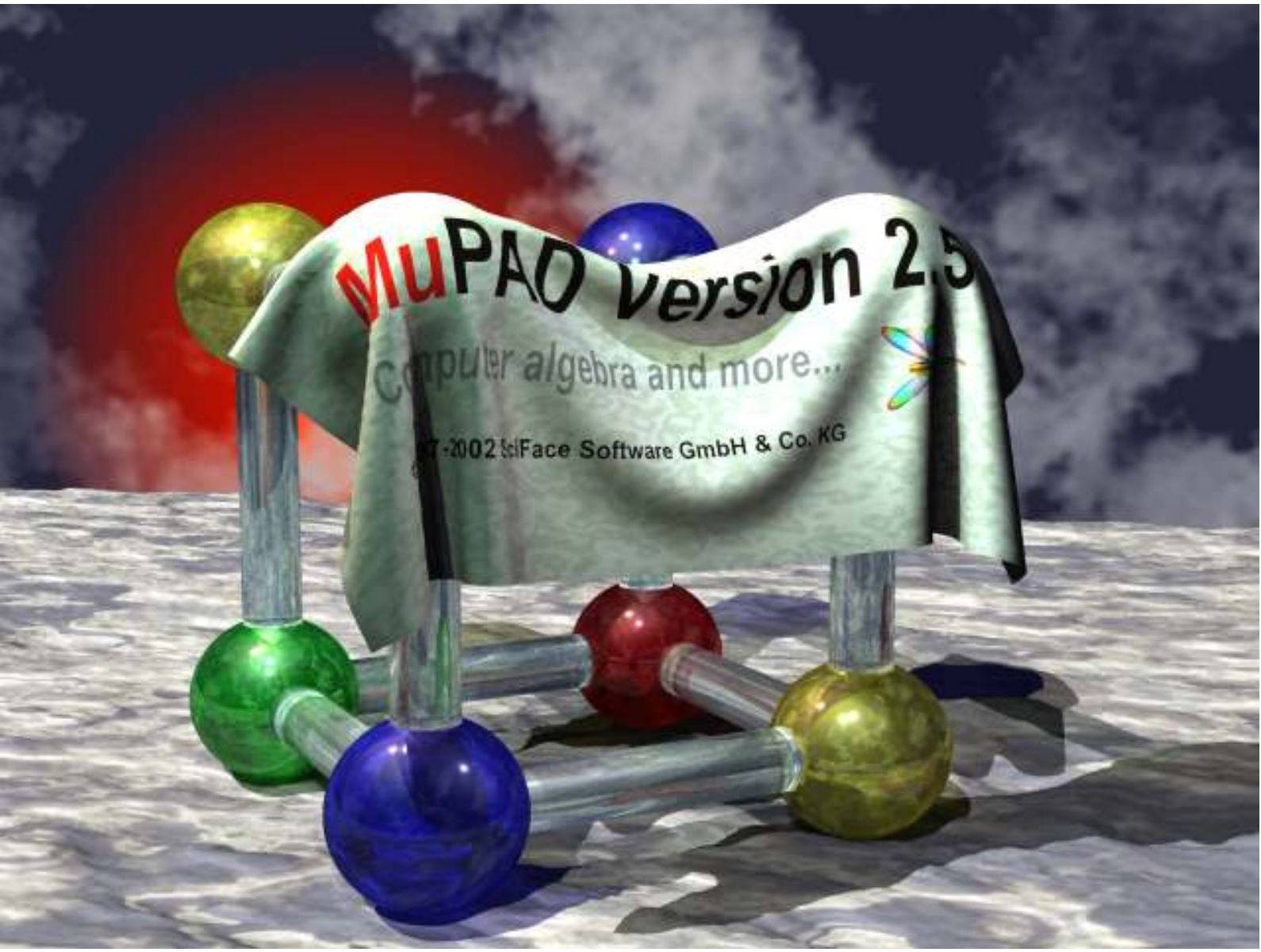
mathPAD

Das MuPAD-Magazin der MuPAD Research Group

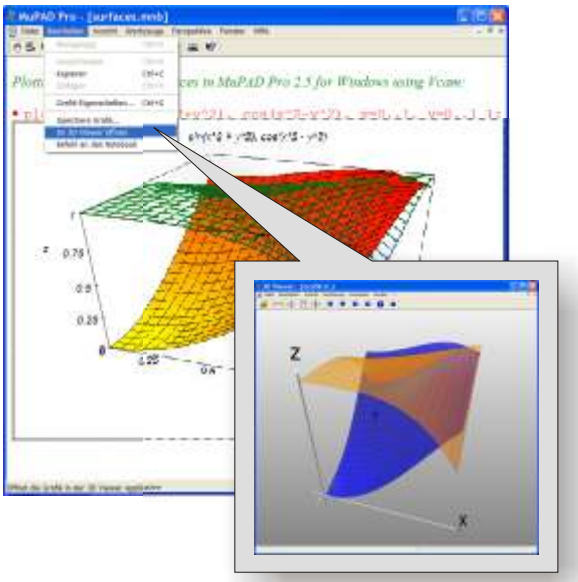
Band 11, Ausgabe 1
August 2002



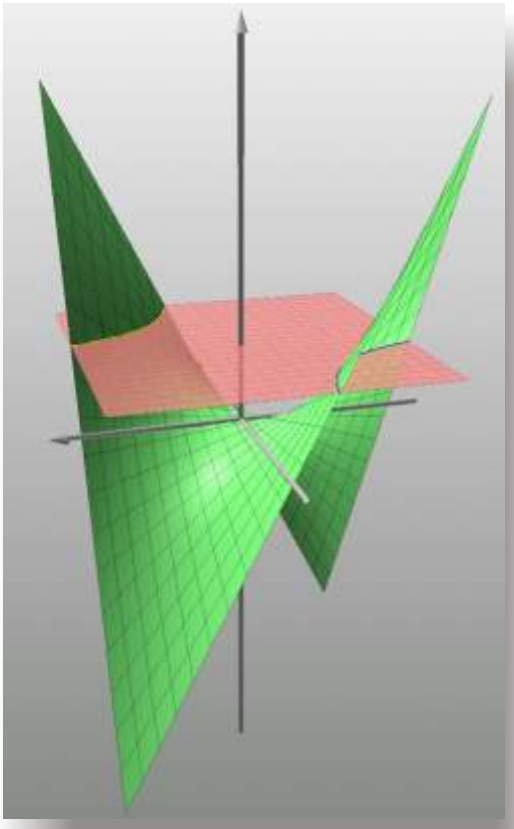
In dieser Ausgabe:

The new 3D-Viewer
in MuPAD Pro 2.5 for Windows:

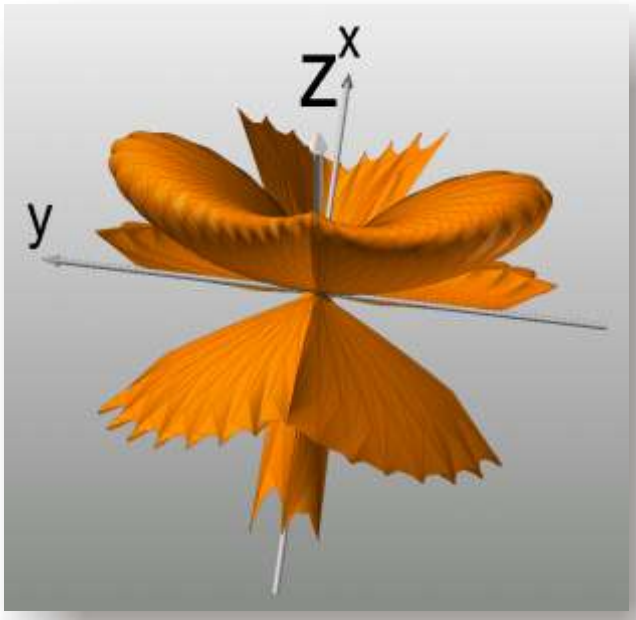
1. Double-click on a MuPAD graphic
2. Open the menu ‘EDIT’
3. Select entry ‘OPEN IN 3D-VIEWER’
4. Explore the beauty of mathematics...



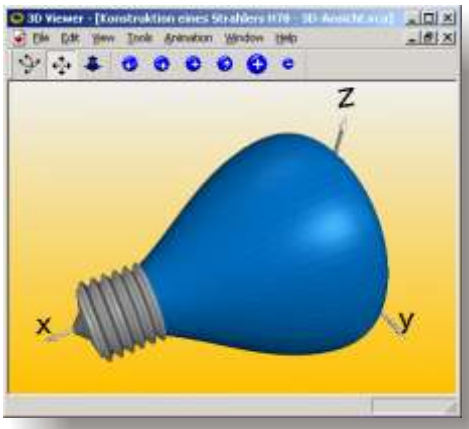
Sine and Cosine Waves



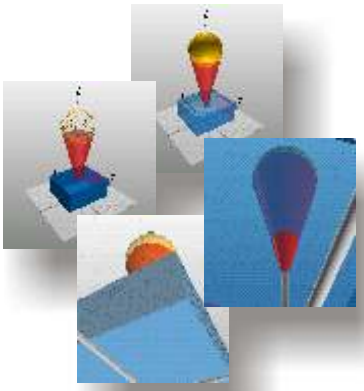
Intersection of Planes: Hyperbola

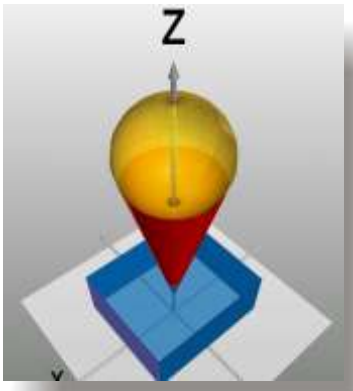


Spherical Object: “Bird”



Application: Electric Lamp





Geometry in R^3

The new dimension in MuPAD Pro 2.5 for Windows !

mathPAD

Band 11, Ausgabe 1, August 2002

In diesem Heft

Redaktion:
Benno Fuchssteiner,
Universität Paderborn
D-33095 Paderborn
email: benno@mupad.de
Andreas Sorgatz
email: andi@mupad.de

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

Liebe Leserin, lieber Leser,

die vorliegende Ausgabe der mathPAD steht ganz im Zeichen des neuen *MuPAD* Release 2.5. Wir berichten über neue Funktionalitäten und Anwendungen sowie über aktuelle Projekte in Forschung und Entwicklung mit *MuPAD*, und um *MuPAD* herum.

Wie bereits in den letzten Ausgaben der mathPAD, so werden sie auch diesmal wieder auf kommerzielle Anzeigen stoßen. Diese sind zur Herausgabe des Magazins notwendig, da die Universität Paderborn die Aufwendungen für eine derart auflagenstarke mathPAD nicht mehr wie in der bisherigen Weise tragen kann. Lassen Sie sich dadurch nicht in Ihrem Lesevergnügen stören.

Wir glauben, auch mit diesem Heft wieder eine interessante Mischung aus englisch- und deutschsprachigen Artikeln mit Informationen zu *MuPAD* und Computeralgebra im allgemeinen zusammengetragen zu haben. Für das Gelingen dieses Heftes möchten wir insbesondere den auswärtigen Autoren herzlich danken!

Das Titelbild dieser Ausgabe zeigt den in Glas modellierten *MuPAD*-Cube. Die Grafik wurde von Christopher Creutzig mit Povray 3.5 auf einem Apple Macintosh generiert.

B.F. & A.S.

