `odesolveGeometric` is the solution of the matrix differential equation

$$\frac{dy}{dt} \;=\; f(t,y)\,y\;, \quad y(t_0) = y_0$$

on the $n \times p$ matrices $\mathbb{C}^{n \times p}$.

The geometric behavior of the integrator is described as follows. It corresponds to the behaviour of the exact solution as described in the introduction for the action $\Lambda(x, \vec{y}) = x\,\vec{y}$ on $\mathbb{C}^n$.

> *Let $f(t, y)$ take values in some Lie algebra $g \subset \mathbb{C}^{n \times n}$, let $G \subset \mathbb{C}^{n \times n}$ be the corresponding Lie group. Then for all times t the numerical solution returned by* `odesolveGeometric` *lies on the homogeneous manifold $M = \{\Lambda(x, y_0)\ ;\ x \in G\} \subset \mathbb{C}^{n \times p}$.*

There is no need to specify the underlying Lie group or the homogeneous manifold. These objects are implicitly given by f: the Lie algebra g is spanned by the matrices produced by f. The statement above holds for any Lie algebra containing the set $\{f(t, y)\ ;\ y \in \mathbb{C}^{n \times p}\}$. This fixes the corresponding subgroup G of $GL(n)$ and defines the manifold M on which both the exact and the numerical dynamics takes place.

We emphasize that the use of the geometric integrator makes only sense, if the matrix output of f lies in a proper subalgebra $g \neq gl(n) = \mathbb{C}^{n \times n}$. If f produces generic matrices without any special structure, then $g = gl(n)$, $G = GL(n)$ and the solution can only be restricted to $M = \{\Lambda(x, y_0)\ ;\ x \in GL(n)\}$. In many cases the dimension of this manifold M is not smaller than the dimension of the ambient space $\mathbb{C}^{n \times p}$ and there is no point in using the quite expensive geometric algorithm. Any classical integration scheme will do just as well.

Internally, the Munthe-Kaas algorithm discussed in the last section is used to integrate the Lie group equation $dx/dt = \tilde{f}(t, x)\,x$ with $\tilde{f}(t, x) = f(t, \Lambda(x, y_0))$. Note that the final result (20) is obtained via exponentials of the matrix output of f. If f produces values in a proper subalgebra $g \neq \mathbb{C}^{n \times n}$, then their exponentials automatically lie on the corresponding proper subgroup $G \neq GL(n)$ and the return value $\Lambda(x(t), y_0)$ automatically lies on the proper homogeneous submanifold $M = \{\Lambda(x, y_0)\ ;\ x \in G\} \neq \mathbb{C}^{n \times p}$. Thus, in principle no a priori knowledge of the underlying group or the manifold is required. The algorithms will automatically restrict the solution to the manifold associated with the initial condition and the group generated by the output of f.

The algorithm in `odesolveGeometric` is based on a traditional Runge-Kutta scheme used for integrating (16). The name of this scheme may be passed to `odesolveGeometric` as an optional argument. There are presently 27 different Runge-Kutta schemes of order 1 through 8 to choose from. The default method is the embedded Runge-Kutta-Fehlberg pair `RKF78` of orders 7 and 8. The order of the Lie group integrator is identical with the order of the classical scheme.

A further optional argument of the form `RelativeError=` *tol* may be passed to the geometric integrator. The numerical value *tol* is a tolerance used to control the local discretization errors of the Runge-Kutta steps: step sizes are chosen adaptively to produce local relative errors below *tol*. Note that the approximation of the solution curve $y(t)$ is controlled by *tol*, whereas the geometric features of the Lie group integration are preserved within working precision $10^{-\text{DIGITS}}$. The default value for *tol* is $10^{-\text{DIGITS}}$. This implies that local errors are of the same magnitude as roundoff errors. Only values *tol* $\geq 10^{-\text{DIGITS}}$ are accepted.

In most applications one should choose $1 \gg$ *tol* $\gg 10^{-\text{DIGITS}}$. For *tol* $\approx 10^{-\text{DIGITS}}$ there is usually no point in using `odesolveGeometric`. Note that also classical algorithms preserve the geometric quantities (locally) to the extent by which the numerical solution approximates the exact solution. For *tol* $\approx 10^{-\text{DIGITS}}$ the classical integrator `odesolve` in the `numeric` package produces results of similar quality, since discretization errors are of the same magnitude as roundoff. For a given tolerance *tol* `odesolve` is usually faster than `odesolveGeometric`. If geometric features are to be preserved well, while the precision of the solution itself is of less importance, then `odesolve` must still control this by a suitably small value of *tol*. Decreasing *tol*, however, is usually much more expensive than increasing the working precision via DIGITS.

Examples

Example 1: We want to integrate the system

$$\frac{d\vec{y}}{dt} = f(\vec{y})\,\vec{y}\ , \quad \vec{y}(0) = \vec{y}_0 = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \ , \quad \vec{y} \in \mathbb{R}^n$$

with the matrix field

$$f(\vec{y}) = \begin{pmatrix} 0 & -y_1 & 0 & \dots & 0 \\ y_1 & 0 & -y_2 & \ddots & \vdots \\ 0 & y_2 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & 0 & -y_{n-1} \\ 0 & \dots & 0 & y_{n-1} & 0 \end{pmatrix} \ , \quad \text{i.e.,} \quad \frac{d\vec{y}}{dt} = \begin{pmatrix} -y_1\, y_2 \\ \vdots \\ y_{i-1}^2 - y_i\, y_{i+1} \\ \vdots \\ y_{n-1}^2 \end{pmatrix} \quad (\leftarrow\ i)\ .$$

This system calls for geometric integration: we note that the matrix field f yields skew symmetric matrices, whence the 2-norm of the vector $\vec{y}$ is preserved by the dynamics:

$$\frac{d}{dt} < \vec{y}, \vec{y} > = < f\,\vec{y}, \vec{y} > + < \vec{y}, f\,\vec{y} > = < \vec{y}, (f^T + f)\,\vec{y} > = \ 0\ .$$

Geometrically speaking, the dynamics takes place on the homogeneous manifold

$$M = \{\vec{y} \in \mathbb{R}^n;\ < \vec{y}, \vec{y} > = < \vec{y}_0, \vec{y}_0 >\}$$

(a sphere) equipped with the standard action of the unitary (orthogonal) group. We define the matrix field f by the following *MuPAD* procedure, which will be called by `odesolveGeometric` in the form $f(t, y)$ with an $n \times 1$ *MuPAD* array y. We note that the components `y[i,1]` of such arrays can also be addressed by `y[i]`. The matrix output of f is an $n \times n$ array:

```
>> f:= proc(t, y)
&>     local n, A, i, j;
&>     begin n:= op(y, [0, 2, 2]);
&>           A:= array(1..n, 1..n, [[0 $ n] $ n]);
&>           for i from 1 to n-1 do
&>             A[i,i+1]:= -y[i]; A[i+1,i]:= y[i];
&>           end_for;
&>           A;
&>     end_proc:
```

We fix the dimension $n = 3$ and define the initial condition $\vec{y}_0$ by a 1-dimensional array y_0. With the initial condition we define the ambient space in which the dynamics takes place: investigating $\vec{y}_0$ the integrator finds out that vectors in $\mathbb{R}^n$ (actually, in $\mathbb{C}^n = \mathbb{C}^{n\times 1}$) are to be considered:

```
>> n:= 3: y0:= array(1..n, [1 $ n]):
>> y:= numeric::odesolveGeometric(0..1, f, y0, RelativeError = 10^(-4));
```

```
          +-                                          -+
          | 0.4689254114, 0.4556785967, 1.603890886 |
          +-                                          -+
```

Since local relative errors are of the magnitude 10^{-4}, only the first 4 digits of the solution are reliable at best. However, due to the geometric nature of the integration, the departure of the solution from the sphere is a pure roundoff effect. It should not be more than the default numerical precision $10^{-\text{DIGITS}} = 10^{-10}$. In this case the Euclidean length of the vector is preserved quite well indeed:

```
>> _plus(y[i]^2 $ i=1..n) - _plus(y0[i]^2 $ i=1..n);

                         -0.000000000000000000002168404345
```

We compare this to the standard integrator `odesolve`, which solves $d\vec{y}/dt = g(t, \vec{y}) = f(t, \vec{y})\, \vec{y}$ by a straightforward Runge-Kutta iteration on $\mathbb{R}^n$. The vector field $g(t, \vec{y})$ is defined by the following procedure, which is called by `odesolve` with a 1-dimensional array y and must produce a list or a 1-dimensional array as output:

```
>> g:= proc(t, y)
&>     local n, A, i;
&>     begin n:= op(y, [0,2,2]);
&>           A:= array(1..n);
&>           A[1]:= -y[1]*y[2];
&>           for i from 2 to n-1 do
&>               A[i]:= y[i-1]^2 - y[i]*y[i+1];
&>           end_for;
&>           A[n]:= y[n-1]^2;
&>           A;
&>     end_proc;
```

We call the standard integrator with the option `RelativeError`. This option has been added to the functionality of `odesolve` for the next *MuPAD* release:

```
>> y:= numeric::odesolve(0..1, g, y0, RelativeError = 10^(-4));

                +-                                       -+
                | 0.4689076829, 0.4556277126, 1.60394316 |
                +-                                       -+
```

The results returned by the 2 integrators differ after the first 4 leading digits, as has to be expected from the setting `RelativeError` $= 10^{-4}$. The departure of the solution from the sphere is of the same order of magnitude as the discretization errors, i.e., about 10^{-4}:

```
>> _plus(y[i]^2 $ i=1..n) - _plus(y0[i]^2 $ i=1..n);

                                    0.0001046904369
```

Example 2: We wish to solve the dynamics (14) considered in the introduction. In order to use the Lie group integrator `odesolveGeometric` the system has to be written in the form $d\vec{y}/dt = f(\vec{y})\, \vec{y}$ with some matrix field f. There are several ways of rewriting the equations. We consider the following two versions:

$$\frac{d\vec{y}}{dt} = \begin{pmatrix} (J_2 - J_3)\, y_2\, y_3 \\ (J_3 - J_1)\, y_3\, y_1 \\ (J_1 - J_2)\, y_1\, y_2 \end{pmatrix} \overset{(a)}{=} \begin{pmatrix} 0 & -J_3\, y_3 & J_2\, y_2 \\ J_3\, y_3 & 0 & -J_1\, y_1 \\ -J_2\, y_2 & J_1\, y_1 & 0 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$$
$$\overset{(b)}{=} \begin{pmatrix} 0 & J_2\, y_3 & -J_3\, y_2 \\ -J_1\, y_3 & 0 & J_3\, y_1 \\ J_1\, y_2 & -J_2\, y_1 & 0 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} .$$

We define the two invariants of the system, choose numerical values for the parameters J_i and fix some initial condition:

```
>> H1:= proc(y) begin float(y[1]^2 + y[2]^2 + y[3]^2) end_proc:
>> H2:= proc(y) begin float(J1*y[1]^2 + J2*y[2]^2 + J3*y[3]^2) end_proc:
>> J1:= 1: J2:= 2: J3:= 3: y0:= [1, 1, 1]:
```

Using the representation (a) we define the matrix field and call `odesolveGeometric` to integrate the system from $t_0 = 0$ to $t = \pi$:

```
>> f:= proc(t, y)
&>     begin
&>       array(1..3,1..3,[[    0    ,-J3*y[3],  J2*y[2]],
&>                         [ J3*y[3] ,    0    , -J1*y[1]],
&>                         [-J2*y[2] , J1*y[1],     0   ]]);
&>     end_proc:
>> y:= numeric::odesolveGeometric(0..PI, f, y0, RelativeError = 10^(-4));

               [0.005644010633, 1.732033364, 0.005345107987]
```

We test the departure from the level manifolds of the conservation laws:

```
>> H1(y) - H1(y0), H2(y) - H2(y0);

             -0.0000000000000000004336808689, -0.000003284676636
```

Apparently the function H_1 is preserved up to roundoff effects, whereas the solution departs from the level set of H_2 by a small but significant amount. This is explained easily: note that f produces skew symmetric matrices. Hence, just as in Example 1, the geometric integrator keeps the solution on the homogeneous manifold given by the unitary (orthogonal) group acting on $\vec{y}_0$ (the sphere of radius $\sqrt{H_1(\vec{y}_0)}$). This manifold is the level set of H_1. There is no reason, why H_2 should be preserved exactly.

