mathPAD

Vol 8 No 1, March 1998

In diesem Heft

Redaktion:
Benno Fuchssteiner
Frank Postel
Gerald Rüscher
Universität-GH Paderborn
D-33095 Paderborn
e-mail: {benno,frankp,jerry}@mupad.de

ISSN 0941-9187
V.i.S.d.P.: Benno Fuchssteiner

Liebe Leserin, lieber Leser,

rechtzeitig zur alljährlichen CeBIT erscheint die neue mathPAD beim Zeitschriftenhändler Ihres Vertrauens. Wie immer sind die Artikel thematisch breit gestreut, wobei jedoch ein kleiner Schwerpunkt dieser Ausgabe auf dem Thema „Multimedia" liegt. Multimedia hat sich in letzter Zeit zu einem Begriff entwickelt, ohne den kaum ein Softwarehaus mehr auskommt, so daß sich auch die Autoren der mathPAD diesem Trend nicht entziehen wollen. Wir hoffen aufzeigen zu können, daß Multimedia - im Kontrast zur inflationären Verwendung dieses Begriffs - keinesfalls eine leere Worthülse sein muß, sondern ganz konkrete und mit Inhalt gefüllte Anwendung findet.

Ergänzt wird das ganze durch aktuelle Berichte rund um das Thema „Scientific Computing". Besonders hervorheben möchten wir das neue, außerordentlich leistungsfähige und effiziente Numerikpaket im *MuPAD*, das unter der Leitung von Dr. Walter Oevel entwickelt wurde, und die Arbeiten unserer französischen Partner, allen voran Dr. Paul Zimmermann, dem das MuPAD Projekt viel zu verdanken hat.

Wie immer möchten wir uns natürlich bei unseren fleißigen Autoren bedanken, die trotz ihres dichtgedrängten Terminplanes und der nahenden Messe noch Zeit fanden, die Artikel für die vorliegende Ausgabe zu schreiben.

F.P. & G.R.

About the Future of *MuPAD*

Benno Fuchssteiner
Universität GH Paderborn, benno@mupad.de

Recently concerns were voiced on the internet as well as by e-mails to the MuPAD group concerning the commercialization of MuPAD. We therefore asked Benno Fuchssteiner, the head of the MuPAD project, to clarify the intention behind that decision and to point out some of the future perspectives of MuPAD. Below you will find the answer we received from him.

Dear MuPAD friend!

Recently one of our users wrote into the MuPAD visitors book:

> *I am also crushed by your decision to make the windows95/NT version commercial. Now yet another promising package will be priced out of reach of the independent researcher.*

Although I do not agree with the conclusion of that statement, I can say that I am as disappointed as he is. Indeed, a part of *MuPAD* goes commercial. As the one who is mainly responsible for that decision I regard this as a serious personal defeat of the many ideas I had when we started the project. I still believe that scientific software encapsulating the basic methods of a science developed by so many generations of mankind should be free. In the following I want to explain briefly what lead to that decision and I also would like to give you a perspective of what will survive of our ideas.

The *MuPAD* group was always suffering from a severe lack of funding. However, this diet with respect to availability of resources not only had disadvantages, but it also strengthened our survival spirit. To give you an idea of what the situation was: until last year the extra amount my university was contributing to the work of the *MuPAD* project was far less than what the same university was spending for the Maple license. This despite the fact that almost 90 percent of the university's own output on the internet came from our small group which had the same standard size as all other groups in mathematics at that institution.

Recently my university decided to contribute more to our research and development, but this only for a limited time and under the condition that *MuPAD* will be commercialized. The same story (with some variations) holds true for other funding sources: they all pushed us towards commercialization, scolded us for concentrating too much on unscientific issues like interfaces (which is not justified at all because a lot of research was done). Another line of argument was that *MuPAD* is such a great success that the project should be ended, since no further development is necessary and, therefore, no more development funds are needed.

Sometimes I regard it a wonder that under such conditions *MuPAD* could prosper at all and reach such a mature state making it a nonnegligible market force. Our success is only due to the determination of my students, my collaborators and many outside developers. The lesson I have learned from our experience is that in academia there is no understanding for you if you not only strive to do scientific research but also insist to cast that research in a form that can be easily used by others. Yet, in contrast to many of my colleagues, I do not consider it to be a mistake to provide a user friendly form of scientific output.

I do believe it worthwhile to spend efforts on interfaces, documentation and internet presence - I would do it again.

Of course, I am not totally a dreamer, so I always knew that some day we would have to commercialize something. But I was sure that, if successful, we could keep the basic tool (*MuPAD* itself) free for an indefinite period of time and that we would be able to build up a commercial structure on peripheral products. I was hoping that such a commercial structure would not only support the continuing development of the system but also bring additional funds for the university which I participated in building up over the last 25 years. But, alas, we were not given the time for that. So last year the decision was due either to make the source code of *MuPAD* freely available on the internet and finish all development in Paderborn or to aim at a commercial structure according to the laws of a free market society. The reasons why I decided for the latter were:

1. I believe such a complex piece of software as the kernel of a computer algebra system has to be developed under efficient management by a dedicated group of people. I was afraid that a continuing development was not possible under the free organizational structures of the internet. And certainly, a system whose kernel is not under constant development is dead, surviving only a couple of years. I did not think that this should be the fate of MuPAD.

2. I also did care for the future of my collaborators, a group to which I and our users owe a lot. We have in Paderborn a group of dedicated people with unique know-how and I could not see what would be gained if this group had to be dispersed. In fact, we already suffered from a severe brain drain due to the uncertain job perspectives I could offer.

3. So I and my collaborators decided for commercialization provided private investors could be found who share our vision. Such investors have been found. Nevertheless I am still in total and unrestricted control of decisions for the future development and market policy. I believe that this a very fortunate situation which eventually can be used for our users as well as for the scientific community.

Believe me that this decision was not easy for me, since, alas, I am not of the age anymore to be attracted by adventures. However, I am rather confident about the future since I am sure we can build up a strong and viable company which will always be committed to the scientific community and will support the interest of that community. To show you that these are not empty words: among the first projects the newly founded company started is the development of the interface for “*MuPAD* light”, a product which will be freely distributed to the scientific community. It will feature all the mathematics of the *MuPAD* system, so you should not mix this up with the restricted versions you know from other systems.

At present our ideas are: there will be two versions of MuPAD, “*MuPAD* pro” and “*MuPAD* light” which only differ in their interfaces but have the same mathematical capabilities. Since commercialization mainly concerns the interfaces, the name of the new company is SciFace (standing for Scientific Interfaces). The CEO of that company is Dr. Kluge, one of the leading *MuPAD* developers. SciFace will also play a leading role in the further development of libraries and the kernel. SciFace hopefully will provide a safe haven for the accumulated know-how of the *MuPAD* group. As soon as resources permit SciFace will start the development of the next-generation-kernel (which, I believe, will open new horizons for doing mathematics on the computer).

The professional version of *MuPAD* will be licensed for a competitive price. Since we will accelerate interface development, we hope that most of our users will find it worthwhile to contribute to the *MuPAD* development by using that version. Those who cannot afford to do so will come to love “*MuPAD* light” as a powerful tool.

Of course, all these words cannot hide the sad fact that SciFace is a commercial company following the laws of the market. I wish good luck for the new company and I hope you do so as well. I sincerely hope that in the future SciFace will not forget its origins and will always feel indebted to the scientific community.

Yours sincerely,

Benno Fuchssteiner

Das Interaktive Buch

Gerald Rüscher
Universität-GH Paderborn, jerry@mupad.de

Der folgende Vortrag wurde auf der IuK-Tagung „Multimedia in den Wissenschaften“ in Würzburg im März '97 gehalten. Er behandelt die Frage, wie man multimediale Dokumente in den Wissenschaften inhaltlich anspruchsvoll gestalten kann. Für die mathPAD wurden einige Passagen gekürzt und leicht modifiziert.

Meine Damen und Herren,

Der Titel meines Vortrags lautet „Das interaktive Buch in Forschung und Lehre”. Der Hamburger Autor Peter Glaser schreibt in einem Artikel im „SPIEGEL special” vom März 97 zum Begriff der Interaktivität:

> „Jede Überlegung zu interaktiven Medien muß genau genommen mit dem Satz enden:
> Es gibt keine Interaktion mit Maschinen”

... und dann weiter ...

> „Es ist nicht interaktiv, ein Feld auf einem Bildschirm anzuklicken. Es gibt keine wirklichen Überraschungen und neuen Entdeckungen”

Alle Auswahlmöglichkeiten von Computerprogrammen, so Glaser, seien unveränderlich festgesetzt. Trotz dieses vernichtenden Urteils eines Fachmanns, möchte ich aber trotzdem am Titel meines Vortrags festhalten und die Aussage von Herrn Glaser vielleicht später ein wenig relativieren.

Beginnen möchte ich aber mit Überlegungen zum Begriff Multimedia — denn unter diesem Motto steht ja diese Tagung. Gibt es eine gängige Definition von Multimedia? Nun, da stellen wir uns mal ganz dumm.

Zunächst: Multimedia hat irgendwas mit Computern zu tun.

Da gibt es Texte, natürlich nicht zu lang und nicht zu kompliziert, am besten in kleine handliche Häppchen zerlegt, damit der geneigte Leser nicht überfordert wird, denn tatsächlich erfordert das Lesen von Texten am Bildschirm ganz andere Qualitäten als auf weißem Papier. Diese Texte sind — je nach Fleiß der Autoren — von reichlich Hyperlinks aller Art durchzogen, die Querverweise en masse bieten. Kritiker behaupten, daß die Mehrzahl der Web-Nutzer kaum noch die wirklichen Texte liest, sondern in erste Linie nur mit den Augen über den Text huscht und nach den wohlbekannten, meist blau-unterstrichenen Links zu suchen. Diese verführen den Leser dazu, einem zappenden Fernsehzuschauer gleich, von Seite zu Seite durch den Text zu hüpfen.

Von Zeit zu Zeit wird der Leser anhalten, um an den Rosen zu riechen, sprich eines der möglichst vielen bunten Bildern zu betrachten, die das Auge von Text wegziehen und Sachverhalte anschaulich visualisieren sollen. Ein Bild sagt nun mal mehr als 1000 Worte. Mittlerweile gibt es stehende Bilder aber eigentlich überall, also muß, wer etwas auf sich hält, seine Multimedia-Dokumente mit hübschen Animationen und Videoclips bestücken, denn bekanntlich reagiert der menschliche Sinn auf Bewegtes schneller und aufmerksamer als auf Statisches, Unbewegtes. Und damit die Animationen nicht an die Stummfilmklassiker mit Buster Keaton und Charlie Chaplin erinnern, wird das ganze dann durch Sounds und Töne untermalt. Kein Rechner mehr ohne Soundkarte, keine Multimedia-Präsentation ohne gefällige Hintergrundgeräusche.

Ganz wichtig ist natürlich auch, daß ein modernes Multimedia-Dokument nicht für sich alleine steht, sondern am besten in einen globalen Kontext gesetzt wird. Deswegen ist es natürlich Pflicht, einen Kontakt zum Internet, eine Webanbindung herzustellen, auf daß der Adressat des Textes auch zu einem kleinen Surfritt in die Weiten der globalen Netzwelt eingeladen werde. Tatsächlich stellen Netzbrowser wie Netscape oder MS Internet-Explorer für viele Leute die Inkarnation eines Multimedia-Tools dar. Und wir haben Glück: in der Tat wachsen die Möglichkeiten, die man mit diesen Systemen hat, dramatisch. So ist man nicht mehr nur passiver Konsument, nein, das multimediale Surfen ist interaktiv geworden. Mit Java-Applets kommt immer mehr Leben in die sonst so statischen Webseiten. Da kann man Rubik's Cube mit der Maus drehen oder virtuelle Avatare durch virtuelle Welten steuern. Und damit nicht genug: durch eine immer größer werdende Anzahl an Drittprogrammen, sogenannten Plug-Ins für die verschiedenen Netzbrowser, können immer mehr externe Applikationen eingebunden werden, um noch mehr Abwechslung in die verschiedenen Präsentationen zu bringen.

Diese Bestandsaufnahme ist in ihrem Ton sicher nicht so kritisch und herablassend gemeint, wie ich sie vorgetragen habe. Und natürlich wollen wir alle hier, daß Multimedia die oben aufgeführte Vielfalt nutzt. Es stellt sich aber die Frage: reicht uns das? In der Tat drängt sich dem arglosen Beobachter nämlich folgendes Bild auf:

Multimedia ist bunt, laut und hat etwas mit sich bewegenden Bildern zu tun.

Abbildung 1: Multimedia im Web?

Ist also Multimedia nur ein besseres Fernsehen? Ist Multimedia nur der Konsum von Bildern, angereichert durch vermeintliche Freiheiten, durch die Möglichkeit, scheinbar die Richtung des Programms wählen zu können und hier und da mal mit der Maus ein wenig Fingerakrobatik zu üben? Ist Multimedia light? Muß Multimedia light sein?

Die Tagung, auf der ich vortragen darf, steht unter dem Titel „Multimedia in den Wissenschaften". Wissenschaft ist per Definition nicht light, sondern tiefgängig, beschäftigt sich mit Ursache, Wirkung und Begründung. Wissenschaft versucht, Inhalte zu vermitteln. Schließt Wissenschaft also Multimedia aus? Kann Multimedia überhaupt einen ernsthaften Beitrag zu den Wissenschaften leisten? Die Antwort lautet — natürlich — ja. Nicht zuletzt die vielen Beiträge, die von den Kollegen hier präsentiert werden zeige, daß Multimedia mitnichten light und inhaltslos sein muß. Aber wollen wir uns bei der Nutzung von Multimedia für die Wissenschaften darauf beschränken, unser Wissen akribisch in Datenbanken zu organisieren? Sicher nicht — wir wollen mehr als nur Konservierung von Inhalten, wir wollen Inhalte neu präsentieren, auf neue Art vermitteln.

Wie kommen nun die Inhalte in die Multimedia-Präsentationen? Nun, zunächst natürlich durch die Autoren. Hier werden in der Tat große Anstrengungen unternommen. Ein Projekt an der Universität Bielefeld der AG Prof. Andreas Dress beschäftigt sich z.B. mit der Herstellung eines interaktiven Lehrbuchs für Differential- und Integralrechnung für die gymnasiale Oberstufe. In diesem Projekt arbeiten 3 Leute an der technischen Realisierung des Multimedia-Dokumentes, während doppelt so viele Autoren — hier Didaktiker und Lehrer — sich um die Inhalte kümmern, sprich das ganze sinnvoll strukturieren und mit wohlüberlegten Texten füllen. Inhalte sind also essentiell und machen in der Tat viel Arbeit.

Nun ist aber wohlbekannt, daß seitenlange Texte — gerade am Bildschirm gelesen — in aller Regel nicht immer in angemessenem Maße dazu beitragen, Sachverhalte verständlich zu machen, insbesondere wenn es sich beim Leser um Studenten oder Schüler handelt. Hier sind Beispiele gefragt, Beispiele und nochmals Beispiele. Und wann können Beispiele ihre Wirkung optimal entfalten? Wenn der Leser die Beispiele selbst ausprobieren, selbst abwandeln kann und damit herumspielen kann. Es besteht als begründeter Bedarf daran, dem Leser eines multimedialen Dokumentes Freiheiten an die Hand zu geben, die Inhalte des Textes selbst ergründen zu können und zwar so weit wie möglich, d.h. am besten sogar außerhalb des Rahmens, den der Autor gesteckt hat. Ein interaktives Buch im Bereich der mathematisch-technischen Fächer — und da schließe ich die klassischen Naturwissenschaften natürlich mit ein — ein solches Buch muß also mehr sein, als eine Sammlung von Texten, Bildern und Videoclips. Es muß den Leser einladen, sich am Stoff zu beteiligen, in das Geschehen mit einzugreifen. Diese Binsenweisheit gilt im übrigen auch für den nichtwissenschaftlichen Bereich.

Interaktion in einem abstrakten Fach, wie es die Mathematik darstellt, ist aber nicht ganz einfach. Viele Sachverhalte sind von den praktischen Erfahrungen des täglichen Lebens derart losgelöst, daß sie nicht oder nur schwer durch Bilder vermittelt werden können. In jedem Fall aber, bietet die Mathematik eine unermeßlich große Anzahl an Sachverhalten. Allein für ein einziges Problem gibt es in aller Regel überabzählbar verschiedene unterschiedliche Manifestationen. Ein Student, dem wir das Differenzieren beibringen wollen, wird sich — so er interessiert ist — nicht mit 2-3 Beispielen zufriedengeben, die man ihm vorgegeben hat. Er möchte frei an den Parametern herumspielen, die Aufgabenstellung modifizieren und so zu eigenen Schlüssen kommen. Kann der Autor des zugrunde liegenden Multimedia-Dokumentes alle Eventualitäten vorhersehen? Kann er wissen, welche Beispiele der Leser als relevant und erhellend erachtet und welche selbstevident erscheinen? Er kann es ahnen, aber nicht wissen! Das Multimedia-System selbst muß also genug inhärentes Wissen besitzen, um auf möglichst viele Aktionen des Lesers angemessen reagieren zu können.

Grundlage für maximale Flexibilität bei der Präsentation mathematisch- technischer Sachverhalte muß also ein System sein, mit dem der Leser eine breite Anzahl an mathematischen Problemen selbst erforschen und ergründen kann — eben ein mathematisches Expertensystem, ein Computer-Algebra-System.

Was ist CA? Stellen Sie sich vor, Sie hätten alle Formeln Ihrer Schulzeit, Ihres Studiums und Ihres Berufes sofort zur Verfügung, Sie würden diese anreichern mit dem algorithmischen Wissen einer großen Zahl professioneller Mathematiker, Ingenieure und Naturwissenschaftler und hätten dann noch jemanden, der Ihnen den Umgang mit diesem Wust von Informationen intelligent erledigt, der Ihnen die Schreibarbeit abnimmt, die Formeln fehlerfrei ineinander einsetzt, Ableitungen bestimmt, Gleichungen löst, Graphiken zeichnet, Geometrie verdeutlicht und benötigte Resultate zur Not in Windeseile tausendstellig ausrechnet. Wenn sie sich das alles vorstellen, dann haben sie einen Ausschnitt aus dem was CA schon heute leistet und einen kleinen Ausschnitt aus dem, was sie morgen leisten wird. Sie sollten sich dabei aber nicht von dem Begriff „Algebra" verwirren lassen, tatsächlich versuchen solche Systeme die gesamte Mathematik abzudecken.

Warum aber braucht man nun solche — zugegebenermaßen sehr aufwendigen — Programmpakete?

Ein Beispiel: In einer Einführungsveranstaltung Mathematik für Wirtschaftswissenschaftler (die dieses Jahr übrigens zum ersten Mal durch den Einsatz eines interaktiven Buchs mit CA-Unterstützung begleitet wird) wurde vorletzte Woche in der Klausur folgende Aufgabe gestellt:

Stellen Sie fest, für welche $\lambda \in \Re$ die Matrix $\begin{pmatrix} \lambda & 1 \\ 2 & \lambda \end{pmatrix}$ nicht invertierbar ist.

Natürlich löst jeder von uns diese Aufgabe durch reines Hinsehen (und auch unsere Studenten konnte diese Aufgabe redlich bearbeiten). Für ein Computerprogramm indes ist dies nicht so einfach. Das Programm muß ein Kriterium für die Singularität von Matrizen kennen (hier die Tatsache, daß die Determinante der Matrix gleich Null sein muß), dann muß es in der Lage sein, eine solche Determinante ausrechnen zu können, und zwar nicht mit festen Zahlen, sondern in Abhängigkeit eines Parameters, eines irgendwie benannten Symbols (hier λ). Und schließlich muß es die so berechnete Determinantengleichung auflösen können, und zwar exakt, denn die Antwort auf obige Frage lautet ja mitnichten $1,4142136$, wie es wohl die meisten Taschenrechner behaupten würden, sondern eben $\lambda \in \{\sqrt{2}, -\sqrt{2}\}$.

Genau diese Folge von Lösungsschritten leisten CA-Systeme. Und dabei beschränken sie sich nicht auf kleine Probleme, auf einfache 2x2 Matrizen, sondern können potentiell beliebig große „Formeln" behandeln, wobei „potentiell" in diesem Zusammenhang bedeutet „nur durch die Leistungsfähigkeit des zugrunde liegenden Rechner eingeschränkt". Dabei ist es wichtig, zu erwähnen, daß die Mehrzahl der existieren CA-Systeme — natürlich auch *MuPAD* — auf handelsüblichen PCs oder Macintoshs läuft und keinesfalls die Kraft einer Workstation erforderlich ist (wenngleich letzteres auch nicht hinderlich ist) . Überflüssig zu sagen, daß man solche Programmsysteme in ihrer Kraft und Flexibiltät nicht in ein paar Zeilen Java implementieren kann.

Bei der Benutzung eines solchen CA-Systems wollen wir den Schülern und Studenten aber keinesfalls alles abnehmen, wir wollen sie nur vom wenig erhellenden technischen Kleinkram entlasten. In der TIMSS Studie (oft fälschlicherweise auch als OECD-Bericht bezeichnet) zum internationalen Vergleich der mathematischen Kenntnisse von Schülern und Studenten wird den westlichen Staaten ein enormes Defizit im Vergleich zu den Konkurrenten aus Fernost bescheinigt. Dieses zeigt sich in der Tatsache, ich zitiere:

> „Die Stärke der asiatischen Schüler liegt nicht im Memorieren von Formeln und der Beherrschung von Rechenroutinen, sondern im mathematischen Denken, im Lösen innermathematischer Probleme und in der Anwendung mathematischer Modelle und Verfahren auf quantitative Alltagsprobleme".

Genau hier setzt CA ein, nämlich bei der Entlastung des Menschen von sklavisch durchzuführender Technik zugunsten der Erfassung größerer, wichtiger Zusammenhänge. Und gleichzeitig bietet sie uns die Möglichkeit, unsere multimedialen Dokumente so zu verfassen, daß sie mehr darstellen, als bloße Aufgabensammlungen. Tatsächlich ist der Nutzer eines solchen Systems sehr frei in seinen Fragen an das System und nutzt mehr als nur eine kleine, vorgegebene Menge von Optionen. Insofern kann ein CA-System tatsächlich Neues bieten, womit die Aussage von Herrn Glaser aus der Einleitung relativiert wird. Und natürlich sind die Adressaten von CA nicht nur Schüler oder Erstsemestler. Gerade bei fortgeschrittenen Problemen ist die Kraft eines CA-Systems sehr hilfreich, denn hier wird man bei wirklich komplexen Fragestellungen signifikant unterstützt.

CA bündelt also Expertenwissen, ein Wissen das sich Autoren von multimedialen Lehrdokumenten (im mathematisch-technischen Bereich) zunutze machen sollten.

Wesentlicher Handlungsbedarf besteht nun noch darin, dieses Wissen den Autoren und Lesern in gut handhabbarer Form zugänglich zu machen. Die meisten CA-Systeme geben sich hier traditionell eher zugeknöpft und bieten dem Nutzer zwar recht mächtige Möglichkeiten, Rechendokumente (sog. Notebooks) zu erstellen, vernachlässigen aber die Ein- und Anbindung in und an andere Welten. Ein Autor möchte — je nach Vorliebe — die Rechenkraft des CA-Systems von seinem Lieblings-Autorensystem aus nutzen, so z.B.:

- für Web-basierte Dokumente (HTML o.ä.)
- in „elektronischen" Slides (PowerPoint o.ä.)
- aus LaTeX-basierten Dokumenten
- in „echten" Autorensystemen (Toolbook, Authorware, Director)
- in Textverarbeitungen (Word, StarWriter etc.)

In *MuPAD* stehen bereits jetzt Möglichkeiten zur Verfügung, um obige Einbindungen elementar zu ermöglichen. Die eingebaute online-Dokumentation ist zum Beispiel 100ist sehr eng an *MuPAD* gekoppelt, so daß erläuternde Beispiele eines Textes auf Mausklick nach *MuPAD* kopiert und dort modifiziert oder einfach so ausgeführt werden können. Auf dieser Basis sind bereits mehrere sog. „interaktive Bücher" geschrieben worden, so z.B. über die Lineare Algebra oder über Zahlentheorie. Diese können vom Leser auch nachträglich noch durch eigene Querverweise, Beispiele oder Anmerkungen erweitert werden. *MuPAD* kann — unter Windows 95 und NT — ebenfalls probemlos als OLE-Server in externe Applikationen wie Word, PowerPoint oder eines der genannten Autorensysteme eingebettet werden.

Nächstes Ziel bei diesen Entwicklungen ist nun, diese Einbettungen und Anbindungen noch enger und vor allem komfortabler zu gestalten. Dazu gehört die Entwicklung einer leistungsfähigen 2D-Ein- und -Ausgabe, sowie eine vernünftige Skalierbarkeit des Systems. Skalierbarkeit bedeutet in diesem Zusammenhang, daß die Möglichkeit bestehen muß, nur einen reduzierten Funktionsumfang zur Verfügung zu stellen oder etwa gewisse Ergebnisse oder Vereinfachungen nicht automatisch durchzuführen (so ist es bei Lehreinheiten für die Sekundarstufe 1 wenig sinnvoll, komplexe Zahlen zu verwenden)

Man kann ohne Übertreibung feststellen, daß die Nutzung von CA-Systemen — hier das System *MuPAD* — für multimediale Texte bereits im vollen Gange ist. Es handelt sich hier nicht um Pläne oder Zukunftsmusik, sondern um Verfahren und Tools die sich bereits im praktischen Einsatz befinden und von vielen Nutzern erprobt und genutzt werden.

Ich danke Ihnen für Ihre Aufmerksamkeit.

Connecting *MuPAD* and Singular with MP

Olaf Bachmann, Hans Schönemann and Andreas Sorgatz
University of Kaiserslautern, {obachman,hannes}@mathematik.uni-kl.de
University of Paderborn, andi@mupad.de

Connecting the Computer Algebra systems MuPAD and SINGULAR *based on the Multi Protocol (MP) is an ongoing joint project of the involved developer teams. This report explains the principals of the existing prototype implementation of such a connection.*

Introduction

SINGULAR is a special-purpose Computer Algebra (CA) system for polynomial computations emphasizing the computational needs of commutative algebra, algebraic geometry and singularity theory. Among others, it features one of the fastest and most general implementations of algorithms for computing Gröbner bases. Making large parts of SINGULAR's functionality available to the users of *MuPAD* has obvious advantages: first of all, it single-handedly exports the know-how and state-of-the-art of an area of CA which, so far, has hardly been supported by *MuPAD*, and, secondly, it enables much more efficient and general computations of Gröbner bases.

However, the requirements of a *MuPAD* - SINGULAR connection make its realization a rather challenging and non-trivial task. In particular, such a connection needs to ensure

Independence: *MuPAD*'s and SINGULAR's kernels may neither be changed nor directly linked together. Instead they must be kept as independently maintained and developed programs.

Efficiency: Due to the nature of the involved computations, the amount of data which needs to be exchanged between the two programs can be very large. Hence, a time-bottleneck in the data exchange may undo any potential efficiency gains, or, may even make such a connection practically unusable.

Usability: SINGULAR's functionality should be provided to *MuPAD* users in a natural and seamless way, i.e. the technicalities of the underlying connection and the particularities of using SINGULAR should be hidden from the *MuPAD* user as far as possible.

These requirements immediately ruled out a number of straight-forward mechanisms for the realization of the connection: linking *MuPAD* directly with SINGULAR violates the independence requirement. Exchanging data between the two programs in an ASCII (i.e. plain text) format cannot ensure the required efficiency. Duplicating SINGULAR's (sometimes peculiar) I/O conventions in its *MuPAD* interface cannot assure the required usability.

Nevertheless, the existing features of *MuPAD* and SINGULAR allow a connection of the two systems which meets the requirements above. The key components in this realization are *MuPAD*'s dynamic modules [12, 13, 14], the Multi Protocol (MP) and SINGULAR's MP interface.

Before we describe how the interplay of these components realizes the desired connection (section *Interfacing*) we introduce MP and SINGULAR in more detail first. At the end, we discuss our results and outline the work which still lies ahead of us.

The Multi Protocol (MP)

The purpose of the Multi Protocol is to support general and efficient communication of mathematical data among scientific computing systems. MP defines a set of basic types and mechanisms for constructing structured data. Numeric data (fixed and arbitrary precision floats and integers) are transmitted in a binary format (2's complement, IEEE float, etc.). Composite tree-like data (such as symbolic expressions) are represented as linearized, annotated syntax trees (*MP trees*) which are transmitted as sequences of *node* and *annotation packets*, where each node packet transmits a node from a syntax tree. A node packet has a header giving the type of the data carried in the packet, the number of arguments (for operators) that follow, a dictionary tag, and the number of annotations attached to the node.

Annotations carry additional information which is either supplementary and can be safely ignored by the receiver, or may contain information essential to the proper decoding of the data. Each annotation is tagged in such a way that the receiver always knows whether it can safely ignore the annotation's content or not.