Kurz notiert

◇ MuPAD Release 2.5: The new MuPAD version has been released in May 2002. Current download versions are available at `www.sciface.com/download.shtml`. For sales information please refer to the web site `www.sciface.com/sales.shtml`. Some of the most important improvements and changes compared to version 2.0 are:

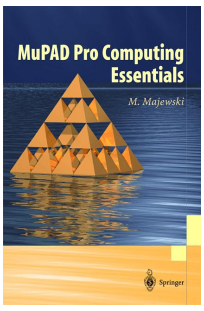
- a new statistics library, `stats`, with about 100 functions. See the article starting on page 57 for details.
- a new `combinat` library including parts of the `http://mupad-combinat.sourceforge.net` package and providing tools for dealing with permutations, subsets, tableaux, compositions, subwords, weighted integer vectors, and more, maintaining a consistent interface for the common types of manipulations.
- an extended `numerics` library with many improvements "under the hood" and integrated with Scilab (optional, see `www.scilab.org`) for fast floating point computations. See the article at page 63 for details.
- a new data type `DOM_INTERVAL` for validated numerical computations, that is, operations with numerical (floating-point) quantities where the results are guaranteed. Type in `?DOM_INTERVAL` to read the details.
- a new data type `Dom::SparseMatrix` which speeds up applications in linear algebra, engineering and numerical research. Read the article on page 69 and type in `?sparsematrix` in MuPAD for details.
- an OpenGL-based 3D-viewer providing a nearly "photorealistic" display of three dimensional objects in MuPAD Pro for Windows. Some examples are printed on the inner side of the cover page.
- HTML-Export of MuPAD-Notebooks under Windows. Using this functionality, users now can write material such as learning units inside MuPAD, export them to HTML format and place them on the web, including typeset formulas, graphics and the like, with just a few mouse clicks. Examples can be found, e.g., on our web site `www.mupad.de/schule+studium/material` (in German language).
- many more... For more information about the improvements with respect to Version 2.0 please refer to the MuPAD online documentation *News and Changes* using the command `?changes` in MuPAD.

◇ MuPAD Pro 2.0 Review: Barry Simon ends his MuPAD Pro 2.0 review in Desktop Engineering, Volume 7, Issue 7, March 2002 with the bottom line: "*MuPAD is a powerful product... If you want a program like this [a Computer Algebra System] around for occasional use, this is an excellent choice...*"

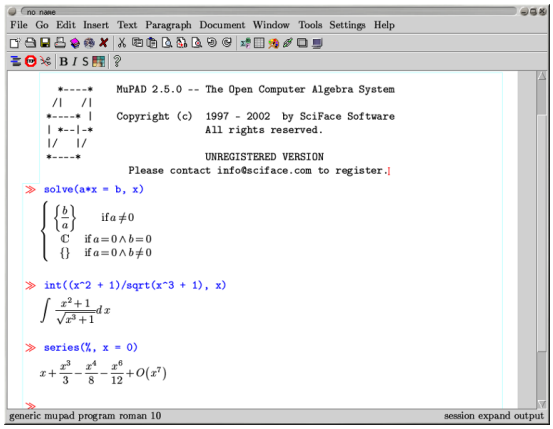
◇ MuPAD as a new Standard: Since spring 2002 the products Scientific Notebook, Scientific Workplace and Scientific Word of MacKichan Software Inc., Seattle, USA are distributed with MuPAD as default math engine. For more informationen refer to `www.mackichan.com`.

◇ Polish MuPAD User Group: On a MuPAD Workshop on July 15.-17. 2002 in Piotrkow Trybunalski, the polish MuPAD User Group started up. The group around Pawel Kwiatkowski and Mirek Majewski develops teaching materials for schools and will translate the user interfaces of MuPAD Pro 2.5 for Windows into Polish. See `www.akademia.piotrkow.pl/MuPAD` or contact `mailto:mupad@akademia.piotrkow.pl`.

◇ MuPAD Pro Computing Essentials: Miroslaw L. Majewski (`mirekmajewski@yahoo.com`), Zayed University, Abu Dhabi, United Arab Emirates, June 2002, Springer-Verlag Heidelberg, Germany, ISBN 3-540-43574-3, 456 Pages, See `www.springer.de`. Prof. Fred Szabo from the Concordia University in Montreal, Canada writes about this book: "*This book is one of the first to provide us with an exciting glimpse into the vast range of possibilities for rethinking what and how we teach in our mathematics courses. The selection of topics in this book is broad enough to satisfy the needs of most college and undergraduate university mathematics majors programs. I am looking forward to introducing my students to new ways of thinking about mathematics using the MuPAD Pro Computing Essentials.*" The book is supported with a special web site called "Selected Examples From the Book" at `www.mupad.com/majewski/examples.html`.



mathPAD



◇ TEXMACS POWERED BY MUPAD: GNU TeXmacs is a free scientific wysiwyg (what-you-see-is-what-you-get) editor with an interface for computer algebra systems. The current program version supports MuPAD 2.0 as well as MuPAD 2.5.

TeXmacs is based on a LaTeX system and is developed as an OpenSource project under Linux. This software can be used, e.g., for publishing scientific mathematical papers on Linux operating systems.

Refer to the web site `www.texmacs.org` for detailed information.

◇ MUPAD INTERACTS WITH MYSQL DATABASE: Your application needs a math engine for symbolic and numerical computing interacting with a full-featured database? Take a look at the new MySQL module interface developed by Christopher Creutzig, University of Paderborn. For detailed information contact the author via `mailto:ccr@mupad.de`.

◇ NEW FEATURES IN THE ODE LIBRARY: New functions has been implemented in MuPAD for solving non-linear Ordinary Differential Equations (ODEs) of first and second order. Five new methods for solving first order ODEs, three derived from Homogeneous equations and the last ones for Abel and Chini equations and a method for finding integrating factors for second order ODEs. Concerning linear ODEs, the work now is focused on the following points: resolution of second order ODEs admitting non-liouvillian solutions and resolution of third (and higher) order ODEs admitting liouvillian solutions. To the end, some tools for linear ordinary differential operators have to be implemented such as computation of invariants and semi-invariants of a given degree, complete factorization and Eigenring. For detailed information please contact the author Olivier Cormier `mailto:cormier@math.uni-paderborn.de`.

◇ MUPAD IN DER MECHATRONIK: Bei der Modellierung mechatronischer Systeme und der damit verbundenen symbolischen Analyse komplexer algebraischer Gleichungssysteme vertraut die Firma iXtronics, Deutschland, ganz auf MuPAD. Als Partner des Mechatronik Laboratorium Paderborn (MLaP) vom AutoMATH Institut der Universität Paderborn ist iXtronics im Projekt “Neue Bahntechnik Paderborn” (`http://www-nbp.upb.de`) involviert. Weitere Informationen zur Firma iXtronics gibt es unter `www.ixtronics.com`.

◇ MUPAD IN SCHULE UND STUDIUM: MuPAD findet zunehmend Verbreitung in der Lehre an Hochschulen, Fachhochschulen sowie in berufs- und allgemeinbildenden Schulen. Als Reaktion auf die vielfältigen Anfragen hat sich im AutoMATH Institut der Uni.-Paderborn das MuPAD “Schule & Studium”-Team formiert. Es arbeitet gezielt für das Anliegen, MuPAD in Schule und Studium lernfördernd einzusetzen und stellt dazu u.a. Infos und Begleitmaterialien auf dem Portal `www.mupad.de/schule+studium` zur Verfügung. Bei Fragen und Anregungen erreichen Sie unser Team unter `mailto:schule@mupad.de`.

◇ LOB FÜR MUPAD: Dr. Karl Sarnow, Gründungsmitglied des BioNet e.V. und des ODS e.V. (1996), dem Träger des Offenen Deutschen Schulnetzes schreibt in seinem Artikel “MuPAD, der Mathe Professor” / “MuPAD 2.0 - Mathe-Ass” im Linux-Magazin, Ausgabe Juni 06 / 2002: “*Dem Schulbereich lassen die MuPAD-Entwickler ganz besondere Pflege angedeihen... Alles in allem handelt es sich bei MuPAD um ein leider seltenes Beispiel hervorragender Software-Entwickung in Deutschland.*”

◇ INITIATIVE ZUR STEIGERUNG DER MEDIENKOMPETENZ: SciFace Software stellt in Kooperation mit dem *MuPAD*-Schule-Team des AutoMATH Institutes der Universität Paderborn allen Schülern und Referendaren, die eine mathematische oder naturwissenschaftliche Facharbeit oder Staatsexamensarbeit mit *MuPAD* bearbeiten und gestalten möchten, eine kostenlose MuPAD Pro 2.5 Lizenz zur Verfügung. Ziel dieser Initiative ist es, vor dem Hintergrund der aktuellen Bildungsstudien, Räume zu schaffen, in denen Schüler und angehende Lehrer ihre Medienkompetenz in den Bereichen Mathematik, Technik und Naturwissenschaft erhöhen können. Weitere Informationen unter www.mupad.de/schule+studium/facharbeit-mit-mupad.

◇ MUPAD - EINE PRAKTISCHE EINFÜHRUNG: Die zweite überarbeitete und erweiterte Auflage des Band 1 aus der Reihe *Mathematik 1× anders – Materialien und Werkzeuge für computerunterstütztes Lernen* steht nun auf dem Web-Portal "MuPAD in Schule & Studium" unter www.mupad.de/schule+studium/literatur/index.shtml#books als PDF-Dokument zum Download bereit. Weitere Ausgaben zur Analysis, Stochastik und E-Technik stehen kurz vor der Veröffentlichung.

◇ MUPAD NEWSLETTER FÜR SCHULE UND STUDIUM: Seit Januar 2002 erscheint 1/4-jährlich der deutschsprachige *MuPAD*-Newsletter (MUN). Neben aktuellen Informationen zu MuPAD und unserem Terminkalender stellt er Begleitmaterialen und Zusatzpakete zu MuPAD vor, die auf dem Web-Portal "MuPAD in Schule & Studium" zur Verfügung gestellt werden. Schreiben Sie uns bitte, wenn Sie selbst Arbeitsmaterialien anbieten oder Lehrerfortbildungen durchführen, damit auch andere MuPAD-Anwender davon profitieren können. Bei Fragen und Anregungen wenden Sie sich bitte an mailto:schule@mupad.de. Der Newsletter wird derzeit rein elektronisch als PDF-Dokument publiziert und kann von folgender Adresse aus dem Web geladen werden: www.mupad.de/schule+studium/news/index.shtml#newsletter.

◇ DAS MUPAD-TUTORIUM: Die zweite überarbeitete und erweiterte Auflage des deutschen MuPAD Tutoriums, ist nun lieferbar (www.springer.de). Der Springer-Verlag beschreibt das 418 Seiten starke Buch mit den Worten: "*MuPAD ist in der Version 2.5 das einzige Computeralgebra-System, welches mit SciLab ein komplettes numerisches Programmpaket integriert hat. Damit wird dem Nutzer insbesondere in der angewandten Mathematik und den Ingenieurwissenschaften ein sehr mächtiges Software-System zur Verfügung gestellt. Das vorliegende MuPAD-Tutorium stellt eine elementare Einführung in dieses System dar. Es richtet sich hauptsächlich an Einsteiger in die Computeralgebra. In einfachen Schritten werden die wichtigsten Bausteine des Systems vorgestellt und an Anwendungsbeispielen demonstriert. Die Benutzung von Systemfunktionen, der Graphik sowie die Programmierung MuPAD's wird an zahlreichen Beispielen eingeübt.*"

◇ MUPAD VERTRIEB: Die ADDITIVE GmbH, Rohrwiesenstraße 2, D-61381 Friedrichsdorf / Ts, Deutschland ist neuer *MuPAD*-Vertriebspartner im deutschsprachigen Raum. Informationen zu Lizenzen, Preisen und Service finden Sie im Web unter www.additive-net.de/mupad.