Next we use representation (b):

```
>> f:= proc(t ,y)
&>     begin
&>       array(1..3,1..3,[[    0,    J2*y[3] , -J3*y[2]],
&>                        [-J1*y[3] ,   0    ,  J3*y[1]],
&>                        [ J1*y[2] ,-J2*y[1],     0   ]]);
&>     end_proc:
>> y:= numeric::odesolveGeometric(0..PI, f, y0, RelativeError = 10^(-5));
```

```
              [0.005462666146, 1.732033275, 0.005526409474]

>> H1(y) - H1(y0), H2(y) - H2(y0);

           -0.0000003502401266, -0.00000000000000000005637851292
```

This time H_2 is preserved up to roundoff effects, whereas the solution departs from the level set of H_1 by a small but significant amount. We have to investigate the output of f in order to understand this behavior. Note that the matrices $u = f(\vec{y})$ are not skew symmetric, but satisfy $u^T J + J u = 0$. It is easily verified that for any fixed matrix J the space

$$g = \{u \in \mathbb{C}^{n \times n} \; ; \; u^T J + J u = 0\}$$

is a Lie algebra. The corresponding Lie group is the "J-unitary" group

$$G = \{x \in GL(n) \; ; \; x^T J \, x = J\} \; .$$

In the geometric integration process the final result is obtained by $\vec{y}(t) = x(t)\, \vec{y}_0$ with some J-unitary $x(t) \in G$. This group leaves the manifolds $M = \{\vec{y} \in \mathbb{R}^n ; < \vec{y}, J\vec{y} >= const\}$ invariant:

$$< x\,\vec{y}, J\, x\vec{y} > = < \vec{y}, x^T J\, x\vec{y} > = < \vec{y}, J\,\vec{y} > \; .$$

In other words, the level sets M of H_2 are homogeneous manifolds with the J-unitary group G acting by standard matrix-vector multiplication. This explains the observation: `odesolveGeometric` automatically preserves homogeneous manifolds with respect to the Lie group generated by the matrix output of f.

Finally, we use the standard integrator `odesolve` to solve the system:

```
>> g:= proc(t, y)
&>     begin [(J2-J3)*y[2]*y[3],
&>            (J3-J1)*y[3]*y[1],
&>            (J1-J2)*y[1]*y[2]];
&>     end_proc:
>> y:= numeric::odesolve(0..PI, g, y0, RelativeError = 10^(-5));

              [0.005494765735, 1.732033651, 0.005494765735]
```

We check the preservation of the two invariants. The solution departs significantly from both level sets:

```
>> H1(y) - H1(y0), H2(y) - H2(y0);

                   0.0000009553044112, 0.000001910608822
```

Example 3: We consider the matrix dynamics

$$\frac{dy}{dt} = f(y)\, y - y\, f(y) \; , \quad y(t_0) = y_0 \; , \quad y, f(y) \in \mathbb{R}^{n \times n} \; . \tag{22}$$

The solution of this system is given by $y(t) = x(t)\, y_0\, x(t)^{-1}$, where the matrix curve $x(t)$ satisfies the Lie group equation

$$\frac{dx}{dt} = f(x y_0 x^{-1})\, x \; , \quad x(t_0) = \mathbb{I} \; . \tag{23}$$

Consequently, for any time t the solution $y(t)$ has the same eigenvalues as y_0.

We can either integrate the Lie group equation (23) to compute $x(t)$ and then compute $y(t)$ from this result. Alternatively, we can use the action $\Lambda(x, y) = x\,y\,x^{-1}$ in `odesolveGeometric` and integrate (22) directly. The solution $y(t)$ lies on the homogeneous isospectral manifold of $G = GL(n)$ acting on y_0 via Λ.

We define the group action:

```
>> LAMBDA:= proc(x,y)
&>          begin x:= Dom::Matrix(Dom::Float)(x);
&>                y:= Dom::Matrix(Dom::Float)(y);
&>                return(x*y*x^(-1));
&>          end_proc:
```

Internally this function will be called with $n \times n$ arrays x and y. For convenience, these arrays are converted to elements of the matrix domain with floating point entries. Thus the conjugation can be computed via the overloaded arithmetical operators `*` and `^`.

Next we consider an interesting matrix field $f(y)$: it keeps the strictly lower triangular part of the $n \times n$ matrix y, sets the diagonal to 0 and fills the upper triangular part with minus the transposed of the strictly lower triangular part. The result is a skew symmetric matrix.

```
>> f:= proc(t, y)
&>     local n, i, j, A;
&>     begin n:= op(y, [0, 2, 2]);
&>           A:= y;
&>           for i from 1 to n do
&>             A[i,i]:= 0;
&>             for j from i+1 to n do
&>               A[i,j]:= -y[j,i];
&>             end_for;
&>           end_for;
&>           A;
&>     end_proc:
```

This rather bizarre function has a deep geometric background. It gives rise to an exciting dynamical system (22), the so-called Toda lattice equation [2, 10].

Since f produces skew symmetric matrices, the solution $x(t)$ of (23) is unitary (orthogonal) and $y = xy_0x^{-1} = xy_0x^T$ is symmetric, if the initial matrix y_0 is symmetric. Geometrically speaking: the dynamics may be restricted to a homogeneous manifold of isospectral symmetric matrices.

We consider the dimension $n = 3$ and choose the following tri-diagonal initial condition y_0:

```
>> y0:= array(1..n, 1..n, [[1, 1, 0],
&>                          [1, 1, 2],
&>                          [0, 2, 1]]);
```

Now we integrate the Toda system from $t_0 = 0$ to $t = 1$:

```
>> y:= numeric::odesolveGeometric(0..1, f, y0, LieGroupAction=LAMBDA,
```

```
&>                                 RelativeError = 10^(-4));

      +-                                                           -+
      |      -1.048326428,     0.6210888824, -0.00000007797189591   |
      |                                                             |
      |      0.6210888824,     0.8255356165,     0.1755266709       |
      |                                                             |
      |  -0.00000007797189585, 0.1755266709,      3.222790812       |
      +-                                                           -+
```

As pointed out above, the symmetry of the matrix $y = xy_0x^{-1} = xy_0x^T$ stems from the fact that x stays on the orthogonal group. Due to the geometric nature of the integration orthogonality of x and hence symmetry of y is preserved up to roundoff effects. By construction the solution must have the same eigenvalues as the initial matrix:

```
>> numeric::eigenvalues(y), numeric::eigenvalues(y0);

    [-1.236067977, 1.0, 3.236067977], [-1.236067977, 1.0, 3.236067977]
```

We remark that the Toda lattice equation is an "integrable system": it can be solved algebraically without integrating any differential equation. Following Symes [10] for symmetric y_0 the orthogonal matrix $x(t)$ in $y(t) = x(t)y_0x(t)^T$ has the following interpretation: its inverse $Q(t) = x(t)^T$ is the Q factor of a QR factorization of the matrix $\exp(-t\, y_0)$. Using the new routine `numeric::expMatrix` we can compute the solution $y(1)$ exactly up to roundoff effects:

```
>> M:= Dom::Matrix(Dom::Float):
>> y0:= M(y0):
>> ([Q,R]):= numeric::factorQR(numeric::expMatrix(-y0)):
>> Q:= M(Q):
>> y:= Q^(-1)*y0*Q;

  +-                                                                      -+
  |          -1.04832646,        0.6210888374, -0.00000000000000001458251  |
  |                                                                        |
  |          0.6210888374,       0.8255356342,         0.1755265687        |
  |                                                                        |
  | -0.000000000000000014474098, 0.1755265687,          3.222790826        |
  +-                                                                      -+
```

The Costs

Geometric integration is quite expensive. We consider $d\vec{y}/dt = f(t, \vec{y})\, \vec{y}$ with $\vec{y} \in \mathbb{C}^n$. Traditional integrators solve this system on the n-dimensional space $\mathbb{C}^n$, whereas `odesolveGeometric` solves the Lie group equation (21) on the $n \times n$ matrices, i.e., on a space of dimension n^2. Following costs arise in solving (16):

- exponentials of $n \times n$ matrices must be computed,
- Λ and f are evaluated,
- `dexpinv` (18) is evaluated via matrix commutators.

Let us consider a traditional explicit Runge-Kutta-scheme of order q with s stages used to solve (16). A single step of the Lie group integrator needs

$$s \text{ matrix exponentials} \quad + \quad s \text{ evaluations of } \Lambda \text{ and } f \quad + \quad (s-1)\,(q-2) \text{ matrix commutators}.$$

Each computation involving a matrix exponential has an asymptotic complexity of $O(n^2)$ for the standard action $\Lambda(\exp(u), \vec{y}) = \exp(u)\,\vec{y}$. For user defined `LAMBDA` this increases to $O(n^3)$. Each commutator costs $O(n^3)$ elementary operations.

If we compare this to the costs of s evaluations of g for a traditional Runge-Kutta step applied to $d\vec{y}/dt = g(t, \vec{y}) = f(t, \vec{y})\,\vec{y}$ on $\mathbb{C}^n$ is is clear that general-purpose geometric integration cannot in general compete with traditional schemes. The natural applications of geometric integrators are numerical simulations in which the preservation of geometric invariants is of vital importance.

As pointed out before, the typical application involves moderate tolerances tol to control the discretization errors: $1 \gg tol \gg 10^{-\text{DIGITS}}$. Here the geometric integrator has the advantage of preserving the invariants up to roundoff effects of the magnitude $10^{-\text{DIGITS}}$. Traditional integrators can achieve this goal only by reducing tol, which is a very expensive measure.

We note that the algorithm `odesolveGeometric` is A-stable, because the linear system $d\vec{y}/dt = f\vec{y}$ with a constant matrix f is solved exactly! Unfortunately, concerning the automatic step size control implemented in `odesolveGeometric`, we note that there are certain thresholds (17) limiting the maximal step size. Even for constant f step size restrictions apply, because otherwise the evaluation of `dexpinv` becomes numerically unstable (despite the fact that this function reduces to the identity map in this special situation). The limitation is comparable to the step size restriction for stiff problems in the traditional approach, if the integrator is not A-stable.

We also note that the expenses in `odesolveGeometric` grow with the stage number of the underlying classical Runge-Kutta scheme. One needs high stage numbers to obtain high order. For moderate tolerances low order schemes may be suitable for a variety of problems and it is sometimes advisable not to use the default method `RKF78` (an embedded Runge-Kutta-Fehlberg pair of orders 7 and 8 with $s = 13$ stages). One may try lower order schemes such as the embedded Cash-Karp pair `CK45` of orders 4 and 5 with $s = 7$ stages to cut the costs.