Composite homogeneous data (such as polynomials, or matrices) is represented in compact encodings based on the concepts of prototypes. In short, a prototype is an MP tree describing the syntactic structure of composite data. It is used to separate the syntactic description of objects from the data of the object. This way, homogeneous data can be represented very compactly by first (and once) encoding their header(s) which is followed by “header-less” encodings of the actual data. The prototypes mechanism is one of the major sources of MP's efficiency.

In a layer above, the data exchange portion of the protocol, MP supports collections of definitions for annotations and mathematical symbols (operators and symbolic constants) in *dictionaries*. Dictionaries address the problem of semantic data integration by defining standardized representation(s) and semantics for mathematical objects. They are identified within node packets through the dictionary tag field. Applications that communicate according to definitions provided in dictionaries do not need to have direct knowledge of each other.

One of the existing MP dictionaries is the MPP (MP-Polynomial) dictionary. It specifies the syntactic encoding of polynomial data and the syntax and semantics of polynomial operations. Besides the straight-forward expression-tree encoding of polynomials, MPP supports compact, prototype-based recursive and distributed encodings of polynomials. Based on these polynomial encodings, more complex objects, such as ideals, modules, or matrices of polynomials can be constructed. Defined operations on polynomial data include computations of Gröbner and Standard bases using various monomial orderings, ideal and module dimensions, syzygy and resolution computations, factorization and gcd of polynomials, and many more.

Applications send (“Put”) and receive (“Get”) data as *messages*, containing an MP tree which is (typically) created by calling routines from the MP Application Programming Interface (API). The MP API is provided as a publicly available, well-documented C-library (`libMP`) which has been ported to all major hard- and software platforms.

An application communicates with other applications through an *MP link*, which is simply an abstraction of an underlying data transport mechanism which is bound to the link at the time of its creation. Up to now, the MP API implements links based on file, socket, PVM, toolbus, email, and WWW transport mechanisms.

MP interfaces exist to the CA systems SINGULAR, Mathematica, *MuPAD* and to various small, special-purpose systems. MP has been, and is mainly, developed at Kent State University (Simon Gray, Paul Wang) with additional contributions by various other people (including the authors). Ongoing MP projects include the use of MP for parallel symbolic computations, support for MPI and shared-memory based links, the development and refinements of dictionaries for various areas (polynomial computations, linear algebra, etc.) and the cooperation with other groups and efforts (like OpenMath) aimed at a standard protocol for communicating mathematical data.

For more information about MP in general, see [9] and [1], about MP prototypes, see [8] and about MPP, see [6]. Always up-to-date information about MP, as well as the latest MP software release can be found at: `http://symbolicnet.mcs.kent.edu/areas/protocols/mp.html`

Singular

SINGULAR is a computer algebra system for polynomial computations with emphasis on the special needs of commutative algebra, algebraic geometry, and singularity theory. Its main computational objects are polynomials, ideals and modules over a large variety of rings. SINGULAR features one of the fastest and most general implementations of various algorithms for computing standard bases. Furthermore, it provides polynomial factorizations, resultant, characteristic set and gcd computations, syzygy and free-resolution computations, and many more . . .

Based on an easy-to-use interactive shell and C-like programming language, SINGULAR's internal functionality is augmented and user-extendible by libraries written in the SINGULAR programming language. A general and efficient implementation of links in general, and of MP links in particular, allows SINGULAR to make its functionality available to, and be easily incorporated into, other programmes.

SINGULAR has been developed by G.-M. Greuel, G. Pfister, and H. Schönemann at the University of Kaiserslautern since 1984. The current developer group consists, further, of O. Bachmann, T. Siebert, T. Wichmann, and W. Pohl.

SINGULAR is publicly available as a binary program for all common Unix platforms, for MS-DOS, Windows 95/NT and for MacOS. The current version number is 1.0. It can be downloaded by anonymous ftp from the site `ftp://www.mathematik.uni-kl.de/pub/Math/Singular`

Besides the executable SINGULAR program, the distribution contains the source code of all SINGULAR libraries and a user manual. For more and always up-to-date information, SINGULAR's home page im WWW can be reached at: `http://www.mathematik.uni-kl.de/~zca/Singular`

Mathematical Features

SINGULAR's primary computational objects are ideals, resp. modules, generated by polynomials, resp. polynomial-vectors, over polynomial rings or, more generally, over the localization of a polynomial ring with respect to any ordering on the set of monomials which is compatible with the semigroup structure. Supported base–rings include:

- polynomial rings with a large variety of polynomial orderings (simple, block, elimination, weight, and general matrix orderings),
- localization of a polynomial ring at a prime ideal generated by a subset of the variables,
- factor rings by an ideal of one of the above,
- rings of tensor products of one of the above,
- exterior algebra.

Supported ground fields for these rings include:

- rational numbers $\mathbb{Q}$,
- finite fields $\mathbb{Z}/p$ (where p is a prime ≤ 32003),
- Galois fields (finite fields with $q = p^n \leq 2^{15}$ elements),
- transcendental extensions ($K(A, B, C, \ldots)$, $K = \mathbb{Q}$ or $\mathbb{Z}/p$),
- algebraic extensions ($K[t]$/minimal-polynomial, $K = \mathbb{Q}$ or $\mathbb{Z}/p$), and
- floating point real numbers with single precision.

The main algorithms implemented in SINGULAR are:

- General standard basis algorithm for *any* monomial ordering which is compatible with the natural semi-group structure of the exponents. This includes well-orderings (Buchberger algorithm) and tangent cone orderings (Mora algorithm) as special cases.
- Traverso's Hilbert–driven and the Trace Gröbner basis algorithms; weighted–ecart–method and high–corner–method, FGLM algorithm for change of ordering.

- Factorizing Buchberger algorithm.
- Intersection, quotient, elimination and saturation of ideals.
- Different algorithms for computations of syzygies and free resolutions of modules.
- Combinatorial algorithms for computations of dimensions of factor rings, Hilbert series and multiplicities of modules.
- Standard multivariate polynomial gcd, resultant, and factorization algorithms.
- Wang's algorithm to compute characteristic sets.
- Computations in the localization by any prime ideal (Mora's Graal).
- Primary decomposition of ideals and modules based on Gianni/Traeger/Zacharias' and on Shimoyama/Yokoyama's algorithms.
- Arnold's classifier of singularities.
- Ring normalization.
- Ring invariants under certain group actions (finite groups, special unipotent groups).
- Invariants of singularities of affine or projective varieties such as Milnor, Tjurina and discriminate numbers, singular locus, T1 and T2 modules, Massey products, etc.
- Algorithms for homological algebras as, for example, for Ext, Tor.

See also [2, 4, 5] for more details on the implemented algorithms.

Computational Features

SINGULAR has a convenient and intuitive interactive user interface (shell). SINGULAR's user interface provides both, access to SINGULAR's mathematical functionality and a convenient, powerful, and C-like programming language (strongly typed, and lexicographically scoped) which includes all the usual programming constructs (like loops, procedures, local/global variables, etc). Based on this programming language, users may extend SINGULAR's functionality by writing their own libraries.

SINGULAR, furthermore, features links as general endpoints of communications, i.e. as something SINGULAR can read from or write to. To this point, the following link types are implemented:

- `Ascii text`: output can conveniently be viewed and manipulated. Read/write is not the fastest.
- `DBM` (Standard Unix database): provides access to data stored in a data base.
- `MP file`: stores data in the binary Multi Protocoll (MP) format [6]. Read/write is very fast.
- `MP TCP` (Socket based, Client/Server links): exchanges data in binary MP format between processes (on the same or different computers); data exchange is very efficient.

The functionality of theses links is provided to the user by a general, consistent and convenient link interface.

Based on MP TCP links, SINGULAR can very efficiently communicate with itself which opens the door for implementations of parallel/distributed algorithms. Furthermore, the same links can be used to communicate with other Computer Algebra programs which have an MP interface. At the moment, there are MP interfaces for *MuPAD*, and Mathematica, enabling the use of one system from the others (e.g. one can use SINGULAR's functionality from within Mathematica or *MuPAD*).

Implementation

SINGULAR is implemented in C/C++ and its kernel consists of approx. 70,000 lines of source code. SINGULAR's main implementation design goal is speed. Therefore, all time-consuming operations like standard bases computations or factorization are implemented in its kernel. As another consequence, SINGULAR has the concept of a global ring which needs to be defined prior to any polynomial operations. Arbitrary precision integer arithmetic is accomplished by linking SINGULAR with the GNU multiple precision library `gmp` and modulo arithmetic is accomplished using look-up tables. Polynomials are internally represented as linked lists of monomials, where a monomial consists of a coefficient and an exponent vector.

To illustrate SINGULAR's speed of Gröbner basis computations, the table below shows the timings of various systems for solving problem 6 of the ISSAC' 97 system challenge (computation of a lexicographical Gröbner basis – see [10, 11] and section *Summary*). All timings were taken on a Pentium 90, 32MB RAM running Linux:

MapleV R4, *MuPAD* 1.3, Reduce, CoCoA (3.0.2), Macaulay 2 (0.8.14)	Mathematica 3.0.1	GB v3[3]	SINGULAR
–[1]	362 sec[2]	34 sec	2 sec

1): interrupted after approx. 15 CPU hours
2): on Pentium Pro 200
3): GB is the program developed by Faugere

Interfacing *MuPAD* and SINGULAR

Based on SINGULAR's MP interface, we connected *MuPAD* and SINGULAR using the following principles. They are explained in more detail in the following subsections.

***MuPAD* – MP interface** An MP interface for *MuPAD* implemented as a *MuPAD* dynamic module, serves a low-level interface for communications with other programs.

***MuPAD* – SINGULAR communication** Based on this *MuPAD*–MP interface, *MuPAD* and SINGULAR are run as independent processes (on the same or different machines) which exchange data and commands using MP-TCP links.

***MuPAD* library** `sing` A *MuPAD* library serves as a high-level interface to the functionality provided by SINGULAR and hides all the technicalities of the underlying communications.

The *MuPAD* – MP interface

The *MuPAD*–MP interface is encapsulated in the *MuPAD* dynamic module `mp`. It consists of the MP library (`libMP`) and a *MuPAD* layer which provides large parts of the MP-API (i.e. the routines provided by `libMP`) in form of *MuPAD* functions in an almost one-to-one fashion. MP links can be opened and closed with `mp::open()` and `mp::close()` where the argument syntax of `mp::open()` is as that of the MP routine `MP_OpenLink()`. On *MuPAD*'s top-level, an MP link is simply represented as an integer. Once a link is opened, the usual “Put” and “Get” operations for transmitting and receiving primitive data (such as integers, floats, operators, etc.) can be performed using the top-level *MuPAD* `mp::Put` and `mp::Get` routines (e.g. `mp::PutSint32Packet` which writes a signed 32-bit integer to a link). The implementation of this layer for communicating primitive data was eased very much by the fact that the `libMP` transparently handles different fixed-length and arbitrary-length integer and float formats (e.g. little/big endian fixed-length integers, or GMP/PARI arbitrary-length integers), i.e., by the fact that `libMP` transparently performs all necessary conversions of numeric data.

Having the primitve routines of the MP API directly available as interactive top-level *MuPAD* functions has the advantage that subsequent implementations can be done in the convenient high-level language of *MuPAD*. This allows rapid prototyping and easy debugging. It, furthermore, provides a well-defined and well-documented MP interface (in the sense that the corresponding, low-level MP API functions are well-defined and well-documented).

However, it has the disadvantage that marshalling and unmarhsalling of complex objects (e.g. polynomials) is done in a piece-mail fashion on the interpreter-level, which, especially for large amounts of data, can result in a significant loss of efficiency. Therefore, we did not stop at lifting MP's "Put" and "Get" routines to *MuPAD*'s top-level user interface, but also implemented the more efficient and general routines `mp::write()` and `mp::read()`.

Once their implementation is complete, the routine `mp::write()` takes as arguments an arbitrary *MuPAD* object and "puts" it to an MP link using C-written (and hence, precompiled), "canned" routines for the different *MuPAD* objects. At the moment, we only support *MuPAD* objects which correspond to the MPP polynomial data objects, that is, to polynomials, ideals, modules, and matricies. Likewise, the routine `mp::read()` is intended to read an arbitrary MP object from a link and to return the corresponding *MuPAD* object and, at the moment, is only implemented for the above–mentioned polynomial objects.

Besides I/O routines, there are, furthermore, routines to check the status of a link (e.g., routines for checking whether there is something to read from a link, or whether a link is ready for writing).

We should point out that the implemented MP interface for *MuPAD* has applications far beyond the SINGULAR – *MuPAD* interface: due to the generality and universality of MP, it, furthermore, enables communications with all other programs which have an MP interface implemented. In particular, it opens the door for accessing Mathematica directly from within *MuPAD*, and vice versa.

The *MuPAD* – SINGULAR communication

Provided that SINGULAR is correctly installed on your local host, and that the `mp` module is loaded into *MuPAD*, a connection from *MuPAD* to SINGULAR can be opened by the *MuPAD* command

```
>> link:=mp::open("-MPtransp","TCP","-MPmode","launch","-MPapplication","Singular -b")
```

which starts the SINGULAR program in batch modus (i.e. all I/O is done through the connecting MP - TCP link) on the local host, and returns the link (identification number) which connects the two programs. With slightly changed arguments, SINGULAR could also be started on a remote host, or a connection could be established to an already running SINGULAR program (consult the documentation of the MP API for a detailed discussion about the meaning of the arguments to the call for opening a link).

Links to SINGULAR are closed like all other MP links, that is, using the following command which also causes the connecting program (e.g., SINGULAR) to exit.

```
>> mp::close(link):
```

Once the connection is established, SINGULAR acts just like an external compute-server for *MuPAD*, i.e., SINGULAR reads input from the connecting link, evaluates (computes) the read expression and writes the results back to the connecting link.

SINGULAR's functionality can be accessed at two levels:

At the MPP level: as explained in sections , the MP Polynomial dictionary (MPP) defines the syntax and semantics of a large variety of polynomial operations. SINGULAR implements almost all of these operations. Hence, on the MPP level, SINGULAR provides its functionality in a system-independent way – nothing needs to be known about SINGULAR to access the MPP operators. For example, the MPP dictionary defines that the (MP) operator `MP_CopPolyStd` takes as arguments the generators of an ideal and, optionally, a monomial ordering, and returns the Standard (resp. Gröbner) basis of the ideal w.r.t. the given monomial ordering. SINGULAR implements this operator, and its functionality can be accessed from *MuPAD* using the command:

```
>> mp::write(link, Std(Ideal(poly(x+y+z,[x,y,z]), poly(x*y*z,[x,y,z]))));
>> gb := mp::read(link);
```

```
                     2    2
   Ideal(poly(y  z   + y   z,  [x,  y,  z]),  poly(x  +  y  +  z,  [x,  y,  z]))
```

See [7] for a detailed description of the operators defined by MPP and the documentation of the `mp::write` routine for the mapping of *MuPAD* operators to MPP operators.

At the SINGULAR level: besides the operators defined in the MPP dictionary, almost all of SINGULAR's top-level operations (i.e. those available from SINGULAR's interactive user shell) can be accessed via its MP interface, and hence, from *MuPAD*, as well. However, their syntax and semantics has to be exactly as specified in the SINGULAR user manual, which, due to the pecularities of SINGULAR's I/O syntax and the *MuPAD* non-existence of many concepts used in SINGULAR, imposes severe limitations on the usability of these operations. Their usage should generally be avoided, or, at least should be restricted to a minimum.

The *MuPAD* library `sing`

Although very flexible and powerful, the direct usage of *MuPAD*'s `mp` module to access SINGULAR's functionality requires that a user needs to be familiar with the basic concepts of MP, MPP and of *MuPAD*'s MP interface. In most cases, however, users are probably only interested in being able to perform certain computations in a reasonable time, and would rather be spared from having to know the details of the underlying communications.

Therefore, we implemented the *MuPAD* library `sing` (which stands for "Singular", or, "Singularity theory") which serves as an additional wrapper around the `mp` module and provides high-level access to most of the MPP operators implemented by SINGULAR: The routines of the `sing` library transparently open a connection to SINGULAR (if one was not already opened), store the connecting link-id in a global variable (`sing::link`) and call the routines `mp::write()` followed by `mp::read()` with the appropriately transformed arguments. In other words, all the details of the underlying MP connection (like opening/closing a link, writing/reading expressions to/from a link, etc.) are totally hidden from the user — the routines of the `sing` library provide their functionality just like any other *MuPAD* library routine.

As an illustration, to compute a Gröbner basis by SINGULAR based on *MuPAD*'s `sing` library can be as easy as:

```
>> loadlib("sing"):
>> gb := sing::std(Ideal(poly(x+y+z,[x,y,z]), poly(x*y*z,[x,y,z]))));
                  2    2
   Ideal(poly(y  z   + y   z,  [x,  y,  z]),  poly(x  +  y  +  z,  [x,  y,  z]))
```

Summary and Conclusions

The implemented *MuPAD*–SINGULAR connection is based on the following components whose interplay is summarized in Figure 2.

1. The Multi Protocoll (MP) which serves as an efficient inter-process communication protocoll for concurrently running instances of the *MuPAD* and SINGULAR programs.
2. SINGULAR's and *MuPAD*'s MP interfaces which enable *MuPAD* to access SINGULAR's functionality.
3. The *MuPAD* library `sing` which hides the underlying communication technicalities from the *MuPAD* users.

Using this connection, we are able to provide much of SINGULAR's functionality to the *MuPAD*–users in a convenient and efficient way. For example, to solve problem 6 of the ISSAC'97 system challenge in *MuPAD*, all that is needed are the following few lines of interactive input:

```
>> p1:= poly(8*w^2+5*w*x -4*w*y+2*w*z+3*w+5*x^2+2*x*y
             -7*x*z-7*x+ 7*y^2-8*y*z-7*y+7*z^2 -8*z+8, [w,x,y,z]):
>> p2:= poly(3*w^2-5*w*x-3*w*y-6*w*z+9*w+4*x^2+2*x*y
             -2*x*z+7*x+9*y^2+6*y*z+5*y+7*z^2+7*z+5, [w,x,y,z]):
>> p3:= poly(-2*w^2+9*w*x+9*w*y-7*w*z-4*w+8*x^2+9*x*y -
             3*x*z+8*x+6*y^2-7*y*z+4*y-6*z^2+8*z+2, [w,x,y,z]):
```

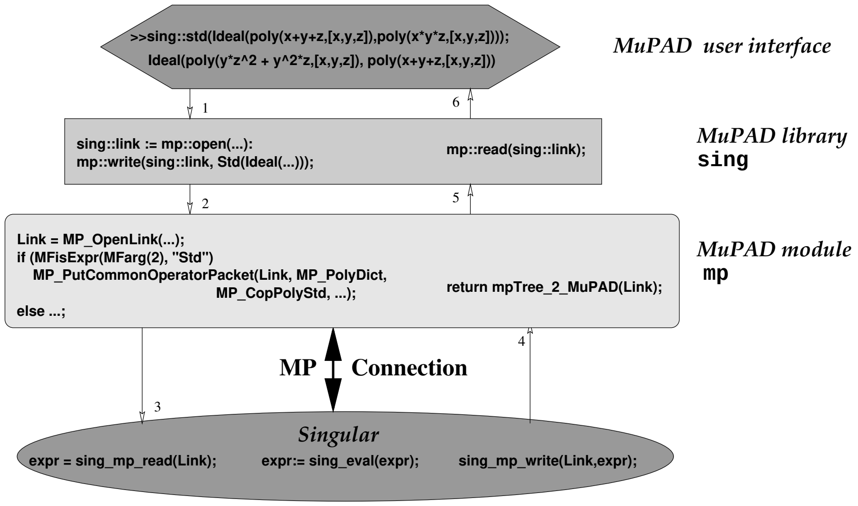


Figure 2: Summary of the *MuPAD* – Singular connection

```
>> p4:= poly(7*w^2+5*w*x+3*w*y-5*w*z -5*w+2*x^2+9*x*y-7*x*z+4*x -
              4*y^2-5*y*z+6*y-4*z^2-9*z+2, [w,x,y,z]):
>> st:= rtime();
>> loadlib("sing"):  gb:=sing::std( Ideal(p1, p2, p3, p4) );
>> (rtime()-st) / 1000.0;
                                    1.5
```

Notice that the entire communication and computation (which includes launching Singular and opening the MP connection) took only 1.5 seconds (on an HP 9000/735). Using *MuPAD*'s "native" `groebner` command to compute the Gröbner basis of the ideal above, takes, with some tricks, almost 2000 seconds, and, without tricks, does not produce a result; even just writing the result of the computation above with *MuPAD* into an ASCII-text file takes almost 1 second (i.e. almost as long as the MP-based *MuPAD* – SINGULAR communication and the SINGULAR computation combined!). Our *MuPAD* – SINGULAR connection made *MuPAD* compute the above Gröbner basis faster than any other system participating in the ISSAC'97 challenge (the only other system which solved this problem was Mathematica – which took over 300 sec. to compute the result) – see [10, 11] for more details on the ISSAC'97 system challenge. — However, much work lies ahead to complete what was started:

- The `mp` module needs to be extended so that the `mp:write()` and `mp::read()` routines are more powerful, i.e., can write (resp. read) a greater class of *MuPAD* (resp. MP) expressions.
- The `sing` library needs to be extended to provide more high-level interface wrappers to the functionality available through the SINGULAR – *MuPAD* connection.
- *MuPAD* domains for the most commonly used polynomial data structures (such as polynomial rings; ideals, modules, matricies with elements from polynomial rings) need to be designed and implemented to allow more convenient manipulations of and operations on these data structures.
- The implemented routines and mechanisms need to be documented in more detail.
- The implementation needs to be ported to operating systems different from Unix.

Nevertheless, as our preliminary results and experience show, the foundation is solid, all basic building blocks are there and in order — all that remains to be done to complete our project is to to put the blocks together and make a robust and large building out of them.

Availability

It is planned to include all components of the *MuPAD* – SINGULAR connection into a forthcoming *MuPAD* release. See also section *MP* and section *Singular* for the availability of MP and SINGULAR.

References

[1] GRAY, S. AND KAJLER, N. AND WANG, P. S., *MP: A Protocol for Efficient Exchange of Mathematical Expressions*, Proc. of the International Symposium on Symbolic and Algebraic Computation (ISSAC' 94), 1994.

[2] H. GRASSMANN AND G.M. GREUEL AND B. MARTIN AND W. NEUMANN AND G. PFISTER AND W. POHL AND H. SCHÖNEMANN AND T. SIEBERT, *On an implementation of standard bases and syzygies in* SINGULAR, Reports On Computer Algebra, Centre for Computer Algebra, University of Kaiserslautern, 1996.

[3] H. SCHÖNEMANN, *Algorithms in* SINGULAR, Reports On Computer Algebra, Centre for Computer Algebra, University of Kaiserslautern, 1996.

[4] G.M. GREUEL AND G. PFISTER, *Advances and improvements in the theory of standard bases and syzygies*, Reports On Computer Algebra, Centre for Computer Algebra, University of Kaiserslautern, 1996.

[5] GREUEL, G.-M. AND PFISTER, G. AND SCHÖNEMANN, HANS, *Singular Reference Manual, Version 1.0*, Reports On Computer Algebra, Centre for Computer Algebra, University of Kaiserslautern, 1996. `http://www.mathematik.uni-kl.de/~zca/Singular`.

[6] BACHMANN, O. AND GRAY, S. AND SCHÖNEMANN, H., *MPP:A Framework for Distributed Polynomial Systems Based on MP*, Proc. of the International Symposium on Symbolic and Algebraic Computation (ISSAC' 96), 1996.

[7] BACHMANN, O. AND SCHÖNEMANN, H., *A Manual for the MPP Dictionary and the MPP Library*, Reports on Computer Algebra, Centre for Computer Algebra, Department of Mathematics, 1996.

[8] BACHMANN, O. AND GRAY, S. AND SCHÖNEMANN, H., *A Proposal for Syntactic Data Integration for Math Protocols*, Proc. of the Second International Symposium on Parallel Symbolic Computation PASCO' 97, ACM Press, 1997.

[9] GRAY, S. AND KAJLER, N. AND WANG, P. S., *Design and Implementation of MP, a Protocol for Efficient Exchange of Mathematical Expressions*, Journal of Symbolic Computing, 1997.

[10] *SIGSAM Bulletin*, Issue 121, Volume 31, Number 3, 1997.

[11] *SIGSAM Bulletin*, Issue 122, Volume 31, Number 4, 1997.

[12] A. SORGATZ, *The Concept of Software Integration in MuPAD*, IMACS' 97, Maui, USA. Available at `http://www.mupad.de/BIB/ONLINE/sorgatz97c.ps.gz`.

[13] A. SORGATZ, *Dynamic Modules in MuPAD*, Poster, ISSAC' 97, Maui, USA. Available at `http://www.mupad.de/BIB/ONLINE/SORGATZ97b/`.

[14] A. SORGATZ, *Dynamische Module – Eine Verwaltung für Maschinencode-Objekte zur Steigerung der Effizienz und Flexibilität von Computeralgebra-Systemen*, *MuPAD* Reports, B.G.Teubner. 1996.

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

MuPAD for Windows 95 / NT

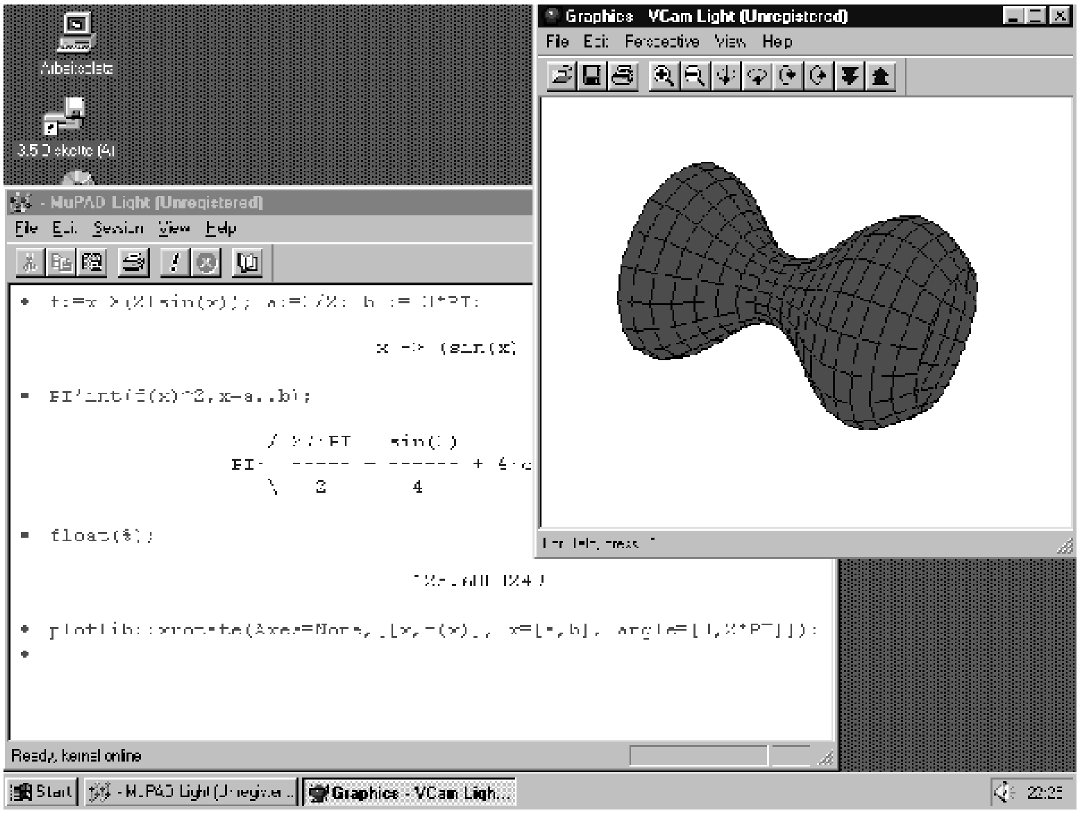
Oliver Kluge
SciFace Software GmbH & Co. KG, `kluge@sciface.com`

At present SciFace offers two versions of MuPAD for Windows, MuPAD Pro *and* MuPAD Light*, which differ in their user interfaces but have the same mathematical capabilities.*

MuPAD Light is freely distributed to members of non-profit educational and scientific institutions. It feature all the mathematical capabilities of MuPAD Pro, so you should not mix this up with the restricted versions you know from other systems.

MuPAD Pro, the professional version of *MuPAD* for Windows, feature notebook functionality, text formatting and fully embedded interactive graphics. Free trial licenses are available. Special licenses for members of non-profit educational and scientific institutions are also available.

The web site of SciFace `http://www.sciface.com` contains detailed information about the license conditions. Download MuPAD Light and trial versions of MuPAD Pro free of charge.