Die neue DUBBEL-CD: interaktiv nun noch funktioneller und anwendungsfreundlicher

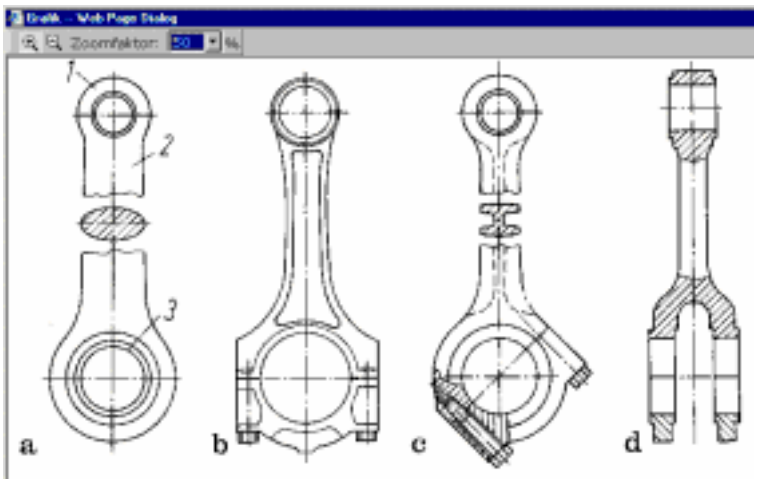
K.-H. Grote und S. Brockop
OvG-Universität Magdeburg, Institut für Maschinenkonstruktion / Konstruktionstechnik
mailto:grote@mb.uni-magdeburg.de

Anfang August 2002 erscheint beim Springer-Verlag die neue DUBBEL-CD – powered by MuPAD. Besonderheit dieser neuen Auflage ist die Möglichkeit, eine innerhalb des DUBBEL auftretende und entsprechend gekennzeichnete Formel per Mausklick zu aktivieren und mit ihr an Ort und Stelle, ohne jegliche Kenntnis im Umgang mit einem Computeralgebra System, einfach zu rechnen. Dazu wird ein MuPAD-Schrittrechner im Kontext der aktuellen DUBBEL-Seite direkt unterhalb der Formel aktiviert. Er unterstützt den Anwender mit einem grafischen Formeleditor sowie mit Menüs und Paletten bei der Ausführung der gewünschten Umformungen und Berechnungen.

Zu den Autoren: Prof. Dr.-Ing. Karl-Heinrich Grote ist der Herausgeber des DUBBEL. Dipl.-Ing. Brockop ist wissenschaftlicher Mitarbeiter am Institut für Maschinenkonstruktion / Konstruktionstechnik.

Seit 1914 ist der DUBBEL, das Taschenbuch für den Maschinenbau, anerkanntes Lehr- und Nachschlagewerk für Studenten und Ingenieure in den produkt- und fertigungsorientierten Fachgebieten. 1 Million verkaufte Exemplare in der 20. Auflage unterstreichen die Bedeutung des technischen Nachschlagewerkes aus dem Springer-Verlag, das der Vielfalt und dem Wissenszuwachs im Maschinenbau Rechnung trägt. Ziel der DUBBEL-Autoren ist es, den lernenden und anwendenden Ingenieur bei der Problemlösung mit neuesten Erkenntnissen zu unterstützen. Aus diesem Grund erfolgte, parallel zur Bearbeitung der 20. Auflage, die Umsetzung der Inhalte in eine interaktive CD-ROM, die dem Benutzer auch alle Möglichkeiten eines elektronischen Nachschlagewerkes bietet. So wird das Auffinden von Stichpunkten durch komfortable Suchfunktionen erleichtert. Hinweise auf andere Kapitel, Literatur und Bilder sind durch Hyperlinks verknüpft. Um den DUBBEL seinen “persönlichen” Erfordernissen anzupassen, kann der Nutzer eigene Notizen und Hyperlinks anbringen.

Der Mehrwert der CD-ROM gegenüber der gedruckten Ausgabe ist das offene Computeralgebra System *MuPAD*. Dieses System verbessert die Alltagstauglichkeit des DUBBEL, da mathematische Lösungsalgorithmen direkt zur Berechnung genutzt werden können. Technische und mathematische Zusammenhänge sind vom Studenten wie vom Ingenieur in der Praxis leichter zu erkennen, komplizierte Gleichungen müssen vom Ingenieur nicht mehr manuell gelöst werden. Somit werden Berechnungsfehler vermieden und es steht mehr Zeit für die eigentlichen Ingenieuraufgaben zur Verfügung. Folgendes Beispiel verdeutlicht die Vorgehensweise unter Anwendung der DUBBEL-CD:



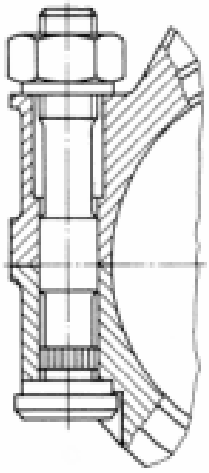
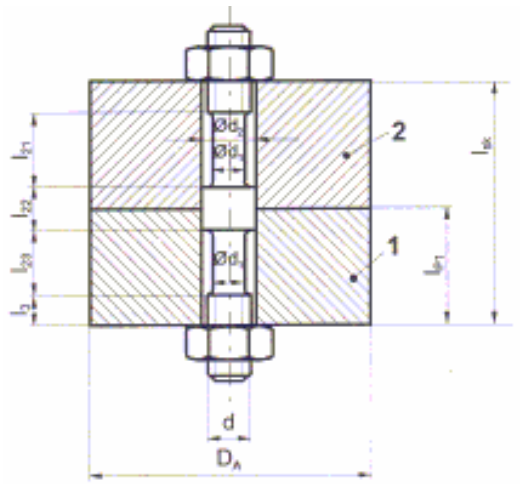


Bild 1 (links): Unterschiedliche Pleuelstangen für verschiedene Einsatzfälle. –
Bild 2 (rechts): Durchsteckschraube in Sonderbauform für Pleuellagerdeckel-Verschraubung.

Es wird eine Pleuelverschraubung ausgelegt, die zunächst nach Bild 1 (übernommen aus DUBBEL P9, Bild 12 a-d) ausgewählt wird und in der Detailzeichnung analysiert, Bild 2, (übernommen aus dem DUBBEL, G40, Bild 56c) - und dann sinnvoll zur Berechnung nach Bild 3 abstrahiert wird. Durch das Anziehen der Mutter entsteht eine Zugkraft in der Schraube, die Vorspannkraft und eine gleich hohe Druckkraft in den “Platten” bzw. Pleuelschalen. Dadurch längt sich der Schraubenbolzen und die Platten werden zusammengedrückt.



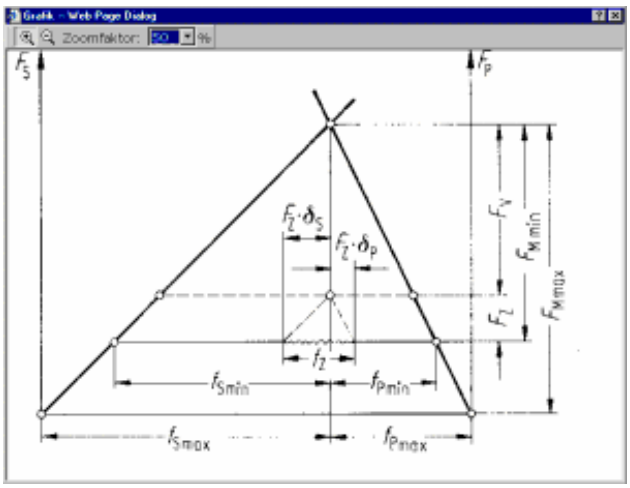
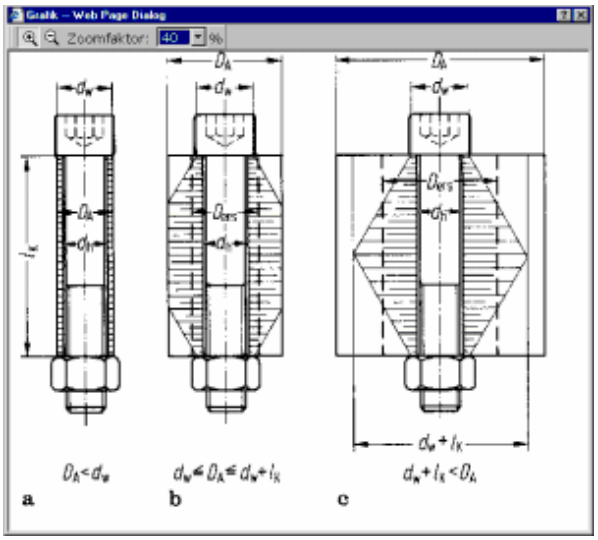


Bild 3 (links): Schraubenverbindung: Abstraktion zur Berechnung.
Bild 4 (rechts): Verspannungsdreieck als grafische Darstellung der Kräfte und Verformungen beim Anziehen. F_s Zugkraft in der Schraube $F_s = F_s(f_s)$; f_s ist Längung der Schraube; F_p Druckkraft in den Platten, $F_p = F_p(f_p)$; f_p ist die Zusammendrückung der Platten; F_M Vorspannkraft bei Montage.

Kräfte und Verformungen nach dem Anziehen richten sich nach der wirksamen Montagekraft F_M. Unter der Annahme linearen Steifigkeitsverhaltens lassen sich die grafischen Einzeldarstellungen der Kraft-Verformungs-Kennlinien für Schraube und Platten in dem sogenannten Verspannungsdreieck zusammenfassen, Bild 4 (Dubbel, G43, Bild 62, hier Screenshot). Mit den angegebenen Bezeichnungen gilt für die Steifigkeit c_S der Schraube $c_S = F_S/f_S$, für die elastische Nachgiebigkeit δ_S der Schrauben $\delta_S = 1/c_S$. Die Steifigkeit der Platten zwischen Schraubenkopf- und Mutternauflage ist $c_P = F_P/f_P$, die elastische Nachgiebigkeit $\delta_P = 1/c_P$ bei zentrischer

Verspannung. Nach dem Anziehen der Mutter gilt für die Montagekräfte in Schraubenbolzen und Platten $F_{MS} = F_{MP} = F_M$. Vorausgesetzt wird, dass Kopf und Mutter vor dem Anziehen allseitig auf den ebenen Platten oder passenden Ansenkungen aufliegen.

Es soll im folgenden mit Hilfe des integrierten Rechners auf der Basis von MuPAD berechnet werden, wie hoch die Nachgiebigkeit sowie die Federsteifigkeit von Schraube und Platten ist. Die Nachgiebigkeit der Platten bei zentrischer Verspannung (δ_P) lässt sich näherungsweise bestimmen, indem man die Nachgiebigkeit des unter einem Winkel α (mit $tan\ \alpha = 0,5$) unter Schraubenkopf und Mutter sich ausbreitenden Doppelkegels mit Bohrung und gleichmäßig verteilter Druckspannung in den einzelnen Querschnitten ermittelt. Für solche Platten gibt auch die VDI-Richtlinie 2230 Näherungsformeln; die Steifigkeit oder die Nachgiebigkeit der Platten werden aus Steifigkeit oder Nachgiebigkeit eines Ersatzzylinders (Bild 5, Screenshot aus DUBBEL G43, Bild 61 a-c) mit einem Querschnitt A_{ers} berechnet. Für den vorliegenden Fall wird Variante b verwendet.