The Exponential of a Matrix

As seen above, *MuPAD*'s geometric integrator makes intensive use of matrix exponentials. In fact, in many cases the evaluation of $f(t, \Lambda(\exp(u), \vec{y}))$ in the geometric integrator does not require the matrix $\exp(u)$, but only the matrix-vector product $\exp(u)\,\vec{y}$.

The function `numeric::expMatrix` [11], available with the next *MuPAD* release, provides fast algorithms for computing $\exp(u)$ or $\exp(u)\,\vec{y}$, respectively. It implements some of the numerous standard algorithms used in numerical analysis [5]. Extensive testing of different methods revealed that the rather “naive” approach of evaluating the Taylor expansion $\exp(u) = \mathbb{I} + u + u^2/2 + ..$ in an appropriate way is often quite competitive. It is used as the default method in `expMatrix`. Special formulas for traceless 2×2 matrices and skew symmetric 3×3 matrices are implemented.

A modern Krylov type algorithm [3, 8] for computing $\exp(u)\,\vec{y}$ is also available in `expMatrix` [11]. It is very fast for larger matrices, if $\vec{y}$ is spanned by few eigenvectors of u. Generally, for matrices u with only few clusters of similar eigenvalues the algorithm reduces the computation to the problem of exponentiating a matrix, whose dimension is small (given by the number of clusters).

We consider the exponential of the $n \times n$ Hilbert matrix $u_{ij} = 1/(i + j - 1)$ as well as $\exp(u)\,\vec{y}$ with $\vec{y} = (1, \ldots, 1)^T$. The matrix $\exp(u)$ is computed by:

```
>> u:= array(1..n, 1..n, [1/(i+j-1) $ j=1..n] $ i=1..n]):
>> numeric::expMatrix(u);
```

The vector $\exp(u)\,\vec{y}$ is computed by:

```
>> y:= array(1..n, [1 $ n]):
>> numeric::expMatrix(u, y);
```

Here the default algorithm based on the Taylor series is used. The Krylov method is invoked by an option:

```
>> numeric::expMatrix(u, y, Krylov);
```

The results are computed with 10 and 100 digits, respectively. The timings in seconds given in the following table clearly show that computing $\exp(u)\,\vec{y}$ by a suitable algorithm can be much faster than computing the matrix $\exp(u)$ and then multiply it with the vector. Corresponding timings for the numerical computation of $\exp(u)$ in *Mathematica 3.0* (Mma) and *Maple V.5* are shown. These systems do not provide special fast algorithms to compute $\exp(u)\,\vec{y}$.

	DIGITS = 10					DIGITS = 100				
	$\exp(u)$			$\exp(u)\,\vec{y}$		$\exp(u)$			$\exp(u)\,\vec{y}$	
				Taylor	Krylov				Taylor	Krylov
n	Mma	*MuPAD*	Maple	*MuPAD*	*MuPAD*	Mma	*MuPAD*	Maple	*MuPAD*	*MuPAD*
2	0.01	0.02	0.03	0.03	0.05	0.14	0.11	0.27	0.12	0.15
10	0.07	0.81	2.3	0.17	0.29	3.4	4.6	41	1.3	2.7
50	1.9	79	290	5.5	2.8	180	530	17 000	45	14
100	8.6	670	3 500	23	8.8	1 100	4 000		180	33

In *Mathematica* the computations with 10 digits are very fast: compiled kernel functions do the job using hardware floats. The *MuPAD* timings for $\exp(y)\,\vec{y}$ show that the Krylov approach is quite efficient for this example: most eigenvalues of high dimensional Hilbert matrices are very small and may be regarded as a single cluster.

References

[1] C.J. BUDD AND A. ISERLES, *Geometric Integration: Numerical Solution of Differential Equations on Manifolds*, Technical Report 1998/NA10, DAMPT, University of Cambridge `http://www.damtp.cam.ac.uk/user/na/reports.html`

[2] H. FLASCHKA, *The Toda lattice*, Phys. Rev. **B9**, pp. 1924-1925 (1974).

[3] M. HOCHBRUCK AND C. LUBICH, *On Krylov Subspace Approximations to the Matrix Exponential Operator*, SIAM J. Numer. Anal. **34**, pp. 1911-1925 (1997).

[4] A. ISERLES, *Insight, not just numbers*, Technical Report 1997/NA10, DAMPT, University of Cambridge `http://www.damtp.cam.ac.uk/user/na/reports.html`

[5] C.B. MOLER AND C.F. VAN LOAN, *Nineteen dubious ways to compute the exponential of a matrix*, SIAM Review **20**, pp. 801–836 (1978).

[6] H. MUNTHE-KAAS, *High Order Runge-Kutta Methods on Manifolds*, J. Appl. Numer. Maths., to appear. `http:://www.ii.uib.no//~hans`

[7] H. MUNTHE-KAAS AND A. ZANNA, *Numerical Integration of Differential Equations on Homogeneous Manifolds*, in *Foundations of Computational Mathematics*, F. Cucker (ed.), Springer 1997.

[8] Y. SAAD, *Analysis of some Krylov Subspace Approximations to the Matrix Exponential Operator*, SIAM J. Numer. Anal. **29**, pp. 209–228, (1992).

[9] J.M. SANZ-SERNA AND M.P. CALVO, *Numerical Hamiltonian Problems*, Chapman & Hall, 1994.

[10] W.W. SYMES, *The QR Algorithm and Scattering for the Finite Nonperiodic Toda Lattice*, Physica **4D**, pp. 275–280, (1982).

[11] R. WAGNER, *Numerik dynamischer Systeme auf Lie-Gruppen: Theorie und Implementation in MuPAD*, diploma thesis, Univ. Paderborn, Germany, December 1998.

Symmetrieberechnungen bei quantenmechanischen Spinketten

Claudia Nolte
Universität Paderborn, claudian@uni-paderborn.de

Der vorliegende Artikel beschreibt eine an der U-GH Paderborn durchgeführte Diplomarbeit [10], deren Ziel die Entwicklung und Implementation eines Programmpaketes in MuPAD zum Auffinden von Symmetrien bzw. Mastersymmetrien im Fall spezieller quantenmechanischer Systeme, den Spinketten zum Spin $1/2$ war.

Einführung

Die ersten Versuche der Implementation von Algorithmen zur Bestimmung von Symmetrien und Mastersymmetrien quantenmechanischer Spinketten an der U-GH Paderborn gehen auf das Jahr 1986 zurück. Damals implementierten Falck[3] und Wittebrok[11] Algorithmen zur Suche von Symmetrien und Mastersymmetrien von Spinketten unter dem Computeralgebrasystem MAPLE [5]. Mit diesen Programmen wurden Symmetrien und Mastersymmetrien sowohl im Fall der Operator Lie Algebra (Falck) als auch im Fall der Vektorfeld Lie Algebra (Wittebrok) bestimmt.

Ziel der hier beschriebenen Arbeit ist es nun, die in den Vorgängerarbeiten implementierten Algorithmen zu übernehmen, zu erweitern und sie auf dem Computeralgebrasystem *MuPAD* zu implementieren.

Ein Unterschied zu den genannten Arbeiten besteht darin, daß in dem hier vorgelegten Programmpaket ein Bearbeiten von Spinoperatoren bzw. Spinvektoren mit mehreren unabhängigen Spinvariablen möglich geworden ist. So konnten z.B. auch Symmetrien für das Hubbard - Modell [6],[7],[8] gefunden werden. Außerdem wurde hier, im Gegensatz zu den Programmpaketen in MAPLE, das Domains - Konzept [9] von *MuPAD* ausgenutzt. Des weiteren wurde großer Wert darauf gelegt, die Schreibweise der Spinoperatoren und Spinvektoren in *MuPAD* und in der geläufigsten Literatur so ähnlich wie möglich zu halten. Dadurch wurde eine leichte, überschaubare Ein- und Ausgabe und ein unkompliziertes Vergleichen von Resultaten des Programmes mit schon bekannten und veröffentlichten Resultaten ermöglicht.

Theorie

Wir betrachten dynamische Systeme der Form

$$A_t = K(A), A \in M$$

Dabei sei $K(A)$ ein C^∞ Vektorfeld auf einer Mannigfaltigkeit M und die obige Gleichung global lösbar. Ein zweites C^∞ Vektorfeld $G(A)$ auf der Mannigfaltigkeit M heißt dann eine **Symmetrie**, oder besser Lie-Bäcklund-Symmetrie (Kurzform für: **infinitesimaler Generator einer einparametrigen Symmetriegruppe**) von $A_t = K(A)$, wenn gilt

$$[K, G] = 0$$

wobei der Kommutator in der Vektorfeld-Liealgebra zu bilden ist.

D.h. die in unserem Programmpaket zu lösende Aufgabe ist die Bestimmung aller oder zumindest möglichst vieler C^∞ Vektorfelder, die mit dem gegebenen Vektorfeld $K(A)$ kommutieren. Die Menge aller dieser Vektorfelder

$$K^{@} = \{G \in \mathcal{L} | [G, K] = 0\}$$

wird als der Kommutant des Vektorfeldes $K(A)$ bezeichnet.

Bei den Spinketten beschränken wir uns ausschließlich auf spezielle dynamische Systeme, nämlich solchen, die quantenmechanischen Systeme sind. Diese Systeme sind dynamische Systeme auf einer Mannigfaltigkeit $M = \mathcal{L}(\mathcal{H}, \mathcal{H})$, die aus allen selbstadjungierten Operatoren eines Hilbertraumes $\mathcal{H}$ besteht. Die Evolutionsgleichung $A_t = K(A), A \in M$ hat in diesem Fall die spezielle Form

$$\frac{d}{dt} A(t) = i \cdot [H, A(t)], A \in \mathcal{L}(\mathcal{H}, \mathcal{H})$$

D.h. das Vektorfeld $K(A)$ ist hier von der Form

$$K(A) = i \cdot [H, A]$$

Es kann nun gezeigt werden, daß alle Vektorfelder des Typs $G(A) = i \cdot [\hat{H}, A]$ die Eigenschaft $[G(A), K(A)] = 0$ genau dann haben, wenn gilt

$$[H, \hat{H}] = 0$$

D.h. zum Auffinden von Symmetrien genügt das Auffinden von Operatoren $\hat{H}$, die mit H kommutieren. Aus diesen Operatoren können dann die Symmetrien mit der Beziehung $G(A) = i \cdot [\hat{H}, A]$ erzeugt werden. Alle Operatoren $\hat{H}$, die mit H kommutieren, werden (wie bei den Vektorfeldern) zum Kommutanten $H^{@} = \{\hat{H} \in \mathcal{L} | [\hat{H}, H] = 0\}$ von H zusammengefaßt.

Es gibt also **zwei Vorgehensweisen bei der Symmetriesuche:**
1) Es können direkt die Vektorfelder gesucht werden, die mit einem gegebenen Vektorfeld $K(A)$ kommutieren.
2) Es können alle Operatoren gesucht werden, die mit einem gegebenem Hamiltonoperator H kommutieren.
Diese beiden Fälle sind im vorliegenden Programmpaket implementiert und getestet worden.

Im Fall, daß der Kommutant $H^{@}$ abelsch ist, kann die Symmetriesuche mit dem Konzept der Mastersymmetrien [4] vereinfacht werden. Eine **Mastersymmetrie** M für $H \in \mathcal{L}$ ist ein Element einer Lie Algebra $\mathcal{L}$ mit den Eigenschaften

$$M \notin H^{@} \text{ und } [[M, H], H] = 0$$

Damit ist $H_1 = [M, H]$ wieder eine Symmetrie [4]. Wenn nun H_2 als $H_2 = [M, H_1]$ definiert ist, folgt mit der Jacobi - Identität $[H_2, H] = 0$. D.h. auch H_2 ist wieder eine Symmetrie. Insgesamt erhält man also eine Folge

$$H_{n+1} = [M, H_n], n \in N$$

von Lie Algebra Elementen, die mit H kommutieren.