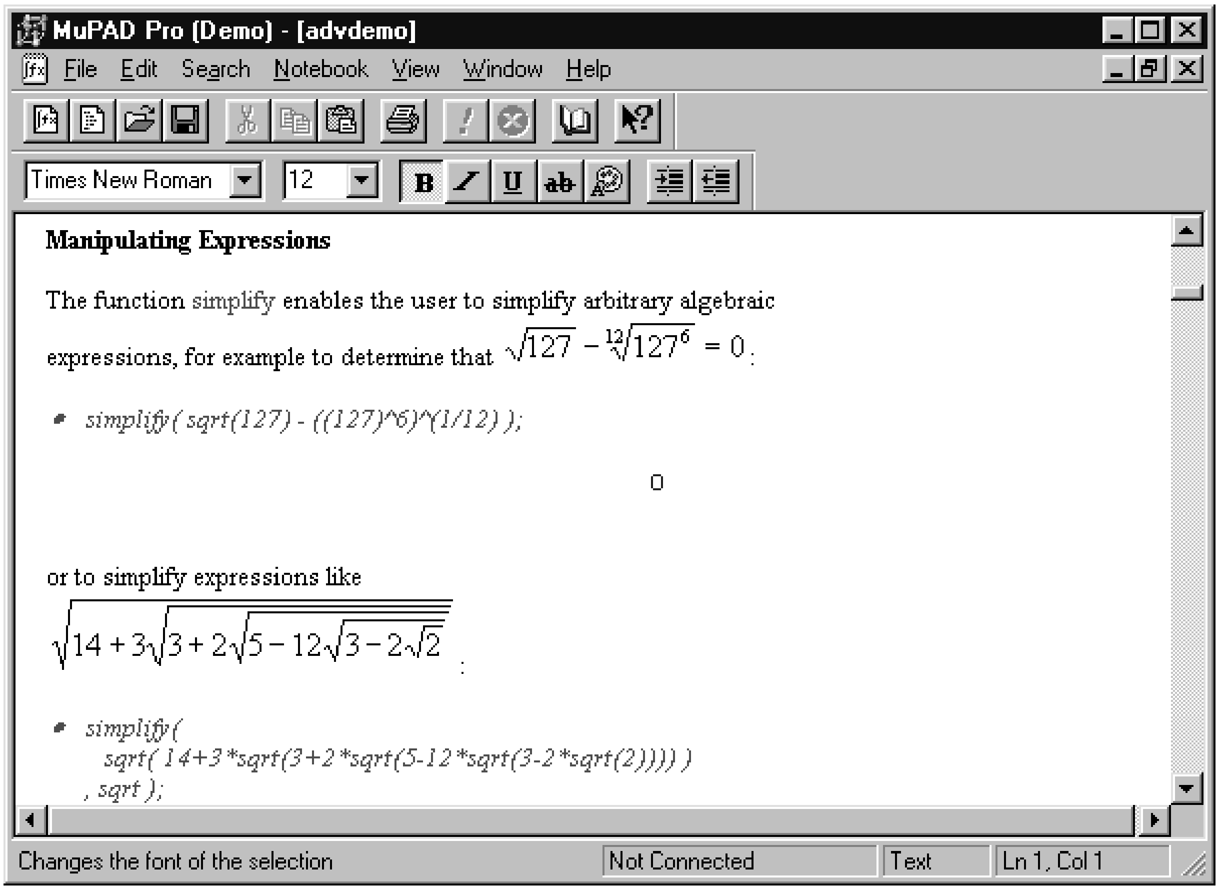


Highlights of *MuPAD* for Windows 95 / NT

Notebooks for Interactive Calculations

- Notebooks combines descriptive text, calculations, graphics and embedded objects
- Notebooks may contain styled text and can be converted to RTF (rich text format)
- *MuPAD* graphics can be embedded into notebooks
- Built-in text editor for writing user-defined procedures
- Drag & Drop
- Full OLE support (MuPAD Pro is an OLE server, container and Active Document server)
- Multiple *MuPAD* sessions can run concurrently
- Notebooks can be saved and reevaluated

Notebooks are available in MuPAD Pro only.

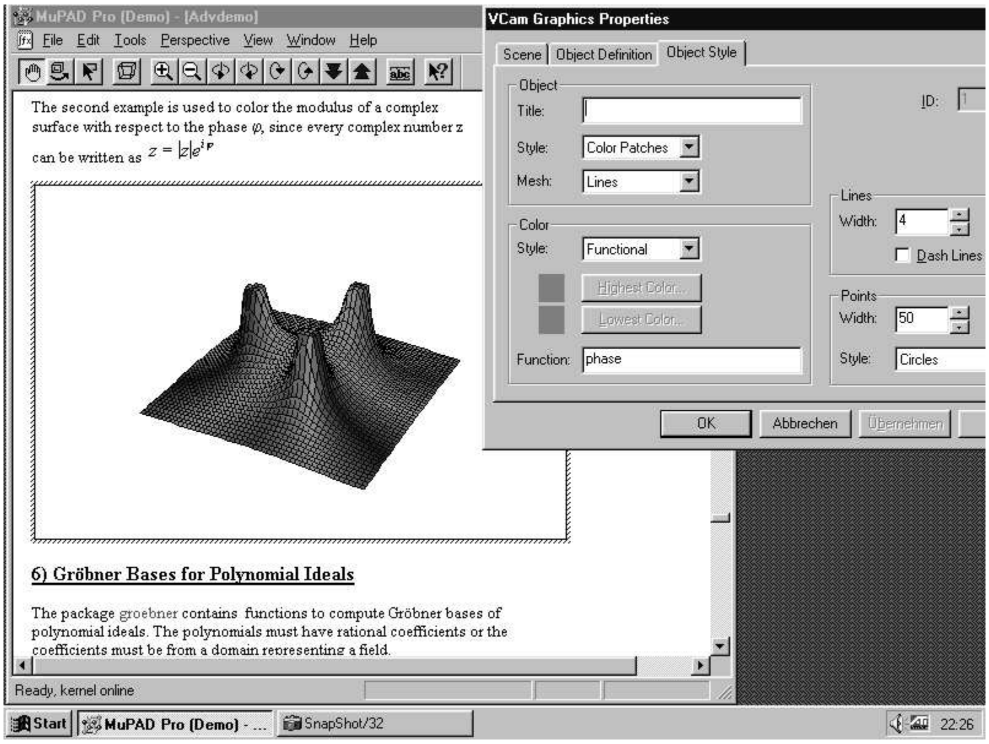


Interactive Graphics Tool for 2D- and 3D-Scenes

- Several objects per scene
- Various style and coloring options
- User definable color functions
- Zooming and rotating of scenes
- Plotting of curves, surfaces and user-definable lists of points or polygons
- Library for contour, cylindrical, implicit, logarithmic, polar, spherical and vector field plots

Additional features of MuPAD Pro:

- Graphics can be embedded into MuPAD Pro notebooks
- New objects can be defined interactively via dialogs
- All properties can be defined interactively via dialogs
- Interactive zooming, rotating and movement of titles
- PostScript output (EPSF)
- Graphics can be saved as Windows Metafiles
- VCam Pro graphics are OLE objects

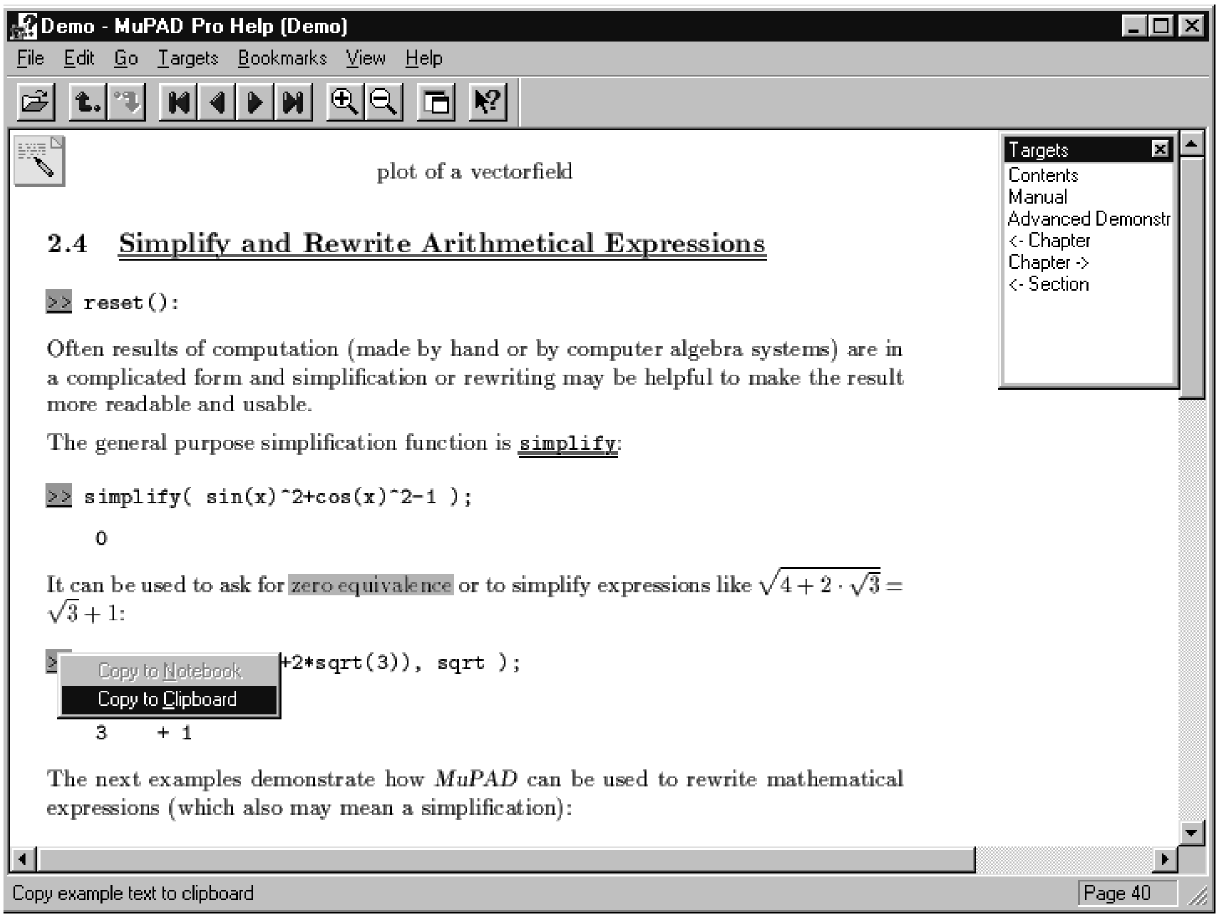


Online Hypertext Documentation

- Complete system documentation included (more than 1000 pages)
- Extensive hyperlink functionality
- Cross document links
- Learning-by-doing with click-and-execute examples

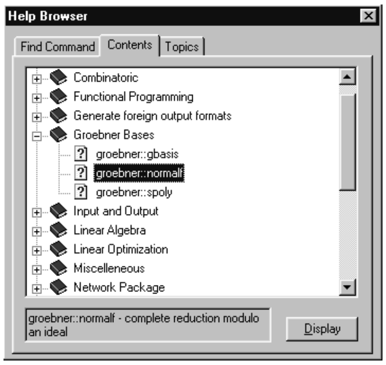
Additional features of MuPAD Pro:

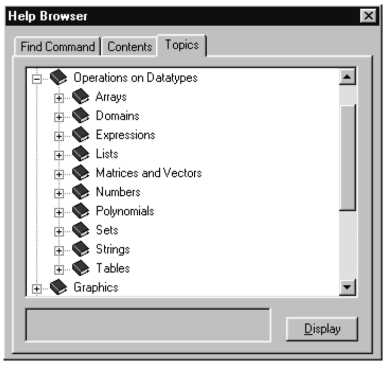
- User-definable hyperlinks
- User-definable bookmarks
- User-definable notes



Help Browser

- Searching of functions, commands, data types or keywords
- Browsing the table of content of the documentation
- Browsing the documentation according to topics





Further Information

SciFace
Scientific Interfaces

SciFace Software	Phone	:	+49 5251 640751
GmbH & Co. KG	Fax	:	+49 5251 640799
Technologiepark 12	Email	:	info@sciface.com
D-33100 Paderborn	WWW	:	http://www.sciface.com
Germany			

CS: a *MuPAD* Package for Counting and Randomly Generating Combinatorial Structures

Alain Denise, Isabelle Dutour and Paul Zimmermann
LRI, Université Paris-Sud, `Alain.Denise@lri.fr`
LORIA, Nancy, {`Isabelle.Dutour,Paul.Zimmermann`}`@loria.fr`

We present a new computer algebra package which permits to count and to generate combinatorial structures of various types, provided that these structures can be described by a specification, as defined in [6].

What is CS ?

CS is a computer algebra package devoted to the handling of combinatorial structures. Its main features are the following: given a combinatorial specification of a class of *decomposable* structures (in the sense of [6]), CS is able to count and uniformly draw at random the structures of any given size n. It can also give some properties of the associated generating series, like recurrences and differential equations.

CS, like COMBSTRUCT (previously known as Gaïa [11]), another package developed by one of the authors, is mainly based on the theory developed in [6] and [7]. But, in addition to the fact that CS is a *MuPAD* package while COMBSTRUCT is a Maple package, CS presents some more advanced features: at the time of submission, CS offers the same fonctionalities than COMBSTRUCT offers concerning unlabelled structures (excepted for cycles and some conditions on cardinality of sets), plus the following:

- Given any decomposable class, CS is able to automatically generate the source of a C program for almost uniform random generation. Then this program can be compiled and used totally independently from *MuPAD*. You are then able to get the approximate counting or draw at random one or several of its elements of desired size, much more efficiently than with a Maple or a *MuPAD* procedure. This is the first implementation of one of the algorithms given in [4].
- When the generating series of a decomposable class is holonomic, CS is able to compute, using Gröbner basis calculation and Gaussian elimination, the minimal order recurrence that its coefficients satisfy. This fairly improves the complexity of counting and enables generating larger structures.

At the time of submission, CS only deals with unlabelled structures. The CS package is currently being developed and we plan to add the labelled case and several other new features; Section describes some of them.

How to install CS on your computer

You can download the CS package (`cs.mu`, one ASCII file) at the address:

```
http://www.loria.fr/ dutour/CS
```

Then you just have to run *MuPAD* (version 1.3 or 1.4) and to read CS by typing, after the prompt “>>”,

```
read("cs.mu"):
```

Counting and Generating

A *specification* of a class of combinatorial structures, as defined in [6], is a set of productions made from basic objects (atoms) (**Epsilon** and **Z** of size 0 and 1 respectively) and from constructions (**Union** for disjoint unions, **Prod** for products, **Sequence** for sequences, **Set** for sets (labelled case) or multisets (unlabelled case)).

Here is the *MuPAD* instruction which creates the specification of plane trees:

```
>> specPT:={T=Prod(Z,Sequence(T))};

                  {T = Prod(Z, Sequence(T))}
```

In other words, a plane tree is a product of a root vertex (the atom `Z`) and a sequence (possibly empty) of plane subtrees. To count the number of plane trees of size, say 100, use the `count` procedure of the CS package:

```
>> cs::count([T,specPT],size=100);

         227508830794229349661819540395688853956041682601541047340
```

We can verify that the first coefficients of the generating series correspond to the first Catalan numbers:

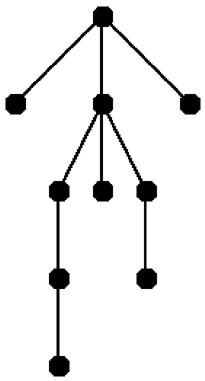
```
>> cs::count([T,specPT],size=i)$i=1..10;

           1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862
```

Now let us draw a random plane tree with 10 vertices:

```
>> cs::draw([T,specPT],size=10);

Prod(Z, Sequence(Prod(Z, Epsilon), Prod(Z, Sequence(

   Prod(Z, Sequence(Prod(Z, Sequence(Prod(Z, Epsilon))))), Prod(Z, Epsilon)

   , Prod(Z, Sequence(Prod(Z, Epsilon))))), Prod(Z, Epsilon)))
```

This description corresponds to the following tree:

Remark: Such a figure can be drawn almost directly from the output of the `draw` procedure by using CalICo [3] or CGraph [1], two programs dedicated to the manipulation and visualization of combinatorial structures. This figure was drawn with CalICo.



Here is another example of a specification, using an argument of cardinality; `P` represents the integer partitions and `N` the integers greater or equal to 1:

```
>> specPART:={P=Set(N),N=Sequence(Z,card>=1)};

                 {N = Sequence(Z, 1 <= card), P = Set(N)}

>> cs::draw([P,specPART],size=5);

Prod(Sequence(Z), Sequence(Z), Sequence(Z), Prod(Sequence(Z, Z), Epsilon))
```

The `count` and `draw` procedures both call the `compile` procedure. This last one produces, given the specification, the recurrences which will be used in order to count and generate the structures. The variable `HOLONOMIC`, when switched to the value `TRUE`, tells CS to test, when compiling a class of structures, if its generating function is holonomic or not. If yes, CS finds a simple linear recurrence to count the structures, so that the counting runs more quickly. (The default value of `HOLONOMIC` is `FALSE`). To illustrate this, compare the time needed to generate Motzkin trees of size 100, when compiling with `HOLONOMIC` equal to `FALSE` or `TRUE` (on a Sun Sparc 10, CPU time in ms):

```
>> specMT:={M=Union(Z,Prod(Z,M),Prod(Z,M,M))};

                 {M = Union(Z, Prod(Z, M), Prod(Z, M, M))}


>> cs::HOLONOMIC:=FALSE:
>> st:=time(): cs::compile(specMT): time()-st;

                                    6330


>> st:=time(): cs::count([M,specMT],size=100); time()-st;

               2494785789912243786801425614600100304678115800

                                    36160
```

```
>> cs::HOLONOMIC:=TRUE:
>> st:=time(): cs::compile(specMT): time()-st;

                              19470


>> st:=time(): cs::count([M,specMT],size=100); time()-st;

              249478578991224378680142561460010030467811580

                              1160
```

In the second case, the call to `compile` is longer, due to the computation using Gröbner basis and Gaussian elimination; but once finished, the counting and random generation are much more efficient. Before compiling with `HOLONOMIC=TRUE`, you can call the *MuPAD* function `setuserinfo` in order to see the operations involved by the calculation of the linear recurrences:

```
>> cs::HOLONOMIC:=TRUE:
>> setuserinfo(Any,1):
>> cs::compile(specMT);
...
standard form is: , {T1 = Prod(Z, M), T3 = ProdPrd(M, M), T2 = Prod(Z, T3)\
, Z = Atom(Z), M = Union(Z, T1, T2)}
...
polynomials are: , [T1*(-1) + M*z, T3*(-1) + M^2, T2*(-1) + z*T3, M*(-1) +\
 z + T1 + T2]
auto reduces list of 4 polynomials
groebner basis is: , [M^2 + M*z^(-1)*(z + (-1)) + 1, T1 + M*z*(-1), T3 + M\
*z^(-1)*(z + (-1)) + 1, z + T2 + M*(z + (-1))]
perform (ordinary) Gaussian elimination
algeq is: , T1*(-1) + z*T1 + z^2 + T1^2
perform (ordinary) Gaussian elimination
diffeq is: , z*2 + T1(z)*(-1) + D(T1)(z)*(-1) + z*T1(z)*(-3) + z*D(T1)(z)*\
2 + z^2*D(T1)(z)*3
recurrence for , T1,  is: , u(n)*(-1) + n*u(n)*2 + u(n + (-1))*(-3) + (n +\
 (-1))*u(n + (-1))*3 + (n + 1)*u(n + 1)*(-1)
...
perform (ordinary) Gaussian elimination
algeq is: , T3*(-1) + z*T3*2 + z^2 + z^2*T3 + z^2*T3^2
perform (ordinary) Gaussian elimination
diffeq is: , T3(z)*2 + z*T3(z)*(-4) + z*D(T3)(z) + z^2*(-4) + z^2*T3(z)*(-\
2) + z^2*D(T3)(z)*(-3) + z^3*D(T3)(z)*(-1) + z^4*D(T3)(z)*3
recurrence for , T3,  is: , u(n)*2 + n*u(n) + u(n + (-1))*(-4) + u(n + (-2\
))*(-2) + (n + (-1))*u(n + (-1))*(-3) + (n + (-2))*u(n + (-2))*(-1) + (n +\
 (-3))*u(n + (-3))*3
...
perform (ordinary) Gaussian elimination
algeq is: , T2*(-1) + z*T2*2 + z^3 + z*T2^2 + z^2*T2
perform (ordinary) Gaussian elimination
diffeq is: , T2(z) + z*T2(z)*(-1) + z*D(T2)(z) + z^3*(-4) + z^2*T2(z)*(-1)\
 + z^3*T2(z)*(-3) + z^2*D(T2)(z)*(-3) + z^3*D(T2)(z)*(-1) + z^4*D(T2)(z)*3
recurrence for , T2,  is: , u(n) + n*u(n) + u(n + (-1))*(-1) + u(n + (-2))\
```

```
*(-1) + u(n + (-3))*(-3) + (n + (-1))*u(n + (-1))*(-3) + (n + (-2))*u(n + \
(-2))*(-1) + (n + (-3))*u(n + (-3))*3

                                    TRUE
```

Equations and Recurrences

The CS package provides some procedures in order to handle generating series. They implement (in *MuPAD*) some fonctionalities of the GFUN package [10]. These procedures are used in `compile` but can also be called independently.

The procedure `speciftoalgeq` returns an algebraic equation that the generating series associated to the given specification satisfies, provided that such an equation exists:

```
>> eq:=cs::speciftoalgeq(specMT,M);
                                              2
                              z - M + M z + M  z
```

(At first glance, this does not look like an equation, but this is to be read as $z - M + Mz + M^2z = 0$.)

The procedure `algeqtodiffeq` converts this equation into a differential equation, while `diffeqtorec` gives the recurrence on the coefficients involved by the differential equation:

```
>> deq:=cs::algeqtodiffeq(eq,M(z));
                                                2              3
       2 z - M(z) + z M(z) - z D(M)(z) + 2 z  D(M)(z) + 3 z  D(M)(z)

>> r:=cs::diffeqtorec(deq, M(z), v(n));

    v(n - 1) - v(n) - n v(n) + 2 (n - 1) v(n - 1) + 3 (n - 2) v(n - 2)
```

We can resume all these steps in one unique call:

```
>> r:=cs::speciftorec(specMT,M,v(n));

    v(n - 1) - v(n) - n v(n) + 2 (n - 1) v(n - 1) + 3 (n - 2) v(n - 2)
```

The procedure `rectoproc` outputs a *MuPAD* procedure that gives the n-th term of the linear recurrence given as input. The `cs::LIM` variable is set to the rank from which the recurrence is valid.

```
>> m:=cs::rectoproc(r, v(n));
proc(n)
  name m;
  local nmax, j, g, ff;
begin
  nmax:=max(op(map([op(op(level(procname), 5))], op, 1)));
  for j from nmax + 1 to n do
    ff:=level(procname);
```

```
    g:=procname(j);
    evalassign(g, (j*(-1) + (-1))^(-1)*(ff(j + (-1))*(j*2 + (-1)) + ff(j +\
 (-2))*(j*3 + (-6)))*(-1), 1)
  end_for
end_proc

>> cs::LIM;
                                     2

>> m(0):=0: m(1):=1: m(2):=1:
>> m(100);
               2494785789912243786801425614600100304678115 80
```

Generation of C code

The Schröder trees are taken as example in this section. Each internal node of theses trees has at least two subtrees. Their enumeration according to their number of leaves leads to the Schröder numbers. Here all the nodes are counted:

```
>> specST:={S=Union(Z,Prod(Z,Sequence(S,card>=2)))}:
```

The `compile` procedure allows additional arguments in order to create the source file of a C program that allows to generate more quickly one or several structures of a given size. This program can be compiled and run either from or out of *MuPAD*. The following example shows how to process the first possibility.

```
>> cs::compile(specST,target=C,file="SchroederTree.c",main=S);

                                   TRUE

>> system("cc -Dsun4 SchroederTree.c -lm; a.out 5");
Prod(Z,Sequence(Z,Z,Z,Z))

                                     1
```

The binary file `a.out` can take two arguments. The first is the size of the structure (n) and the second is the number of structures you want to generate. If this second argument is not present, the program generates only one structure, and if it is equal to 0, it gives the approximate number of structures of size n:

```
>> system("a.out 1000 0");
5.1077756867799034*10^471..5.1077756867845379*10^471

                                     1
```

In fact, the result consists in an interval; we can guarantee that the exact number of structures lies in this interval, because the program does respect the IEEE specifications concerning floating-point calculations. This can be easily verified for our example:

```
>> cs::count([S,specST],size=1000);
```

```
51077756867821111314107471883520487646459962939615212132140562283586531550\
85712531917972151855853338148428884854040708738499122927307310995223206972\
46647726596273843155464894031285558286190795796393695165581530434292537730\
59361751753258007702832280113282117601364859322363650398274479548733624782\
45018768527999752095954909173205985617348262129533441008570673176572255375\
47121689339885351209588801649129568970987115615063798368477104478098170599\
5549956379942013897484633052

>> length(%);
                                        472
```

The C program allows to manipulate very large numbers (see above). Normally, it can be compiled on all machines. Presently, the supported architectures are SunOs (-Dsunos), Solaris (-Dsun4), DecAlpha (-Dalpha), IRIX (-DIRIX64), HP (-Dhp700).

The following table compares the CPU time (in seconds on an IRIX machine) needed for all computations. The two columns "count" give the time required for the preprocessing, without or with linear recurrences, while the column "draw" gives the average time for one random generation (over 100 generations, except for the size 1000000 for which just one generation was processed).

n	count (without rec.)	count (with rec.)	draw
100	0.037	0.007	0.002
1000	4.158	0.036	0.027
10000	499.210	0.358	0.346
100000	–	4.054	4.364
1000000	–	45.219	57.352

In the current version of CS, the C program only performs *almost uniform* random generation, because the coefficients used in the generation procedures are computed in floating-point arithmetic. (However, since the procedures are numerically stable, the rounding errors involved by floating-point arithmetic do not strongly affect the quasi-uniformity of generation; for details, see [4]). A future version of CS will allow to produce exactly uniform generators in C language (see Section).

Future Features

The CS package is presently at the beginning of its development (the version number, at the time of submission, is 0.91). Our aim is to add, within the next months, several new features. Here are some of them.

- A future version of CS will handle labelled structures as well as unlabelled ones.
- One of the authors has developed a Maple package, named qAlGO, mainly based on the notion of object grammars [5], and that leads to enumeration and random generation of objects according to non algebraic parameters. The packages qAlGO and COMBSTRUCT complement each other. We plan to integrate also the qAlGO technics in CS.
- Recently, two of the authors investigated a method, based on a strategy of "lazy evaluation", which fairly improves the average complexity of the exactly uniform random generation of decomposable structures [4]. We plan to use this method in CS in order to get more efficient generation procedures.

- In the current version of CS, the C programs written by the `draw` procedure only perform almost uniform random generation. We intend to use a multiprecision mathematical library, like PARI [9] or GMP [8], so that CS can write true random generators in C language.

References

[1] FRANÇOIS BERTAULT, *Génération et tracé de structures décomposables*, Université Henri Poincaré Nancy 1, 1997.

[2] M. DELEST AND J.M. FÉDOU AND N. ROUILLON, *Human Interaction for Symbolic Computation*, Computation and Images in Combinatorics, To appear. Texts and Monographs in Symbolic Computation, 1996.

[3] M. DELEST AND J.M. FÉDOU AND G. MELANÇON AND N. ROUILLON *Computation and images in Combinatorics*, RIACA Amsterdam, HISC book, Springer Verlag, To appear.

[4] A. DENISE AND P. ZIMMERMANN, *Uniform Random Generation of Decomposable Structures Using Floating-Point Arithmetic*, INRIA, 1997.

[5] I. DUTOUR AND J.M. FÉDOU, *Object grammars and random generation*, LaBRI, Université Bordeaux I, 1997. To appear in Discrete Mathematics and Theoretical Computer Science.

[6] P. FLAJOLET AND P. ZIMMERMANN AND VAN CUTSEM, B., *A Calculus for the Random Generation of Labelled Combinatorial Structures*, Theoretical Computer Science, 1994.

[7] P. FLAJOLET AND P. ZIMMERMANN AND VAN CUTSEM, B., *A Calculus of Random Generation: Unlabelled Structures*, In preparation.

[8] TORBJÖRN GRANLUND, *GNU MP: The GNU Multiple Precision Arithmetic Library*, 1996.

[9] C. BATUT AND D. BERNARDI AND H. COHEN AND M. OLIVIER, *User's Guide to PARI-GP*, 1995. `ftp://megrez.math.u-bordeaux.fr/pub/pari`.

[10] B. SALVY AND P. ZIMMERMANN, *Gfun: A Maple Package for the Manipulation of Generating and Holonomic Functions in One Variable*, toms, 1994.

[11] P. ZIMMERMANN, *Gaffa: a package for the random generation of combinatorial structures*, MapleTech, 1994.