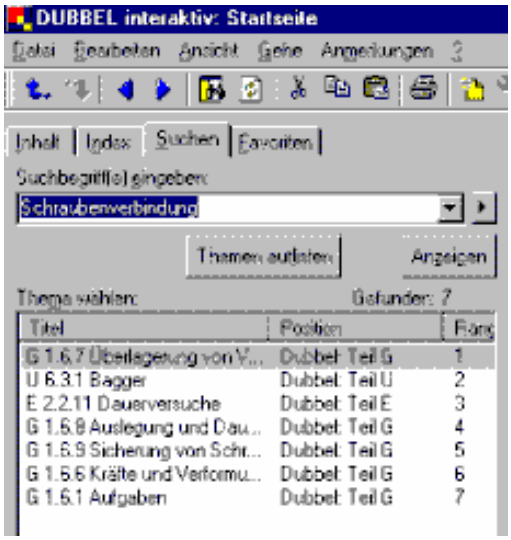
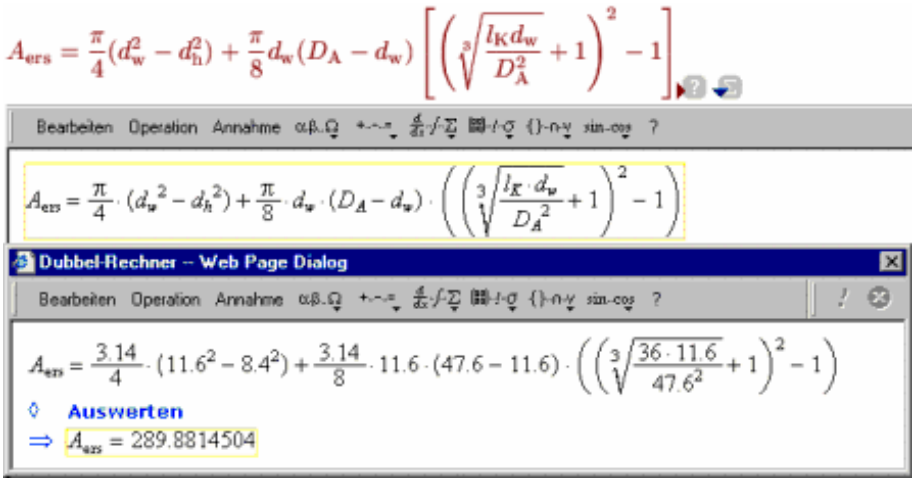
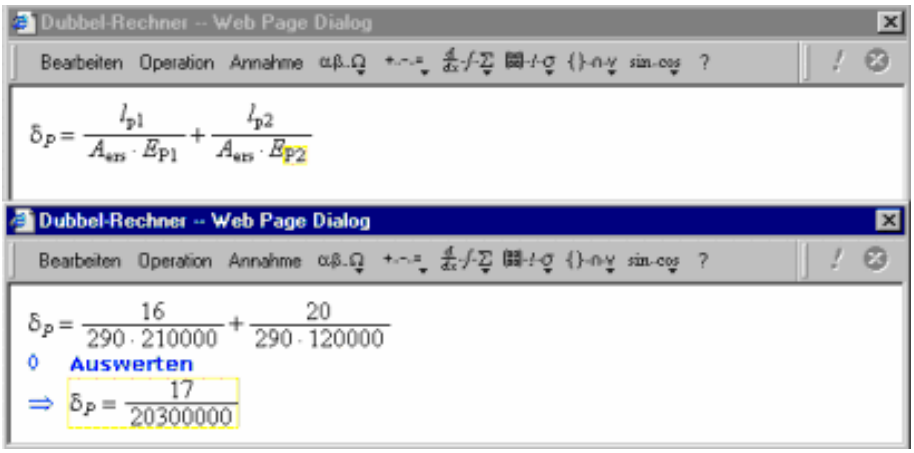


Bild 5 (links): Ersatzdruckzylinder zur Berechnung der elastischen Nachgiebigkeit von verspannten Platten.
Bild 6 (rechts): Screenshot zur komfortablen Stichpunktsuche.

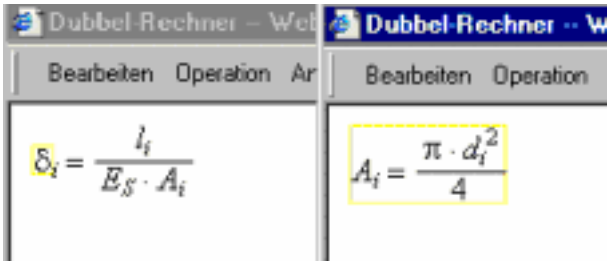
Die aktive Gleichung kann nach einer Suche “Schraubenverbindung” (Bild 6) aufgerufen und alle notwendigen Werte eingesetzt werden.



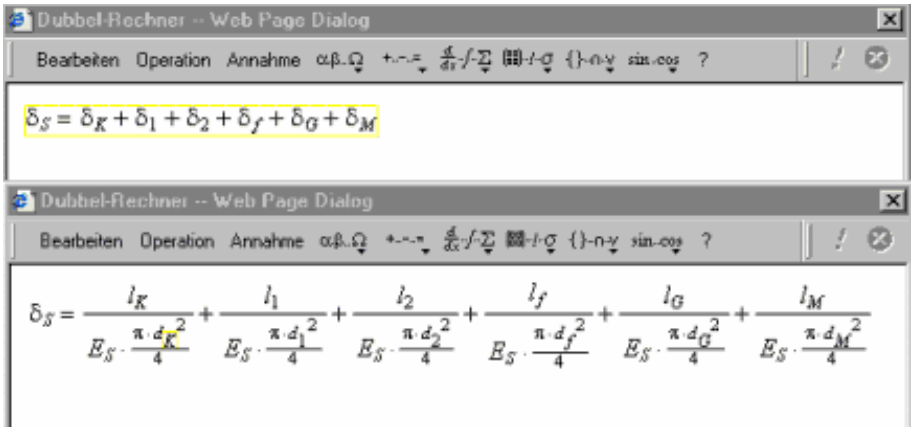
Anschließend werden die Werte in die Gleichung für die Nachgiebigkeit der Platten eingefügt und mit *MuPAD* berechnet.



Anhand der Berechnung der Nachgiebigkeit der Schraube wird demonstriert, wie die Funktionen *Kopieren* und *Einfügen* genutzt werden können. Es muss die Nachgiebigkeit jedes einzelnen Abschnittes berechnet werden. Die notwendige Gleichung sowie die Berechnung des Querschnitts kann im Rechner aufgerufen werden. Anschließend werden die einzelnen Werte addiert.



Das Computeralgebra System *MuPAD* vereinfacht dies ebenfalls, indem es ermöglicht, die einzelnen Formeln zu kopieren und in die Gesamtgleichung einzusetzen.



Die Federsteifigkeit ist der Kehrwert der Nachgiebigkeit und beträgt: für die Platten $c_p = 1,1943 * 10^6 N/mm$ und für die Schraube $c_S = 0,1463 * 10^6 N/mm$.

An die symmetrisch gestaltete und (zentrisch) vorgespannten Schraubenverbindung soll nun eine axiale Zugkraft (Betriebskraft) $F_A = 7200N$ zentrisch unter Kopf und Mutter der Durchsteckschraube angreifen, Bild 3 und 7. Dadurch wird die Schraube um einen Betrag f_{SA} zusätzlich verlängert und die Verkürzung der Platten f_{PA} um den gleichen Betrag vermindert; d.h. Schraube und Platten sind weggleich (parallel) bezüglich der Zugkraft geschaltet, solange kein Klaffen der Schraubenverbindung in der Trennfuge auftritt. Für die Schraubenzusatzkraft (F_{SA}) gilt folgende Beziehung: $F_{SA} = (c_S/(c_S + c_P))F_A = 785,5N$. Sie sagt aus, wie groß die zusätzliche Belastung der Schraube durch die auftretenden axialen Kräfte ist. Ist diese Kraft zu groß, kommt es zum Bruch der Schraube, besonders bei dynamischer Betriebskraft, und die Pleuelstange kann dann keine Kräfte weiterleiten und es würde z. B. ein Motorschaden auftreten. Die Klemmkraft F_{KR} ist die zwischen den Platten wirkende Kraft, sie hält die Pleuelschalen im Beispiel zusammen. ($F_{KR} = F_V - F_{PA}$, mit $F_{PA} = F_A - F_{SA} = 6414,5N$, F_V muss noch mit Hilfe der maximal erlaubten Werkstoffbeanspruchung bestimmt werden, worauf hier verzichtet wird.) Wird diese Kraft F_{KR} zu klein, dann heben die Platten ab und die Verbindung erfüllt nicht mehr ihren Zweck z.B. als Lagerschale. Bei dieser Schraubenverbindung einer Pleuelstange konnte in der weiteren Berechnung nachgewiesen werden, dass es während des Betriebes zu keinem Bruch der Schraube und zu keinem Lösen der Platten kommt. Somit kann diese Konstruktion zur Kraftübertragung genutzt werden.

Bild 7: Videodarstellung zu den wirkenden Kräften in der Schraubenverbindung (Bildsequenz aus Video der interaktiven CD-ROM entnommen) erleichtern das Verständnis für einige technische Zusammenhänge.

Das Beispiel einer Schraubenverbindungsberechnung zeigt, wie der Einsatz von *MuPAD* das Anwendungsspektrum des DUBBEL in der neuen Version der CD-ROM erweitert. Neben dem schnellen Auffinden der notwendigen Gleichungen mit den gezeigten Funktionen ist es möglich, die Berechnung für verschiedene Fälle zügig zu wiederholen, da einmal vorgenommene Modifikationen einfach geändert werden können. Die Arbeit wird erheblich beschleunigt und vereinfacht.

Als Nachschlagewerk ist der DUBBEL aus den Büros und Hörsälen nicht mehr wegzudenken. Nun hat die 20. Auflage des DUBBEL eine neue Dimension mit der interaktiven, verbesserten mathematischen Berechnungssoftware *MuPAD* in die digitale Welt erreicht. Die CD ist ab August 2002 im Buchhandel oder über den Springer-Verlag (`www.springer.de`) erhältlich.

Der Firma SciFace Software sei hier nochmals für die vorbildliche Zusammenarbeit bei der Realisierung der CD gedankt, wie aber auch allen beteiligten Mitarbeiterinnen und Mitarbeitern beim Springer-Verlag, bei der Firma Stürtz und an meinem Lehrstuhl.

Individuelle Leistungskontrolle bei mathematischen Massenvorlesungen

Benno Fuchssteiner
AutoMATH, Universität Paderborn, mailto:benno@mupad.de

Inhalt des Projektes ist ein webbasierter Übungsbetrieb mit individueller Aufgabengenerierung, Fehleranalyse und Leistungskontrolle bei mathematischen Massenvorlesungen für Anwender. Es wurden bisher die Themenbereiche aus der Vorlesung Mathematik I für Informatiker realisiert. Diese umfassen u.a.: Aussagenlogik,Lineare Algebra,Modulares Rechnen,RSA Verfahren,Mengenalgebra, Polynome. In einer unvollendeten Vorversion des Projektes wurden die Themenbereiche der Vorlesung Mathematik I für Wirtschaftswissenschaftler für einen webbasierten Übungsbetrieb aufbereitet.

Wir haben ein System für ein webbasiertes Training mathematischer Fertigkeiten entworfen, als Prototyp realisiert und, nach dem Prinzip des *Blended Learning* als Ergänzung zum klassischen Präsenzübungsbetrieb, im Masseneinsatz erprobt.

Am besten man sieht sich das auf dem Web an (am Ende des Artikels findet man eine kurze Gebrauchsanweisung zur eigenen Erprobung).