D.h. nach der Bestimmung einer Mastersymmetrie ist es möglich eine ganze Folge von Symmetrien, sowohl im Operator- als auch im Vektorfeldfall zu bestimmen.

Die zur Implementierung der Symmetrie- und Mastersymmetriesuche verwendeten Algorithmen und Verfahren sowie eine genauere Beschreibung der zugrundeliegenden Theorie können ausführlich in [10] nachgelesen werden.

Die Benutzung des Programmpaketes

Für die Bearbeitung von **Operatoren** steht im Programmpaket `SPIN` das Domain `spinop` zur Verfügung. Die Eingabe des Hamiltonoperators des XYh - Modells (anisotroper Heisenbergscher Ferromagnet in externem magnetischen Feld) [1],[4]

$$H0 := \sum_{n \in \mathbb{Z}} J1 \sigma_n^x \sigma_{n+1}^x + \sigma_n^y \sigma_{n+1}^y + h \sigma_n^z$$

erfolgt in *MuPAD* in folgender Weise:

```
>> H0:=spinop(J1*S[n,x]*S[n+1,x]+J2*S[n,y]*S[n+1,y]+h*S[n,z]);

    spinop(h S[n, z] + J1 S[n, x] S[n + 1, x] + J2 S[n, y] S[n + 1, y])
```

Man sieht, daß hier statt dem Buchstaben σ der Buchstabe S verwendet wird. Es ist allerdings möglich, diese Voreinstellung zu verändern.

Die erste Symmetrie kann dann mit der Prozedur `SYMGEN` bestimmt werden:

```
>> H1:=SYMGEN(H0,H0);

spinop(((- 2 I) h J1 - par2) S[n, x] S[n + 1, y] +

   (2 I h J1 + par2) S[n, y] S[n + 1, x] +

   ((- 2 I) J1 J2) S[n, y] S[n + 1, z] S[n + 2, x] +

   (2 I J1 J2) S[n, x] S[n + 1, z] S[n + 2, y])
```

Zur Kontrolle kann dann nachgerechnet werden, ob der Kommutator zwischen H0 und H1 wirklich 0 ist, d.h. ob der von der Prozedur bestimmte Operator wirklich eine Symmetrie ist:

```
>> com(H0,H1);

                             spinop(0)
```

Mit der Prozedur `GHOGEN` kann dann eine erste Mastersymmetrie bestimmt werden:

```
>> M0:=GHOGEN(H0,H1);

      /                        / J1          J1 par7 \
spinop| (h n - par7) S[n, z] + | -- + n J1 - ------- | S[n, x] S[n + 1, x]
      \                        \ 2              h    /

  / J2          J2 par7 \                         \
+ | -- + n J2 - ------- | S[n, y] S[n + 1, y] |
  \ 2              h    /                         /
```

wobei die `parn` freie Parameter der Mastersymmetrie sind. Solche müssen auftreten, denn Mastersymmetrien sind nur modulo von Symmetrien eindeutig. Mit Hilfe dieser Mastersymmetrie ist es nun möglich eine Folge von Symmetrien zu bestimmen. Weitere Ergebnisse können in [10] nachgelesen werden.

Für die Bearbeitung von **Vektorfeldern** steht das Domain `spinvec` zur Verfügung. Das zum XYh - Modell zugehörige Vektorfeld hat die Form:

$$H0 := \begin{pmatrix} -2h\sigma_n^y + 2J2\sigma_n^z\sigma_{n+1}^y + 2J2\sigma_{n-1}^y\sigma_n^z \\ 2h\sigma_n^x - 2J1\sigma_n^z\sigma_{n+1}^x - 2J1\sigma_{n-1}^x\sigma_n^z \\ -2J2\sigma_n^x\sigma_{n+1}^y - 2J2\sigma_{n-1}^y\sigma_n^x + 2J1\sigma_n^y\sigma_{n+1}^x + 2J1\sigma_{n-1}^x\sigma_n^y \end{pmatrix}$$

Im Domain spinvec gibt es zwei Möglichkeiten dieses Vektorfeld zu erzeugen. Zum einen kann dieses Vektorfeld direkt eingegeben werden, zum anderen kann dieses Vektorfeld aber auch aus dem oben angegebenen Hamiltonoperator des XYh - Modells erzeugt werden. Das geschieht mit Hilfe der Prozedur `trans`:

```
>> H0:=trans(H0);

                              spinvec(

(- 2 h) S[n, y] + (2 J2) S[n, z] S[n + 1, y] +

(2 J2) S[n - 1, y] S[n, z]


(2 h) S[n, x] + (- 2 J1) S[n, z] S[n + 1, x] +

(- 2 J1) S[n - 1, x] S[n, z]


(2 J1) S[n, y] S[n + 1, x] + (2 J1) S[n - 1, x] S[n, y] +

(- 2 J2) S[n, x] S[n + 1, y] + (- 2 J2) S[n - 1, y] S[n, x]

                                 )
```

Auch in diesem Fall kann mit der Prozedur `SYMGEN` eine erste Symmetrie berechnet werden:

```
>> H1:=SYMGEN(H0,H0);

                              spinvec(

(4 J1 J2 + par8) S[n - 1, x] S[n, y] S[n + 1, y] +

   (- 4 J1 J2 - par8) S[n - 1, y] S[n, y] S[n + 1, x] +

   (4 J1 J2 + par8) S[n, z] S[n + 1, z] S[n + 2, x] +

   (- 4 J1 J2 - par8) S[n - 2, x] S[n - 1, z] S[n, z]


(4 J1 J2 + par8) S[n - 1, y] S[n, x] S[n + 1, x] +

   (- 4 J1 J2 - par8) S[n - 1, x] S[n, x] S[n + 1, y] +

   (4 J1 J2 + par8) S[n, z] S[n + 1, z] S[n + 2, y] +

   (- 4 J1 J2 - par8) S[n - 2, y] S[n - 1, z] S[n, z]
```

```
(4 J1 J2 + par8) S[n - 2, x] S[n - 1, z] S[n, x] +

   (- 4 J1 J2 - par8) S[n, x] S[n + 1, z] S[n + 2, x] +

   (4 J1 J2 + par8) S[n - 2, y] S[n - 1, z] S[n, y] +

   (- 4 J1 J2 - par8) S[n, y] S[n + 1, z] S[n + 2, y]

                                       )
```

Die Bestimmung einer Matersymmetrie soll wieder mit Hilfe der Prozedur GHOGEN erfolgen. In diesem Fall kann das Programmpaket allerdings keine Mastersymmetrie bestimmen:

```
>> M0:=GHOGEN(H0,H1);

Error: GHO:nonexistent [spinvec::GHOGEN]
```

Das Bestimmen weiterer Symmetrien mit der Prozedur SYMGEN ist aber möglich. Eine genauere Beschreibung der Ergebnisse kann wieder in [10] nachgelesen werden.

Diese Sammlung von Beispielen zur Benutzung des Programmpaketes ist natürlich nicht vollständig. So ist es z.B. möglich die Bezeichnungen für die Spinvariablen, die Gitterpunkte und die räumlichen Koordinaten (hier: x,y,z) zu verändern und bei Vektoren eine für den Algorithmus notwendige Highest Order Projektion auszuwählen. Außerdem können Spinvektoren und -operatoren addiert, mit einem Skalar malgenommen und nichtkommutativ miteinander multipliziert werden. Für diese Operationen sollen hier jedoch keine Beispiele angeführt werden, da das den Rahmen dieses Artikels sprengen würde. Alle diese Operationen sind allerdings auf den Help-Seiten des Programmpaketes und in [10] beschrieben.

Resultate

Für die bekannten Modelle (XYh- und XYZ-Modell [1],[4],[2]) konnten viele Symmetrien und Mastersymmetrien berechnet werden (s.o). Die gefundenen Symmetrien entsprechen den in den Vorgängerarbeiten entdeckten bekannten Symmetrien. Im Fall des Hubbard - Modells [6],[7],[8] war nun auch die Berechnung einiger Symmetrien möglich, die hier allerdings aus Platzgründen nicht ausgeschrieben werden sollen. Mastersymmetrien konnten bisher allerdings nicht gefunden werden.

Ausblick

Das hier vorgestellte Programmpaket wurde unter *MuPAD* 1.3 entwickelt und auch schon unter *MuPAD* 1.4 getestet. Es steht in der jetzt veröffentlichten Version noch nicht zur Verfügung. Auf Grund der bei den Berechnungen auftretenden, sehr großen Gleichungssysteme war die Berechnung höherer Symmetrien bisher sehr zeit- und speicheraufwendig. Mit den im Moment an der U-GH Paderborn durchgeführten Überlegungen und Implementationen zur Verbesserung des Gleichungslösers im Falle linearer Gleichungen, müsste aber eine deutliche Verbesserung des Laufzeitverhaltens möglich werden.

Literatur

[1] Barouch, E. Statistical Mechanics of the XY - Model. In W.E. Brittin, editor, *Mathematical Methods in Theoretical Physics*, pages 1–55, 1972.

[2] Baxter, R.J. One - Dimensional Anisotropic Heisenberg Chain. *Annals of Physics*, 70:323–337, 1972.

[3] Falck, U. Computeralgorithmen zur Behandlung vollstaendig integrabler Spinketten. *Diplomarbeit, UGH-Paderborn*, 1986.

[4] Fuchssteiner, B., Falck, U. Computer Algorithms for the Detection of completely integrable Quantum Spin Chains. *Symmetries and Nonlinear Phenomena*, 1988.

[5] Geddes, K.O., Gonnet, G.H., Char, B.W. *MAPLE Benutzerhandbuch*. Watcom Publications Limited, 1994.

[6] Hubbard, J. *Proc. Royal Society*. A 276:238, 1963.

[7] Hubbard, J. *Proc. Royal Society*. A 277:237, 1964.

[8] Hubbard, J. *Proc. Royal Society*. A 281:241, 1964.

[9] Naundorf, H. Domains. *mathPAD*, 4(1):20, March, 1994.

[10] Nolte, C. Algorithms for symmetries of quantum mechanical spin chains. *Diplomarbeit, UGH-Paderborn*, 1997.

[11] Wittebrok, Gisela. Behandlung quantenmechanischer Spinketten als klassische Fluesse auf Operatormannigfaltigkeiten: Computeralgorithmen zur Bestimmung einparametriger Symmetriegruppen. *Diplomarbeit, UGH-Padeborn*, 1987.

Dynamic Modules meet Windows 95/NT

Andreas Sorgatz
Universität-GH Paderborn, andi@mupad.de

This article gives a brief overview over Dynamic Modules in MuPAD Pro for Windows 95/NT which is available in the current developer version and will be part of a new major MuPAD release.

Introduction

The concept of dynamic modules [1] enables users to extend the MuPAD kernel with user-defined C/C++ routines and to integrate external software packages in an easy, flexible and very efficient way. This technology was introduced in mathPAD Vol.7/No.1 page 16. Applications of dynamic modules were demonstrated in mathPAD Vol.8/No.1 with articles about interacting with the *Singular* system using *MP* (see page 8) and implementing the *Macro Parallelism* in MuPAD 1.4.1 using *PVM* (see page 33) [2].