Ten Consecutive Primes in Arithmetic Progression

Paul Zimmermann
LORIA, Nancy, zimmerma@loria.fr

This story begins in September 1997. Two colleagues from Lyon, Nik Lygeros and Michel Mizony, invited me at that time to give a talk about the state of the art in factoring polynomials. During my short visit in Lyon — just one day —, they told me about a number theory problem they were working on:

$$1181 = 6 \cdot 194 + 17, 1187 = 6 \cdot 195 + 17, 1193 = 6 \cdot 196 + 17.$$

In the above, the three numbers on the left (1181, 1187 and 1193) are primes, with a common difference of 6: we say these are primes in *arithmetic progression*. In addition, there is no other prime inbetween: we say these are *consecutive* primes in arithmetic progression. The original problem had one more constraint, namely that the Euclidean division of each prime by its rank — 1181 is the 194th prime, so its rank is 194 — should give the same remainder. In other words, we were looking for consecutive primes in arithmetic progression such that the integer quotient of each prime by its rank equals the common difference — here 6.

During the afternoon while I was in Lyon, we played with MAPLE and PARI, and found several new occurrences of three and four primes in arithmetic progression answering to the original problem. Then I came back to Nancy, improved the PARI program, and found one solution with five primes, namely

$$30109571823599,\ 30109571823629,\ 30109571823659,\ 30109571823689,\ 30109571823719$$

with a common difference of 30. After a few days, we noticed that the original problem was not much harder than finding consecutive primes in arithmetic progression. Indeed, if $p, p + a, p + 2a, \ldots, p + (k - 1)a$ is a sequence of k consecutive primes in arithmetic progression, we just need to check that $a \leq q/\pi(q) < a + 1$ for any q in that sequence, where $\pi(q)$ is the number of primes less or equal to q (i.e. q is the $\pi(q)$-th prime). Using the following inequalities from Dusart [2],

$$k\left(\ln k + \ln\ln k - 1 + \frac{\ln\ln k - 2.25}{\ln k}\right) \leq p_k \leq k\left(\ln k + \ln\ln k - 1 + \frac{\ln\ln k - 1.8}{\ln k}\right) \text{ for } k \geq 27076,$$

where p_k denotes the k-th prime, one gets some rather large windows, for example

$$4.36 \cdot 10^{91} \leq q \leq 11.8 \cdot 10^{91} \tag{1}$$

for $a = 210$. This means that any sequence of consecutive primes in arithmetic progression in that range will also be a solution of the original problem.

Looking in the literature, we found that the problem of finding consecutive primes in arithmetic progression had already been studied. This is in fact the subject of a whole chapter (A6) in the 2nd edition of the book *Unsolved Problems in Number Theory*, from Richard K. Guy [3]. While searching on the net, we also found that a new record had been set by Harvey Dubner and Harry Nelson, who found seven consecutive primes in arithmetic progression in August 1995,[1] and that the corresponding article was in print (it has been published now, see [1]). Those seven primes were 97 digits long. The key idea was to search the first prime of the form

$$p = x + N \cdot m \tag{2}$$

[1] http://felix.unife.it/Root/d-Mathematics/d-Number-theory/t-7-primes-in-progression

where x and m are fixed, and N goes from 0 to ∞. In practice, m is the product of small primes, and x is "well-chosen" using a method invented by Nelson in 1975 [4].

We then contacted Harvey Dubner, who contacted in turn Tony Forbes who had designed a very efficient program for PCs under Windows to search for special patterns of primes. In a few days, Tony adapted his program to search for consecutive primes in arithmetic progression. In the same time, I designed a program for Unix machines, using the free and portable multiprecision library GMP.[2] As our programs were about 6 to 12 times faster than the UBASIC program Harvey Dubner and Harry Nelson used in 1995, it became clear to us that setting a new record was possible. This was in the middle of October. After about two weeks, I found with the GMP program a sequence of eight consecutive primes in arithmetic progression, namely $p, p+210, \ldots, p+7\cdot 210$ where p is the following 92-digit number:

$$43804034644029893325717710709965599930101479007432825862362446333961919524977985103251510661.$$

As it satisfies Eq. (1), it was also a solution to the original problem.

Not happy with that new record, we then decided (Harvey Dubner, Tony Forbes, Nik Lygeros, Michel Mizony, and me) to go for nine primes. As the estimated time was about 20 times larger than for eight primes, we asked for help in the scientific community, mainly with machines cycles. About hundred people answered us positively, with about two hundred machines, half Unix machines and half PCs. Some members of the *MuPAD* group helped too: Gerald (alias Jerry) Ruescher, Torsten Metzner and Stefan Wehmeier. On January 15th, just at the time we were becoming discouraged about finding a new record, a "helper" from Klagenfurt in Austria, Manfred Toplic, found nine consecutive primes in arithmetic progression, namely $p, p+210, \ldots, p+8\cdot 210$ where p is the following 92-digit number:

$$99679432066701086484490653695853561638982364080991618395774048585529071475461114799677694651.$$

This also was a solution to the original problem.

Not happy either with that 9-primes record, we decided once more to go to ten primes. I was personally a bit suspicious about this enterprise, since the total estimated cpu time was about 300 years. But when people from Distributed Net[3] told us they would be ready to help us in April or March, and it would take them only a few days, I was confident the project was doable. We then decided to start it without waiting for Distributed Net, with all the helpers from the 9-primes project. The 10-primes project therefore started in the middle of January. On March 3rd, we were at only about 3% of the total expected work, when we got an email from Manfred Toplic (yes, again him!) telling us he had found ten consecutive primes in arithmetic progression! We were very surprised, firstly because we did not expect to find a solution so quickly, and secondly because Manfred had already found the 9-primes record. The ten consecutive primes in arithmetic progression are $p, p+210, \ldots, p+9\cdot 210$ where p is the following 93-digit number:

$$100996972469714247637786655587969840329509324689190041803603417758904341703348882159067229719.$$

Here again, this is also a solution to the original problem.

Now you should think: why not eleven primes ? This is in fact much harder. The reason is that, up to ten primes, one can have a common difference of 210 between primes, whereas for eleven primes, the common difference must be at least 2310. The optimal size of the primes to search for is then about 600 digits instead of 90 digits for 7 to 10 primes. As a consequence, the expected search time is about 10^{13} times larger than for 10 primes !

[2] http://www.matematik.su.se/ tege/gmp
[3] http://www.distributed.net

Of course, this is only an estimation on average. It does not prevent lucky people like Manfred Toplic to find a 11-primes record in a few weeks !

This is the end of that story. In retrospect, it has been for me a very exciting and interesting experience. It has been exciting to work together — by exchanging several emails a day — with people like Harvey Dubner and Tony Forbes I never met before (and I still did not encounter them physically). It has been exciting to get each day several results from helpers, expecting one of them had found a new record. It has been exciting to discuss on the net with helpers, who I don't know either, to learn and to explain them details of number theory. It has been very interesting to discover how powerful the "Internet machine" can be, even for a project managed by hand like ours. All this pushed me to start a new project, called ECMNET.[4] But this is another story.

Acknowledgements. We used a *MuPAD* program implementing Nelson's method to determine "good" values of the parameter x in Eq. (2), and we thank the *MuPAD* team for making this software available. We thank also Torbjörn Granlund for making GMP freely available, on which the Unix version of the search program was based, and François Morain who checked the primality of the 9-primes and 10-primes solutions using his ECPP program.[5] (Our search program just found probable primes.) The PARI program, freely available too,[6] was also very useful for our first experiments; by the way, *MuPAD* uses PARI for its arithmetic.

References

[1] DUBNER, H., AND NELSON, H. Seven consecutive primes in arithmetic progression. *Mathematics of Computation* (Oct. 1997), 1743–1740.

[2] DUSART, P. Le k^{e} nombre premier est plus grand que $k(\ln k + \ln\ln k - 1)$ pour $k \geq 2$, 1997.

[3] GUY, R. K. *Unsolved Problems in Number Theory*, 2nd ed. Springer-Verlag, 1994.

[4] NELSON, H. L. There is a better sequence. *J. Recreational Math. 8*, 1 (1975), 39–43.

[4] http://www.loria.fr/ zimmerma/records/ecmnet.html
[5] http://www.lix.polytechnique.fr/ morain/Prgms/ecpp.english.html
[6] ftp://megrez.math.u-bordeaux.fr/pub/pari

News about Macro Parallelism in *MuPAD* 1.4

Andreas Sorgatz, Torsten Metzner et al
University of Paderborn, {andi,tom}@mupad.de

MuPAD macro parallelism [7] is based on the concepts of message passing *and* network *(*global*)* variables*. This article describes the user interface as well as the implementation of the macro parallelism in MuPAD 1.4 and gives examples for parallel algorithms.*

Introduction

The concept of macro parallelism follows the *master/worker* paradigm, where the first interactive cluster (sequential or parallel *MuPAD* kernel) of the network is the master. While the master is connected to a terminal, slaves use a so-called *work* queue to successively fetch and evaluate tasks (figure 3).

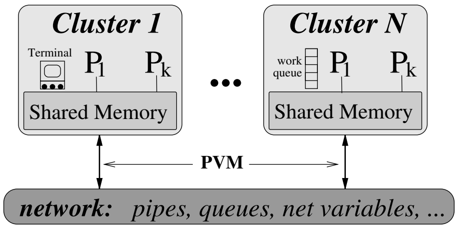


Figure 3: Macro Parallelism in *MuPAD*

MuPAD data -i.e. objects, programs and tasks- can be distributed by writing to and reading from so-called *pipes* and *queues*. Writing can be done locally as well as to remote pipes and queues, while reading is a strictly local operation. Queues may receive data from each cluster of the network, whereas pipes are always one-way connections between two specific clusters (figure 4). Neither the number of pipes and queues nor their naming conventions impose any restrictions to the user. The concept makes it easy to place tasks, input data and results on any cluster of the network.

Furthermore, sharing data between all clusters of the network can be done by use of so-called *global* (*network*) *variables*. In contrast to message passing, this is a more convenient method to distribute and share information. Write access to global variables is realized as an *atomic* operation (see function `global`). Thus global variables can be used to synchronize tasks over the network and to introduce *locks* and *semaphores*. Figure 4 gives a brief demonstration how to use macro parallelism in *MuPAD*.

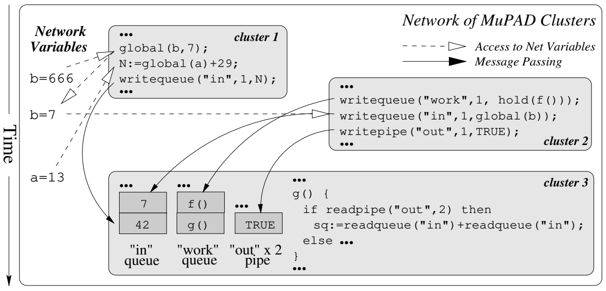


Figure 4: Using *MuPAD* Macro Parallelism

Quick Start

The macro parallelism is provided by the dynamic module [12], [11], [10], [9] `net`. For the following examples, the module `net` is expected to be installed with *MuPAD* Release 1.4.0. To use a network of $4+1$ *MuPAD* clusters, the user inserts -for example- the following lines into the personal *MuPAD* startup file `~/.mupadinit`.

```
NETCONF:= [ "wiwianka"=2, "gauss"=1, "poisson"=1 ]:
module( net ):
```

Calling *MuPAD* in the form `mupad -U master` respectively `xmupad -U master` starts a local kernel as the interactive master, two remote kernels on the host `wiwianka` and one kernel on each of both `gauss` and `poisson`. The following commands give a first example on using the macro parallelism:

```
>> topology();                              // get number of clusters available
   5
>> net::ismaster();                         // is this cluster the master?
   TRUE
>> topology(Cluster);                       // determine the local cluster-id
   1
>> writequeue("work",3,hold(                // ask cluster #3 whether it is master
    writepipe(YesIam,1,net::ismaster()) )); // let it write into the pipe 'YesIam'
>> readpipe(YesIam,3,Block);                // read the answer from the pipe
   FALSE
>> global(a,42):                            // set a global variable
>> global(a,global(a)+1), global(a);        // increment a global variable
   42, 43
```

User Interface

The central features of the macro parallelism are message passing and the access to global variables. Beside this, it provides methods to interactively configure, start, shutdown and restart the network as well as to set network preferences. This allows users to (re-)configure the network at run-time according to their need. This may also be controlled by a *MuPAD* program, e.g. a modular algorithm.

Message Passing

In the following description, `name` is a *MuPAD* object naming a queue or pipe, `cid` is a cluster identification number (integer) and `Block` is an option (identifier).

- `net::readqueue(name [,Block])`: reads from the local queue `name`. `Block` effects a blocking read. Otherwise the object of type `DOM_NULL` is returned if the queue with name `name` is empty.
- `net::writequeue(name, cid, value)`: writes `value` into the queue `name` of cluster `cid`.
- `net::readpipe(name, cid [,Block])`: reads from the local pipe `name`×`cid`. `Block` effects a blocking read. Otherwise the object of type `DOM_NULL` is returned if the pipe with name `name` is empty.
- `net::writepipe(name, cid, value)`: writes `value` into the pipe `name` of cluster `cid`.

Global Variables

Below, `var` is -respectively evaluates to- an identifier. `level` is an integer defining the substitution level.

- `net::global( var )`: reads out the global variable `var`.
- `net::global( var, value )`: assigns `value` to the global variable `var`. In contrast to `value`, `var` is not evaluated. During the evaluation of `value`, `var` cannot be changed by others. `global` returns the previous value of `var` respectively the object of type `DOM_NULL` if `var` was not assigned a value before.
- `net::globale( level, var [, value] )`: like `global`, but evaluates `var` with level `level`.

Locks and Semaphores

Since writing to globals is an atomic operation, it can easily be used to implement *locks* and *semaphores*.

- `net::initlock( var )`: initializes the lock variable `var`.
- `net::setlock( var )`: tries to set the lock `var` and returns `TRUE` if it was successful.
- `net::spinlock( var )`: waits (busy waiting) until it is allowed to set the lock variable `var`.
- `net::unsetlock( var )`: resets the lock variable `var`.
- `net::initsem( var )`: including `net::spinsem`, `net::acquiresem` and `net::freesem`.

Topology Information

Network topology information are provided by the following functions. `Cluster` is an option (identifier).

- `net::topology([Cluster])`: returns the number of clusters currently available respectively the local cluster identification number, if option `Cluster` is used.
- `net::ismaster()`: returns `TRUE` on the master of macro parallelism and `FALSE` on a slave cluster.

Network Configuration

The network configuration specifies which hosts and how many clusters are part of the macro parallelism. It is either required before the module `net` is loaded into a *MuPAD* session for the first time -to start the network automatically- or before the function `net::master` (see next section) is called.

- `NETCONF:=["`$\texttt{host}_1$`-`$\texttt{clusters}_1$`,...]`: configures the network. $\texttt{host}_k$ is a hostname and $\texttt{clusters}_k$ specifies the number of clusters to be started on this host.
- `net::pref((LogError|ComType)[=value])`: gets and sets network preferences.

- ∘ `LogError` is `TRUE` (default on master) or `FALSE`. It controls whether the master informs the user about errors which occured on any slave cluster of the macro parallel network.
- ∘ `ComType` is `"MCODE"` (default) or `"ASCII"`. It controls whether *MuPAD* data are transferred over the network as text or in a *MuPAD* specific binary format called MCODE.

Startup and Shutdown

At any time in a *MuPAD* session the network can be shutdown and restarted with a new configuration.

- `net::master()`: starts the network. It requires a network configuration specified in the *MuPAD* variable `NETCONF` and returns `TRUE` if the network was started. Otherwise it returns `FALSE`.
- `net::shutdown()`: shuts down the network and returns `TRUE`. The master simulates a macro parallel network of topology 1, if the module `net` is loaded but no network is running.
- `net::halt()`: halts the network instantly and quits the master. This can be used as emergency exit.

Other Functions

- `net::map( list, func [,arg`$_1$` ...] )`: applies function `func` to each element of `list` and returns the list of results. The results are computed in parallel.

Using these basic set of routines, macro parallelism can be extended with user library functions to make programming even more convenient. See section *Reflection and Outlook* for examples and future plans. A detailed description of all functions is available as online documentation when the module `net` is loaded. Refer to `net:doc()` respectively `net::doc("`*`func`*`")`.

Currently Implemented Algorithms

Starting to write macro parallel algorithms, we first concentrated on typical mathematical problem classes like a parallel sieve algorithm for factoring integers, a modular algorithm for computing greatest common divisors over algebraic number fields and a Gröbner Basis algorithm. This testsuite was (and still is) used to check the correctness as well as the performance of the implementation of macro parallelism.

Factoring Integers by Means of a Quadratic Sieve

MuPAD's library package on number theory `NUMLIB` contains an implementation of the so-called *multi polynomial quadratic sieve* [8] for factoring integers. The key step consists in finding out which values of certain polynomials are *smooth*, i.e. factor completely into small primes. Each polynomial is of the form $ax^2 + bx + c$, with the coefficients a, b, c depending on a certain parameter d. In order to sieve such a polynomial, the *MuPAD* procedure `numlib::sievepoly` has to be called with the list `[a,b,c,d]`, the integer `n` to be factored, the number of values to examine `M`, the factor base `fb`, the product of all elements of the factorbase `fbprod`, and two parameters `T` and `largefactorbound` that define which prime factors are considered as "small".

Since the polynomials can be investigated independently from each other, writing a parallel version of the sieve is now trivial. With `abcdlist =` $[[a_1, b_1, c_1, d_1], \ldots, [a_k, b_k, c_k, d_k]]$ being a list of parameter lists defining k polynomials, the *MuPAD* syntax of the sieve step is simply

```
>> net::map( abcdlist, numlib::sievepoly, n, M, fb, fbprod, T, largefactorbound );
```

The result is a list of length k. Its i-th entry is the list of smooth values of $a_i x^2 + b_i x + c$. The algorithm stops sieving if the number of sieve reports obtained is too small for factoring n.

Although it is impossible to compete with `C` implementations of this algorithm, the running time of `mpqs` compares well to `ifactor`, the implementation of the elliptic curve method in the *MuPAD* kernel. Compared to a sequential version of `mpqs`, the speedup is roughly proportional to the number of clusters (as one should expect). Using this implementation on eight clusters, it is possible to factor numbers up to about 35 digits in roughly 10–15 minutes.

Gcd over Algebraic Number Fields

In [5], Langemyr and McCallum propose a modular algorithm for computing the greatest common divisor of two polynomials $f, g \in (\mathbb{Q}[x]/(r))[y]$, $r \in \mathbb{Q}[x]$. The basic idea is to compute many gcd's of homomorphic images of f and g in residue class rings $F_p[x]/(\bar{r})$, where $\bar{r} \in F_p[x]$ is obtained from r by reducing all coefficients modulo p. (This requires replacing r with a polynomial over $\mathbb{Z}$ first, as well as removing all denominators from f and g.) Each time after processing a prime, the results are put together by means of the Chinese remainder algorithm.

Unlike in the case of the sieve algorithm, it is not determined in advance how many p will be necessary (the theoretical upper bound is too conservative). The algorithm proceeds until a trial division of both inputs by the result succeeds; such trial division is attempted whenever considering a new prime does not change the previously obtained result.

The implementation uses a procedure `gcdmodp` which computes the modular $\gcd$ of f and g for a given input p. The huge polynomials f and g as well as r are not passed as parameters, but stored in global variables, which means that they have to be broadcast over the net only once. The *MuPAD* source code of the Langemyr algorithm looks like this:

```
// Do some initializations first, e.g. make f, g and r global ///////
// Some Initializations, make f, g and r global ////////////////////
[...]
net::global(fff, f);  net::global(ggg, g);  net::global(rrr, r);

// Another way to make some object known everywhere: ////////////////
for j from 2 to topology() do
   writequeue("work", j, hold(_assign)(hold(gcdmodp), gcdmodp))
end_for;

// First, get all clusters to work /////////////////////////////////
for j from 2 to topology() do
   repeat  lastp:= nextprime(lastp+1)
   until   [...] // until lastp is a "good" prime in some sense /////
   end_repeat;

   // Let cluster 'j' send the master the following information: ////
   //      [the modular gcd mod lastp, lastp, its cluster-id 'j'] ////
   writequeue("work", j, hold(writequeue)("resq",1,[hold(gcdmodp)(lastp),lastp,j]));
end_for;

while TRUE do
   repeat lastp:= nextprime(lastp+1) until [...] # lastp is "good" # end_repeat;

   // Wait for a cluster which can compute this new task 'p' ////////
   result:= readqueue("resq", Block);
   p:= result[2];   j:= result[3];   r:= result[1];
   writequeue("work", j, hold(writequeue)("resq",1,[hold(gcdmodp)(lastp),lastp,j]));
```

```
    // Combine the result 'r' with the previously received results. //
    // For this, relatively simple Chinese remainder algorithms are //
    // carried out by the master.
    [...]

    // Then test for termination, return if it succeeds  ////////////
    [...]
end_while;
```

The above algorithm showed nearly full speed up, compared to a sequential version. Using a network of one UltraSparc-4, two UltraSparc-II and three UltraSparc-I, the gcd of two polynomials of degree 110 over $\mathbb{Q}(i)$ (with real and imaginary part of each coefficient between -10^{12} and 10^{12}) was found in less than one minute real time. The degree of the gcd in our example was 10.

Karatsuba's Method for Multiplying Polynomials

Using a well-known trick of Karatsuba and Ofman ([4]), it is possible to reduce one polynomial multiplication to three multiplications of polynomials of half of the degree of the original polynomials (plus some additions). We used this as an example for an algorithm that involves a lot of recursive calls. Our parallel implementation works as follows: whenever a polynomial is split, the multiplications to be done are appended to a global list of problems, together with the number of the cluster that is waiting for the result. While waiting for a result, every cluster including the master takes jobs from this list of problems; in addition, all slaves are told to read permanently from this list until the computation ends. Since taking a task just means redistributing three smaller tasks and taking further ones while waiting for the result, the length of the polynomials in the problem list decreases quickly, while any cluster is waiting for a cascade of results. Polynomials of degree smaller than a certain limit are not distributed further, but handled by a sequential version of the Karazuba algorithm.

This approach causes a lot of tasks and results to be communicated on the network. It is especially useful as a benchmark for communication and the efficiency of accesses to the global variable `problemheap`, which frequently occur. The code of the main program `karazmult` looks like this:

```
karazmult:=proc( f, g )
local i, res;
begin
    net::global(problemheap, []); // start with an empty list of problems
    for i from 2 to topology() do
        // Broadcast some procedure definitions
        writequeue("work", i, hold(_assign)(hold(karazuba), karazuba));
        writequeue("work", i, hold(_assign)(hold(workuntilfinish), workuntilfinish));
        // Cause cluster 'i' to permanently take tasks from problemheap
        makework(i);
    end_for;

    // Start the computation on the master
    res:=karazuba([coeff(f,i)$hold(i)=0..degree(f)],[coeff(g,i)$hold(i)=0..degree(g)]);
    // Tell the slaves to finish polling on problemheap
    for i from 2 to topology() do dismiss(i) end_for;
    // Return the result, converted to polynomial
    poly(zip(res,[$0..nops(res)-1],DOM_LIST),op(f,2..3))
end_proc:
```

The polling on the global `problemheap` (executed by the slave worker clusters) is implemented as follows:

```
makework:= proc( clusternr )
```

```
begin  writequeue("work", clusternr, hold(fp::apply)(workuntilfinish))
end_proc:

workuntilfinish:= proc()
local  dotasklocal;
begin  dotasklocal:=net::global(dotask);
       while readqueue("finish")<> TRUE do dotasklocal(); end_while;
end_proc:

dismiss:= proc( clusternr )
begin  writequeue("finish", clusternr, TRUE)
end_proc:

net::global(dotask, proc()
local task, localcopy;
begin
   if nops(net::global(problemheap))>0 then
      net::global(problemheap, ( // atomic access to problemheap
         localcopy:=net::global(problemheap);
         if nops(localcopy)=0 then [] // problemheap is empty
         else task:=localcopy[1];
              [op(localcopy, 2..nops(localcopy))]
         end_if)
      ); // other clusters are allowed to write to problemheap again
      if task <> hold(task) then
         op(task,1);
         writequeue(op(task,3), op(task,2), eval(%)) ;
      end_if
   end_if
end_proc):
```

The procedure `makework` shows how to make a cluster execute a procedure without assigning it to a local variable of the cluster: it may be passed as an argument to `fp::apply`, which is globally known because it is part of the *MuPAD* library. This style is not recommended if a procedure is to be executed by any slave (and therefore sent to it) more than once.

Now, the multiplication algorithm `karazuba` itself reads as follows:

```
karazuba:=proc( f, g )                          // input : coefficient lists   f,g
begin                                           // output: coefficient list of f*g
   resqu:=genident("result");
   if   nops(f)<BOUND1 or nops(g)<BOUND1 then [...] // classical multiplication
   elif nops(f)<BOUND2 or nops(g)<BOUND2 then [...] // sequential Karazuba
   else // parallel Karazuba algorithm
      n:=max(nops(f), nops(g)) div 2;               // splitting point
      if   nops(f)<=n then [...] // f is short, subdivide g into two parts
      elif nops(g)<=n then [...] // g is short, subdivide f into two parts
      else // Split both f and g
           a:=[op(f,n+1..nops(f))];  b:=[op(f,1..n)];
           c:=[op(g,n+1..nops(g))];  d:=[op(g,1..n)];
           // Write the three necessary multiplications to the problemheap
           zip(a,b,_plus,0);  zip(c,d,_plus,0);

           // A problem (task of the problemheap) is specfied via a list:
           //  - the function call to evaluate
           //  - the integer 1, 2 or 3 to distinguish the three products
           //  - the cluster who wants the result (this cluster)
```

```
            //  - the queue where to write the results

            net::global(problemheap, append(net::global(problemheap),
               [[hold(karazuba)(%, %2), 1], topology(Cluster), resqu]));
            net::global(problemheap, append(net::global(problemheap),
               [[hold(karazuba)(a,  c), 2], topology(Cluster), resqu]));
            net::global(problemheap, append(net::global(problemheap),
               [[hold(karazuba)(b,  d), 3], topology(Cluster), resqu]));

            // Read incoming results; if there are none, do useful work in meantime
            missing:=3;
            while missing > 0 do
               res:=readqueue(resqu);
               if res<>null() then
                  p.(op(res,2)):= op(res,1);
                  missing:=missing-1;
               else
                  net::global(dotask)()
               end_if
            end_while;
            // Some additions of polynomials (carried out locally)
            [...]
         end_if
      end_if
end_proc:
```

Since this algorithm was chiefly written in order to test this kind of programming paradigm, we could not expect it to be very fast. At least we managed to beat the multiplication of the (sequential) *MuPAD* kernel: using the same configuration as above (six UltraSparc ranging from UltraSparc-I to UltraSparc-4), two dense univariate integral polynomials of degree 2000 each were multiplied in 40 seconds real time (most of it communication time for accessing the global `problemheap`), while the multiplication on a single Ultra-4 took 107 seconds. Implementing global work queues directly in C/C++ in the module `net` would mean a major improvement here.

Gröbner Basis Computation

A very first version of a Gröbner algorithm is implemented. The algorithm used is the second version of the parallel Buchberger algorithm described in [2] (using additional search parallelism). At the copy deadline the algorithms was just finished. Speedups seems to be rather poor, but no significant tests were performed at present.

About the Implementation

Technical Notes

The current implementation is tested with *MuPAD* Release 1.4.0 (sequential version) in a homogeneous network of Sun workstation using SunOS 5.5/5.6.

Since the binary code of the macro parallelism is relatively large ($>$ 350Kb), it was not statically integrated into the *MuPAD* kernel but implemented as a dynamic module [10], which can be *loaded on demand*. For this, the *MuPAD* kernel 1.4.0 itself was not changed in any sense. The new functionality is fully provided by the dynamic modules `net`.

As transportation layer PVM [1] is used, since it is portable, easy to use and supports us in managing the clusters. To use PVM in the context of a dynamic module, it was compiled with the GNU g++ compiler with option `-fpic` to create an library archive of position independent object code. Due to security reasons, we use `ssh` instead of `rsh` for spawning `PVM` tasks.

With the current version of the module `net`, *MuPAD* objects can be transferred either ASCII or binary encoded (default), using a *MuPAD* specific binary format called MCODE. The user can change the encoding because in *MuPAD* 1.4.0, the MCODE cannot be used in a heterogeneous environment. On the other hand, *MuPAD* objects like `funs` have no text representation and cannot directly be sent ASCII encoded. Future MCODE versions will be architecture independent to avoid these problems.

Concept and Realization

Figure 5 shows a simplified scheme of the internal design of the dynamic module `net`. It is built upon the PVM library 3.4, which is used to start and manage a network of arbitrarily many *MuPAD* clusters and to send and receive network messages.