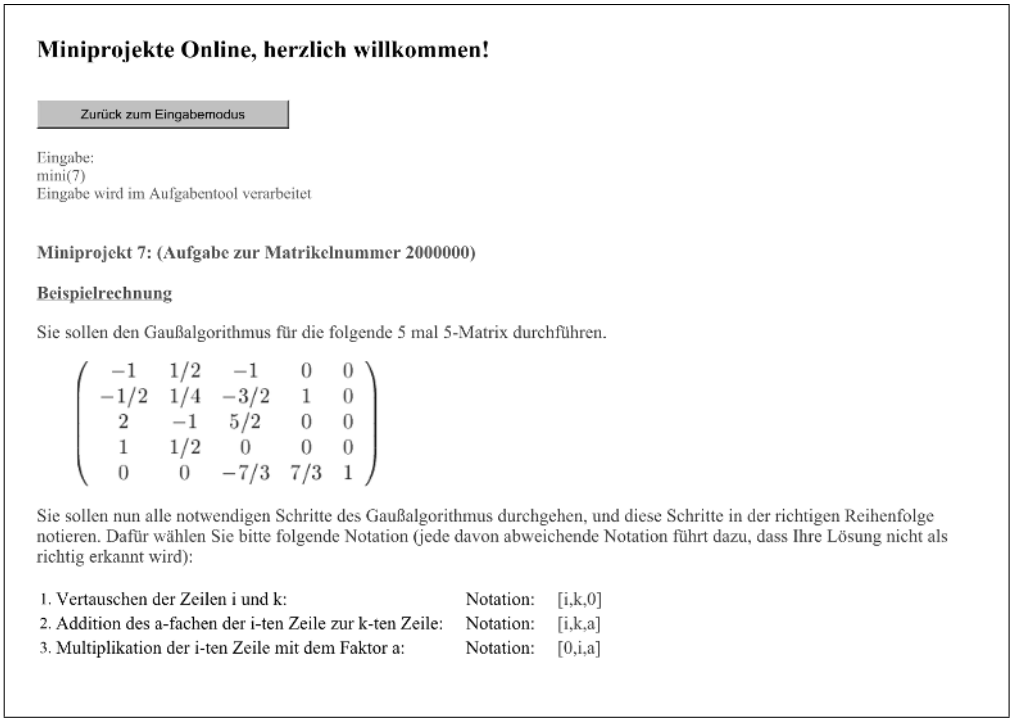
Das System, oder besser, die einzelnen Softwarekomponenten, dienen der Leistungskontrolle und Leistungsförderung bei mathematischen Massenvorlesungen. Inhalte sind ein webbasierter Übungsbetrieb mit individueller Aufgabengenerierung, individueller Fehleranalyse und Leistungskontrolle. Die mathematischen Inhalte des Systems werden im Hintergrund vom Computeralgebrasystem *MuPAD* erzeugt. Es werden beim Nutzer aber keine Kenntnisse der Computeralgebra vorausgesetzt, natürlich auch keine Kenntnisse über *MuPAD*. Das System ist für den Einsatz bei mathematischen Vorlesungen für Anwender ausgelegt und wurde in diesem Bereich bereits mit großem Erfolg erprobt. Nach anfänglicher Skepsis waren die studentischen Stellungnahmen enthusiastisch.

Der webbasierte Übungsbetrieb läuft folgendermaßen ab: Der Studierende registriert sich auf einer Webseite und bekommt für die weitere Arbeit mit dem Online-Übungsbetrieb eine Identität und ein Passwort zugewiesen. Er kann dann im Laufe des Semesters eine vom Dozenten festgelegte Anzahl von Übungsproblemen auf dem Web abrufen. Die einzelnen Probleme gehören zu Themenkreisen, die für alle Studenten gleich sind, das jeweilige Problem für den einzelnen Studenten wird allerdings gemäss dessen Identität während des Abrufs generiert. Durch die Zuteilung individuell generierter Aufgaben wird das Kopieren von Übungsleistungen verhindert.

In der folgenden Abbildung sehen Sie den Anfang der Webseite mit der Aufgabenstellung zum Thema Gaußalgorithmus. Es ist die Aufgabenstellung, welche der Matrikelnummer 2000000 zugewiesen wurde. Die Ausgabe ist Plattform-unabhängig und weitgehend Browser-unabhängig. Eine unangenehme Schwierigkeit stellt die dynamische Formeldarstellung im Web dar. Da es noch keinen Browser gab, welcher MathML rendern konnte[1], mußten

[1] Mozilla 1.0 war noch nicht verfügbar. Die Installation von MathML-fähigen Plugins durch die Studenten sollte vermieden werden.

hier durch einen Hack Graphiken dynamisch eingebunden werden, denn die Formeln mussten ja für alle Studierenden verschieden sein und werden daher erst direkt vor dem Aufbau der Webseite generiert. Einstweilen (und das bleibt noch eine Weile so) werden die Formeln auf folgende Weise erstellt: *MuPAD* erzeugt TEX-Output, TEX erzeugt dann eine dvi-Datei, diese wird dann mehrfach behandelt und am Ende zu einer png-Grafikdatei umgewandelt, welche dann in die Webseite eingebunden wird. Für mich war es überraschend, wie schnell das auch auf einem langsamen Server funktioniert.



Miniprojekte Online, herzlich willkommen!

Zurück zum Eingabemodus

Eingabe:
mini(7)
Eingabe wird im Aufgabentool verarbeitet

Miniprojekt 7: (Aufgabe zur Matrikelnummer 2000000)

Beispielrechnung

Sie sollen den Gaußalgorithmus für die folgende 5 mal 5-Matrix durchführen.

$$\begin{pmatrix} -1 & 1/2 & -1 & 0 & 0 \\ -1/2 & 1/4 & -3/2 & 1 & 0 \\ 2 & -1 & 5/2 & 0 & 0 \\ 1 & 1/2 & 0 & 0 & 0 \\ 0 & 0 & -7/3 & 7/3 & 1 \end{pmatrix}$$

Sie sollen nun alle notwendigen Schritte des Gaußalgorithmus durchgehen, und diese Schritte in der richtigen Reihenfolge notieren. Dafür wählen Sie bitte folgende Notation (jede davon abweichende Notation führt dazu, dass Ihre Lösung nicht als richtig erkannt wird):

1. Vertauschen der Zeilen i und k:	Notation:	[i,k,0]
2. Addition des a-fachen der i-ten Zeile zur k-ten Zeile:	Notation:	[i,k,a]
3. Multiplikation der i-ten Zeile mit dem Faktor a:	Notation:	[0,i,a]

Zusammen mit der Problemstellung wird der/dem Studierenden eine spezielle Abgabeform des Ergebnisses vorgeschrieben. Diese Vorschrift zwingt dazu, auch charakteristische Zwischenergebnisse anzugeben, die natürlich von der jeweiligen Aufgabe abhängen. Eine Abgabe des Endergebnisses reicht i.A. nicht, denn sonst könnte die Lösung ja in vielen Fällen sehr einfach mit einem Computeralgebrasystem erzeugt werden. Die/der Studierende hat dann einen großzügig bemessenen Zeitraum für die Abgabe der Übung. Die Abgabe der Lösung geschieht durch Eintrag in eine Webseite, dabei wird die Richtigkeit der Lösung, wieder bezogen auf die Identität des einzelnen Studenten, überprüft. Falsche Lösungen werden nicht akzeptiert, allerdings werden den Studenten bei fehlerhaften Lösungen Fehleranalysen gegeben. Die Fehleranalyse erlaubt es den Ort des ersten auftretenden Fehlers (oder auch die Orte mehrerer Fehler) zu lokalisieren. Die Fehleranalyse kann auch lokal auf dem eigenen Computer des Studierenden durchgeführt werden, er muß dafür allerdings ein entsprechendes *MuPAD*-Notebook herunterladen (dafür braucht er dann eine MuPAD Lizenz - aber wie gesagt, es geht auch ohne das). Je nach Konfiguration des Übungstools werden Fehlversuche mit Sanktionen belegt, bei dem eingesetzten Prototypen blieben Fehlversuche ohne Sanktionen und wurden einfach nur verworfen (was m. E. die richtige Alternative ist). Bei Abgabe der richtigen Lösung wird diese in ein Protokoll eingetragen. Der registrierte Nutzer kann dann jederzeit seinen Kontostand richtiger Lösungen abfragen.

Das didaktische Konzept, welches dem Projekt zugrunde liegt, ist simpel. Es besteht darin:

- zusätzlich zur Vermittlung eines begrifflichen und methodischen Apparates der Mathematik die mathematischen **Fertigkeiten** zu verbessern,
- in der Erkenntnis dass Wissen, welches nicht angewandt wird, ein frühes Verfallsdatum hat,

- und im Wissen, dass eine Erhöhung des Selbstvertrauens die Fähigkeiten im Umgang mit Mathematik steigert, und dass eine Erhöhung des Selbstvertrauens nur durch die Erfahrung des erfolgreichen Umgangs mit komplexen mathematischen Sachverhalten möglich ist, wofür eigene und vollständige Lösungen gestellter Probleme nötig sind.

Also bestand die Realisierung des didaktischen Konzepts darin, Werkzeuge zu schaffen, mit denen mathematische Fertigkeiten besser und intensiver als bisher trainiert werden können. Mathematische Hausübungen herkömmlicher Art können in Massenvorlesungen diese Aufgabe m.E. nicht erfüllen, da sie, sofern sie in irgendeiner Weise obligatorisch sind, zu fast 80 Prozent abgeschrieben werden, und, sofern sie nicht obligatorisch sind, meist gar nicht gemacht werden. Also müssen Werkzeuge geschaffen werden, die allen Studierenden eigene individuelle Übungsaufgaben zuteilen und die außerdem eine schnelle und umfassende Antwort über die Richtigkeit der Lösung geben.

Das Projekt ist meines Wissens das erste Projekt bei dem eine völlig individualisierte und nicht datenbankabhängige Vergabe, Fehleranalyse und Kontrolle mathematischer Aufgaben und Übungen für große Studentengruppen durchgeführt wird. Als Konkurrenzprojekt ist mir nur das LON CAPA Projekt bekannt (siehe www.lon-capa.org, welches von einem Konsortium amerikanischer Universitäten durchgeführt wird. LON CAPA hat allerdings eine umfangreichere Zielsetzung, ist aber bei der Realisierung nicht so weit wie das hier beschriebene Projekt.

Die Motivation für das Projekt

Ich kämpfe seit Jahrzehnten gegen die nach meiner Meinung unsinnig hohen und ökonomisch unverantwortbaren Abbrecherzahlen in technischen und naturwissenschaftlichen Fächern an deutschen Hochschulen (Details siehe weiter unten). Diese Abbrecherzahlen werden meiner Meinung nach durch unzureichende Fertigkeiten im Umgang mit mathematischen Sachverhalten und mathematisch formulierten Inhalten der Fachwissenschaften verursacht. Diesem Sachverhalt ist durch Verdichtung des mathematischen Begriffs- und Methodenapparates in der Lehre nicht beizukommen, sondern nur durch ein intensives Training der mathematischen Fertigkeiten und des Basiswissens. Ich mache deshalb seit Jahren Eingangstests zur mathematischen Vorbildung von Studienanfängern und zur Nachhaltigkeit der schulischen Ausbildung. Auf der Basis der daraus gewonnen Erkenntnisse führe ich seit 1997 Experimente bezüglich der sinnvollen Nutzung mathematischer Expertensysteme in mathematischen Massenvorlesungen durch.