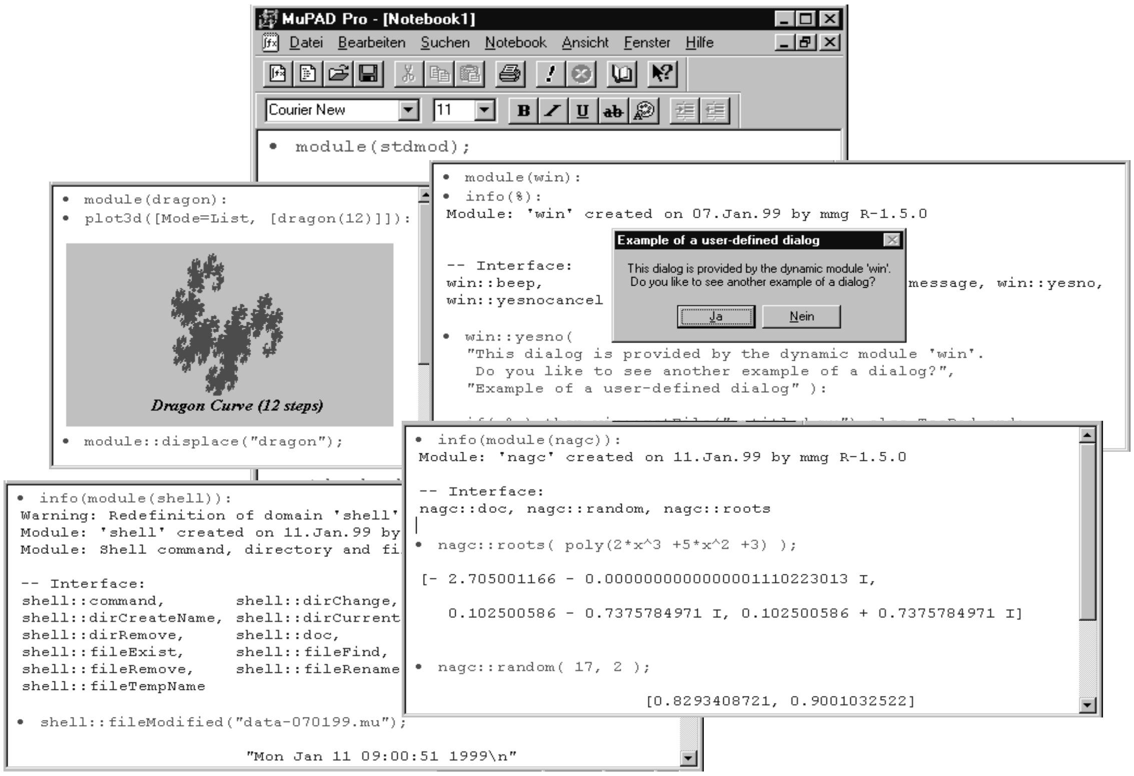


Fig.1: Using dynamic modules in MuPAD Pro for Windows 95/NT

Besides the integration of special purpose tools and fast numeric functions, e.g. using the NAG-C library or user-defined numeric code, dynamic modules also enable users to access all features of the computer hardware as well as of the operating system from within MuPAD, e.g. for creating and using specific Windows 95/NT message boxes and dialogs in MuPAD algorithms as shown in the Figure 1.

Creating and Using Dynamic Modules

For creating dynamic modules, a so-called *module generator* (mmg) is provided. It extends the user C/C++ source code with additional information needed by the MuPAD module manager and compiles and links it to a special kind of a dynamic link library (DLL).

The module generator is designed as a command line tool. On Windows 95/NT it can be used in exactly the same way as on UNIX operating systems. For this, the user opens a MSDOS command shell and calls the MSDOS batch file `mmg.bat` with at least a module source file as argument:

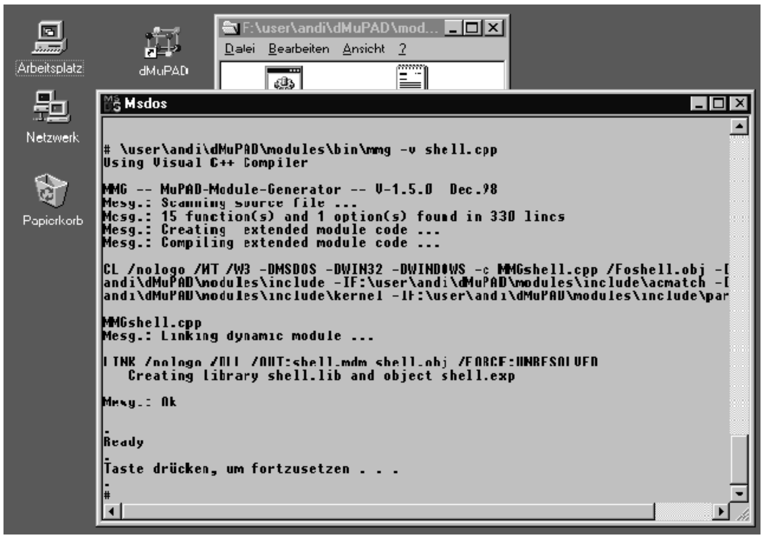


Fig.2: Calling the module generator using the UNIX syntax

However, often it is more convenient to use the drag & drop feature of Windows 95/NT to create module binaries. Here, the module source file is selected with the mouse cursor and dropped on the module generator `mmg.bat` as demonstrated in Figure 3(a). With this, the module generator is started in a new MSDOS command shell window, the module source code is compiled and the executable module file (with the suffix `.mdm`) is placed on the desktop – refer to Figure 3(b).

mathPAD

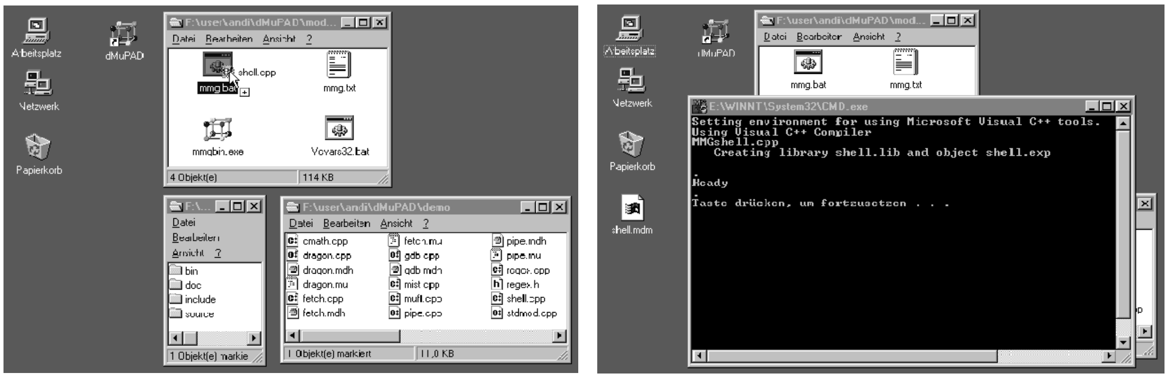


Fig.3: Calling the module generator using drag & drop

Using the drag & drop feature is also supported by so-called *module inline options* which allow the user to insert the mmg command line options into the module C/C++ source file, e.g. the line `MMG( generate="debug" )` instructs the module generator to create a debug version of this module.

After copying a dynamic module into the MuPAD *modules* subdirectory, it can be used in the same way as on UNIX systems, which means: after loading it with the command `module()`, it is used like a MuPAD library.

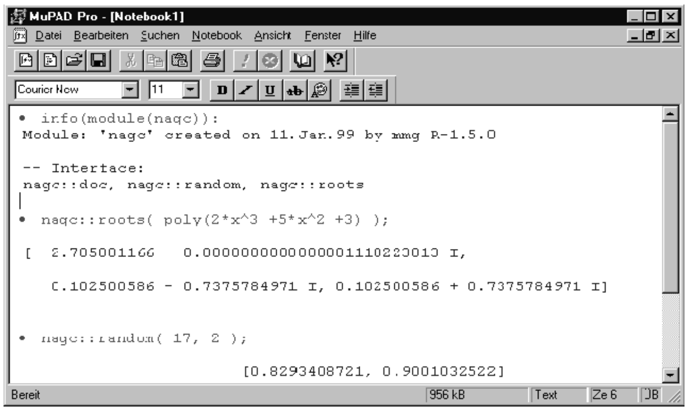


Fig.4: Using NAG-C routines within a dynamic module

Technical Information

Currently, only the *Microsoft Visual C++* Compiler (Version 5.0) is supported for compiling module binaries. This has to be installed and configured before the MuPAD module generator can be used.

Documentation

The book *Dynamic Modules — User's Manual and Programming Guide for MuPAD 1.4* [1] describes the user and programming interface of dynamic modules emphasizing UNIX platforms. However, it is also suitable (and needed) for implementing dynamic module on Windows 95/NT systems as well as for the coming MuPAD releases.

Special extensions and features which are specific to the Windows 95/NT platform will be described in a separate document. This document will also contain all updates, extensions and new features of dynamic modules in the developer version of MuPAD with respect to the dynamic modules Manual 1.4.

For detailed information refer to the MuPAD web site: `http://www.mupad.de/PAPER/MODULES/`

References

[1] A. Sorgatz. *Dynamic Modules — User's Manual and Programming Guide for MuPAD 1.4*. October 1998, Springer Verlag, Berlin · Heidelberg · New York. 234 pages, with a CD-ROM, ISBN 3-540-65043-1. Web: `http://www.mupad.de/~andi/Pub/Intro/sor98.shtml`

[2] T. Metzner, M. Radimersky, A. Sorgatz, S. Wehmeier. *User's Guide to Macro Parallelism in MuPAD 1.4.1*. B.G. Teubner Verlag Stuttgart · Leibzig. With a CD-ROM, ISBN 3-933764-01-7. Web: `http://www.mupad.de/~andi/Pub/Intro/mpb98.shtml`

[3] A Sorgatz. *Dynamische Module – Ein Konzept zur Effizienten Software Integration im General Purpose Computer Algebra System MuPAD*. Mai 1998, Sonderforschungbereich 376, Universität Paderborn, Technical Report tr-rsfb-98-056. Web: `http://www.mupad.de/~andi/Pub/Intro/sfb98.shtml`

Die Fachgruppe Computeralgebra

Johannes Grabmeier
IBM Deutschland ISG, Heidelberg, grabm@de.ibm.com

Die Themen und Aufgaben sowie die Struktur und die Aktivitäten der gemeinsamen Fachgruppe Computeralgebra der DMV (Deutsche Mathematiker-Vereinigung), der GAMM (Gesellschaft für angewandte Mathematik und Mechanik) und der GI (Gesellschaft für Informatik) werden dargestellt.

Aufgaben und Zielsetzungen

Historie

Am 7. November 1987 wurde in Deutschland die Fachgruppe Computeralgebra gegründet. Die Initiative dazu ging von u.a. Albrecht Blaser, Heidelberg, (GI, IBM), Benno Fuchssteiner, Paderborn, (DMV), Reiner Janßen, Heidelberg (IBM), Joachim Neubüser, Aachen, (DMV) aus und wurde unterstützt von Willi Törnig, Darmstadt, (DMV) und Wolfgang Walter, Karlsruhe, (GAMM). Es galt der stürmischen Entwicklung dieses Gebietes im Überlappungsbereich von Informatik, Mathematik und Anwendungsgebieten in den Natur-, Ingenieur- und Wirtschaftswissenschaften eine gemeinsame Plattform zu geben.