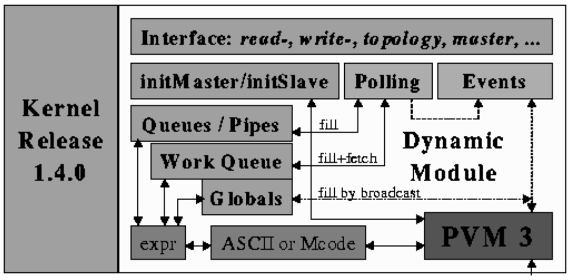


Figure 5: Internal Sceme of the Dynamic Module `net`

`Queues` are an internal data structure of the *MuPAD* kernel. They can be accessed by dynamic modules on a C/C++ language level and are used here to represent local network queues and pipes on each cluster. Remote writing to these queues is realized by PVM messages. Since the macro parallelism provides named queues and pipes, where a name may be any arbitrary *MuPAD* object, *hash* tables (`DOM_TABLE`) are used to associate these names with corresponding internal queues.

Global variables are represented as entries of local tables (`DOM_TABLE`). All values of global variables are held on each cluster, so that reading can be done locally, which is very fast. Writing a global variable is realized by a network *broadcast*, which updates the corresponding table entry on each cluster of the network. To guarantee that this transaction is an atomic operation, the master of macro parallelism works as a *permit* server for write access to global variables. The feature atomic operation is meant in the following sense: Writing to a global variable is a blocking operation which returns after all clusters have updated their local table entry. It prevents any other write access to this variable and ensures that in a call `global(varname,value)`, the expression `value` is evaluated in a protected context. Thus, `global(v,global(v)+1)` is a safe operation. In effect reading globals is cheap while writing is expensive.

Reflection and Outlook

The straight forward strategy to realize global variables was chosen because its implementation is more or less trivial and allows us to get first experiences with the macro parallelism and to write parallel algorithms at a very early state of the development. The performance is reasonable, if globals are used very deliberately. But they may become a terrible bottle-neck when used extensively. More efficient solutions are needed for the future.

The experiences we gained when implementing parallel algorithms showed that it would be useful to extend the user interface to make programming of parallel algorithms even more convenient. Currently missing features are:

- a user friendly concept to support the programmer of parallel algorithms in handling errors.
- functions to check the current status of a cluster's *worker status*: still working?, number of tasks in the work queue?, used memory?, available memory?, ...
- global work queues which may be accessed by each cluster of the network to fetch a new task, if its local work queue is empty. This eases programming and is also a trivial way of load-balancing. The global work queue may be realized as a *priority queue*, which supports trivial but better load-balancing. E.g. in some cases the priority may be specified by a *time consumption estimation* to support the *shortest processing time first* strategy, which is known as a good strategy for batch systems. Based on a *worker status* more intelligent load-balancing strategies could be realized via *MuPAD* library functions.
- support of broadcasts, e.g. to easily distribute procedures and other definitions to a given set of clusters.
- parallel debugging and tracing features, e.g. a macro parallel `userinfo`.

However, already the current implementation provides an easy way to develop and run distributed computations with *MuPAD*. In combination with the technique of integrating special purpose C/C++ algorithms for real world applications with the concept of dynamic modules, *MuPAD* offers an efficient and flexible parallel programming environment for mathematical algorithms.

The C/C++ code of the dynamic module `net` as well as the *MuPAD* source code of the algorithms listed here, will be published with the Manual *Dynamic Modules - User's Manual and Programming Guide for MuPAD 1.4* [12].

References

[1] AL GEIST ADAM BEGUELLIN JACK DONGARRA WEICHENG JIANG ROBERT MANCHEK AND VAIDY SUNDERAM. *PVM: Parallel Virtual Machine A User's Guide and Tutorial for Networked Parallel Computing*. MIT Press, 1994. Also available at: ftp:://ftp.netlib.org/pvm3/book/pvm-book.ps.

[2] B.AMRHEIN, O., AND W.KUECHLIN. A case study of multi-threaded groebner basis completion. In *Proceedings of the ISSAC, Zürich* (1996), L. Y. N., Ed., ACM Press, pp. 95–102.

[3] HECKLER, C., METZNER, T., AND ZIMMERMANN, P. Progress report on parallelism in *MuPAD*. Tech. Rep. tr-rsfb-97-042, SFB 376, Universität-Gesamthochschule Paderborn, june 1997. also published as Technical Report No. 3154 of INRIA, Lorraine, Villers-les-Nancy, France, and presented as a poster on ISSAC'97.

[4] KARATSUBA, A., AND OFMAN, Y. Multiplication of multidigit numbers by automata. *Soviet Physics-Doklady 7* (1963), 595–596.

[5] LANGEMYR, L., AND MCCALLUM, S. The computation of polynomial greatest common divisors over an algebraic number field. *J. Symbolic Computation 8* (1989), 429–448.

[6] METZNER, T. Computeralgebra auf heterogen verteilten Systemen. Diplomarbeit, Universität – GH Paderborn, Fachbereich Mathematik-Informatik, Dezember 1996.

[7] NAUNDORF, H. Parallelism in mupad. In *Electronic Proceedings of the 1st International IMACS Conference on Applications of Computer Algebra* (may 1995), M. Wester, S. Steinberg, and M. Jahn, Eds.

[8] SILVERMAN. The multiple polynomial quadratic sieve. In *Math.of Comp.* (1987), vol. 48, pp. 329–339.

[9] SORGATZ, A. *Dynamische Module – Eine Verwaltung für Maschinencode-Objekte zur Steigerung der Effizienz und Flexibilität von Computeralgebra-Systemen*, vol. 1 of *MuPAD Reports*. B.G.Teubner Stuttgart, Oct. 1996. ISBN 3-519-02195-1.

[10] SORGATZ, A. Dynamic modules - software integration in mupad. poster/abstract, ISSAC'97, Maui, USA, July 1997. `http://www.mupad.de/BIB/ONLINE/SORGATZ97b/`.

[11] SORGATZ, A. Dynamic modules - the concept of software integration in mupad. in session 13, Computer Algebra and Software Integration", IMACS'97, Maui, Hawaii, USA, July 1997. `http://www.mupad.de/BIB/ONLINE/sorgatz97c.ps.gz`.

[12] SORGATZ, A. *Dynamic Modules - User's Manual and Programming Guide for MuPAD 1.4*. Preprint: contact the author `andi@mupad.de`, Feb '98. includes source code.

[13] THE *MuPAD* GROUP, BENNO FUCHSSTEINER ET AL. *MuPAD User's Manual - MuPAD Version 1.2.2*, first ed. John Wiley and sons, Chichester, New York, March 1996. ISBN 0-471-96716-5.

Differentialgleichungen, Differentialgeometrie, Tensoren: neue Funktionen in der *MuPAD* -Library

Stefan Wehmeier
Universität GH Paderborn, stefanw@uni-paderborn.de

Während MuPAD schon seit längerem mit einem Differentialgleichungslöser aufwarten kann, fehlten bislang die entsprechenden Funktionen aus dem Bereich der Numerik ebenso wie passende Domains zur Repräsentation der bei solchen Problemstellungen auftretenden Objekte (wie etwa Differentialoperatoren) und zu differentialgeometrischen Problemstellungen (z.B. Oberflächen als Definitionsbereiche). Der vorliegende Artikel gibt einen Überblick, wie diese Lücken geschlossen wurden.

Neues zur symbolischen Lösung von Differentialgleichungen

Das Library-Paket `ODE` [1] und erste Versuche, Differentialgleichungen in MuPAD symbolisch zu lösen, gehen bis auf das Jahr 1995 zurück. Hauptsächlich tätig waren hier Paul Zimmermann sowie MuPAD's Kooperationspartner Winfried Fakler und Eckhard Pflügel. Nach der ausführliche Darstellung in der letzten Ausgabe (siehe [1]) sollen hier einige weitere Beispiele folgen.

Für Differentialgleichungen gibt es in MuPAD einen eigenen Datentyp: `ode`; eine Gleichung `eq`, in der Ableitungen vorkommen, muß zunächst in diesen Typ konvertiert werden, um weiter bearbeitet werden zu können. „Bearbeitet" heißt natürlich meistens „gelöst".

Ein einfaches Beispiel:

```
>> eq:=ode(y'(x)=y(x)/x^2,y(x)): solve(eq);

                               {          /   1 \ }
                               { C1 exp| - -  | }
                               {          \   x / }
```

In der Eingabe dürfen neben der gesuchten Funktion auch weitere unbekannte Funktionen und Parameter auftreten:

```
>> eq:=ode(y'(x)=c*g(y(x)),y(x)): sol:=solve(eq);
```

[1] Abkürzung für *ordinary differential equation*

```
          /     /  1       \                \
     solve| int| ----, y | = C1 + c x, y |
          \     \ g(y)     /                /
```

Wenn *MuPAD* nicht weiterkommt, gibt es einen unevaluierten (d.h. nicht ausgeführten) Aufruf der Funktion `solve` zurück. In manchen Fällen ist dies genau der Aufruf, der vom Benutzer eingegeben wurde; oft gelingen aber wenigstens ein paar Schritte auf dem Weg zur Lösung. In unserem Beispiel ist ohne Kenntnis von `g` in der Tat nichts zu machen; hat `g(y)` später einen bestimmten Wert, so wird automatisch bei der nächsten Auswertung von `sol` weitergerechnet:

```
>> g(y):=y^2: sol;

                  {      1      }
                  { ---------- }
                  { - C1 - c x }
```

In welchen Fällen kann eine Gleichung nun gelöst werden? Dies hängt von der „Einfachheit" erstens der Gleichung und zweitens der Lösung ab. Die hauptsächlichen Fortschritte in der neuen Version liegen in der Lösbarkeit von linearen Differentialgleichungen[2].

```
>> Ty:=diff(y(x),x$3)+(2*x^2-1)/(2*x*(x-1))*diff(y(x),x$2)+
&>      (196*x^3-491*x^2+295*x-98)/(196*x^2*(x-1)^2)*diff(y(x),x)+
&>      (-x^2+3*x-1)/(196*x^2*(x-1)^2)*y(x):
```

```
>> solve(ode(Ty, y(x)));

{       /       /      1          \ \          /    /       1          \ \
{       |    int| -----------, x | |          | int| -----------, x | |
{       |       |   2     1/2     | |          |    |   2     1/2     | |
{       |       \ (x  - x)        / |          |    \ (x  - x)        / |
{ C2 exp| - --------------------- | + C3 exp| --------------------- | + C4
{       \            14           /          \            14           /
{
{
{
{
{

      /    /       1          \ \ /     /      /       1          \ \     /
      | int| -----------, x | | |     |   int| -----------, x | |     |
      |    |   2     1/2     | | |     |      |   2     1/2     | |     |
      |    \ (x  - x)        / | |     |      \ (x  - x)        / |     |
   exp| --------------------- | | exp| - --------------------- | int|
      \           14           / |     \            7           /     |
                                  |                                   |
```

[2] d.h. linear in der gesuchten Funktion $y(x)$ und deren Ableitung, aber nicht unbedingt in x

```
                             |                                   |
                             |                                   |
                             |                                   \
                             \

                                                   /
                                                   |
/                   /    /        2        \ \ \   |
|            1/2    |    | 5 x - x  - 2    | | |   |
| (x (x - 1))    exp| int| ------------, x | | | / |
|                   |    |     2           | | |   \
\                   \    \    x  - x       / / /

         /      /      1         \ \
         |   int| -----------, x | |
         |      |   2     1/2    | |
    2    |      \ (x  - x)       / |
28 x  exp| - --------------------- | -
         \            14           /

         /      /      1         \ \
         |   int| -----------, x | |
         |      |   2     1/2    | |
    3    |      \ (x  - x)       / |
56 x  exp| - --------------------- | +
         \            14           /

         /      /      1         \ \ \    \      / /
         |   int| -----------, x | | |    |      | |
         |      |   2     1/2    | | |    |      | |
    4    |      \ (x  - x)       / | |    |      | |            1/2
28 x  exp| - --------------------- | |, x | - int| | (x (x - 1))
         \            14           / /    |      | |
                                          |      | \
                                          |      |
                                          |      |
                                          /      |
                                                 \

                                 /      /      1         \ \ \
                                 |   int| -----------, x | | |
   /    /        2        \ \    |      |   2     1/2    | | |
   |    | 5 x - x  - 2    | |    |      \ (x  - x)       / | |
exp| int| ------------, x | | exp| - --------------------- | | /
   |    |     2           | |    \            14           / |
   \    \    x  - x       / /                                /

                           \ \ }
                           | | }
                           | | }
     2       3       4     | | }
(28 x  - 56 x  + 28 x ), x | | }
```

```
| | }
| | }
| | }
| | }
| | }
/ / }
```

Das vorstehende Beispiel ist typisch für eine sogenannte *liouvillesche* Lösung, d.h. eine Funktion, die aus ineinander verschachtelten algebraischen Funktionen, Exponentialfunktionen und Integralen aufgebaut ist.[3]

Numerik gewöhnlicher Differentialgleichungen

In der neuen *MuPAD*-Version 1.4 ist eine wichtige Lücke geschlossen worden: mit Hilfe der Funktion `numeric::odesolve` können Differentialgleichungen nun auch numerisch gelöst werden. Implementiert wurde diese neue Funktionalität von Walter Oevel.

Gegeben sei eine gewöhnliche Differentialgleichung $y' = f(x, y)$ mit Anfangsbedingung $y(x_0) = y_0$; gesucht ist der Wert der Lösung y an einer gegebenen Stelle $x_1 \neq x_0$. (Wir nehmen hier zunächst an, daß die DGL eindeutig lösbar ist.) Diese Daten sind wie folgt einzugeben: als erstes Argument der Bereich $x_0 \ldots x_1$, dann die Funktion f, schließlich der Wert y_0. Als sehr einfaches Beispiel betrachten wir $f(x, y) = \frac{y(x)}{x}$ und die Anfangsbedingung $y(1) = 3$; gesucht ist $y(2)$. Die Lösung in MuPAD sieht hier wie folgt aus:

```
>> numeric::odesolve(1..2, func([y[1]/x],x,y), [3]);

                          [6.0]
```

In der Tat überzeugt man sich leicht (etwa mit Hilfe des symbolischen Differentialgleichungslösers), daß $y(x) = 3x$ die eindeutig bestimmte Lösung der DGL ist und somit $y(2) = 6$ gilt. Aber was haben die eckigen Klammern zu bedeuten? Hier zeigt sich die Allgemeinheit von `odesolve`: es kann vektorwertige Lösungen $y : \mathbb{R} \mapsto \mathbb{R}^n$ von Differentialgleichungen im $\mathbb{R}^n$ für beliebiges $n \geq 1$ berechnen, auch im Spezialfall $n = 1$ werden die Funktionswerte von y daher als Vektoren geschrieben. Insbesondere kann *MuPAD* damit nun auch Systeme von Differentialgleichungen sowie Differentialgleichungen höherer Ordnung numerisch lösen. Gegeben sei etwa das Anfangswertproblem

$$y''(x) = y'(x) * y(x) + cos(y(x) * x^2) + x^3, \quad y'(0) = y(0) = 0.$$

Mit $y[1] = y,\ y[2] = y'$ wird daraus

```
>> numeric::odesolve(0..2, func([y[2],y[2]*y[1]+cos(y[1]*x^2)+x^3],x,y),
>& [0,0]);

                [7.06780213, 29.95374367]
```

Dies bedeutet fur die Lösung y des Anfangswertproblems: $y(2) \approx 7.06, y'(2) \approx 29.95$.

[3]Nach solchen Lösungen kann man neuerdings mittels `ode::liouvillianSolutions` auch direkt suchen.

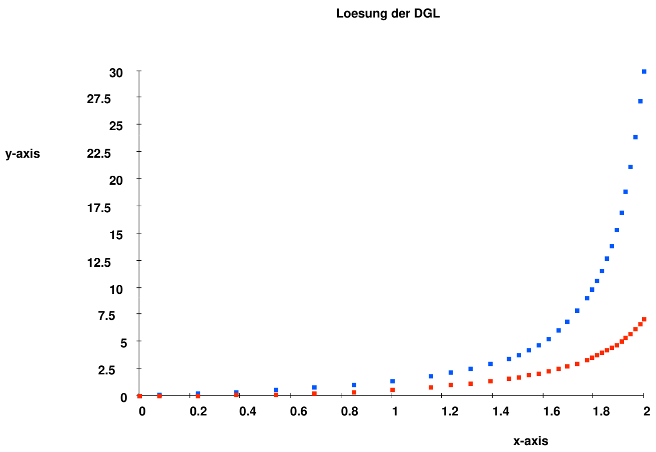


Abbildung 6: Numerisch bestimmte Werte der Lösung

Oft wird nicht nur der Wert der Lösung an einer bestimmten Stelle, sondern eine umfangreiche Wertetabelle gewünscht, etwa wenn ein Funktionsgraph geplottet werden soll. Hierfür existiert die Option `Alldata`, bei der eine solche Wertetabelle als Liste zurückgegeben wird. Wir betrachten wieder das Beispiel von eben:

```
>> liste:=numeric::odesolve(0..2, func([y[2],y[2]*y[1]+cos(y[1]*x^2)+x^3],x,y),
>& [0,0], Alldata):
```

Jetzt besteht `liste` aus Listen der Form $[x, [y(x), y'(x)]]$; zuerst müssen daraus Punkte der Form $(x, y(x))$ bzw. $(x, y'(x))$ erzeugt werden:

```
>> ypunkte:=  map(liste, func(point(l[1], l[2][1]), l)):
>> dypunkte:= map(liste, func(point(l[1], l[2][2]), l)):
```

Man kann sich nun mittels

```
>> plot2d(Title="Loesung der DGL", Scaling=UnConstrained, Labeling=TRUE,
>& [Mode=List, ypunkte], [Mode=List, dypunkte]):
```

die Punkte der Wertetabelle plotten lassen. Das Ergebnis sieht man in Abbildung 6.

Interessant ist auch der Vergleich mit den vorhandenen Verfahren zur numerischen Integration in MuPAD (letztlich ja ein Spezialfall des numerischen Lösens von Differentialgleichungen). So kann etwa das Integral

$$\int_1^2 x^{\frac{1}{x}}\,dx$$

auf zwei verschiedene Arten berechnet werden:

```
>> float(hold(int)(x^(1/x),x=1..2));

                              1.277065148
```

```
>> numeric::odesolve(1..2, func([x^(1/x)],x,y), [0]);

                    [1.277065148]
```

Was steckt nun an numerischen Verfahren hinter `odesolve`? Hauptsächlich Runge-Kutta-Methoden in allen erdenklichen Variationen mit verschiedenen Konvergenzordnungen. Als optionaler vierter Parameter kann der anzuwendende Algorithmus vom Benutzer selbst bestimmt werden, wobei nicht weniger als 23 Möglichkeiten zur Auswahl stehen. Genaueres findet man in *Walter Oevel,* **Numerical Computations in MuPAD 1.4**, in diesem Heft. Zusätzlich besteht die Möglichkeit, die Schrittweite selbst vorzugeben, wodurch das Ergebnis natürlich (bei zu großer Schrittweite) sehr ungenau werden kann. Im folgenden wird dasselbe Integral wie zuvor mit nur zehn Schritten des „klassischen" Runge-Kutta-Verfahrens vierter Ordnung berechnet:

```
>> numeric::odesolve(1..2, func([x^(1/x)],x,y), [0], RK4, Stepsize=0.1);

                    [1.277065062]
```

Zum Schluß dürfen einige warnende Hinweise nicht fehlen: eine numerische Funktion wie `numeric::odesolve` muß zwangsläufig in Schwierigkeiten kommen, wenn sie auf Singularitäten oder den Rand des Definitionsbereichs stößt, und sie kann auch nicht entscheiden, ob die gegebene DGL eindeutig lösbar ist. So war etwa als Bestandteil der ISSAC Challenge 1997 das folgende Anfangswertproblem numerisch zu lösen

$$y''(x) = y'(x)^3 + y(x)^3 + x^3, y(0) = 0, y'(0) = 0$$

und der Wert von y an der (ebenfalls zu bestimmenden) Singularität der Ableitung auszuwerten. Wie man dies in *MuPAD* mit Hilfe gewisser Vorüberlegungen tut, wurde bereits in [2] beschrieben; jedenfalls ist es unzulässig, einfach z.B.

```
numeric::odesolve(0..2, func([y[2],y[2]^3+y[1]^3+x^3],x,y), [0,0]);
```

einzutippen, weil die adaptive Schrittweitensteuerung in diesem Fall die Schrittweite unmittelbar links von der Singularität beliebig weit verkleinert (und damit die Anzahl der Schritte entsprechend ins Unendliche vergrößert). Dagegen erhält man durch ein Verfahren mit fester Schrittweite auch ohne Vorüberlegungen (Singularitätenanalyse) die Information, daß die Singularität offenbar zwischen 1.64 und 1.65 liegen muß:

```
>> numeric::odesolve(0..1.64, func([y[2],y[2]^3+y[1]^3+x^3],x,y),
>& [0,0], RK4, Stepsize=0.005);

               [0.838250797, 10.66298702]


>> numeric::odesolve(0..1.65, func([y[2],y[2]^3+y[1]^3+x^3],x,y),
>& [0,0], RK4, Stepsize=0.005);

           [1.417321651e904, 5.159415622e2749]
```

Somit erhält der Benutzer mit dem numerischen Differentialgleichungslöser ein gutes Hilfsmittel an die Hand, auch „häßliche” Gleichungen schnell untersuchen zu können.

Integration in das Domains–Konzept

Neben dem eigentlichen Ziel, Differentialgleichungen lösen zu können, gilt es auch, zunächst Methoden zur Behandlung der darin auftretenden Terme zu entwickeln. Die einfachste Möglichkeit besteht darin, einfach den Datentyp des Kerns `DOM_EXPR` für Ausdrücke zu verwenden, ähnlich wie ja auch reelle Funktionen am besten auf diese Art geschrieben werden. — Nebenbei muß erwähnt werden, daß auch die im Kern definierten Polynome Ableitungen als Unbestimmte zulassen, zum Beispiel

```
>> poly(diff(y(x),x)+diff(y(x),x)^2+3, [y(x)]);

                          2
      poly(diff(y(x), x)  + diff(y(x), x) + 3, [diff(y(x), x)])
```

Seit einiger Zeit gibt es hierzu nun in *MuPAD* eine Alternative, die auf dem bewährten Domains-Konzept beruht. Für die Ausdrücke, die auf der rechten Seite einer expliziten DGL auftreten können, wurden eigene Datentypen eingeführt. Entwurf und Implementation stammen von Werner Seiler, MuPAD's Kooperationspartner in Mannheim. Vorteil ist (ähnlich wie bei den speziellen Datentypen für Polynome), daß spezielle Methoden implementiert und algebraische Strukturen genauer abgebildet werden können. Im Unterschied zu Ausdrücken des Kerns sind die Abhängigkeiten der Variablen von einander hier ein Bestandteil des Typs: je nachdem, ob y eine Funktion von x ist oder umgekehrt, hat $y + x$ jeweils eine verschiedene Bedeutung. Nachdem diese Abhängigkeit einmal definiert ist, reicht es, `y` statt `y(x)` einzugeben. Die obige Eingabe wird damit etwas übersichtlicher:

```
>> V:=Dom::DifferentialVariable([x],[y]):
>> P:= Dom::DifferentialPolynomial(V, Dom::Rational);

Dom::DifferentialPolynomial(Dom::DifferentialVariable([x], [y]),

   Dom::Rational, DegreeOrder)

>> P(D(y)^2+D(y)+3);

                                    2
                            y([x])   + y([x]) + 3
```

Eckige Klammern ersetzen hier die sonst üblichen unteren Indizes, d.h. statt y_{xxx} schreibt man `y([x,x,x])`. Natürlich ist die Anzahl der abhängigen und unabhängigen Variablen nicht beschränkt.

Genauso wie `DifferentialPolynomial` zu gewöhnlichen Polynomen, bildet `DifferentialExpression` das Gegenstück zu allgemeinen Ausdrücken. Ein Spezialfall sind diejenigen Ausdrücke, in denen nur *unabhängigen* Variablen auftreten (`Dom::IndependentDifferentialExpression`).

Über die Darstellung von Differential*operatoren* durch spezielle Datentypen war schon in [1] die Rede. Zu ergänzen bleibt, daß es nicht nur das Domain `Dom::LinearOrdinaryDifferentialOperator` für gewöhnliche, sondern auch `Dom::LinearDifferentialOperator` für partielle Differentialoperatoren gibt.

Zum Abschluss muß erwähnt werden, daß es in *MuPAD* auch formale Differentialkörper gibt. Sei etwa $F = \mathbb{Q}(x)$ ausgestattet mit der üblichen Ableitungsoperation. Wir adjungieren $y := \sqrt(x)$ zu diesem Körper, d.h. eine Nullstelle des Polynoms $y^2 - x \in F[y]$. Die Fortsetzung der Ableitungsoperation ist dann eindeutig bestimmt (natürlich ist $[\sqrt(x)]' = \frac{1}{2\sqrt(x)} = \frac{\sqrt(x)}{2x}$).

```
>> F:=Dom::Fraction(Dom::DistributedPolynomial([x], Dom::Rational));

        Dom::Fraction(Dom::DistributedPolynomial([x], Dom::Rational))

>> A:=Dom::AlgebraicExtension(F, poly(y^2-x,[y],F));

Dom::AlgebraicExtension(Dom::Fraction(

                                                           2
   Dom::DistributedPolynomial([x], Dom::Rational)), y  - x = 0, y)

>> c:=A(y);

                                    y

>> diff(c,x);

                                    y
                                   ---
                                   2 x
```

Auf diese Weise ist es z.B. auch möglich, Differentialkörper endlicher Charakteristik zu definieren.

Differentialgeometrie

Das Differentialgeometrie-Paket `diffgeom` beschäftigt sich mit Mannigfaltigkeiten und darauf definierten Funktionen, d.h. Skalar-, Vektor- und Tensorfeldern. Diese allgemeine Beschreibung umfaßt eine ganze Reihe interessanter Spezialfälle, wie etwa Felder im $\mathbb{R}^n$ oder konstante Felder, d.h. Skalare, Vektoren und Tensoren. Das Paket beruht auf dem objektorientierten *Domains*-Konzept.

Eine m-dimensionale *Mannigfaltigkeit* im n-dimensionalen Raum ist das Bild einer unendlich oft differenzierbaren Funktion $U \mapsto \mathbb{R}^n$, wo U eine rechteckige Teilmenge des $\mathbb{R}^m$ ist.[4] Anders ausgedrückt: eine Mannigfaltigkeit ist gegeben durch einen Vektor $[a_1(t_1, \ldots t_m), \ldots, a_n(t_1, \ldots t_m)]$, dessen Koordinaten Funktionen von m Parametern sind, und m Definitionsbereichen für die Parameter; sie besteht aus allen Punkten, die man durch Einsetzen von Werten für die Parameter erhält. Alle Mannigfaltigkeiten im n-dimensionalen Raum bilden das Domain `diffgeom::Manifold(n)`. Einige Beispiele im dreidimensionalen Raum:

```
>> M:=diffgeom::Manifold(3);
```

[4] Dies ist eine Abwandlung des üblichen Begriffs: normalerweise setzt man eine Familie von (im allgemeinen unendlich vielen) offenenen Teilmengen $U_i \subseteq \mathbb{R}^m$ und auf jeder eine *injektive* Abbildung in den $\mathbb{R}^n$ mit noch weiteren Eigenschaften voraus - die Vereinigung der (einander überlappenden) Bilder heißt dann Mannigfaltigkeit. Es dürfte schwer sein, diesen Begriff in eine Datenstruktur umzusetzen.

```
                              diffgeom::Manifold(3)

>> K:=M([t, cos(2*PI*t), sin(2*PI*t)],[t],[0..1]);

                  [t, cos(2 t PI), sin(2 t PI)], t = 0..1

>> Z:=M([cos(ph), sin(ph), z], [ph,z], [0..PI, -infinity..infinity]);

       [cos(ph), sin(ph), z], ph = 0..PI, z = -infinity..infinity

>> R3:= M();

[X1, X2, X3], X1 = -infinity..infinity, X2 = -infinity..infinity,

   X3 = -infinity..infinity

# Einsetzen von Parameterwerten z.B: #

>> Z([PI,1]);

                               +-         -+
                               | -1, 0, 1 |
                               +-         -+
```

Durch diese Definitionen ist K nun eine parametrisierte Kurve (eine Kreisspirale), Z (anschaulich eine Zylinderhälfte) eine parametrisierte Fläche, und $R3$ ist der $\mathbb{R}^3$ selbst.