Eine bessere Ausbildung in Bezug auf die Beherrschung konkreter mathematischer Fertigkeiten an deutschen Hochschulen ist dringend nötig. Wie weiter unten ausgeführt, haben wir in vielen mathematisch orientierten Studiengängen an deutschen Universitäten eine Erfolgsquote von nur 20-40 Prozent, also eine Abbrecherquote von 60-80 Prozent. Da außerdem die Studienzeiten zu lang sind, lohnt es sich einmal einen Blick auf einen der wenigen Studiengänge mit erträglichen Studienzeiten und deutlich geringerer Abbruchquote zu werfen: die Ausbildung der Juristen. Die dort deutlich bessere Ergebnissituation hat damit zu tun, dass neben der Vermittlung des begrifflichen und methodischen Wissens in Vorlesungen eine erhöhte Vermittlung von Fertigkeiten im Umgang mit den primären Ausbildungsinhalten durch die allgemein üblichen Repetitorien stattfindet. Da mathematische Fertigkeiten eine unabdingbare Voraussetzung für das Studium der Natur- und Ingenieurwissenschaften sind, müssen solche Fertigkeiten besser und intensiver als bisher trainiert werden. Es muss gewissermaßen ein mathematischer Drill installiert werden, der allerdings so gestaltet sein muss, dass er nicht abschreckt, sondern den Studierenden Freude bereitet. Der Drill muss also einhergehen mit Maßnahmen zur Erhöhung der Motivation für den Umgang mit mathematischen Sachverhalten, weiterhin muss den Studierenden ein größeres Selbstvertrauen in ihre eigenen Fähigkeiten zur Lösung komplexerer mathematischer Sachverhalte vermittelt werden. Genau diese Effekte werden (und wurden) mit dem hier vorgestellten Projekt erzielt. Was die Erhöhung der Motivation angeht, sei hier ein Beispiel aus dem Wintersemester 2001/2002 angeführt, welches zeigt, welche Begeisterung der webbasierte Übungsbetrieb bei meinen Studenten auslöste: An einem Sonntag Abend konnte ich um 22 Uhr eine neues “Miniprojekt” ins Netz stellen. In der Nacht fiel mir ein Fehler auf, der für einige Spezialfälle auftreten konnte. Damit bei meinen Studenten dieser seltene Fehler nicht auftreten könne, setzte ich mich am Montag Morgen um 6 Uhr an den Rechner, um die Reparatur auszuführen. Zu meiner großen Verwunderung hatten zu dieser Zeit (also während der Nacht an einem Wochenende) schon über 250 Studenten das relativ komplexe Problem bearbeitet.

Eine Verbesserung des Erfolgs mathematischer Lehre für Anwender wurde in diesem Falle also erzielt durch:

- Vermittlung besserer Fertigkeiten im Umgang mit mathematischen Methoden
- Erhöhung der Freude am Lösen mathematische Probleme
- Verkürzung der Korrekturzeiten auf (fast) Null und damit eine Erhöhung der Aktualität von Fehlerhinweisen
- Erhöhung des Selbstvertrauens in die eigenen Fähigkeiten zur Lösung komplexer mathematischer Sachverhalte.

Ich hoffe, dass das System eine effizientere Nutzung personeller Ressourcen und eine bessere Ausbildung in Bezug auf die Beherrschung konkreter mathematischer Fertigkeiten ermöglicht. Die wichtigste strukturelle Auswirkung liegt allerdings in einer Erhöhung studentischer Erfolgsquoten: In vielen mathematisch orientierten Studiengängen haben wir, wie schon oben angeführt, bundesweit eine Erfolgsquote von nur 25-40 Prozent zu verzeichnen, zum Beispiel haben wir in der Informatik eine Erfolgsquote von nur 35 Prozent, oder in den Wirtschaftswissenschaften von ca. 40 Prozent. In der Mathematik selbst liegt sie bei 20 Prozent. Man kann sich darüber unter anderem informieren bei:

`http://www.uni-essen.de/isa/fg_naturwiss/informatik/informatik_hs_frm.htm`

Die bittere Wahrheit ist einfach die: Unsere Universitäten bekommen die intellektuelle Spitze von circa 30 Prozent eines Altersjahrganges, und schicken davon in manchen Fächern bis zu zwei Drittel erfolglos weg. Dies ist ein ökonomischer Malus ersten Ranges. Meiner Meinung nach liegt hier ein ernstes Problem, ein Problem zu dessen rationaler Lösung das Fach Mathematik einen nicht geringen Beitrag leisten kann, denn häufig sind die Schwierigkeiten, die zum Abbruch führen, im Umgang mit Mathematik begründet, nicht nur im Fach selbst, sondern, und vielleicht hauptsächlich, in der mathematisch-strukturellen Durchdringung anderer Fächer. Dieses Problem kann deshalb nur durch eine gezielte Erhöhung der mathematischen **Fertigkeiten** der Studierenden (nicht durch Vergrößerung des mathematischen Begriffsapparates) in Angriff genommen werden. Im Rahmen der Erprobung der Projektarbeiten konnten die Erfolgsquoten in mathematischen Vorexamensklausuren von bisher zwischen 45 und 75 Prozent auf über 90 Prozent gesteigert werden, dies ohne das mathematische Anspruchsniveau zu senken (die Klausuren sind im Web einsehbar). Zudem konnte beobachtet werden, das der Prozentsatz der Studierenden, die sich zum frühest möglichen Zeitpunkt zur Vorexamensklausur meldeten, beachtlich höher als sonst lag (im Wintersemester 2001/2002 meldeten sich von ca. 700 Studienanfängern mehr als 600 zum ersten Klausurtermin an, der nur drei Wochen nach Semesterschluss lag). Es lässt sich m.E. kaum eine größere strukturelle Auswirkung auf unsere Hochschulen denken, als sie durch eine Erhöhung der studentischen Erfolgsquoten gegeben wäre. Da ich weiß, dass eine gezielte Erhöhung von studentischen Erfolgsquoten keineswegs überall auf Beifall stößt, erlaube ich mir hier noch eine Anmerkung: Hohe studentische Erfolgsquoten sind per se nicht ungewöhnlich, zum Beispiel haben wir in England vielerorts eine Erfolgsquote von über 85 Prozent, was sich auch in den vom angelsächsischen Bildungssystem beeinflussten Ländern zeigt (Indien hat z.B. eine Erfolgsquote von mehr als 80 Prozent bei einer sehr, sehr großen Universitätsdichte). Es macht wenig Sinn, aus Ländern, in denen wir zum Teil massiv Entwicklungshilfe leisteten, und die einfach ein anderes und zahlenmäßig effizienteres Ausbildungssystem haben - aber trotzdem kein selektiveres (wie über 1000 Universitäten in Indien zeigen) - Spitzenkräfte für Innovationen im Hochtechnologiebereich gezielt zu importieren und bei uns hingegen einen großen Teil eines Altersjahrganges ohne Hochschulabschluss - zum Teil nach erheblicher Studiendauer - von den Hochschulen wegzuschicken. Man möge dies nicht als Votum gegen eine weltweite Liberalisierung des Arbeitsmarktes verstehen, sondern als Votum dafür, in einem weltweit liberalisierten Arbeitsmarkt erfolgreich mitzuspielen - bzw. unsere Studenten mitspielen zu lassen.

Erprobung

Eine indirekte Evaluation wurde durch Erprobung in der Praxis durchgeführt. Es nahmen im WS 2001/2002 fast 650 Studierende an den individualisierten elektronischen Hausübungen (Miniprojekte) teil, und von diesen wurden über 16000 verschiedene Lösungen abgegeben. Es wurden dabei insgesamt ca. 8000 unterschiedliche Lösungen als richtig anerkannt und den Studierenden gutgeschrieben.

Probieren Sie es aus!

Sie können die Miniprojekte zur Mathematik für Informatiker I ausprobieren ohne MuPAD installiert zu haben. Sie müssen dafür ins Web gehen:

`http://math-www.upb.de/LOCAL/LEHRE_UNIPB/WIWI_WS01/informatik.shtml`.

Dann sehen Sie zuerst einmal in Miniprojekte Info hinein, oder Sie gehen gleich zum Miniprojekte Tool. Dafür brauchen Sie eine Nutzeridentität und ein Passwort. Beides können Sie sich selbst generieren (siehe unten). Zu Ihrer Bequemlichkeit habe ich Ihnen eine Superuseridentität angelegt:

Matrikelnummer: `2000000`, Passwort: `probe12`

Im Tool geben Sie Matrikelnummer und Passwort ein, dann lassen Sie sich z. B. das Miniprojekt 7 zu Ihrer Matrikelnummer durch Eingabe von `mini(7)` erzeugen (Button abschicken aktivieren!). Es gibt gegenwärtig Aufgaben zu 16 Themen. Mit der Superuser Identität können Sie sich Miniprojekte zu anderen Matrikelnummern erzeugen, dies geschieht durch Eingabe von z.B. `mini(7,123456)`, wobei 123456 die gewünschte Matrikelnummer ist. Wenn Sie mathematische Schwierigkeiten oder Verständnisschwierigkeiten haben, dann gehen Sie auf den Link Beispielrechnung. Danach, oder später, lösen Sie die Aufgabe und geben das Ergebnis in der gewünschten Form ein. Wenn Sie eine Lösung abgeben wollen, dann tun Sie dies bitte in der vorgeschriebenen Form in demselben Fenster. Sie können dies als Superuser für jede Studentenidentität tun, ohne Superuser Status können Sie nur Lösungen für die Matrikelnummer abgeben unter der Sie sich eingeloggt haben. Korrekte Abgaben (für fiktive Studenten) finden Sie in den meisten Beispielrechnungen, einige werden unten angeben.

Beim ersten Mal machen Sie wahrscheinlich entweder syntaktische (manchmal auch mathematische Fehler), das Tool gibt Ihnen dazu Hilfen. In zukünftigen Versionen wird die Übungsabgabe weiter vereinfacht.

Wenn Sie Ihren Punktestand abfragen wollen, so geschieht das durch Eingabe von `status`. Als Superuser können Sie auch fremde Punktestände abfragen. Dafür geben Sie einfach `status_1234560` ein, wenn Sie z. B. für den Nutzer mit der Matrikelnummer 1234560 interessieren.

Neue und eigene Identitäten legen Sie sich selbst durch Anklicken der Sektion Orga Übungen, und da dann wieder den Link Orga Übungen, an. Sie sollten sich eine Zahl zwischen 2000000 und 3000000 wählen, die durch 11 oder 10 teilbar ist. Die Zahl darf noch nicht belegt sein (was bei diesen Zahlen auch nicht sehr wahrscheinlich ist). Bei selbst generierten Identitäten können Sie sich neue Passwörter zuteilen lassen, diese gehen dann an die von Ihnen angegebene Email Adresse. Das Superuserpasswort können Sie nicht ändern.

Hier folgen nun einige richtige Abgaben für den Studenten 1234560 (was Sie am zweiten Eintrag der Liste sehen). Sie sollten diese Einträge nutzen und eventuell ändern, wenn Sie sehen wollen, was bei falschen Lösungen passiert.

Korrekte Resultate für den Studenten Nr. 1234560 sind:

```
Problem 1:
 [1,1234560, [{[V, z], [z, q], [1,R], [R, W], [W, 2], [q, 1], [2, V]},
 {[V, W], [z, 2], [1, z], [R,q], [W, 1], [q, V], [2, R]} ]]

Problem 3:
 [3, 1234560, [1,[938054, 642949, 295105, 52739, 31410, 21329,
 10081, 1167, 745, 422, 323, 99, 26, 21, 5, 1, 0]]]

Problem 16:
 [16,1234560,[-2,-3,-1]]

Problem 12:
 [12,1234560, [sqrt(1/2), [[26/55, 2], [-1/2, 1], [1,3], [-13/55, 0]]]]
```

Viel Spaß! Und wenn Sie genügend Punkte erhalten, dann stelle ich Ihnen einen Übungsschein aus.