Zum ersten Sprecher der Fachgruppenleitung wurde Fritz Schwarz, St. Augustin, gewählt, zum Stellvertreter Joachim Neubüser, Aachen, die die Ämter von 1987 bis 1990 innehatte. 1990-1993 leitete Volker Weispfenning, Passau, die Fachgruppe, sein Stellvertreter war Johannes Grabmeier, Heidelberg, der dann auch in der Zeit 1993-1999 das Sprecheramt ausübte. Sein Stellvertreter war B. Heinrich Matzat, Heidelberg. In der konstituierenden Sitzung am 5. März 1999 in Leipzig wurde H. Michael Moeller, Dortmund, zum neuen Sprecher für die Amtsperiode 1999-2002 gewählt, zum Stellvertreter Michael E. Pohst, Berlin.

Ziele

In der *Ordnung* der Fachgruppe werden die folgenden Aufgaben und Ziele formuliert:

> Die Fachgruppe sieht es als ihre Aufgabe an, Forschung, Lehre und Entwicklung, Anwendungen, Informationsaustausch und Zusammenarbeit auf dem Gebiet der Computeralgebra zu fördern. Die Computeralgebra ist ein Wissenschaftsgebiet, das sich mit Methoden zum Lösen mathematisch formulierter Probleme durch Algorithmen zum symbolischen und algebraischen Rechnen und deren Umsetzung in Soft- und Hardware sowie ihren Anwendungen beschäftigt. Die Computeralgebra beruht auf der exakten endlichen Darstellung endlicher oder unendlicher mathematischer Objekte und Strukturen und ermöglicht deren symbolische und formelmäßige Behandlung durch eine Maschine.

Im Report Computeralgebra von 1993 – siehe weiter unten im Text – wird das Gebiet weiter präzisiert:

Strukturelles mathematisches Wissen wird dabei sowohl beim Entwurf als auch bei der Verifikation und Aufwandsanalyse der betreffenden Algorithmen verwendet. Die Computeralgebra kann damit wirkungsvoll eingesetzt werden bei der Lösung von mathematisch modellierten Fragestellungen in zum Teil sehr verschiedenen Gebieten der Informatik und Mathematik, sowie in den Natur- und Ingenieurwissenschaften.

Computeralgebra – eine Säule des Wissenschaftlichen Rechnens

Das Rechnen mit beliebig langen Zahlen, das Rechnen mit Symbolen, Unbestimmten und Polynomen, Faktorisieren von Zahlen und Polynomen, Ableitungen polynomialen und anderer Funktionen das wesentlich schwierigere Auffinden von Stammfunktionen, das exakte Lösen polynomialer Gleichungssystems oder einer Differentialgleichung sind einige konkrete Beispiele für Problemstellungen in der Computeralgebra.

Diese verschiedenartigen Beispiele haben eines gemeinsam: Im Gegensatz zur numerischen Mathematik mit ihrer Gleitkommaarithmetik und Rundungsfehlerproblematik stehen hier *exakte*, *algebraische* und *symbolische Rechnungen* sowie die symbolische Manipulation von Formeln im Mittelpunkt. Es erscheint erstaunlich, daß nach 50 Jahren des Rechnens auf dem Computer *wissenschaftliches Rechnen* immer noch für weite Kreise fast ausschließlich mit numerischem Rechnen verbunden ist, oder gar gleich gesetzt wird. Eine der Ursachen dafür ist mit Sicherheit, daß in den ersten Jahren und Jahrzehnten die Computertechnik nicht den hohen Speicher- und Prozessoranforderungen des symbolische Rechnens genügte, und sich damit Erwartungen, wie sie schon im Jahre 1842 von ADA AUGUSTA COUNTESS OF LOVELACE[1] über die Analytische Maschine von C. BABBAGE formuliert wurden, zunächst schnell als scheinbar unerfüllbar erwiesen:

> Many persons . . . imagine that the business of the engine is to give results in numerical notations, the nature of its processes must consequently be arithmetical and numerical rather than algebraical and analytical. This is an error. The engine can arrange and combine its numerical quantities exactly as if they were letters or other general symbols; and in fact might bring out its results in algebraical notation *were* provisions made accordingly.

Die Visionen der Countess of Lovelace haben sich mittlerweile erfüllt. Es ist Aufgabe der Fachgruppe die weitere Entwicklung dieses vielseitigen Gebietes als *Säule des wissenschaftlichen Rechnens*[2] zu fördern.

Anwendungen der Computeralgebra

Bei den Anwendungen mathematischer Methoden in Naturwissenschaft und Technik stehen traditionell numerischen Methoden im Vordergrund. Mit den symbolischen Methoden der Computeralgebra haben sich neue Anwendungsgebiete eröffnet, bei denen es auf *exakte* Lösungen ankommt und bei denen strukturmathematische Überlegungen, z.B. zur Beschreibung von Symmetrien eingehen; ferner die Behandlung von Problemen, die von unbestimmten *Parametern* abhängen.

Dazu gehören etwa[3]:

- In der *Physik* und ihren technologischen Anwendungen: Gemischt symbolisch-numerische Berechnungen komplexer Probleme in der Himmelsmechanik, der Hocheenergiephysik (Feynman–Integrale) und der Relativitätstheorie (Differentialgeometrie); Integration und Lösung von Differentialgleichungen in geschlossener Form;

[1] Zitiert nach *Boyle, A.; Caviness, B.F. (Hrsg.)*: Future Directions for Research in Symbolic Computation. Report of a Workshop on Symbolic and Algebraic Computation, 1988, Washington, DC. Philadelphia: SIAM Reports on Issues in the Mathematical Sciences, 1990, p. 10.

[2] Siehe auch *Grabmeier, J.:* Computeralgebra – eine Säule des Wissenschaftlichen Rechnens, it+ti, Informationstechnik und Technische Informatik, 6/95, Seite 5–20, Oldenbourg, 1995.

[3] Nach Weispfenning, V.: Memorandum für das BMFT.

symbolische Berechnungen in den Algebren der Quantenmechanik; Klassifikation höherdimensionaler kristallographischer Gruppen zur Beschreibung von inkommensurabel modulierten Strukturen, Quasikristallen und magnetischen Strukturen.

- In der *Chemie*: Anwendungen der Darstellungstheorie auf die Klassifikation von Graphen chemischer Verbindungen insbesondere von Isomeren; Lösung grosser Gleichungssysteme zur Bestimmung chemischer Reaktionsgleichgewichte bei variablen Reaktionsbedingungen, z.B. bei Verbrennungsprozessen und der Abgasregulierung.
- In der **Sicherheitstechnik**: Algebraische Methoden der Fehlererkennung und -korrektur bei der Nachrichtenübertragung; kryptographische Kodierung von vertraulichen Nachrichten durch Public–Key–Verfahren mit Methoden der Zahlentheorie und der algebraischen Gruppen; Verifikation von Sicherheitsmechanismen (Protokollen).
- In der **Robotik**: Bewegungsplanung und Regulierung autonomer Roboter z.B. in der Raumfahrt; zylindrisch algebraische Zellenzerlegung des $\mathbf{R}^n$; geometrische Bildverarbeitung beim Maschinensehen.
- Im **computergestützten Entwurf (CAD)**: Flexible Inferenzsysteme für die geometrische Modellierung parametrisierter Probleme; Konstruktion von Übergangsflächen.
- In der Kontrolltheorie: Untersuchung der Stabilität und Sicherheit von Kontrollsystemen mit Rückkopplung.
- In der **Genforschung**: Klassifikation von DNA-Strukturen.

Die Methoden der Computeralgebra erlauben in diesen Anwendungsbereichen eine *automatische Behandlung* von Problemen, die sonst nur mühsam mit Ad hoc-Ansätzen angegangen werden konnten. Das große Potential dieser Methoden ist dabei noch lange nicht ausgeschöpft. Kontinuierliche Förderung dieser Anwendungen und der zugrundeliegenden Algorithmen kombiniert mit immer leistungsstärkerer Hardware werden dem Gebiet Computeralgebra weitere Entwicklungsmöglichkeiten geben.

Aufgaben

Die Fachgruppe gibt den Computeralgebra-Rundbrief heraus, der zweimal im Jahr erscheint und an die mehr ca. 600 Mitglieder versandt wird. Die Fachgruppe betreibt im Internet ein elektronisches Informationssystem CAIS. Sie organisiert und unterstützt Workshops, Seminare, Tagungen und andere Aktivitäten auf dem Gebiet der Computeralgebra. Die Koordination der Aktivitäten und die Herausgabe des Rundbriefs liegt in den Händen der *Fachgruppenleitung*.

Struktur und Ordnung

Die Ordnung

Die Ordnung der Fachgruppe regelt die Ziele, die Organisation und Struktur der Fachgruppe. Sie ist im Internet unter der URL

`http://www.gwdg.de/~cais/mitteilungen/mitteilungen.html`

zu finden.

Mitgliedschaft

Die Mitgliedschaft in der Fachgruppe steht Mitgliedern der Trägergesellschaften, aber auch Nichtmitgliedern offen. Die Fachgruppe strebt aber an, daß jedes ihrer Mitglieder auch Mitglied in mindestens einer der Trägergesellschaften ist. Mitgliedschaft wird durch Erklärung des Beitritts an eine der Geschäftsstellen der Trägergesellschaften erworben. Im Anhang haben wir ein Beitrittsformular beigefügt, siehe Seite 77. Die Mitgliedschaft steht auch institutionellen Einrichtungen offen. Zur Zeit[4] beträgt der Jahresbeitrag DM 15, für Mitglieder, die nicht Mitglied einer Trägergesellschaft sind DM 18.

Die Fachgruppenleitung

Die Fachgruppenleitung hat mindestens zwölf Mitglieder. Von jeder beteiligten Trägergesellschaften wird ein Vertreter benannt. Die restlichen neun werden von den Mitgliedern der Fachgruppe gewählt. Die Amtszeit der Fachgruppenleitung beträgt drei Jahre. Die Ordnung sieht weiterhin vor, daß bis zu drei weitere Fachexperten in die Fachgruppenleitung berufen werden können.

Aktivitäten und Schwerpunkte

Der Rundbrief

Die Fachgruppe gibt zweimal jährlich einen Rundbrief (ISSN 0933-5994) heraus, der an alle Mitglieder verschickt wird. Exemplare darüber hinaus können über die GI bezogen werden. Ältere Ausgaben können im CAIS auf der Seite

`http://www.gwdg.de/~cais/CAR`

abgerufen werden. Die folgenden Rubriken bilden das Gerüst des Rundbriefs.

- Mitteilungen der Sprecher
- Hinweise auf Konferenzen
- Berichte von Konferenzen
- Themen und Anwendungen der Computeralgebra
- Netzinformationsdienste zu Mathematik und Computeralgebra
- Neues über Systeme und Hardware
- Berichte über Arbeitsgruppen
- Publikationen über Computeralgebra
- Unterrichtsmaterialien mit Computeralgebra
- Besprechungen zu Büchern der Computeralgebra
- Lehrveranstaltungen über Computeralgebra

[4] 1999

- Kurze Mitteilungen

Jedes Mitglied ist eingeladen Beiträge, Buchbesprechungen und Informationen zum Rundbrief beizusteuern. Redaktionsschluß ist jeweils der 28.02. und der 30.09. eines jeden Jahres.