Natürlich kann ein und dieselbe Punktmenge auf verschiedene Arten parametrisiert werden (und *MuPAD* hat keine Möglichkeit, die Gleichheit zweier Mannigfaltigkeiten als Punktmengen zu erkennen). Soll zwischen verschiedenen Parametrisierungen gewechselt werden, muß zunächst definiert werden, wie der Parameterwechsel zu geschehen hat. Zu diesem Zweck gibt es das Domain `diffgeom::Parametrization`. Diejenige Parametrisierung Q, die bei der Definition der Mannigfaltigkeit verwendet wurde, wird als „kanonisch" ausgezeichnet, und jede andere Parametrisierung P wird durch die beiden Übergänge $Q \mapsto P$ und $P \mapsto Q$ definiert.

```
>> Zparams:=diffgeom::Parametrization(Z);

diffgeom::Parametrization([cos(ph), sin(ph), z], ph = 0..PI,

   z = -infinity..infinity)

# definiere P: jedem Paar (y,z) wird das Paar (cos(x),z) zugeordnet,
  die Umkehrabbildung ordnet jedem Paar (x,z) das Paar (acos(x),z) zu #

>> P:=Zparams([x,z],[cos(x),z],[acos(x),z]);

                             (x, z) -> (acos(x), z)

>> P([-1,1]);

                                   [cos(-1), 1]
```

```
>> Z((P^(-1))([-1,1]));

                          +-          -+
                          | -1, 0, 1  |
                          +-          -+
```

P bewirkt hier den Übergang von sphärischen zu kartesischen Koordinaten (wobei der Radius bzw. die y-Koordinate fehlen). Der Benutzer ist selbst dafür verantwortlich, daß die beiden Koordinatenwechsel einer Parametrisierung zueinander invers sind.

Mannigfaltigkeiten und ihre verschiedenen Parametrisierungen sind für sich genommen nicht besonders interessant. Sie dienen aber als Definitionsbereich für verschiedene Arten von Funktionen (Feldern).

Skalarfelder ordnen jedem Punkt der Mannigfaltigkeit eine Zahl zu (physikalisches Beispiel: jeder Punkt des oben definierten Halbzylinders `Z` hat eine bestimmte Temperatur). Ein Skalarfeld ist also nichts weiter als ein arithmetischer Ausdruck, der von den Koordinaten einer gegebenen Parametrisierung abhängt (wobei diese Darstellung gewechselt werden kann). Angenommen, der Punkt mit kartesischen Koordinaten (x, y, z) hat Temperatur $T(x, y, z) = 2 * x^2 + y^2 - 3 * z^2$. Es ist möglich, dies als Skalarfeld im $\mathbb{R}^3$ zu definieren[5], oder auch nur auf der Untermannigfaltigkeit Z (wo noch zusätzlich $x^2 + y^2 = 1$ gilt, was uns Tipparbeit erspart):

```
>> S:=diffgeom::ScalarField(Z):
>> Temperatur:=S(x^2+1-2*z^2, P^(-1));

                     2      2
                    x  - 2 z  + 1, (x, z) -> (acos(x), z)

>> S::changeparam(Temperatur, Zparams::one);

                     2            2
                - 2 z  + acos(ph)  + 1 , (ph, z) -> (ph, z)
```

Die Eingabe erfolgte in kartesischen Koordinaten: da bei Definition von Z Zylinderkoordinaten für „kanonisch" erklärt wurden, werden diese mit P^{-1} bezeichnet, da P^{-1} die Transformation ist, die Zylinderkoordinaten in kartesische überführt.

Vektorfelder ordnen jedem Punkt der Mannigfaltigkeit einen Tangentenvektor an diesen Punkt zu. Zu der Darstellung von Vektorfeldern als Datenstruktur ist zunächst zu sagen, daß das Differentialgeometrie-Paket kein Domain für Tangentenvektoren an einen bestimmten Punkt enthält; vielmehr werden die partiellen Ableitungen nach den Parametern der Mannigfaltigkeit als Basis für jeden Tangentenraum gewählt, und Vektoren an verschiedenen Aufpunkten, die bei dieser Wahl der Basis die gleichen Koordinaten haben, werden miteinander identifiziert. Wenn z.B. ein Teilchen die oben definierte spiralförmige Bahn $K : t \mapsto (t, cos(2\pi t), sin(2\pi t))$ durchläuft, wird dadurch jedem Zeitpunkt t eine Momentangeschwindigkeit zugeordnet; diese ist *der* kanonische Basisvektor des Tangentenraums an den Punkt $K(t)$; seine Darstellung ist einfach `[1]`:

```
>> VF:=diffgeom::VectorField(K);

diffgeom::VectorField([t, cos(2 t PI), sin(2 t PI)], t = 0..1,
```

[5]man gibt dann einfach `S:=diffgeom::ScalarField(R3):` `Temperatur:=S(`$2 * x^2 + y^2 - 3 * z^2$`):` ein

```
   contravariant)

>> v:=VF([1]);

                              [1], [t], [t]


# sieht unabhaengig von t aus, im R^3 sieht man jedoch, dass die
  Tangentenvektoren sich mit der Spirale "mitdrehen" #

>> VF::cartesianCoordinates(v);

                 [1, -2 PI sin(2 t PI), 2 PI cos(2 t PI)]
```

Kovektorfelder ordnen jedem Punkt der Mannigfaltigkeit eine lineare Abbildung vom Tangentenraum an diesen Punkt nach $\mathbb{R}$ zu.

```
>> VFZ:=diffgeom::VectorField(Z):
>> CVFZ:=diffgeom::VectorField(Z, "covariant"):
>> v:=VFZ([sin(ph),z*cos(ph)]);

                  [sin(ph), z cos(ph)], [ph, z], [ph, z]


>> cv:=CVFZ([ph+z, ph-z]);

                   [z + ph, - z + ph ], [ph, z], [ph, z]


>> v*cv;

       sin(ph) (z + ph) + z cos(ph) (- z + ph ), (ph, z) -> (ph, z)
```

Als Skalarprodukt wird defaultmässig das Standardskalarprodukt verwendet — das heist, es wird davon ausgegangen, daß die Basis `[phi, z]` der Kovektoren zu der gleichnamigen Basis der Vektoren dual ist. Das Skalarprodukt kann durch die Methode `setScalarProduct` der zugrunde liegenden Mannigfaltigkeit `Z` umdefiniert werden.

Natürlich sind auch für Vektoren und Kovektoren verschiedene Koordinatensysteme verfügbar.

```
>> v::changeparam(v, 2=P, 3=P);

     [-sin(x) sin(acos(x)), z cos(acos(x))], [cos(x), z], [cos(x), z]
```

Weitere wichtige Methoden sind die verschiedenen Formen von Ableitungen (totales Differential, Richtungsableitungen).

```
>> diff(v, VFZ([cos(ph),z*sin(ph)]));

                                    2
                      [cos(ph) , 0], [ph, z], [ph, z]
```

Tensorfelder sind Tensorprodukte von Vektor- und Kovektorfeldern, d.h. Abbildungen, die zu je einem Punkt und einem Tupel von Tangentenvektoren und -kovektoren an diesen Punkt eine reelle Zahl zurückliefern.

```
>> T:=diffgeom::TensorField([VFZ, CVFZ]):
>> t:=T([[z*sin(ph), z*cos(ph)],[-z*cos(ph), z*sin(ph)]]);

                        +-                          -+
                        |   z sin(ph), z cos(ph)    |
                        |                           |
                        |  -z cos(ph), z sin(ph)    |
                        +-                          -+
```

Das Ergebnis einer Überschiebung von Tensor und Vektor ist formal kein Vektor, sondern ein 1-Tensor; jedoch ist die Umwandlung möglich.

```
>> t.v;

             array(1..2,
                              2    2         2
               (1) = z sin(ph)  + z  cos(ph) ,
                                            2
               (2) = - z cos(ph) sin(ph) + z  cos(ph) sin(ph)
             )


>> VFZ(%);

          2    2         2                          2
[z sin(ph)  + z  cos(ph) , - z cos(ph) sin(ph) + z  cos(ph) sin(ph) ],

   [ph, z], [ph, z]
```

Wie oben zu sehen, werden Tensorfelder mit Hilfe von Arrays realisiert und als solche ausgegeben.

Weitere Methoden für Tensoren sind zum Beispiel Indexpermutationen, Kontraktion oder Tensorprodukt. Bestimmte Funktionen (wie etwa das äußere Produkt $\wedge$) sind nur für rein kontra- und rein kovariante Tensoren definiert.

Verfügbarkeit

Das ODE-Paket sowie die neuen numerischen Funktionen sind bereits in *MuPAD* 1.4 integriert. Die neuen Domainkonstruktoren aus obigem Abschnitt (siehe „Integration in das Domains–Konzept“) sind in einer undokumentierten Experimentalversion ebenfalls enthalten. Das Differentialgeometrie-Paket wird demnächst veröffentlicht.

Ausblick

Was bleibt zu tun? Im Moment fehlen noch Methoden zur (symbolischen und numerischen) Behandlung partieller Differentialgleichungen. Ebenso können die in diesem Artikel vorgestellten Funktionen sicherlich noch weiterentwickelt werden, etwa im Hinblick auf spezielle Typen von Differentialgleichungen, numerische Verfahren für Randwertprobleme (Schießverfahren) oder weitere Funktionen für Tensoren. Die Entwicklung wird fortgesetzt und erst in einer nachfolgenden Version ihren Abschluss erreichen.

Literatur

[1] *W.Fakler*, *Algorithms for Solving Linear Ordinary Differential Equations*. In: MathPAD Vol.7 No.1, September 1997, S.50-59.

[2] *Walter Oevel, Stefan Wehmeier, Paul Zimmermann*, *MuPAD's solutions to the ISSAC challenge*, August 1997. Im Web unter `http://www.mupad.de/CONF/ABSTRACTS/ISSAC97C/`.

[3] *Werner Seiler*, *Computer Algebra and Differential Equations - An Overview*. In: MathPAD Vol.7 No.1, September 1997, S.34-49.

Numerical Computations in *MuPAD* 1.4

Walter Oevel
Universität GH Paderborn, `walter@uni-paderborn.de`

Some of the algorithms in the numeric package of MuPAD 1.4 are discussed. In particular the routines for computing eigenvalues, for numerical quadrature and for solving ordinary differential equations are compared to the corresponding routines provided by Mathematica 3.0 and Maple V.4.

Introduction

The new *MuPAD* release 1.4 features a variety of numerical routines collected in the library domain `numeric`:

`bairstow`	all roots of a polynomial via Bairstow's algorithm,
`bisect`	compute a root of a function by bisectioning,
`butcher`	Butcher parameters of explicit Runge-Kutta schemes,
`det`	determinant of a matrix,
`eigenvalues`	eigenvalues of a matrix,
`eigenvectors`	eigenvalues and vectors of a matrix,
`factorCholesky`	Cholesky factorization of a positive definite matrix,
`factorLU`	LU factorization of a matrix,
`factorQR`	QR factorization of a matrix,
`fint`	numerical integration,
`fsolve`	all zeros of a polynomial,
`gldata`	weights and abscissae of Gauss-Legendre quadrature,
`gtdata`	weights and abscissae of Gauss-Tschebyscheff quadrature,
`inverse`	inverse of a matrix,
`ncdata`	weights and abscissae of Newton-Cotes quadrature,
`newton`	Newton search for the root of an expression,
`odesolve`	solve the initial value problem of a dynamical system,
`quadrature`	numerical quadrature,
`singularvalues`	singular values of a matrix,
`singularvectors`	singular value decomposition of a matrix,
`vonMises`	spectral radius of a matrix (the power method).

There is some overlap with corresponding library routines such as `linalg::det`, `linalg::eigenvalues`, `linalg::inverse`, ... In fact, many of the `numeric` routines can handle symbolic data as well as numeric data. However, it should be noted that the `numeric` functions are tuned to handle numeric input in a most efficient way. While the `linalg` routines are designed to allow the computation of determinants, inverses etc. over various coefficient rings, the `numeric` routines try to avoid the overhead of computations in such domains and treat all data as of some numerical type such as `DOM_FLOAT`, `DOM_INT`, `DOM_RAT` or `DOM_COMPLEX`. Symbolic data, if accepted by the routines, are regarded as being of domtype `DOM_EXPR`. Most functions return data of domtype `DOM_FLOAT` by default. However, in some cases options may be passed to the routines preventing the conversion of exact data (domtype `DOM_INT` or `DOM_RAT`) to floats.

For floating point computations the global variable `DIGITS` controls the numerical precision of the `numeric` routines. Some of them do not use extra digits internally to increase the precision of intermediate steps. Consequently, due to roundoff it should not be expected that the returned values are always correct up to the very last decimal place specified by `DIGITS`.

Nevertheless, those routines featuring an adaptive error control mechanism (numerical quadrature, the Runge-Kutta methods of `odesolve`, eigenvalue routines) try to push the discretization error of the method below a relative precision of $10^{-\mathtt{DIGITS}}$. This is often possible despite the presence of roundoff errors of this magnitude. It is recommended that a numerical result is compared to the result obtained by another test run with slightly increased `DIGITS`. The stable decimals may be regarded as reliable.

Some of the routines such as `quadrature` or `odesolve` are collections of methods which may be specified by the user via suitable options. However, often a user does not want to be bothered with the technical details of the computation, or he does not have the expertise to choose the method most suitable for his particular application. Special care was taken to choose a default mechanism which yields acceptable results for large classes of "typical" input data.

Further, we point out that in the numerical context one of the most attractive features of symbolic environments is their arbitrary precision arithmetic. If standard hardware precision suffices, then symbolic systems cannot compete with numerical packages using compiled code and hardware support. For this reason we regard it important that numerical routines in a computer algebra system should be able to compute high precision results efficiently. Consequently we tried to tune *MuPAD*'s `numeric` routines for efficiency far beyond machine precision. This is quite in contrast to Mathematica and Maple where the numerics is tuned to be efficient at low digits.

In the following sections we shall have a closer look at some of the more advanced algorithms implemented in the *MuPAD* 1.4 `numeric` package and compare them to the corresponding functions provided in Mathematica 3.0 and Maple V.4. In these tests we used the default mechanisms of the routines whenever possible. We consider low precision goals of 10 digits as well as high precision goals with several hundred digits. All tests were carried out on a SUN Enterprise 450 Server workstation equipped with a 300 MHz processor and 512 MByte of main memory.

Eigenvalues

Compared to *MuPAD* 1.3 the new release 1.4 has gained a lot of efficiency reading from and writing into (large) matrices thanks to appropriate improvements in the *MuPAD* kernel. Simple tests reveal that for matrices of domtype `DOM_ARRAY` the time used by indexed reading and writing in the form `A[i,j]:=formula(A[k,l])` may be neglected in *MuPAD* 1.4, when comparing these costs to the typical floating point operations involved. However, large objects of domtype `Dom::Matrix` should be avoided in the numerical context, since reading and writing is significantly more expensive for this data type. As a general rule, the `numeric` routines dealing with matrices internally use the data type `DOM_ARRAY` to hold the data. Input matrices of domtype `Dom::Matrix` are converted to arrays before any numerical algorithm is applied. Matrix output is re-converted to `Dom::Matrix` allowing user-friendly further processing of the data.

We have a closer look at the numerical algorithms for computing eigenvalues and eigenvectors of matrices. The numerical standard method consists of the following steps:

i) "Balance" the matrix to improve numerical stability of the problem.
ii) Apply a similarity transformation to convert the matrix to (upper) Hessenberg form, i.e., to a matrix with a single sub-diagonal band. For hermitean matrices the resulting Hessenberg matrix is real and tridiagonal.
iii) Use some variant of the QR-method with spectral shifts in order to compute the spectral data of the Hessenberg matrix.
iv) If eigenvectors are requested, then apply the inverses of the previous transformations to the eigenvectors of the

	digits $= 10$	digits $= 20$	digits $= 100$	digits $= 500$
$n = 10$	0.3 / 0.0 / 0.1	0.3 / 0.2 / 0.2	0.5 / 0.6 / 1.0	3.4 / 8.4 / 10.6
$n = 25$	1.8 / 0.0 / 0.2	2.0 / 1.6 / 1.2	3.5 / 4.7 / 7.2	25.5 / 72.0 / 71.8
$n = 50$	10.0 / 0.2 / 1.1	10.7 / 12.0 / 7.9	18.0 / 30.0 / 59.3	127.0 / 418.0 / 352.0
$n = 100$	67.2 / 0.7 / 7.0	70.5 / 97.0 / 67.8	109.0 / 200.0 / 641.0	725.0 / 2610.0 / 2570.0

Table 1: *Times (in seconds) to compute all eigenvalues of the symmetric $n \times n$ matrix $A_{ij} = (i + j - 1)^{-1}$ using MuPAD 1.4 / Mathematica 3.0 / Maple V.4.*

	digits $= 10$	digits $= 20$	digits $= 100$	digits $= 500$
$n = 10$	0.6 / 0.0 / 0.9	0.9 / 0.6 / 1.3	2.6 / 1.7 / 7.8	30.9 / 27.0 / 108.0
$n = 25$	6.0 / 0.1 / 11.0	7.2 / 6.3 / 15.4	18.9 / 19.0 / 115.0	240.8 / 270.0 / 1160.0
$n = 50$	63.5 / 0.3 / 110.0	62.8 / 44.4 / 142.0	111.0 / 110.0 / 1200.0	1240.0 / 1760.0 / 8380.0
$n = 100$	488.0 / 1.6 / 1670.0	538.0 / 350.0 / 2730.0	692.0 / 900.0 / 23600.0	6950.0 / 12100.0 / >24 hrs

Table 2: *Times (in seconds) to compute all eigenvalues of the complex $n \times n$ matrix $A_{ij} = (i + j\,I - 1)^{-1}$ using MuPAD 1.4 / Mathematica 3.0 / Maple V.4.*

Hessenberg matrix.

The complexity of this algorithm is $O(n^3)$ for $n \times n$ matrices. For hermitean matrices only a single $O(n^3)$ step is required, when computing the transformation to real tridiagonal form. The remaining diagonalization of the tridiagonal form only needs $O(n^2)$ operations, if only eigenvalues are requested. For arbitrary (non-hermitean) matrices the computational cost is higher, since also the diagonalization of the Hessenberg matrix needs $O(n^3)$ operations.

Standard implementations of this strategy exist for various computer languages [5],[6]. We have implemented and optimized this code for the *MuPAD* language. Special care was taken to deal with the case of complex input data efficiently. In Tables 1, 2, 3 and 4 we list the timings for test runs in *MuPAD* 1.4 and the corresponding timings in Mathematica 3.0 and Maple V.4.

For testing the hermitean case we applied our `numeric` routines to the $n \times n$ Hilbert matrices $A_{ij} = 1/(i+j-1)$. These are notoriously ill-conditioned positive definite matrices: the ratio of the largest to the smallest eigenvalue is approximately $e^{3.5\,n}$ for large n. At best the relative error $|\Delta\lambda|/\lambda$ in the approximation of an eigenvalue λ by a numerical computation with `DIGITS` significant decimal places is of the magnitude

$$\frac{|\Delta\lambda|}{\lambda} > \frac{\lambda_{max}}{\lambda}\,10^{-\mathtt{DIGITS}}\;, \tag{1}$$

whence a computational precision of at least `DIGITS` $\approx 1.5\,n$ is needed to catch the first few decimal places of the small eigenvalues correctly. The test example for the non-hermitean case is the complex matrix $A_{ij} = 1/(i + jI - 1)$. The statements used for the eigenvalue computation were:

MuPAD 1.4:	`DIGITS:=digits: numeric::eigenvalues(A);`
Mathematica 3.0:	`Eigenvalues[N[A,digits]]`
Maple V.4:	`Digits:=digits: linalg[eigenvalues](map(evalf,A));`

All returned results were accurate within the expected precision given by (1). The timings clearly show the superiority of Mathematica at very low digits. This should not be surprising: within hardware precision compiled kernel functions do the work using hardware floats. Also Maple makes use of hardware floats at low digits. No such facility is presently implemented in *MuPAD*'s `numeric` library. Beyond hardware precision *MuPAD*'s eigen routines turn out to be competitive. In fact, for some applications with high precision *MuPAD* is faster than its competitors. Maple reveals severe difficulties in dealing with large complex non-hermitean matrices and high digits.

	digits = 10	digits = 20	digits = 100	digits = 500
$n =$ 10	0.7 / 0.0 / 0.3	0.9 / 0.3 / 0.3	1.5 / 0.9 / 2.1	10.4 / 13.6 / 31.9
$n =$ 25	8.6 / 0.0 / 3.3	10.2 / 2.8 / 4.2	18.9 / 8.5 / 23.5	139.0 / 140.0 / 346.0
$n =$ 50	63.0 / 0.2 / 27.3	71.3 / 22.1 / 32.4	137.0 / 56.9 / 218.7	1030.0 / 882.0 / 2540.0
$n =$100	516.0 / 1.0 / 299.0	550.0 / 227.0 / 374.0	1020.0 / 393.0 / 3450.0	8020.0 / 5560.0 / 15100.0

Table 3: *Times (in seconds) to compute all eigenvectors of the symmetric $n \times n$ matrix $A_{ij} = (i + j - 1)^{-1}$ using MuPAD 1.4 / Mathematica 3.0 / Maple V.4.*

	digits = 10	digits = 20	digits = 100	digits = 500
$n =$ 10	1.3 / 0.0 / 1.6	1.7 / 1.1 / 2.0	4.6 / 3.4 / 14.5	50.6 / 54.8 / 231.0
$n =$ 25	13.8 / 0.1 / 23.4	16.2 / 13.1 / 29.6	41.5 / 40.8 / 290.0	495.0 / 635.0 / 2870.0
$n =$ 50	141.0 / 0.4 / 268.0	138.0 / 93.2 / 320.0	257.0 / 244.0 / 3260.0	2930.0 / 4080.0 / 21200.0
$n =$100	1180.0 / 1.1 / 4950.0	1280.0 / 860.0 / 7050.0	1690.0 / 1940.0 / 61600.0	18100.0 / 28500.0 / >24 hrs

Table 4: *Times (in seconds) to compute all eigenvectors of the complex $n \times n$ matrix $A_{ij} = (i + j\,I - 1)^{-1}$ using MuPAD 1.4 / Mathematica 3.0 / Maple V.4.*

Quadrature

The function `quadrature` in the `numeric` package does numeric integration:

$$\texttt{numeric::quadrature(f,x=a..b,Options)} \approx \int_a^b f\,dx\;.$$

There is another function `numeric::fint` for the same purpose. The procedure `fint` starts with a symbolic analysis of the integrand, trying to detect and eliminate singularities. The final numerical computation is based on the Clenshaw-Curtis algorithm. In contrast to `fint` the function `quadrature` does not use any symbolic pre-processing of the integrand. It applies a straightforward numerical scheme.

The implementation of `quadrature` is tuned to be efficient for smooth integrands f and high precision goals with up to 200 `DIGITS` of relative precision. It is common folklore that for high precision quadrature a variable order method such as Clenshaw-Curtis should be used [3]. However, our own tests have shown that the performance of this algorithm (or of its implementation) is not very satisfactory in practice. For this reason `quadrature` chooses a different strategy by using a fixed (but very high) order quadrature rule Q_n defined by n nodes $c_1, \ldots, c_n \in [0,1]$ and weights $b_1, \ldots, b_n$:

$$Q_n(f,\alpha,\beta) := (\beta - \alpha)\sum_{j=1}^{n} b_j\, f(\alpha + c_j\,(\beta - \alpha))\;.$$

Following [2] a simple error control mechanism is used to build an adaptive quadrature algorithm from this rule:

1) Compute a rough estimate $Q_{est} := Q_n(f,a,b)$ of the integral.

2) The following pseudo-implementation

```
quadrature:=proc(f,x=α..β)
local Q1,Q2,γ;
  γ:=(α+β)/2; Q1:=Qn(f,α,β); Q2:=Qn(f,α,γ)+Qn(f,γ,β);
  if abs(Q1−Q2) < Q_est * 10^(−DIGITS)
    then return(Q2)
    else return(quadrature(f,x=α..γ)+quadrature(f,x=γ..β))
  end_if;
end_proc;
```

leads to a numerical approximation `quadrature(f,x=a..b)`$\approx \int_a^b f\,dx$ based on the estimate Q_{est} which should be accurate within a relative precision of $10^{-\text{DIGITS}}$.

Recursively, bisected intervals are treated with the fixed quadrature rule Q_n. The process stops, when bisectioning of a subinterval does not change the estimate of the subintegral significantly, i.e., when it does not make any difference whether `Q1` or `Q2` is used as the contribution to the whole integral $\int_a^b f\,dx \approx Q_{est}$: via rounding with `DIGITS` significant decimal places one would find Q_{est}+`Q1`$=Q_{est}$+`Q2`.

This implementation may be regarded as an adaptive variant of the composite quadrature rule

$$Q_{N,n}(f,a,b) = \sum_{i=0}^{N-1} Q_n\Big(f, a+i\,\frac{b-a}{N}, a+(i+1)\,\frac{b-a}{N}\Big)\ ,$$

where N is of the form $N=2^m$ after m bisectioning steps. If Q_n is a consistent rule, i.e., if

$$Q_n(f,\alpha,\alpha+h) - \int_\alpha^{\alpha+h} f\,dx = o(h) \quad \text{(uniformly in } \alpha\text{)},$$

then clearly

$$\lim_{N\to\infty} Q_{N,n}(f,a,b) \ \longrightarrow\ \int_a^b f\,dx\ .$$

The speed of convergence is determined by the order p of the underlying quadrature Q_n:

$$Q_n(f,\alpha,\alpha+h) - \int_\alpha^{\alpha+h} f\,dx = O(h^{p+1}) \quad\Longrightarrow\quad Q_{N,n}(f,a,b) - \int_a^b f\,dx = O\Big(\frac{1}{N^p}\Big).$$

The qualitative features of this quadrature mechanism may by summarized as follows:

+ It is extremely simple.

+ It is very robust: the convergence of $Q_{N,n}$ is granted for all smooth integrands. In principle there is no need to treat tricky cases such as highly oscillatory integrands by special methods.

± It even works for non-smooth integrands provided these are bounded. However, the convergence rate will be poor, since the standard quadrature rules Q_n have very low order, if the integrand or some of its derivatives are not continuous.

− It is unsuitable for non-bounded integrands. The sample points may be chosen too close to a singularity leading to numerical blow-up. Generally, singular integrands call for symbolic pre-processing of the integral to eliminate the singularities by some kind of transformation technique.

− Its efficiency depends on the order of the underlying quadrature Q_n, which should be sufficiently large when aiming at high precision results. The data (weights b_i and nodes c_i) of Q_n need to be available. In case of Gauss-Legendre rules Q_n these data have to be computed numerically.

MuPAD's implementation of `quadrature` allows various choices of the basic quadrature rule Q_n. It is amusing to see the result for $\int_0^1 1\,dx$ obtained by the non-adaptive Newton-Cotes (`NC`) rule Q_n with a large number of nodes $n = 100$, say. The order of this rule is $p = 100$. The option `Adaptive=FALSE` causes `quadrature` to return $Q_n(f, a, b)$ immediately:

```
>> DIGITS:=10: numeric::quadrature(1,x=0..1,NC=100,Adaptive=FALSE);

                              2176.0
```

This strange result is not due to a faulty implementation: it just reflects the well known numerical instability of the Newton-Cotes formulae with large n:

```
>> DIGITS:=50: numeric::quadrature(1,x=0..1,NC=100,Adaptive=FALSE);

         0.99999999999999999999999999999999834490395404221 9
```

Consequently, aiming at high precision, one needs to use numerically stable quadrature rules Q_n of high order. The natural candidates are Gauss-Legendre (`GL`) rules with n nodes (order $p = 2n$):

```
>> DIGITS:=10: numeric::quadrature(1,x=0..1,GL=50,Adaptive=FALSE);

                              1.0
```

The default mechanism of *MuPAD*'s `quadrature` is the adaptive version of Gauss-Legendre quadrature with

DIGITS ≤ 10	$10 <$ DIGITS ≤ 50	$50 <$ DIGITS ≤ 100	$100 <$ DIGITS
$n = 20,\ p = 40$	$n = 40,\ p = 80$	$n = 80,\ p = 160$	$n = 160,\ p = 320$

If the integrand is sufficiently smooth, then only a few bisection steps are needed to reach the stopping criterion of the adaptive implementation. The data (b_i, c_i) of the Gauss-Legendre rules with $n = 20, 40, 80, 160$ are pre-computed with 200 significant decimal digits, whence no extra time is needed to generate these data, if the default mechanism with `DIGITS` ≤ 200 is used.

We look at the following test cases:

$$
\begin{array}{ll}
a) & f(x) = \sin(x)\,\exp(\cos(x)),\ x = 0\,..\,\pi\ , \\
b) & f(x) = 1/(1+x^2),\ x = 0\,..\,1\ , \\
c) & f(x) = 1/(1+10^{10}\,x^2),\ x = 0\,..\,1\ , \\
d) & f(x) = \exp(-x^{100}),\ x = 0\,..\,1.1\ , \\
e) & f(x) = x^2\,\sin(x^3),\ x = 0\,..\,10\ , \\
f) & f(x) = \sqrt{x},\ x = 0\,..\,1\ , \\
g) & f(x) = \max(\sin(x), \cos(x)),\ x = 0\,..\,1\ , \\
h) & f(x) = 1/\sqrt{x},\ x = 0\,..\,1\ .
\end{array}
$$

The problems a) and b) are rather trivial, all other problems feature some property making numerical quadrature difficult. In c) the integrand is localized in a very small neighbourhood of $x = 0$. In d) there is a very steep descent from 1 to 0 when passing the point $x = 1$. In e) the integrand is highly oscillatory, a fact that usually calls for special numerical methods. The problems f) and g) are not smooth: the derivative of the integrand has a singularity and a jump discontinuity, respectively. In h) the integrand is unbounded.