Einführung in die Finanzmathematik mit MuPAD: Implementierung der Black-Scholes Formel

Elmar Lukas
Universität Paderborn, mailto:Elukas@notes.upb.de

Ziel der Finanzmathematik von Derivaten ist es, eine Antwort auf die folgende Frage zu geben: Was ist ein Optionsrecht bei gegebener Laufzeit und gegebenem Basiswert heute wert? Das theoretische Fundament der Optionspreistheorie bildet das in den siebziger Jahren von Fisher Black (†1995), Myron S. Scholes und Robert C. Merton vorgestellte Modell, welches auf der präferenzfreien Bewertung derivater Finanztitel beruht. Ausgezeichnet mit dem Wirtschaftsnobelpreis 1997, veränderte dieses Modell und die daraus abgeleitete Black-Scholes Formel die finanzmarkttheoretische Forschung nachhaltig. Vor diesem Hintergrund verfolgt der Artikel das Ziel, die Grundgedanken innerhalb der Finanzmathematik von Derivaten aufzuzeigen sowie eine Implementierung der gängigen Black-Scholes Formel unter MuPAD vorzuschlagen.

Einleitung

Schon vor über 2000 Jahren sicherte sich Thales von Milet, so berichten die Schriften des Philosophen Aristoteles[1], das Recht, Olivenpressen während der nächsten Ernteperiode nutzen zu können. Obwohl eher philosophisch motiviert, spiegelt dieses Beispiel schon den Grundgedanken des heutigen Handels mit Derivaten an Finanzmärkten wider, nämlich die gezielte Steuerung von Risiken durch den Einsatz solcher Instrumente.
Um Finanzrisiken handeln zu können, hat die Finanzwelt spezielle Instrumente erfunden, sogenannte Derivate. Unter dem Begriff Derivat versteht man allgemein ein Finanzinstrument, dessen Zahlungsstrom von anderen, ihm zugrundeliegenden Wertpapieren abhängt. Beispiele für derivative Instrumente sind Terminverträge, Swaps und Optionen. Auf letztere soll im Rahmen dieses Artikels näher eingegangen werden. Das mit Optionen verbundene Optionsrecht sichert dem Inhaber das Recht, nicht aber die Verpflichtung, einen bestimmten Vermögensgegenstand (Basisobjekt), z. B. eine Aktie, zu einem vorab festgelegten Preis (Basispreis E) innerhalb oder bei Ablauf einer vorab definierten Frist (Laufzeit T) zu erwerben oder zu veräußern.
Man unterscheidet bei Standard-Optionen (engl. Plain-Vanilla Options) zwischen zwei Arten. Dem Inhaber einer Kaufoption (engl. Call-Option) steht das Recht auf den Kauf des Basisobjektes, dem Inhaber einer Verkaufsoption (engl. Put-Option) das Recht auf Verkauf des Basisobjektes zu. Standard-Optionen weisen noch ein weiteres Differenzierungsmerkmal auf. Kann das Recht, egal ob Verkaufs- oder Erwerbsrecht, während der Laufzeit ausgeübt werden, so spricht man von Optionen amerikanischen Typs. Kann dieses Recht jedoch nur am Ende der Frist ausgeübt werden, so spricht man von Optionen europäischen Typs.

[1] Vgl. [1, S.16].

Allen Optionen gleich ist die Tatsache, dass ein Ausüben des Optionsrechtes nach Ende der Laufzeit nicht mehr möglich ist. Die Option ist dann wertlos. Zur Zeit der Ausübung orientiert sich der Inhaber an der Auszahlungsfunktion des jeweiligen Optionstyps (Put/Call), welche nur vom Wert des Basisobjektes (S) und dem Basispreis (E) abhängt. Liegt bspw. der momentane Wert des Basisobjektes unter dem des Basispreises, so ist es für den Inhaber einer Call-Option günstiger, das Optionsrecht nicht auszuüben.[2] Der resultierende Erlös ist somit null. Ist der Wert des Basisobjektes hingegen größer als der im Vertrag zugesicherte Basiswert, so wird der Inhaber das Optionsrecht ausüben und das Basisobjekt zu dem vereinbarten Preis erwerben. Die Differenz zwischen momentanem Wert des Basisobjektes und dem Basispreis kann der Inhaber durch unmittelbaren Verkauf des Basisobjektes als Gewinn realisieren. Die Auszahlungsfunktion $C_T(S,E)$ kann in *MuPAD* durch den Aufruf

`plot2d(IntrinsicValueP(S,E) $ S=0..N)` für eine Put-Option bzw.
`plot2d(IntrinsicValueC(S,E) $ S=0..N)` für eine Call-Option visualisiert werden.[3]

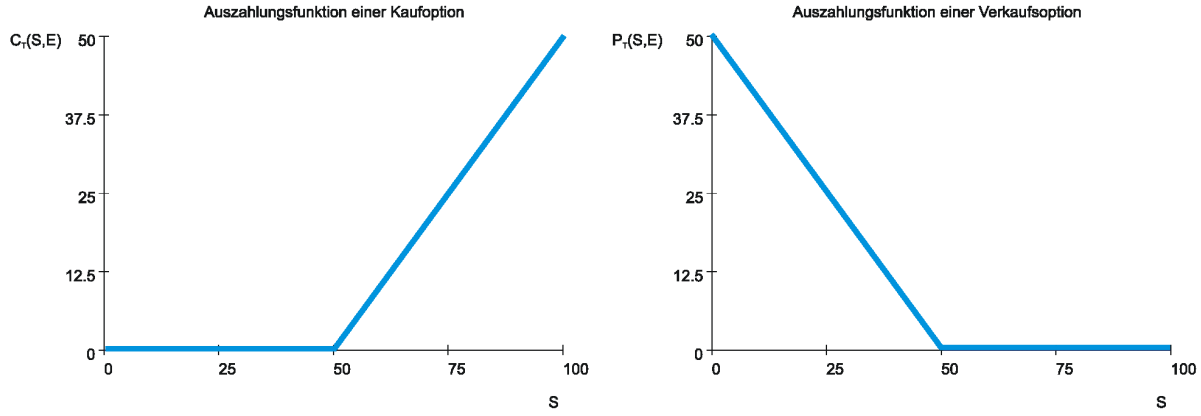


Abbildung 1: Auszahlungsfunktion einer a) Call-Option und b) Put-Option mit $E = 50$ und $N = 100$

Zur Bestimmung des momentanen Wertes eines Optionsrechtes muß die Tatsache berücksichtigt werden, dass dessen Wert eine vom Wert des Basisobjektes S, z. B. einer Aktie, abgeleitete Größe darstellt. Aktienkurse reagieren wiederum auf eine Vielzahl von neuen Informationen durch entsprechende Kursbewegungen. Aussagen über deren zeitlichen Verlauf müssen also entsprechend mathematisch formuliert werden.
Diesem Problem schon sehr nahe gekommen war Louis Bachelier (1870-1946). Als Doktorand des berühmten Mathematikers Henri Poincaré (1854-1912) hatte Bachelier (1900) in seiner Doktorarbeit "*Théorie de la Spéculation*" [3], Aktienkursverläufe mit Hilfe der von Norbert Wiener (1894-1964) mathematisch formulierten Brown'schen Bewegung modelliert. [4] Obwohl die von Bachelier postulierte Verlaufshypothese von Basiswerten seiner Zeit weit voraus war, blieb sie doch lange Zeit unbemerkt.
Über ein halbes Jahrhundert später erfuhr die Idee der Modellierung spekulativer Preisprozesse durch die Arbeiten von Sprenkle (1961) [13], Ayres (1963) [2] und Samuelson (1965) [11] neuen Auftrieb. Jedoch erwies es sich noch immer aus ökonomischer Sicht als problematisch, den Wert, den die Individuen dem immanenten Risiko solcher Finanzprodukte beimessen, "richtig" zu erfassen. 1973 gelang es Fisher Black, Myron Scholes[4] und Robert Merton[9], eine geschlossene Lösung zur Bewertung von Optionen anzugeben. Weltweite Anerkennung erfuhr diese Leistung 1997 mit der Verleihung des Nobelpreises in Wirtschaftswissenschaften durch die königlich schwedische Akademie der Wissenschaften.

[2] Der Inhaber kann das Basisobjekt günstiger, als im Optionsvertrag zugesichert, beziehen.
[3] Für eine Call-Option bzw. Put-Option gilt:

$$C_T(S,E) = \begin{cases} 0 & if \quad S \leq E \\ S-E & if \quad S > E \end{cases}, P_T(S,E) = \begin{cases} S-E & if \quad S > E \\ 0 & if \quad S \leq E \end{cases}$$

[4] Die Brown'sche Bewegung geht auf den schottischen Botaniker Robert Brown (1773-1858) zurück, der beobachten konnte, wie in Flüssigkeit aufgelöste Blütenpollen, angestoßen durch Molekülbewegungen, irreguläre Bewegungen vollziehen.

Aktienkursverlaufshypothese

Vor diesem zeitlichen Hintergrund hat sich die Random-Walk-Hypothese innerhalb der Ökonomik manifestiert. Die Random-Walk-Hypothese besagt, dass zeitlich aufeinanderfolgende Kursänderungen voneinander unabhängig sind und dem Erscheinungsbild einer nach dem Zufallsprinzip erzeugten Zahlenreihe entsprechen. Dies hat zur Folge, dass, ausgehend von den historischen Daten der Kursverläufe, keine Rückschlüsse auf die zukünftigen Ereignisse gezogen werden können. Der jeweils letzte Kurs repräsentiert die bestmögliche Schätzung aller zukünftigen Kurse. Grund dafür ist die These, das zur momentanen Kursfeststellung alle zur Verfügung stehenden Informationen bereits berücksichtigt worden sind (Effizienzthese des Kapitalmarktes).
Die am häufigsten anzutreffenden stochastischen Prozesse, mit denen sich dies modellieren läßt, können unter dem Begriff *Itô-Prozess* subsummiert werden. Ein *Itô-Prozess* hat die Form:

$$dX = a(X,t)dt + b(X,t)dW. \tag{1}$$

Hierbei ist $a(X,t)dt$ der deterministische und $b(X,t)dW$ der stochastische Anteil der SPDE (Stochastic Partial Differential Equation). Der Term dW kennzeichnet das Wiener Inkrement. Es gilt:

$$dW = z\sqrt{dt} \tag{2}$$

mit $z \sim \mathcal{N}(0,1)$ als standardnormalverteilte Zufallsvariable und dt als infinitesimales Zeitinkrement.
In der Finanzmathematik findet die geometrisch Brown'sche Bewegung zur zeitlichen Darstellung von relativen Aktienkursänderungen vielfache Verwendung. Ein solcher Prozess wird als spezieller Fall eines Itô-Prozesses aufgefasst und hat die Form:

$$dS = \mu S dt + \sigma S dW \tag{3}$$

mit μ als Erwartungswert der Rendite einer Aktie und σ als Volatilität der Rendite.
Zur Visualisierung eines derartigen stochastischen Prozesses ist die Erzeugung standardnormalverteilter Zufallszahlen (vergleiche Gleichung (2)) notwendig. In *MuPAD* ist ein solcher Generator nicht implementiert, so dass sich hier die Programmierung einer Prozedur anbietet. Eine dafür geeignete Methode stellt der von Box-Muller (1958) vorgeschlagene Algorithmus dar, welcher sich schnell in *MuPAD* einbinden läßt und durch `Random::Norm()` aufgerufen werden kann.[5]

```
Random::Norm := proc()
   local rho, theta, U1, U2;
begin
   U1 := random(0^DIGITS..3^DIGITS)/float(3^DIGITS);
   U2 := random(0^DIGITS..3^DIGITS)/float(3^DIGITS);

   theta := float(2*PI*U2());
   rho    := float(sqrt(-2*ln(U1())));

   float(rho*cos(theta));
end_proc:
```

[5] Diese Prozedur kann unter *MuPAD 2.5* entfallen, da ab dieser Version standardnormalverteilte Zufallszahlen mittels des Aufrufs `stats::normalRandom(0,1)` generiert werden können.