Die Beiträge sollten idealerweise in TEX gesetzt per Email an den Fachexperten Rundbrief, Dr. Ulrich Schwardmann, Göttingen, gesendet werden:

`uschwar1@gwdg.de.`

Die Anschrift lautet: Dr. Ulrich Schwardmann, Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen (GWDG), Am Fassberg, 37077 Göttingen, Telefax: 0551-21119.

Es ist möglich auf den Umschlagseiten Anzeigen zu schalten. Derzeit[5] kostet die Außenseite DM 1000, die Umschlaginnenseite DM 800. Die aktuelle Preisliste dafür sowie weitere Informationen können bei der Gesellschaft für Informatik (GI), Wissenschaftszentrum, Ahrstr. 45, 53175 Bonn, Telefon 0228-302-164, Telefax 0228-302-167,

`gs@gi-ev.de`

angefordert werden. Die GI nimmt auch die Anzeigenaufträge entgegen.

CAIS

Im Internet ist die Fachgruppe mit dem elektronischen Informationssystem CAIS (Computeralgebra-Informationsystem) vertreten. Die Seiten sind unter der URL

`http://www.gwdg.de/~cais`

zu finden. Konferenzankündigungen, Mitteilungen und einzurichtende Links können an

`cais@rz.uni-karlsruhe.de`

gesandt werden. Gerhard Schneider ist als Referent für das CAIS verantwortlich.

Die Mailingliste der Fachgruppe

Im Rahmen von CAIS ist auch eine Mailinglist/Verteilerliste

`cais-l@rz.uni-karlsuhe.de`

eingerichtet. Die Subskription geschieht durch Senden einer Note mit Text

`SUBSCRIBE CAIS-L <Vorname> <Nachname>`

[5] 1999

an

`listserv@rz-uni-karlsruhe.de.`

Beispielsweise werden die Eingänge von Büchern zur Computeralgebra, die von den Verlagen der Fachgruppe zur Besprechung im Rundbrief zur Verfügung gestellt werden, über diesen Verteiler bekannt gemacht.

Der Report Computeralgebra

Die Fachgruppe hat in den Jahren 1991-1993 einen **Report** zur Computeralgebra zusammengestellt. Auf 320 Seiten haben dazu 99 Autoren mit Grundsätzlichem, mit Übersichtsartikeln, Anwendungsbeschreibungen, Berichten zu den Systemen und sonstigen Informationen beigetragen. Damit war zum ersten Mal ein umfassender Überblick über dieses Gebiet gegeben. Zur Zeit wird von den Herausgebern Johannes Grabmeier, Heidelberg, Erich Kaltofen, Raleigh (North Carolina), und Volker Weispfenning, Passau, dieser Report ins englische übersetzt, überarbeitet, erweitert und ergänzt und wird demnächst als *Handbook of Computer Algebra* siehe

`http://www.springer.de/math/compalg.htm.`

Aktivitäten der Jahre 1996-1999

Die Fachgruppenleitung der Amtsperiode 1996-1999 konstituierte sich auf ihrer Sitzung am 1. März 1996 in Darmstadt. Zu den von den Mitgliedern in die Fachgruppenleitung gewählten Vertretern Johannes Grabmeier, Heidelberg, Adalberg Kerber, Bayreuth, Wolfram Koepf, Leipzig, Wolfgang Küchlin, Tübingen, B. Heinrich Matzat, Heidelberg, H. Michael Möller, Dortmund, Michael Pohst, Berlin, Volker Weispfenning, Passau, und Horst Günter Zimmer, Saarbrücken, wurden von den Gesellschaften die Vertretern Gerhard Schneider, Göttingen, (GI), Benno Fuchssteiner, Padernborn (DMV) und Karl G. Roesner, Darmstadt (GAMM), entsandt. Als Fachexperten wurden Friedrich Hehl, Köln (Physik), Ulrich Schwardmann, Göttingen (Rundbrief), und Karl Hantzschmann, Rostock (ISSAC'98) gewählt.

Nach dem Ausscheiden von Benno Fuchssteiner 1997 wurde B. H. Matzat als neuer Vertreter von der DMV benannt. Als neuer Fachexperte Fachhochschulen wurde 1997 Wilhelm Werner aus Künzelsau berufen.

Für die weitere Schwerpunktthemen Chemieanwendungen, Lehre & Didaktik und CAIS waren die Referenten A. Kerber, W. Koepf und W. Küchlin sowie G. Schneider verantwortlich.

Die halbjährlichen Sitzungen der Fachgruppe fanden in Darmstadt (Frühjahr 1996), Jena (Herbst 1996), Dortmund (Frühjahr 1997), Heidelberg (Herbst 1997), Göttingen (Frühjahr 1998) und Künzelsau (Herbst 1998) statt.

Konferenzen

Die Fachgruppe Computeralgebra hat neben der Konferenz ISSAC'1998 und Thurnau'1998 – siehe weiter unten im Text – eine Reihe von Konferenzen und Sektion zur Computeralgebra initiiert. Fast regelmäßig gibt es eine Sektion Computeralgebra bei den Jahrestagungen der DMV Jena 1996 (Leitung G. Hiß, Heidelberg bzw. Aachen, und M. Pohst, Berlin), Mainz 1999 (Leitung G. Pfister, Kaiserslautern, und W. Plesken, Aachen) und der GAMM Prag 1996, Regensburg 1997 (Leitung jeweils S. M. Rump, Hamburg) und Bremen 1998 (Leitung J. Grabmeier und K. Roesner). Bei der GI-Jahrestagung 1996 in Klagenfurt wurde zum ersten Mal ein Hauptreferat dem Thema

Computeralgebra gewidmet (A.M. Odlyzko) sowie ein Workshop Computeralgebra unter der Leitung von F. Winkler, Linz und J. Grabmeier durchgeführt. Im Jahr 1996 fand ein Seminar zum Thema Computeralgebra-Software (Leitung J. Buchmann, Darmstadt) in Dagstuhl statt. Beim International Congress of Mathematicans (ICM'98) in Berlin wurde zum ersten Mal eine *Special Session on Mathematical Software* mit sehr großem Anteil Computeralgebra eingerichtet, die Leitung des Programmkomitees lag in den Händen von J. Grabmeier. Weiter hat die Fachgruppe die von W. Werner geleitete Konferenz *Computeralgebra in Forschung und Lehre*, Künzelsau 1998, die vor allem den Fachhochschulbereich angesprochen hat, unterstützt. Eine weitere Konferenz ist im März 2000 in Konstanz geplant, Leitung Elke-Dagmar Heinrich.

ISSAC'1998 Rostock

Bei einer der weltweit wichtigsten Konferenzreihen zum Thema Computeralgebra ist die Fachgruppe ganz wesentlich vertreten. Die Verantwortung für das das jährlich stattfindete *International Symposium on Symbolic and Algebraic Computations* (ISSAC) wird von einem Steering Comittee getragen. Die Fachgruppe ist seit 1995 institutionelles Mitglied und war von 1995-1998 durch J. Grabmeier, 1998-1999 durch B.H. Matzat und nun durch N.N. vertreten.

Nach Bonn 1991 fand die ISSAC-Konferenz zum zweiten Mal 1998 in Deutschland statt. Die Konferenz ISSAC'1998 in Rostock wurde von der Fachgruppe initiiert und mit der Unterstützung der GI organisiert. Die lokalen Organisation stand unter der Leitung von Karl Hantzschmann aus Rostock. General Chair war Volker Weispfenning aus Passau. Die vorbildlich Organisation und fachliche Leitung machten dieses Symposium nicht nur zu einem Höhepunkt für die etwa 130 Teilnehmer aus der ganzen Welt, sondern auch zu einem Höhepunkt in der Amtszeit der Fachgruppenleitung. Nicht zuletzt durch die hervorragende Arbeit des Schatzmeisters der ISSAC'1998, A. Widiger, Rostock, wurde die Konferenz auch zum finanziellen Erfolg der Fachgruppe.

Computeralgebra im Unterricht

Der weiten Verbreitung von Computeralgebra-Systemen und der damit verbundenen größten Anwendergruppe, die Lehrenden und die Lernenden wurde durch folgende Intitiative Rechnung getragen. Auf ihrer Sitzung im Herbst 1997 hat die Fachgruppenleitung den Dialog zum Thema *Computeralgebra im Unterricht* mit den Fachgesellschaften GDM (Gesellschaft für Didaktik der Mathematik) und MNU (Verein zur Förderung des mathematischen und naturwissenschaftlichen Unterrichts) sowie zur Fachgruppe Didaktik der Mathematik der DMV begonnen. Prof. Dr. Weigand aus Gießen von der GDM und G. Schmidt aus Mainz vom MNU referierten über die Erfahrungen und Aktivitäten ihrer Gesellschaften.

Die Fachgruppe hatte zu diesem Thema im Sommer 1997 die Ministerien der 16 Bundesländer angeschrieben und angeboten, zum Thema *Curriculumsentwicklung Computeralgebra* mit ihnen in die Diskussion zu treten. Die Resonanz bislang dazu war sehr positiv. Im April 1998 wurde deshalb ein Symposium zu Thema *Computeralgebra in Lehre, Ausbildung und Weiterbildung* auf Schloß Thurnau bei Bayreuth veranstaltet. Die beteiligten Gesellschaften haben dazu Vertreter der Ministerien und Schulbehörden eingeladen. Die organisatorische Verantwortung lag bei A. Kerber aus Bayreuth.

Die Fachgruppe wird sich weiterhin mit diesem wichtigen Thema beschäftigten. Im Frühjahr 2000 soll wieder ein solcher Symposium veranstaltet werden. W. Koepf aus Leipzig hat weitere Informationen zu diesem Thema im Internet gesammelt, beispielsweise eine Adressensammlung der verantwortlichen Ministerien und Institutionen, Hinweise zu interessante Projekte und ihren URL-Adressen, Übersicht über die Verbreitung von Computeralgebra an Schulen in Deutschland und Europa sowie über Schulversuche. Diese Seite kann über die Internetseite der Fachgruppe

`http://www.gwdg.de/~cais/ca-brd/ca-brd.html` angesteuert werden und hat die Adresse

`http://www.imn.htwk-leipzig.de/~koepf/ca.html.`

Die Fachgruppe hat vor, weitere Links einzurichten, insbesondere solche, die Unterrichtsmaterialien enthalten, bei denen Computeralgebra eingesetzt wird. Die Erstellung dieser Links soll allerdings einem Referierungsprozeß unterworfen werden, um die Qualität der Materialien sicherzustellen. Parallel hierzu soll im Rundbrief eine Rubrik Unterrichtsmaterialien mit Computeralgebra etabliert werden, in der regelmäßig interessante Themen vorgestellt werden.

Die Benchmark–Initiative

Die Initiative der Fachgruppe zum sehr weit gefaßten Thema *Benchmarks* in der Computeralgebra hat mit der Einberufung eines Treffens unter der Leitung von Heinz Kredel aus Mannheim am Rande der ISSAC-Konferenz in Rostock einen wichtigen Schritt unternommen. Sein Referat und eine strukturierte Zusammenstellung der bislang bekannten Aktivitäten finden Sie im CAIS unter

`http://www.gwdg.de/~cais/systeme/systeme.html`

bzw.

`http://krum.rz.uni-mannheim.de/cafgbench.html.`

Die neue Fachgruppenleitung 1999–2002

Neuwahl der Fachgruppenleitung

Die Wahlleitung für die Neuwahl der Fachgruppenleitung durch die Mitglieder im Herbst 1998 hatten Karl Hantzschmann und B. Heinrich Matzat übernommen. Die Neuwahl an der sich 141 Mitglieder beteiligten brachte folgendes Ergebnis.