The statements used for the quadrature of $\int_a^b f\,dx$ were:

	f(x)	a .. b	digits=10	digits=100	digits=200
a)	$\sin(x)\ \exp(\cos(x))$	0 .. π	0.6 / 0.2 / 0.1	1.5 / 13.6 / 4.5	9.3 / 86.7 / 45.1
b)	$1/(1+x^2)$	0 .. 1	0.0 / 0.0 / 0.1	0.1 / 0.7 / 1.8	0.4 / 29.2 / 21.0
c)	$1/(1+10^{10}x^2)$	0 .. 1	0.4 / 0.7 / 0.1	2.8 / 46.5 / 748.0	10.8 / 229.0 / FAIL
d)	$\exp(-x^{100})$	0 .. 1.1	0.2 / 0.3 / 0.1	5.2 / 43.5 / 421.0	30.6 / 1030.0 / FAIL
e)	$x^2\ \sin(x^3)$	0 .. 10	2.1 / 38.2 / 1.7	23.1 / 1200.0 / 149.0	68.9 / >24 hrs / FAIL
f)	$\sqrt{x}$	0 .. 1	0.3 / 0.2 / 0.1	34.0 / 24.3 / 0.1	277.0 / 137.0 / 0.4
g)	$\max(\sin(x), \cos(x))$	0 .. 1	1.0 / 0.9 / 0.9	225.0 / 318.0 / FAIL	2790.0 / 6950.0 / FAIL
h)	$1/\sqrt{x}$	0 .. 1	3.7 / 0.4 / 0.3	FAIL / 30.3 / 0.5	FAIL / 167.0 / 0.8

Table 5: *Times (in seconds) to approximate $\int_a^b f\,dx$ using MuPAD 1.4/Mathematica 3.0/Maple V.4.*

	DIGITS =10	DIGITS =50	DIGITS =100	DIGITS =500	DIGITS =1000
$n = 100$	4.0	6.7	10.4	38.9	105.0
$n = 160$	0.1	0.1	0.1	70.0	240.0
$n = 500$	159.0	218.0	311.0	1140.0	2900.0

Table 6: *Times (in seconds) to compute* `numeric::gldata(n,DIGITS)`.

MuPAD 1.4:
```
DIGITS:=digits:  numeric::quadrature(f,x=a..b);
```
Mathematica 3.0:
```
NIntegrate[f,{x,a,b},MaxRecursion->10^6,
               PrecisionGoal->digits,
               WorkingPrecision->digits+10]
```
Maple V.4:
```
evalf(Int(f,x=a..b),digits);
```

We note that the option `WorkingPrecision->digits+10` prevented Mathematica to use hardware floats in our test cases. For digits=200 we had to set `WorkingPrecision->digits+60` to obtain quadrature results in Mathematica. The precision of all returned results was a small multiple of $10^{-digits}$. The timings shown in Table 5 indicate that the *MuPAD* implementation performs remarkably well. For the non-smooth case f) Maple managed to trivialize the integrand by a symbolic transformation. Both *MuPAD* and Mathematica carried out the numerical quadrature using the original integrand. In g) Maple did not manage to eliminate the non-smoothness and consequently failed at higher digits. Both *MuPAD* and Mathematica were successful, but due to the non-smoothness the quadrature converged slowly.

Only in the unbounded case h) *MuPAD*'s `quadrature` failed to deliver a result. Because of the slow convergence close to the singularity the recursive depth of `quadrature` reached the admissible limit. Maple managed to trivialize the integrand by a symbolic transformation.

Although the default mechanism of *MuPAD*'s `quadrature` uses only a few Gauss-Legendre schemes Q_n with $n = 20, 40, 80, 160$, precomputed with 200 digits, there is no principal obstacle to aim for higher precision goals using higher order quadrature rules. However, `quadrature` calls the routine `numeric::gldata` to compute the data of the requested Gauss-Legendre rule numerically with the required precision. Some time will be spent on this process before the actual quadrature starts. Due to a remember mechanism in `gldata` this extra time will arise only once in each *MuPAD* session for each choice of n and `DIGITS`. Some timings are given in Table 6. Aiming at high precision result, it is indeed worthwhile to use quadrature rules of higher order than implemented in the default mechanism. For example, the computation

```
>> DIGITS:=1000: numeric::quadrature(1/(1+x^2),x=0..1,GL=500);
```

takes about 2970 seconds, of which about 2900 seconds are spent on computing the corresponding Gaussian data. The default mechanism

```
>> DIGITS:=1000: numeric::quadrature(1/(1+x^2),x=0..1);
```

equivalent to

```
>> DIGITS:=1000: numeric::quadrature(1/(1+x^2),x=0..1,GL=160);
```

needs a total of 3860 seconds. Here only 240 seconds are spent on computing the Gaussian data, but due to the lower order the quadrature itself needs about 3620 seconds.

Solving Ordinary Differential Equations

Runge-Kutta (RK) schemes for solving the initial value problem

$$\frac{d\vec{x}}{dt} = \vec{f}(t,\vec{x}) \ , \quad \vec{x}(t_0) = \vec{x}_0$$

approximate time steps $\vec{x}(t) \to \vec{x}(t+h)$ by a mapping

$$\vec{x}(t+h) \approx \vec{x}(t) + h \sum_{j=1}^{s} b_j \, \vec{k}_j \ ,$$

where the "internal stages" $\vec{k}_1, \ldots, \vec{k}_s$ are defined by

$$\vec{k}_i = \vec{f}\Big(t + c_i\, h \ , \ \vec{x}(t) + h \sum_{j=1}^{n} a_{ij}\, \vec{k}_j\Big) \ , \quad i = 1, \ldots, s \ .$$

Appropriate choices of the Butcher parameters c_i, a_{ij}, b_j lead to various methods of practical interest ([1, 4]). If the matrix a_{ij} is strictly lower triangular, then the intermediate stages $\vec{k}_i$ are easily computed recursively: the method is explicit. Certain "embedded pairs" of RK schemes are known in the literature. These consist of two methods which share the parameters c_i, a_{ij} but differ in the weights b_j. Since the internal stages are the same for both methods, one can compute the result of both schemes for (almost) the same costs arising from each individual scheme. Comparing the results of the two methods one gets a very cheap error estimate for the lower order process.

MuPAD's `numeric::odesolve` implements the class of explicit RK schemes. It calls `numeric::butcher` to obtain the appropriate Butcher data. The default mechanism uses the embedded Runge-Kutta-Fehlberg pair RKF78 with $s = 13$ stages consisting of submethods of order 7 and 8 (see `numeric::butcher(RKF78)`). Using the error estimate of the pair the stepsize is chosen adaptively to keep the local truncation error below a relative precision of $10^{-\texttt{DIGITS}}$. However, no control of the global error is implemented.

The test case is the planar Kepler problem

$$\frac{d^2\vec{x}}{dt^2} = -\frac{\vec{x}}{\|\vec{x}\|_2^3} \ , \quad \text{i.e.}\ , \quad \frac{d\vec{x}}{dt} = \vec{p} \ , \quad \frac{d\vec{p}}{dt} = -\frac{\vec{x}}{\|\vec{x}\|_2^3}$$

with initial conditions $\vec{x}(0) = (1,0)$, $\vec{p}(0) = (0,p)$. The initial momentum $p = 1$ leads to a circular orbit, whereas other values of $p \in (0, \sqrt{2})$ lead to eccentric elliptic orbits with period $T = 2\,\pi\,/(2-p^2)^{3/2}$. We computed one

p	digits	timings	global errors
1	10	3.1 / 1.2 / 3.4	$1.0 \cdot 10^{-11}$ / $3.7 \cdot 10^{-9}$ / $2.5 \cdot 10^{-9}$
10^{-2}	10	15.8 / 13.8 / 14.2	$2.5 \cdot 10^{-8}$ / $1.6 \cdot 10^{-5}$ / $1.5 \cdot 10^{-5}$
10^{-4}	10	29.4 / 25.3 / 24.8	$1.1 \cdot 10^{-3}$ / $6.3 \cdot 10^{-2}$ / $2.3 \cdot 10^{-2}$
1	20	38.4 / 5.9 / 375.0	$1.0 \cdot 10^{-20}$ / $2.0 \cdot 10^{-19}$ / $2.5 \cdot 10^{-19}$
10^{-2}	20	285.0 / 105.0 / 1790.0	$2.4 \cdot 10^{-17}$ / $5.8 \cdot 10^{-15}$ / $1.6 \cdot 10^{-16}$
10^{-4}	20	536.0 / 198.0 / 3180.0	$2.7 \cdot 10^{-13}$ / $2.2 \cdot 10^{-11}$ / $2.4 \cdot 10^{-12}$

Table 7: *Times (in seconds) and global errors when computing one numerical revolution of the Kepler problem using MuPAD 1.4 / Mathematica 3.0 / Maple V.4.*

numerical revolution $t = 0\,..\,T$ for various choices of p. Small values of p correspond to highly eccentric orbits leading close to the center of gravity. There the system becomes quite sensitive to small perturbations, whence the local truncation orders lead to much larger global errors upon return to the starting position. Finally the

$$\text{global error} := \max\Big(\|\vec{x}(T) - \vec{x}(0)\|_2, \|\vec{p}(T) - \vec{p}(0)\|_2\Big)$$

is computed.

We compare the ode solvers of *MuPAD* 1.4, Mathematica 3.0 and Maple V.4. The statements used for the integration of the system were:

MuPAD 1.4:

```
T:=2*PI/(2-p^2)^(3/2);
f:=func([x[3],x[4],-x[1]/(x[1]^2+x[2]^2)^(3/2),
                   -x[2]/(x[1]^2+x[2]^2)^(3/2)],t,x):
DIGITS:=digits:  numeric::odesolve(0..T,f,[1,0,0,p]);
```

Mathematica 3.0:

```
T:=2*Pi/(2-p^2)^(3/2)
X:=NDSolve[{x1''[t]==-x1[t]/(x1[t]^2+x2[t]^2)^(3/2),
             x2''[t]==-x2[t]/(x1[t]^2+x2[t]^2)^(3/2),
             x1[0]==1,x1'[0]==0,x2[0]==0,x2'[0]==p},
            {x1,x2},{t,0,T},MaxSteps->Infinity,
            PrecisionGoal->digits,WorkingPrecision->digits+10]
Timing[x1[T]/.X]
```

Maple V.4:

```
T:=2*Pi/(2-p^2)^(3/2):
System:={ diff(x1(t),t)=p1(t),
          diff(x2(t),t)=p2(t),
          diff(p1(t),t)=-x1(t)/(x1(t)^2+x2(t)^2)^(3/2),
          diff(p2(t),t)=-x2(t)/(x1(t)^2+x2(t)^2)^(3/2),
          x1(0)=1,x2(0)=0,p1(0)=0,p2(0)=p }:
Digits:=digits+10:
X:=dsolve(System,{x1(t),x2(t),p1(t),p2(t)},type=numeric,
          maxfun=-1,relerr=10^(-digits)):
X(T);
```

These settings ensure that the ode solvers control the local truncation error trying to keep it below a relative precision of $10^{-digits}$. According to Table 7 all systems yield similar results at a precision of 10 digits. The *MuPAD* results are slightly more accurate. At 20 digits Mathematica performs best. However, taking into account the smaller errors, *MuPAD* is still competitive. The Maple routines are significantly slower.

The default mechanism of *MuPAD*'s `odesolve` was suitable to perform all the computations. In Mathematica and Maple the options `MaxSteps` and `maxfun` were needed, since the default mechanisms failed to deliver results for digits=20. Also, a suitable working precision needs to be chosen by the user.

Summary

We have demonstrated by a few examples that the new `numeric` package of *MuPAD* 1.4 provides efficient routines for a variety of standard problems. In contrast to Mathematica and Maple no facility to use hardware floats is presently provided by *MuPAD*'s `numeric` library. Consequently, aiming at low precision result, the *MuPAD* routines cannot be as fast as their counterparts in the other systems. However, at higher precision the *MuPAD* implementations are quite competitive. In fact, for many high precision applications *MuPAD* turns out to be the most efficient system.

References

[1] J.C. BUTCHER, *The Numerical Analysis of Ordinary Differential Equations*, John Wiley and Sons, Chichester, 1987.

[2] W. GANDER, *Computermathematik*, Birkhäuser, Basel, 1985.

[3] W.M. GENTLEMAN, *Algorithm 424: Clenshaw-Curtis Quadrature*, Comm. ACM **15(5)** pp. 353-355 (1972).

[4] E. HAIRER, S.P. NØRSETT AND G. WANNER, *Solving Ordinary Differential Equations I: Nonstiff Problems*, Springer, Berlin, 1987.

[5] W.H. PRESS, B.P. FLANNERY, S.A. TEUKOLSKY AND W.T. VETTERLING, *Numerical Recipes*, Cambridge University Press, Cambridge 1992.

[6] J.H. WILKINSON AND C. REINSCH, *Linear Algebra (Handbook for Automatic Computation)*, Springer, 1971.

Eine moderne Vorlesung zur mathematischen Ausbildung von Wirtschaftswissenschaftler

Frank Postel, Gerald Rüscher
Universität-GH Paderborn, {frankp, jerry}@mupad.de

Neue Medien in der Hochschullehre sind in Paderborn nicht nur ein Schlagwort. Im WS 97/98 wurden im Rahmen der Pilotprojekt-Lehrveranstaltung „Mathematik für Wirtschaftswissenschaftler“ neue inhaltliche Schwerpunkte gesetzt und unter Einsatz multimedialer Hilfsmittel vermittelt.

Die Veranstaltung (in) der Vergangenheit

Die Lehrveranstaltung *Mathematik für Wirtschaftswissenschaftler* in Paderborn zählt zu den sogenannten Massenveranstaltungen. Über 400 Studenten und Studentinnen, zumeist im ersten Semester, drängeln sich zweimal pro Woche im großen Vorlesungssaal der Universität Paderborn und ein weiteres Mal in derselben Woche – aufgeteilt in etwa 6 Gruppen – zu den vorlesungsbegleitenden Übungen.

Für die Studierenden ist diese Veranstaltung Pflicht. Die Mathematik – ein Schreckgespenst für viele – stellt für die meisten dieser jungen Leute eine große Hürde dar, die in den vergangenen Jahren nur durchschnittlich 40 bis 50 Prozent erfolgreich überspringen konnten, die wenigsten mit großer Sicherheit.

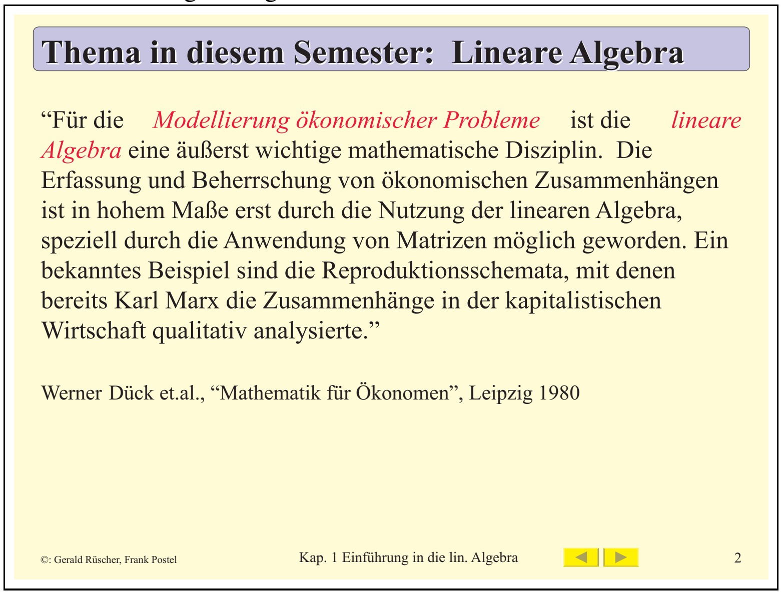
Worin liegen die Ursachen für diese Misere? Gibt es einen signifikanten Leistungsabfall der Studenten zu beklagen? Bilden die Schulen die Studenten hinsichtlich den Anforderungen in der Universität nur unzureichend aus?

Auch die Studenten stellen Fragen, auf die wir oft keine zufriedenstellenden Antworten geben können:

- Brauche ich diese Definition oder ein angeblich so fundamentales Theorem wirklich noch später in meinem Berufsleben?
- Warum soll ich eine Formel auswendig lernen, wenn ich sie doch in vielen Büchern nachlesen kann?
- Warum soll ich das mit der Hand rechnen, wenn ich doch einen Taschenrechner oder meinen Computer zu Hause habe, mit dem alles so einfach und so sicher, nahezu automatisch, geht?
- Was hat diese Mathematik überhaupt mit meinem Berufsziel zu tun?

Die Ursachen sind sicherlich zu komplex, um sie hier diskutieren und beantworten zu können. Wir haben uns aber die Frage gestellt, welche Möglichkeiten die Universitätsausbildung aufzeigen kann, Wege aus dieser untragbaren Situation zu finden.

Es ist sicherlich wichtig, daß ein angehender Ökonom durch die Veranstaltung befähigt wird, sich mit elementaren mathematischen Zusammenhängen auseinanderzusetzen oder sich in solche gar selbstständig einzuarbeiten. Mehr aber noch sollte er/sie in der Lage sein, einem Kollegen aus der Mathematik ein Problem angemessen zu schildern, was eine gewisse Basis gemeinsamer Sprache zur Kommunikation zwischen diesen beiden so unterschiedlichen Fachgruppen voraussetzt. Eine Veranstaltung Mathematik für Wirtschaftswissenschaftler sollte also zur Orientierung, nicht zur Verwirrung beitragen.



Wir glauben, daß eine moderne Lehrveranstaltung befriedigende Antworten auf diese Fragen geben kann, ohne die zu lehrenden Inhalte grundlegend verändern zu müssen. Wichtig sind vielmehr eine moderne Art der Wissensvermittlung, eine stärkere Motivation der mathematischen Sachverhalte und eine stärkere praxisbezogene Auswahl an Inhalten.

Insbesondere hinsichtlich der mathematischen Ausbildung müssen wir uns auf die Frage zurückbesinnen, welche *Grundvorstellungen* von zentralen Begriffen hinsichtlich der Vermittlung mathematischer Sachverhalte bei den Studenten aufgebaut werden sollen.

Aufbau geeigneter inhaltlicher Vorstellungen

Lassen Sie uns die Bedeutung der Konzentration auf geeignete inhaltliche Vorstellungen an einem Beispiel diskutieren. Wir werden einige Möglichkeiten aufzeigen, den Begriff des *Grenzwertes einer Folge* einzuführen und beginnen mit einer streng formalen Definition:

> Es sei $\langle a_n \rangle$ eine Folge. Dann ist g Grenzwert dieser Folge genau dann, wenn gilt:
>
> $$\forall \epsilon > 0 \;\; \exists n_0 \in \mathbb{N} \;\; \forall n \geq n_0 : \;\; a_n \in U_\epsilon(g),$$
>
> wobei $U_\epsilon(g)$ die ϵ-Umgebung von g ist.

Diese Definition (oder eine äquivalente Beschreibung mit Hilfe des Cauchy-Kriteriums) bietet sich an, um an späteren Stellen Beweise (z.B. zu Konvergenzuntersuchungen oder im Rahmen der Differential- und Integralrechnung) zu führen.

Eine ebenso exakte aber etwas weniger formale Beschreibung des Grenzwertbegriffes liefert die folgende Definition:

> Es sei $\langle a_n \rangle$ eine Folge. Dann ist g Grenzwert dieser Folge genau dann, wenn man zu jeder noch so kleinen Umgebung von g stets ein Index n_0 angeben kann, so daß alle Folgenglieder von $\langle a_n \rangle$ ab dem Index n_0 in dieser Umgebung liegen.

Man würde diese Definition nur als Umschreibung der ersten Definition auffassen, man sieht aber bereits hier, daß der Verzicht auf die dem Mathematiker sehr vertrauten Symbole und explizite Verwendung von Existenz- und Allquantoren dem Nichtexperten sympathischer erscheint und erste Hemmschwellen abbauen kann.

Eine dritte Variante stellt die folgende Definition dar:

> Es sei $\langle a_n \rangle$ eine Folge. Dann ist g Grenzwert dieser Folge genau dann, wenn in jeder noch so kleinen Umgebung von g unendlich viele Folgenglieder liegen, außerhalb dieser Umgebung aber nur endlich viele.

Diese Definition liefert sicherlich die beste anschauliche und inhaltliche Vorstellung über den Grenzwertbegriff und eignet sich darüberhinaus für gewisse Beweise – wie beispielsweise die Eindeutigkeit des Grenzwertes – und erste Konvergenzbetrachtungen. Sie ist jedoch ungeeignet für das Beweisen von Konvergenzkritierien oder anspruchsvolleren Konvergenzuntersuchungen.

Es ist üblich, im Rahmen der Mathematik für Wirtschaftswissenschaftler auf die meisten Beweise zu verzichten und die Schwerpunkte auf andere Themen zu setzen wie beispielsweise der Modellierung mathematischer Sachverhalte. Daher sollte man stets abwägen, ob nicht eine exakte, aber anschauliche Formulierung einer streng formalen vorzuziehen ist, insbesondere weil ein strenger Formalismus nur selten dazu beiträgt, das Verständnis seitens der Studierenden zu erhöhen.

Als ein weiteres Beispiel sei die Einführung in die Differentialrechnung genannt. Der Begriff der Ableitung einer Funktion läßt sich als lineares Approximationsproblem, als lokale Änderung einer Größe und rein formal als Grenzwert eines Differenzenquotienten einführen. Die Frage ist also, welche Grundvorstellungen über den Ableitungsbegriff für einen angehenden Ökonom von Bedeutung sind?

Sicherlich wird es an einigen Stellen notwendig sein, eine formale Beschreibung zu liefern (am Beispiel des Grenzwertbegriffes wäre man sicherlich gezwungen, den Begriff der ϵ-Umgebung formal einzuführen). Diese Stellen sollten jedoch hinsichtlich der zu lehrenden Grundvorstellungen gut ausgewählt sein.

Geht man diesen Weg, so ergibt sich ein weiterer unschätzbarer Vorteil: Man gewinnt Zeit!

Betrachten wir beispielsweise das wichtige und insbesondere für die Studenten sehr spannende Kapitel der Linearen Optimierung. Klassische Vorlesungen beginnen – nach einer kurzen motivierenden Einleitung – Begriffe wie „konvexe“ und „konkave“ Punkthülle formal zu definieren. Diesen Begriffen folgen dann eine Reihe weiterer formaler Aspekte, die den Studieren zeigen sollen, warum der das Simplex-Verfahren als der zentrale Algorithmus dieses Kapitels funktioniert. Bereits an dieser Stelle ist die durch die Einleitung aufgebaute Motivation der Studenten meist wieder abgebaut, immer mehr Studenten vermeiden von nun an den Besuch der Vorlesungen.

Abgesehen von der Frage, ob durch Einführung dieser Begriffe wirklich die notwendigen Grundvorstellungen aufgebaut werden, so zeigt die Realität, daß nur selten die Zeit bleibt, spannende und anwednungsnahe Aspekte der Linearen Optimierung wie beispielsweise Transportprobleme (Stepping-Stone-Verfahren) oder Zuordnungsprobleme zu behandeln.

Struktur und Organisation des Projektes

Mit unserem Pilotprojekt wurde eine Veranstaltung angeboten, die parallel zur Vorlesung Mathematik A für Wirtschaftswissenschaftler gelesen wird. Der Pilotcharakter ergibt sich primär aus der Tatsache, daß erstmals zwei getrennte Veranstaltungen zu einem Prüfungsgebiet stattfanden.

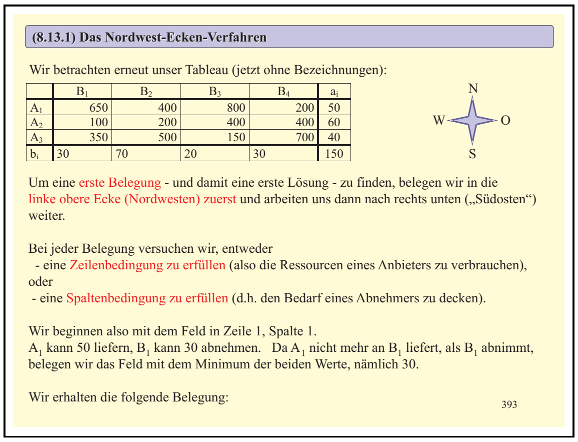
Das Vorlesungsmaterial wurde in Form von PowerPoint-Folien aufbereitet. Es sprachen einige Gründe für die Benutzung von PowerPoint, u.a.:

- PowerPoint stellt zahlreiche Möglichkeiten zur Erstellung von elektronischen Folien-Präsentationen auf einem PC zur Verfügung.
- Die Einarbeitungszeit in PowerPoint ist relativ gering.
- Die Erstellung von Folien ist nicht nur Experten vorbestimmt.
- PowerPoint ist unter Windows 95/NT weit verbreitet.
- Es bietet Möglichkeiten zur Integration von Objekten, die mit anderen Applikationen erstellt worden sind.
- Es gibt Möglichkeiten zur Integration von PowerPoint-Folien in das WWW (u.a. existieren verschiedene, frei erhältliche Plugins für Netscape und den Internet Explorer).

Die Veranstaltung fand in einem gewöhnlichen Seminarraum ohne besondere technische Ausstattung statt. Zu Beginn einer Vorlesung mußten daher Laptop und Beamer, mit denen die PowerPoint-Folien vorgeführt wurden, mitgebracht werden.[1]

Sämtliche Vorlesungs- und Übungsunterlagen wurden auf dem Internet zugänglich gemacht oder wurden auf Wunsch direkt auf Disketten kopiert. Zusätzlich wurden die Unterrichtseinheiten stets in ausgedruckter Form als Kopiervorlage zur Verfügung gestellt.

Für spontane Bemerkungen und eingeschobenen Berechnungen wurde auf die Tafel zurückgegiffen. Die Studenten konnten sich diese Notizen mitschreiben. Damit die Studierenden eine Zuordnung zu einer aufliegenden Folie treffen konnten, wurden die Folien fortlaufend numeriert.



(8.13.1) Das Nordwest-Ecken-Verfahren

Wir betrachten erneut unser Tableau (jetzt ohne Bezeichnungen):

	B_1	B_2	B_3	B_4	a_i
A_1	650	400	800	200	50
A_2	100	200	400	400	60
A_3	350	500	150	700	40
b_i	30	70	20	30	150

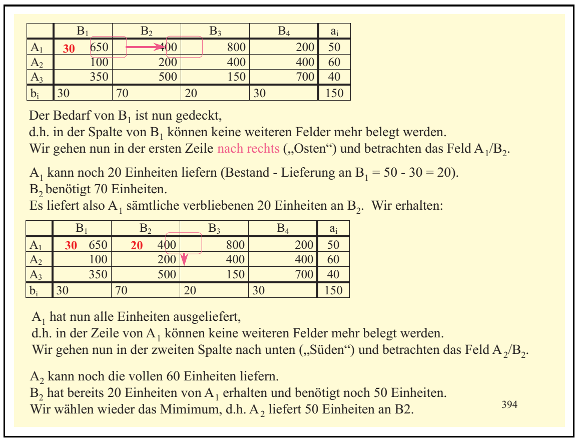
Um eine erste Belegung - und damit eine erste Lösung - zu finden, belegen wir in die linke obere Ecke (Nordwesten) zuerst und arbeiten uns dann nach rechts unten („Südosten") weiter.

Bei jeder Belegung versuchen wir, entweder
- eine Zeilenbedingung zu erfüllen (also die Ressourcen eines Anbieters zu verbrauchen), oder
- eine Spaltenbedingung zu erfüllen (d.h. den Bedarf eines Abnehmers zu decken).

Wir beginnen also mit dem Feld in Zeile 1, Spalte 1.
A_1 kann 50 liefern, B_1 kann 30 abnehmen. Da A_1 nicht mehr an B_1 liefert, als B_1 abnimmt, belegen wir das Feld mit dem Minimum der beiden Werte, nämlich 30.

Wir erhalten die folgende Belegung:

393



	B_1	B_2	B_3	B_4	a_i
A_1	**30** 650	400	800	200	50
A_2	100	200	400	400	60
A_3	350	500	150	700	40
b_i	30	70	20	30	150

Der Bedarf von B_1 ist nun gedeckt,
d.h. in der Spalte von B_1 können keine weiteren Felder mehr belegt werden.
Wir gehen nun in der ersten Zeile nach rechts („Osten") und betrachten das Feld A_1/B_2.

A_1 kann noch 20 Einheiten liefern (Bestand - Lieferung an B_1 = 50 - 30 = 20).
B_2 benötigt 70 Einheiten.
Es liefert also A_1 sämtliche verbliebenen 20 Einheiten an B_2. Wir erhalten:

	B_1	B_2	B_3	B_4	a_i
A_1	**30** 650	**20** 400	800	200	50
A_2	100	200	400	400	60
A_3	350	500	150	700	40
b_i	30	70	20	30	150

A_1 hat nun alle Einheiten ausgeliefert,
d.h. in der Zeile von A_1 können keine weiteren Felder mehr belegt werden.
Wir gehen nun in der zweiten Spalte nach unten („Süden") und betrachten das Feld A_2/B_2.