Dr. Johannes Grabmeier	Heidelberg	99
Prof. Dr. Volker Weispfenning	Passau	94
Prof. Dr. G.-M. Greuel	Kaiserslautern	66
Prof. Dr. H. Michael Moeller	Dortmund	57
Prof. Dr. Adalbert Kerber	Bayreuth	54
Prof. Dr. Michael E. Pohst	Berlin	52
Prof. Dr. Gerhard Schneider	Göttingen	49
Prof. Dr. Wolfram Koepf	Leipzig	46
Prof. Dr.-Ing. Dr. rer.nat. Volker Strehl	Erlangen-Nürnberg	46
Dr. Joachim Apel	Leipzig	44
Heiko Knechtel	Stadthagen	43
Dr. H.-W. Henn	Karlsruhe	42
Prof. Dr. Wilhelm Werner	Heilbronn	42
Dr. Hans-Gert Gräbe	Leipzig	36
Prof. Dr. Gerhard Hiß	Aachen	35
Prof. Dr. Gunter Malle	Kassel	33

Damit sind die ersten neun Kandidaten gewählt. Die weiteren sieben Kandidaten sind in der Reihenfolge der Stimmenverteilung auf der Nachrückliste. Die Vertreter der Trägergesellschaften waren bereits vorab delegiert worden: von der DMV Prof. Dr. B. Heinrich Matzat, Heidelberg von der GAMM Prof.Dr. S. M. Rump, Hamburg–Harburg, und von der GI Prof. Dr. Karl Hantzschmann, Rostock.

Die konstituierende Sitzung der neuen Fachgruppenleitung fand am Freitag, dem 5. März 1999 in Leipzig statt. Zum neuen Sprecher wurde B. Michael Moeller, Dortmund, zum stellvertretenden Sprecher wurde Michael E. Pohst, Berlin, gewählt.

Für wichtige Fachthemen wurden Fachreferenten benannt und Fachexperten berufen. Als Fachreferenten wurden wieder berufen A. Kerber für Chemieanwendungen, G. Schneider für CAIS, W. Koepf für Lehre & Didaktik sowie neu V. Strehl für Computational Engineering. Auch das Thema Benchmark soll demnächst besetzt werden. Als Fachexperten wurden W. Werner für Fachhochschulen und F. Hehl für Physik wieder in die Fachgruppenleitung gewählt. Dazu kommt neu H. Knechtel, Stadthagen, für das Thema Schulen. U. Schwardmann wird weiter als verantwortlicher Redakteur den Rundbrief betreuen.

Aufnahmeantrag für Mitgliedschaft in der Fachgruppe Computeralgebra

Der auf der Seite 77 abgedruckte Antrag auf Mitgliedschaft in der Fachgruppe Computeralgebra kann ausgefüllt an eine der drei Trägergesellschaften geschickt werden:

Gesellschaft für Informatik e.V. (GI) Wissenschaftszentrum		Deutsche Mathematiker-Vereinigung e.V. (DMV)		Gesellschaft für Angewandte Mathematik und Mechanik e.V. (GAMM)
Ahrstraße 45 53175 Bonn	oder	Mohrenstraße 39 10117 Berlin	oder	NWF I – Mathematik, Univ. Regensburg Universitätsstr. 31 96053 Regensburg
Tel.: 0228-302-149 Fax.: 0228-302-167 e-mail: gs@gi-ev.de		Tel.: 030-20377-306 Fax.: 030-20377-307 e-mail: dmv@iaas-berlin.d400.de		

(Im folgenden jeweils Zutreffendes bitte im entsprechenden Feld [ ] ankreuzen bzw. ____________ ausfüllen.)

Name: ______________________ Vorname: ______________

Akademischer Grad/Titel: ______________________

Privatadresse

Straße/Postfach: ______________________

PLZ/Ort: ______________________ Telefon: ______________

e-mail: ______________________ Telefax: ______________

Dienstanschrift

Firma/Institution: ______________________

Straße/Postfach: ______________________

PLZ/Ort: ______________________ Telefon: ______________

e-mail: ______________________ Telefax: ______________

Gewünschte Postanschrift: [ ] Privatadresse [ ] Dienstanschrift

1. Hiermit beantrage ich zum 1. Januar 1999/200__ die Aufnahme als Mitglied in die Fachgruppe

Computeralgebra (CA) (bei der GI: 2.2.1).

2. Der Jahresbeitrag beträgt DM 15,00 bzw. DM 18,00. Ich ordne mich folgender Beitragsklasse zu:
   [ ] **15,00 DM**. für Mitglieder einer der drei Trägergesellschaften
   - [ ] GI Mitgliedsnummer: ______________
   - [ ] DMV Mitgliedsnummer: ______________
   - [ ] GAMM Mitgliedsnummer: ______________

   Der Beitrag zur Fachgruppe Computeralgebra wird mit der Beitragsrechnung der Trägergesellschaft in Rechnung gestellt. (Bei Mitgliedschaft bei mehreren Trägergesellschaften wird dies von derjenigen durchgeführt, zu der Sie diesen Antrag schicken.) [ ] Ich habe dafür bereits eine Einzugsvollmacht erteilt. Diese wird hiermit für den Beitrag für die Fachgruppe Computeralgebra erweitert.

   [ ] **15,00 DM**. Ich bin aber noch nicht Mitglied einer der drei Trägergesellschaften. Deshalb beantrage ich gleichzeitig die Mitgliedschaft in der

   [ ] GI [ ] DMV [ ] GAMM.

   und bitte um Übersendung der entsprechenden Unterlagen.

   [ ] **18,00 DM** für Nichtmitglieder der drei Trägergesellschaften. [ ] Gleichzeitig bitte ich um Zusendung von Informationen über die Mitgliedschaft in folgenden Gesellschaften:

   [ ] GI [ ] DMV [ ] GAMM.

3. Die in dieses Formular eingetragenen Angaben werden elektronisch gespeichert. Ich bin damit einverstanden, daß meine Postanschrift durch die Trägergesellschaften oder durch Dritte nach Weitergabe durch eine Trägergesellschaft wie folgt genutzt werden kann (ist nichts angekreuzt wird c. angenommen).
   [ ] a. Zusendungen aller Art mit Bezug zur Informatik, Mathematik bzw. Mechanik.
   [ ] b. Zusendungen durch wissenschaftliche Institutionen mit Bezug zur Informatik, Mathematik bzw. Mechanik.
   [ ] c. Nur Zusendungen interner Art von GI, DMV bzw. GAMM.

Ort, Datum: ______________________ Unterschrift: ______________________

Kalender

CeBIT '99

Hannover, Germany, March 18 –24, 1999

Information:
`www.messe.de/cb99/index_e.html`

ECCAD '99

East Coast Computer Algebra Day, Raleigh, North Carolina, USA, April 24, 1999

Information:
`http://vega.math.ncsu.edu/`

3. Pfingsttagung in Münster (mit MuPAD Workshop)

Münster, Germany, May 25 –28, 1999

Information:
`http://www.uni-muenster.de/Lehrerausbildung/ti92tag.html`

CASC 99

Workshop on Computer Algebra in Scientific Computing, München, Germany, May 31 –June 4, 1999

Information:
`http://wwwmayr.informatik.tu-muenchen.de/konferenzen/CASC99/`

IMACS ACA'99

Conference on Applications of Computer Algebra, El Escorial, Spain, June 24 –27, 1999

Information:
`http://math.unm.edu/ACA/1999.html`

Curves & Surfaces

Fourth international conference organized by A.F.A., Saint-Malo, France, July 1 –7, 1999

Deadline for paper submissions:
February 28, 1999

Information:
`http://emma.enst.fr:8081/ afa/saint-malo/`

ICIAM 99

International Congress on Industrial and Applied Mathematics, Edingburgh, Scottland, July 5 –9, 1999

Information:
`http://www.ma.hw.ac.uk/iciam99/`

FoCM'99

Conference on Foundations of Computational Mathematics, Oxford, UK, July 18-28, 1999

Information:
`http://www.damtp.cam.ac.uk/user/na/FoCM/Oxford.html`

ISSAC'99

International Symposium on Symbolic and Algebraic Computation, Vancouver, Canada, July 28 –31, 1999

Information:
`http://www.cecm.sfu.ca/ISSAC99/`

ICTMT4

International Conference on Technology in Mathematics Teaching, Plymouth, UK, August 9 –13, 1999

Information:
`http://www.tech.plym.ac.uk/maths/CTMHOME/ictmt4.html`

Euro-Par '99

Workshop on Parallel Symbolic Computation, Toulouse, France, August 31 –September 3, 1999

Information:
`http://www.enseeiht.fr/europar99/`

13th AAECC Symposium

On Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes, University of Hawaii, USA, November 14 –19, 1999

Information:
`http://www.irit.fr/ACTIVITES/AAECC/aaecc13.htm`

Surfin' the Web

Interessante, nützliche und kuriose Plätze im „global village".

Die Tips in dieser Ausgabe stammen von Andreas Sorgatz (`andi@mupad.de`).

Computer Algebra

`http://www-sop.inria.fr/saga/conf.html`
List of conferences, connected to symbolic computations.

Forschung

`http://elfi.rz.ruhr-uni-bochum.de/elfi/`
ELFI – Servicestelle für elektronische Forschungsförderinformationen.

In Eigener Sache

`http://www.sciface.com`
Mit Information zum kommerziellen Vertrieb und Support von *MuPAD*.

`http://www.sciface.com/news.shtml`
Letzte Meldungen zum Thema *MuPAD*.

`http://www.mupad.de/EDU/`
MuPAD in Education; Von Mailinglist bis Workshop.

`http://www.mupad.de/EDU/gallery.shtml`
MuPAD Gallery; MuPAD Notebooks für den Mathematikunterricht – im Aufbau.

Verlage, Bücher und Zeitschriften

`http://www.spektrum.de/`
Spektrum der Wissenschaft; Interessantes vom Spektrum Ticker über Aktuelles Heft bis Archive.

`http://www.heise.de/`
Heise Verlag; Neues zum Thema Computer sowie Online-Artikel der Zeitschriften *iX* und **c't**

`http://www.linux-magazin.de/`
Linux Magazin; Mit aktuellen Informationen zum Thema Linux und vielen Online-Artikeln.

`http://www.lob.de/`
Lehmanns Online Bookshop; Mit ausführlichen Informationen zu vielen Büchern.

`http://www.buchhandel.de/`
Buch+Medien Online; Verweise auf Datenbanken und Verlage mit eigenen Online Angeboten.

Reisen

http://www.travel-overland.de/it.html
Reisen Online; Von Flugplan bis Hotelreservierung weltweit.

http://www.lycos.de/webguides/reise/m_airlines.html
Verweise auf Fluggesellschaften weltweit.

Allerlei

http://www.adfc.de/
ADFC Allgemeiner Deutscher Fahrrad Club e.V.

http://www.met.fu-berlin.de/deutsch/Wetter/
Aktuelle Wetterinformationen des Institutes für Meteorologie der Freien Universität Berlin

http://www.billiger-telefonieren.de/
Wer ist der billigste Telefonanbieter im Land? Ein interaktives Auskunftssystem.

http://www.ids-mannheim.de/reform/
Informationen zur Neuregelung Rechtschreibung.

http://www.europa.eu.int/euro/html/entry.html
In aller Munde: Der Euro – eine Währung für Europa; Aktuelles und Wissenswertes.

http://kochbuch.unix-ag.uni-kl.de/
Die Rezeptsammlung der UNIX-AG Kaiserlautern; Gaumenfreuden tausenfach.