A_2 kann noch die vollen 60 Einheiten liefern.
B_2 hat bereits 20 Einheiten von A_1 erhalten und benötigt noch 50 Einheiten.
Wir wählen wieder das Mimimum, d.h. A_2 liefert 50 Einheiten an B2.

394

Die Möglichkeit, Gestaltungselemente einer Folie schrittweise ein- und ausblenden zu können (sogenannte *benutzerdefinierte Animationen*), eröffnete bei der Präsentation des Stoffes neue, bessere Möglichkeiten als bei

[1] Im kommenden Sommersemester ist die Fortsetzung des Projektes geplant. Dann stehen Hörsäle und Seminarräume mit fest installierten Beamern zur Verfügung, was den Aufwand für den Tutor reduzieren wird.

gewöhnlichen Hardcopy-Folien. Dabei soll der Begriff „Animation“ nicht mit bunten, sich bewegenden Bildern assoziiert werden. Vielmehr haben wir die Animationen verwendet, um

- Zeilen schrittweise aufzubauen, beispielsweise zur Herleitung einer Formel oder zum Lösen einer Gleichung;
- Text- oder Grafikelemente ein- und auszublenden, beispielsweise um Teilausdrücke farbig einzukreisen oder die Lösung zu einer Frage einzublenden, die zuvor mit den Studenten diskutiert worden ist;
- größere Absätze zeilenweise einzublenden, um ein verzögertes Mitlesen des Textes zu ermöglichen.

Die Tatsache, daß die elektronischen Folien zu Beginn der Veranstaltung bereits vollständig vorbereitet vorlagen, spart Zeit, die bei herkömmlichen Vorlesungen für das Anschreiben verloren geht. Insbesondere bei schreibtechnisch aufwendigen Objekten wie Matrizen oder Simplextableaus ist die Zeitersparnis zum Teil erheblich.

Die gewonne Zeit konnte für zusätzliche Beispiele, für zusätzliche Bemerkungen des Dozenten und für Diskussionen mit und unter den Studenten sinnvoll genutzt werden. Hiermit werden also Möglichkeiten für stärkere Aktivitäten der Studierenden geschaffen, die dem Lernprozeß zu Gute kommen.

Teile der Veranstaltung, insbesondere der Übungen (siehe unten), fanden in PC-Pools statt. Aufgrund begrenzter PC-Ausstattungen mußte die Anzahl an Teilnehmer der Veranstaltung auf ca. 30 Personen begrenzt werden. Die Auswahl der Studierenden aus rund 150 eingegangenen Einschreibungen erfolgte durch Losentscheid. Für die Studierenden präsentierte sich die Veranstaltung sowohl inhaltlich als auch prüfungsrechtlich als eine vollwertige Alternative zur regulären Vorlesung.[2]

Inhalte der Vorlesung

Es wurden schwerpunktmäßig die folgenden Inhalte behandelt:

- Die Matrixschreibweise und Anwendungen in der Ökonomie.
- Die Matrixrechnung und Probleme aus der Ökonomie (Produktionsmatrizen, Kostenmatrizen, etc.).
- Die allgemeine Lösung von linearen Gleichungsssystemen.
- Weitere Eigenschaften von Matrizen: Rang und Invertierbarkeit
- Lineare Optimierung und ihre Anwendungen.
  Simplexverfahren mit Spezialfällen, Transport- und Zuordnungsprobleme.
- Ein Streifzug durch die Analytische Geometrie: Die Geometrie der Ebene und des Raumes.
- Der Funktionsbegriff und Beispiele aus der Ökonomie (Angebots- und Nachfragekurve, Gleichgewichtspreis, etc.).
- Elementare Funktionsklassen.
- Folgen und Reihen, Grenzwerte, Beispiele aus der Finanzmathematik.

Üblicherweise wird die Vorlesung mit der Differentialrechnung beendet, jedoch fielen durch den mehrwöchigen Streik der Studenten im November und Dezember 1997 über 6 Stunden Vorlesungszeit aus!

[2] Mittelfristiges Ziel ist es natürlich, Stil und Inhalte der Pilotprojektveranstaltung in möglichst großem Umfang auf die Hauptveranstaltung übertragen zu können.

Wir haben uns damit an die klassischen Inhalte der Vorlesung gehalten. Neu sind insbesondere die verwendeten Medien und die Auswahl geeigneter Grundvorstellungen (siehe oben) mit den Zielen, den Schwerpunkt auf inhaltliches statt formales Verständnis der Begriffe zu legen, die Gewinnung von Einsicht auf inhaltlicher Ebene zu fördern und bei anwendungsnahen Grundvorstellungen die Fähigkeit zur Modellierung zu schulen.

Über die mathematischen Inhalte hinaus konnten die Studenten die modernen Kommunikationsmöglichkeiten des Internets verwenden. So wurden beispielsweise von einigen Studenten per E-Mail Fragen an die Tutoren gestellt und Lösungen zu Übungsaufgaben an den Übungsleiter geschickt. Von der Möglichkeit, Diskussionsrunden und Nachrichtengruppen für die Studenten aufzubauen, wurde aufgrund des kleinen Teilnehmerkreises nicht Gebrauch gemacht. Verfügen hinreichend viele Studenten über einen Internetzugang, so wäre es sicherlich eine spannende Frage, ob solche Kommunikationsmöglichkeiten von den Studenten im Rahmen einer solchen Veranstaltung genutzt werden würden.

Übungen und der Einsatz eines mathematischen Expertensystems

In jeder Woche erhielten die Studenten ein Übungsblatt mit Aufgaben zu den aktuellen Vorlesungsinhalten. Die Übungsmaterialien wurden mit Microsoft Word erstellt und konnten auf freiwilliger Basis bearbeitet werden. Die Lösungen konnten handschriftlich oder elektronisch – z.B. in Form einer Word-Datei – bearbeitet werden. Die Abgabe konnte auch über E-Mail erfolgen. Musterlösungen wurden vorlesungsbegleitend auf dem Internet veröffentlicht.

In Teilen sollten die Studierenden auch mit dem Computer-Algebra-System *MuPAD* vertraut gemacht werden. Eine intensivere Beschäftigung mit dem System sollte jedoch auf freiwilliger Basis erfolgen. Entsprechende Veranstaltungen wurden daher in einen Block sogenannter *freier Übungen* gelegt. In einer solchen freien Übung ist auch jenen Studierenden ohne eigenen PC die Möglichkeit gegeben, an einem Computer zu arbeiten. Die freien Übungen sollten unter Betreuung eines Tutors stattfinden.

Mit dem Einsatz des Systems *MuPAD* wurden mehrere Ziele verbunden:

- Der Umgang mit einem modernen PC wird geschult. Das betrifft den Umgang mit Windows 95 als auch das Benutzen von Programmen wie Mailtools und Web-Browser.
- Einarbeitung in ein modernes mathematisches Expertensystem, das einem angehenden Ökonomen als ein sinnvolles Werkzeug zur Seite stehen kann.
- Durch *MuPAD* steht dem Lehrenden die Möglichkeit offen, praxisnahe Anwendungen in den Unterricht zu integrieren. Solche Systeme arbeiten mit Matrizen und Gleichungssystemen wie Taschenrechner mit Zahlen und trigonometrischen Funktionen.

Zu unserem Bedauern konnten wir den Einsatz von *MuPAD* in dieser Form nicht realisieren. Uns wurde ein Rechnerraum zur Verfügung gestellt, der über etwa 20 vernetzte PC's verfügt. Die Rechner laufen unter Windows 95. Bekannte Schwächen dieses Betriebssystems im Netzwerkbetrieb und ein mangelhafter Beamer führten zu erschwerten Rahmenbedingungen, die uns letzendlich davon abgehalten haben, die ersten Versuche einer freien Übung am PC fortzuführen.[3]

Der Einsatz elektronischer Folien

Nach Ablauf der regulären Vorlesungszeit können wir stolz auf rund 800 Folien zurückblicken. Dazu kommen 12 Übungsblätter mit entsprechenden Musterlösungen und einer Probeklausur, die etwa zur Halbzeit der Vorlesung

[3] Auch hier versprechen wir uns im Rahmen der Fortsetung des Projektes im kommenden Sommersemester eine Verbesserung, da uns dann die Informatik einen neuen Rechnerraum mit PC's, die unter Windows NT laufen, zur Verfügung stellen wird.

gestellt worden ist.

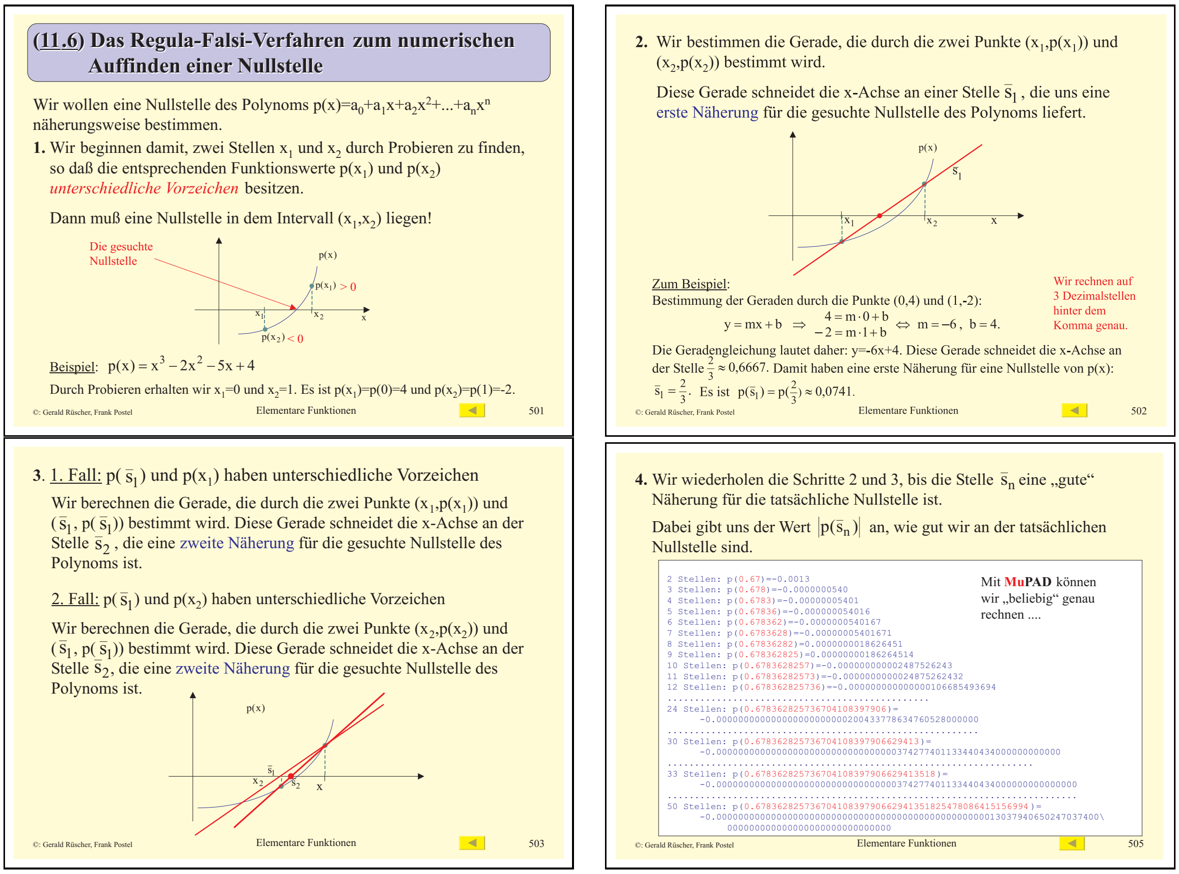
Sämtliche Materialien und die Informationen, die den Studierenden zur Verfügung gestanden haben, sind auf dem Internet unter der Adresse

`http://www.mupad.de/LOCAL/LEHRE_UNIPB/WIWI_WS97/`

einsehbar.

Wir wollen im folgenden kurz über unsere gewonnenen Erfahrungen zur Erstellung und Präsentation der Folien berichten.

Aufwand. Die Erstellung der Unterrichtsmaterialien in Form von PowerPoint-Folien bedeutet zunächst wesentlich mehr Arbeit! Bedenkt man, daß wir zum einen die Inhalte der Vorlesung neu strukturieren – es gab nur sehr wenige Vorlagen, nach denen wir uns richten konnten – und darüberhinaus die Folien anfertigen mußten, so ergab sich zu Beginn des Semesters ein Aufwand von 12 bis 16 Stunden zur Erstellung der Vorlesungsmaterialien für zwei Vorlesungsstunden, die Erstellung der Übungsmaterialien nicht eingerechnet. Jedoch konnte sich der Aufwand nach gewonnener Erfahrung im Umgang mit PowerPoint und durch Verwendung von Teilen bereits erstellter Materialien reduzieren. Bei zukünfitgen Vorlesungen reduziert sich der Arbeitsaufwand aufgrund der nun vorhandenen Materialien natürlich drastisch.



Layout. Wir setzten bestimmte von PowerPoint zur Verfügung gestellten Layoutstile wie Farben, Schriftgrößen, benutzerdefinierte Animationen (siehe oben) und Ein- und Ausblendung von Textpassagen und Grafiken ein. Allerdings achteten wir darauf, daß sich die Folien dem Studenten stets in einem einheitlichen Erscheinungsbild präsentierten, d.h. wir assoziierten bestimmte Aussagen mit entsprechenden Layoutstilen. So setzten wir Definitionen stets in ein rot untermaltes Rechteck, während wesentliche Eigenschaften in blau untermalten Rechtecken präsentiert wurden. Schriftfarben wurden zur Texthervorhebung verwendet.

Die vielen Layout- und Animations-Möglichkeiten, die PowerPoint einem Autor zur Hand gibt, konnten wir nur eingeschränkt nutzen, da es stets möglich sein sollte, die Folien den Studenten in Form eines ausgedruckten Skripts zur Verfügung zu stellen.

Pro und Contra. Durch die sehr gute schriftliche Ausarbeitung der Folien ist man leicht geneigt, während einer Vorlesung inhaltlich zu schnell vorzugehen. Man sollte genügend Zeit zwischen einer Animation oder einem Folienwechsel lassen, Zwischenfragen stellen und auf die Reaktionen der Studenten achten.

Es hat sich gezeigt, daß durch die sorgfältige Vorbereitung und insbesondere die Art der Präsentation einer Unterrichtseinheit etwa 10 bis 30 Minuten pro Vorlesungeinheit – im Vergleich zu einer gewöhnlichen Vorlesungseinheit – gewonnen werden konnte. Diese Zeit sollte man jedoch nicht dafür verwenden, wesentlich mehr Stoff zu behandeln. Vielmehr bietet die restliche Zeit Möglichkeiten zu Diskussionen mit und unter den Studenten, die durch gezielte Fragestellungen oder durch intuitives Herausarbeiten von mathematischen Zusammenhängen angeregt werden können.

Sehr umfangreiche Sachverhalte (ein größeres Beispiel, ein Beweis, etc.) lassen sich in der Regel schlecht mit Folien behandeln, da der Platz einer Folie doch sehr beschränkt ist und man nur eine Folie zu einem Zeitpunkt darstellen kann. Es hat sich gezeigt, daß exzessives Vor- und Rückblättern recht verwirrend wirken kann. Hier kommt es dann darauf an, die Folien gut zu organisieren. Gegebenenfalls müssen Inhalte auf einer Folge von Folien wiederholt werden, um dem Studenten eine (auch anschauliche) Merkhilfe zuliefern.

Da wir bisher sehr wenig Gebrauch von den Hypertextmöglichkeiten in PowerPoint (ab Version 97) gemacht haben, konnten wir zum Teil nur sehr schlecht auf vorhergehende Inhalte verweisen. Ein normales Rück- und insbesondere Vorblättern während einer Vorlesung erwies sich als zu schwerfällig. Wir empfehlen daher die Nutzung der Hypertextmöglichkeiten von PowerPoint, um flexibler durch die Folien navigieren zu können.

MuPAD. Bei unseren Vorbereitungen zu dem Projekt planten wir auch den Einsatz von *MuPAD* in einer Unterrichtseinheit durch direkte Einbindung in die PowerPoint-Folien. Die bisherige Erfahrung hat jedoch gezeigt, daß der dadurch entstehende Kontextwechsel (PowerPoint $\rightarrow$ *MuPAD* $\rightarrow$ PowerPoint $\rightarrow$ ...) zu Irriationen der Studenten führen kann. Unruhe kam auf. Man sollte daher nur an bestimmten Stellen sehr gezielt (und gut vorbereitet) den Einsatz von *MuPAD*[4] innerhalb einer Vorlesung vornehmen.

Es bietet sich sicherlich an, die Unterrichtseinheiten mit *MuPAD* gesondert – beispielsweise in Form einer freien Übung wie innerhalb dieses Projektes geplant – abzuhalten. Damit schafft man zudem den Studenten die Möglichkeit, sich mit diesem Thema zu beschäftigen, die ein wirkliches Interesse zeigen.

An einigen Stellen wurden Ausschnitte einer *MuPAD*-Sitzung in eine Folie integriert, beispielsweise zur Visualisierung von Funktionen oder zur numerischen Auswertung einer Folge[5]. Dazu wurden meist die zugehörigen *MuPAD*-Befehle angegeben, damit interessierte Studenten in der Lage sind, die Beispiele nachzurechnen.

Was ließe sich noch tun? Die Folien und weitere Unterrichtsmaterialien präsentieren sich zur Zeit auf dem Internet exakt in der Form, wie sie während des Pilotprojektes entstanden sind. Hier lassen sich noch einige sinnvolle Erweiterungen und Verbesserungen, nicht nur inhaltlicher Art, vornehmen. So fehlen ein Inhaltsverzeichnis, ein

[4]Oder eines anderen geeigneten Computer-Algebra-Systems

[5]die Berechnung von 100 Folgenglieder mit einer Genauigkeit von z.B. 50 Dezimalstellen erweckt auch bei den müden Studenten in der letzten Reihe große Aufmerksamkeit ...

Gesamtindex und die Einarbeitung von Hyperlinks und Navigationshilfen (siehe oben). Letzteres kann durch Querverbindungen von PowerPoint zu *MuPAD*-Notebooks erweitert werden, so daß auf Mausklick *MuPAD* gestartet und ein entsprechendes Notebook geöffnet wird.

Weiterhin bietet es sich an, die Folien in Dateien nicht nach Vorlesungsterminen sondern nach inhaltlichen Gesichtspunkten zu unterteilen sowie Übungen und Musterlösungen an die entsprechenden Kapitel anzubinden.

Und was meinten die Studenten?

In der letzten Vorlesungswoche haben wir eine ausführliche Evaluation unseres Projektes in Form eines Fragebogens durchgeführt. Wir gliederten die Fragen in folgende Kategorien:

- Mathematisches Vorwissen und Computerkenntnisse
- Allgemeine Bewertung der Vorlesung
- Bewertung der Dozenten
- Fragen zu den Übungen
- Bewertung der elektronischen Folien als Unterrichtsmedium
- Fragen zum Skript (= Sammlung der elektronischen Folien)
- Computereinsatz
- Allgemeines

Das Echo der Studierenden war durchweg positiv bis sehr positiv. Wir möchten im folgenden näher auf die letzten 4 Kategorien eingehen und die Ergebnisse der Umfrage präsentieren.

An der Evaluation beteiligten sich 26 Studenten und Studentinnen, von denen nach eigenen Angaben 22 über einen eigenen *Heim-PC* verfügten und 17 Zugriff auf einen Rechner der Universität hatten. Desweiteren verfügten zur Vorlesungszeit 10 Studenten und Studentinnen über einen eigenen *Internetanschluß*, während 15 Teilnehmer das Internet über die Universität nutzen konnten.

Bewertungen der elektronischen Folien als Unterrichtsmedium.

Frage	1	2	3	4	5	Bewertung
Slides besser als Tafel?	0	0	0	7	19	1=überflüssig, 5=wesentlich besser
Fördern Aufmerksamkeit?	0	3	6	6	11	1=ablenkend, 3=egal, 5=besser zu verfolgen
elektronische Slides vs. Hardcopy	0	0	1	7	18	1=Hardcopy besser, 5=elektronisch besser
Übersichtlichkeit?	0	0	2	12	12	1=konfus, 5=wohlstrukturiert
Farbeinsatz?	0	0	2	11	13	1=verwirrend, 5=sehr nützlich
Einsatz Animationen?	0	1	4	9	12	1=verwirrend, 5=sehr nützlich
Informationsmenge?	0	0	17	7	2	1=zu wenig, 3=angemessen, 5=überladen
Problem der „single slide"	0	7	10	5	4	1=erschwerend, 3=egal, 5=erleichternd
Mehr „Multimedia" gewünscht?	0	0	1	10	15	1=auf keinen Fall, 5=auf jeden Fall

Fragen zum Skript

Frage	1	2	3	4	5	Bewertung
Nutzen des Skripts?	0	0	2	10	14	1=überlüssig, 5=notwendig
Nutzen für Nachbereitung	0	0	2	10	14	1=nicht geeignet, 5=perfekt geeignet
Skript auf Heim-PC genutzt?	15	8	3			1=ja, 2=nein,3=kein PC vorhanden
Internet-Präsenz	2	0	11	1	12	1=nutzlos, 3=egal, 5=sehr nützlich
Internet-Nutzung	11	3	5	7		1=nie, 4=regelmäßig

Computereinsatz

Frage	1	2	3	4	5	Bewertung
Interesse, den Einsatz zu vertiefen?	0	2	8	11	4	1=nein, 5=sehr interessiert
Interesse, *MuPAD* kennzulernen?	0	1	8	12	4	1=nein, 5=sehr interessiert

Allgemeines

Frage	1	2	3	4	5	Bewertung
Computervorkenntnisse	1	7	11	6	1	1=schlecht, 5=sehr gut
Allg. Zufriedenheit	0	0	0	2	23	1=sehr unzufrieden, 5=sehr zufrieden
Geeignet für Audimax?	3	3	6	3	10	1=nein, 3=mit Änderungen, 5=ja

Insgesamt zeigte sich eine sehr kleine Fluktuation an Teilnehmern. Trotz des recht unbeliebten Themas und der – besonders im Wintersemester – unangenehmen Vorlesungszeit von 7:30 (dreimal die Woche), waren in aller Regel mehr als 2/3 der Studierenden in der Vorlesung anwesend. Fairerweise muß an dieser Stelle erwähnt werden, daß nicht nur die modifizieren Schwerpunkte und die neue Art der Vermittlung positiv aufgenommen wurden, sondern daß natürlich auch die kleine Gruppengröße von ca. 30-36 Personen das Pilotprojekt attraktiv machte.

Danksagung

Ein Projekt dieser Größenordnung konnte nicht allein durch die Arbeit der beiden Autoren realisiert werden. Auch andere Personen haben dazu beigetragen, daß diese Veranstaltung zu einem Erfolg für die Studenten und Dozenten geworden ist. Herzlich danken möchten wir:

- Unseren Übungsleiter Torsten Metzner, der viel Zeit und liebevolle Mühe in die Aufbereitung der Musterlösungen investiert hat.
- Claudia Nolte und Kai Widua, die uns bei der Aufbereitung einiger Folien unterstützt haben und wohl am besten wissen, wieviel Aufwand dies bereitet.
- Frau Ingrid Jux, die freundliche Dame aus dem Prüfungssekretariat des FB 5, die uns stets konstruktiv und unbürokratisch zur Seite gestanden hat und uns geholfen hat, die rechtliche Grundlage des Pilotprojekt zu sichern.

MuPAD in Education - The Euclid Programme

Klaus Drescher
SciFace Software GmbH & Co. KG, drescher@sciface.com

Special MuPAD licenses are available for students, teachers, public schools and universities. Additionally SciFace offers the Euclid educational programme featuring free course licenses.

What is the Euclid Programme?

SciFace tries to encourage teachers and lecturers to use computer algebra in their courses by offering the new Euclid programme. This programme offers free course licenses for *MuPAD* for courses held at public schools and universities. Additionally SciFace will install a discussion forum such that lecturers and teachers can share their computer algebra classroom experiences with others.

Why the Euclid Programme was Installed

Computer algebra systems are a fact - here and now. Their impact on working with mathematics and mathematical problem solving will even increase in the future. Computer algebra systems will become standard daily used tools for all branches of science and engineering dealing with mathematics. Teachers and lecturers can not ignore the importance of these systems for the future of their students as they need to be able to work with these systems.

Many teaching issues are still not resolved. No-one is able to offer a simple recipe how to integrate computer algebra into current curricula and courses. Non the less practitioners have found pragmatic ways to successfully use computer algebra for teaching mathematics.

Taking part in the Euclid Programme

Free course licenses for *MuPAD* are available for public schools and universities for the duration of a course or lecture. So teachers and lecturers of such institutions can use *MuPAD* for free during a course and additionally give each student a free copy of *MuPAD* for use at home which may be used during the course.

Only one condition is linked to this offer: You as a teacher or lecturer should tell us about your experiences, about problems you encountered and insights you gained. We want that you share your knowledge with other lecturers and will make your reports available at our Web site `http://www.sciface.com`.

Take a look at our Web site for further information and material about Computer Algebra in Education. You can order course licenses online.

Kalender

CeBIT '98

Hannover, Germany, March 19 –25, 1998

Information:
www.messe.de/cb98/index_e.html

MNU '98

MNU-Jahrestagung 1998,
Leipzig, Germany, April 6 – 9, 1998

Information:
http://www.informatik.uni-leipzig.de/~graebe/lfb/mnu.html

CASC '98

Workshop on Computer Algebra in Scientific Computing,
Saint-Petersburg, Russia, April 20 – 24, 1998

Information:
http://www.pdmi.ras.ru/EIMI/comalg

Computer-Algebra in Lehre, Ausbildung und Weiterbildung

Bayreuth, Germany, April 20 – 24, 1998

International Conference on Teaching of Mathematics

International Conference on Teaching of Mathematics,
Samos, Greece, July 3 – 6, 1998

Information:
http://icg.fas.harvard.edu/~samos98

ISSAC '98

International Symposium on Symbolic and Algebraic Computation,
Rotstock, Germany, August 13 – 15, 1998

Information:
http://wwwteo.informatik.uni-rostock.de/

ICM '98

International Congress of Mathematicians 1998,
Berlin, Germany, August 18 – 27, 1998

Information:
http://elib.zib.de/ICM98

ICTMT-11

New Orleans, November 19-22, 1998

Software-Ergonomie 99

Design von zugänglichen, benutzbaren, integrierten, multimedialen,menschengerechten, virtuellen und realen Informationswelten, 9. Fachtagung
Walldorf (Baden), Germany, March 8 – 11, 1999

Deadline for paper submissions:
May 30, 1998

Information:
http://www.sap.com/germany/se99

Surfin' the Web

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

Die Tips in dieser Ausgabe stammen von Andreas Sorgatz[1] (so), Frank Postel[2] (fp) und Gerald Rüscher[3] (gr)

Computer Algebra

`http://www.sciface.com`
Mit Information zum kommerziellen Vertrieb und Support von *MuPAD* und *MuPAD* Pro für Windows 95/NT.(so)

`http://euler.uni-koblenz.de/`
Insitut für Mediendidaktik. Sie finden hier u.a. Artikel und Poster zur ICTMT-3. (fp)

`http://www.learn-line.nrw.de/Faecher/ma.htm`
Ein Bildungsserver vom Land NRW. (fp)

`http://metric.ma.ic.ac.uk/came/`
CAME: Computer Algebra in Mathematics Education (fp)

`http://www.openmath.org/`
The OpenMath Server. (fp)

Mathematics

`http://www.w3.org/Math/`
The MathML server. (fp)

`http://archives.math.utk.edu/`
MathArchives: Eine umfangreiche Datenbank von Materialien und Software. (fp)

Allerlei

`http://www.whiskyweb.com/maltfile/maltfile_search.html`
Eine der ersten Adressen für Liebhaber eines guten Single Malts. Suchen Sie Ihren *perfect dram* während *MuPAD* integriert. (so)

`http://bahn.hafas.de/`
Die Fahrplanauskunft der Deutschen Bahn AG. (gr)

[1] andi@uni-paderborn.de
[2] frankp@uni-paderborn.de
[3] jerry@uni-paderborn.de

`http://www01.teleauskunft.de/cgi-bin/tron.cgi`
Deutschlandweite Telefonauskunft. (gr)

`http://www.nads.de:82/WWW/PLZ.html`
Postleitzahlenauskunft. (gr)

`http://www.postag.de/postag/service/preise/`
Preisauskunft der Post-AG. (gr)

`http://www.bringdienst.de/`
Nie wieder abends hungrig vor' m Rechner. (gr)

`http://tpsearch.ntnu.no/`
Flotte, web-gestütze Dateisuche (Web-Archie). (gr)