Eine Diskretisierung mittels einer Euler-Approximation[6] erlaubt die Darstellung des modellierten Aktienkurses im Zeitverlauf.

- ```
  unassign(GeomBrown):
  m:= 800:
  GeomBrown[1]:= 50:    /* Startwert */
  DeltaT:= 1/m:
  mu:= 0.01:
  Sigma:= 0.4:
  for i from 1 to m-1 do
     GeomBrown[i+1]:= GeomBrown[i]+mu*GeomBrown[i]*DeltaT+Sigma
                      *GeomBrown[i]*float(Random::Norm()*sqrt(DeltaT));
     Stock[i]:= float((i-1)*DeltaT);
  end_for:
  ```

Die nachfolgende Grafik zeigt drei Realisierungen des stochastischen Prozesses, welche auf die oben beschriebene Weise generiert worden sind.

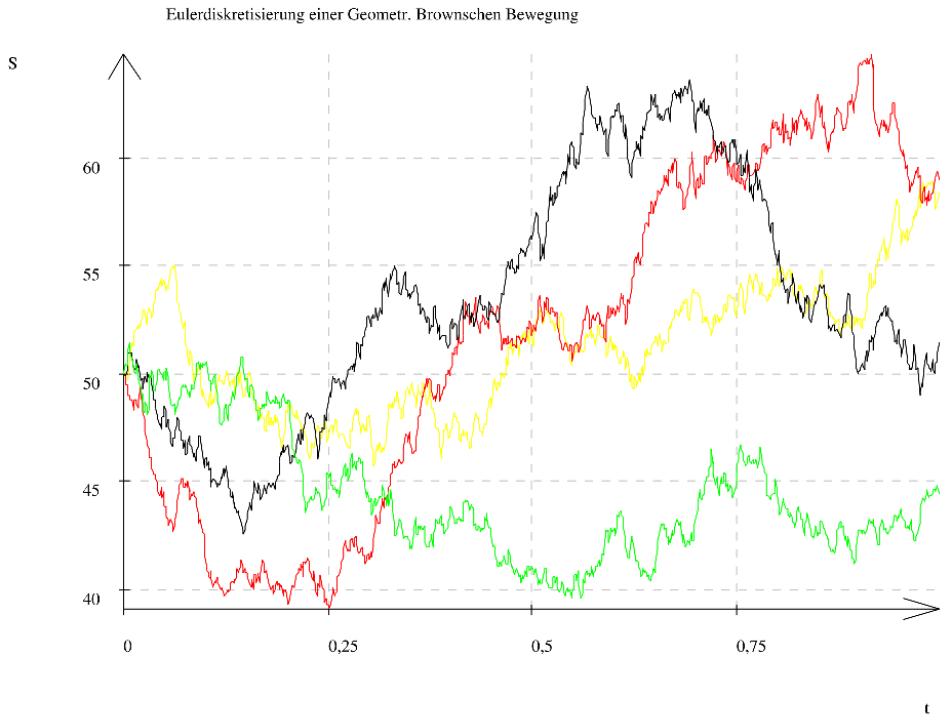


Abbildung 2: Drei Realisierungen eines stochastischen Prozesses mit $\mu = 0.1$, $\sigma = 0.2$, $S_0 = 50$und $\Delta t = 1/800$

Während bislang der Schwerpunkt auf dem Erfassen der Unsicherheit bezüglich zukünftiger Werte bzw. zukünftiger

[6] In der Literatur auch als Euler-Maruyama Approximation bekannt. Eine Euler-Approximation ist ein zeitkontinuierlicher stochastischer Prozess $Y = \{Y(t), t_0 \leq t \leq T\}$ bei gegebener Diskretisierung $t_0 = \tau_0 < \tau_1 < \ldots < \tau_N = T$ innerhalb $[t_0, T]$. Dann folgt aus Gleichung (1): $Y_{n+1} = Y_n + a(\tau_n, Y_n)(\tau_{n+1} - \tau_n) + b(\tau_n, Y_n)(W_{\tau_{n+1}} - W_{\tau_n})$, für $n = 0, 1, \ldots, N$ und $Y_0 = X_0$ als Startwert. Vgl. z. B. [7, S.305 ff.].

[7] Ab MuPAD Version 2.5 verwenden Sie statt `unassign(GeomBrown);` den Befehl `delete GeomBrown;`.
  ```

Auszahlungen eines Basisobjektes lag, bildet bei Optionsrechten die Abhängigkeit von diesen Basiswerten die Grundlage der Bewertung. Somit stellt die Bewertung von Optionen einen relativen Bewertungsansatz dar. Wichtiges Instrumentarium dabei ist das Lemma von Itô, auf das an dieser Stelle nur kurz eingegangen werden kann.[8]
Sei die Aktienkursentwicklung S durch eine geometrische Brown'sche Bewegung gegeben. Ferner sei $F(S,t)$ der Entwicklungsprozess eines derivaten Finanztitels in Abhängigkeit der Zeit t und S. Dann liefert das Lemma von Itô das totale Differential der Funktion $F(S,t)$. Es gilt:

$$dF = \left(\mu S\frac{\partial F}{\partial S} + \frac{\partial F}{\partial t} + \frac{1}{2}\sigma^2 S^2\frac{\partial^2 F}{\partial S^2}\right)dt + \sigma S\frac{\partial F}{\partial S}dW. \tag{4}$$

Herleitung der Black-Scholes Optionspreisformel mit Hilfe der präferenzfreien Bewertung

Das Black-Scholes Modell (1973) beruht auf fünf grundlegenden Annahmen:

1. Das Optionsrecht ist europäischen Typs,
2. der Kurs des Basiswertes folgt einer geometrischen Brown´schen Bewegung,
3. der risikofreie Zinssatz r und die Volatilität σ des Basiswertes sind bekannt und über die Laufzeit konstant. Dividendenauszahlungen des Basiswertes während der Optionslaufzeit werden nicht berücksichtigt,
4. es liegen perfekte Märkte vor und Leerverkäufe[9] sind in jedem gewünschten Umfang zulässig. Es fallen keine Transaktionskosten oder Steuern an,
5. der Handel mit Wertpapieren erfolgt kontinuierlich in Zeit und Menge.

Unter diesen Annahmen konstruierten Black und Scholes ein Portfolio Π, welches aus einer Kaufoption $C(S,t)$ und Δ-Anteilen von Leerverkäufen des Basiswertes besteht.[10]

$$\Pi = C(S,t) - \Delta S. \tag{5}$$

Kleine Änderungen $d\Pi$ des Portfolios innerhalb des Zeitintervalls $[t, t+dt]$ entsprechen kleinen Wertänderungen beider Portfoliopositionen. Somit gilt:

$$d\Pi = dC(S,t) - \Delta dS. \tag{6}$$

Man kann zeigen, dass die Menge Δ an Leerverkäufen

$$\Delta = \frac{\partial C}{\partial S} \tag{7}$$

entsprechen muß, damit die risikoinduzierten Terme dW aus Gleichung (6) sich gegenseitig aufheben.
Da das so kreierte Portfolio nun risikofrei ist, muß ferner dessen Rendite $d\Pi dt$, der einer sicheren Anlage entsprechen, da sonst Arbitragemöglichkeiten existieren. Mit r als risikolosem Zinssatz folgt

$$r\Pi dt = d\Pi dt, \tag{8}$$

[8] Der interessierte Leser sei verwiesen auf z. B. [10].
[9] Der Leerverkauf ist ein Short-Geschäft, bei dem ein Wertpapier mit der Verpflichtung entliehen wird, es nach Ablauf einer Leihfrist zurückzugeben.
[10] Die Gegenposition ΔS bewirkt, dass die Veränderungen des Optionskurses laufend neutralisiert werden. Man spricht in diesem Fall von Delta-Hedging.

was unmittelbar das Duplikationsprinzip der Optionsbewertung verdeutlicht. Einsetzen der Gleichungen (5), (6) und (4) für $dC(S,t)$ in (8) ergibt:

$$r\left(C(S,t)-\frac{\partial C}{\partial S}S\right)dt=\left(\frac{\partial C}{\partial t}+\frac{1}{2}\sigma^2S^2\frac{\partial^2C}{\partial S^2}\right)dt. \tag{9}$$

Durch Umstellen erhält man folgende Differentialgleichung:

$$\frac{\partial C}{\partial t}+Sr\frac{\partial C}{\partial S}+\frac{1}{2}S^2\sigma^2\frac{\partial^2C}{\partial S^2}-rS=0. \tag{10}$$

An dieser Stelle läßt sich die wissenschaftliche Leistung von Black, Scholes und Merton am besten verdeutlichen. Durch die Konstruktion des Portfolios entfallen die individuellen Kurserwartungen μ und Risikopräferenzen: Die Bewertung einer Option erfolgt somit präferenzfrei.
Gleichung (10) repräsentiert eine lineare partielle Differentialgleichung, welche unter den folgenden Randbedingungen[11]:

$$C_T(S,T) = [S-E]^+, \tag{11}$$
$$C(0,t) = 0, \tag{12}$$
$$\lim_{S\to\infty} C(S,t) \sim S \tag{13}$$

eine analytische Lösung besitzt, die als Black-Scholes Formel bezeichnet wird. Es gilt:

$$C(S,t) = SN(d_1)-Ee^{-r(T-t)}N(d_2) \tag{14}$$
$$d_1 = \frac{log\left(\frac{S}{E}+(r+\frac{1}{2}\sigma^2)(T-t)\right)}{\sigma\sqrt{T-t}} \tag{15}$$
$$d_2 = \frac{log\left(\frac{S}{E}+(r-\frac{1}{2}\sigma^2)(T-t)\right)}{\sigma\sqrt{T-t}} \tag{16}$$

mit $N(.)$ als kumulierte univariate Standardnormalverteilungsfunktion.

Einbindung der Black-Scholes Optionspreisformel in *MuPAD*

Um die Black-Scholes Formel in *MuPAD* einbinden zu können, muß zunächst berücksichtigt werden, dass zur Bestimmung des Optionswertes die kumulierte Standardnormalverteilungsfunktion von d_1 und d_2 ermittelt werden muß. Diese Verteilungfunktion fehlt leider unter *MuPAD* 2.0,[12], was jedoch kein allzu großes Problem darstellt, da die Berechnung der Verteilungsfunktion über die in *MuPAD* implementierte Fehlerfunktion `erf()` erfolgen kann. Es gilt:[13]

$$N(z) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{z}e^{-\frac{1}{2}y^2}dy \tag{17}$$
$$= \frac{1}{2}\texttt{erf}\left(\frac{z}{\sqrt{2}}\right)+\frac{1}{2}. \tag{18}$$

Nachfolgend ist die Prozedur zum Bewerten von europäischen Kaufoptionen aufgeführt:[14]

[11] Hier exemplarisch für eine Call-Option. Für eine Put-Option vgl. z. B. [12, S.37ff].
[12] In *MuPAD* 2.5 ist die kumulierte Standardnormalverteilungsfunktion bereits als Funktion `stats::normalCDF` implementiert.
[13] Vgl. [8, S. 630].
[14] Der Ausdruck $(T-t^*)$ in Gleichung (14) wurde durch die Restlaufzeit t substituiert.

```
• BlackScholes::CallEurop := proc(S,E,sigma,r,t)
    local d1, d2, a, b;
  begin
    d1:= (ln(S/E)+r*t)/(sigma*sqrt(t))+1/2*sigma*sqrt(t);
    d2:= d1-sigma*sqrt(t);
    a := erf(d1/sqrt(2))/2+1/2;
    b := erf(d2/sqrt(2))/2+1/2;
    float(S*a-E*exp(-r*t)*b);
  end_proc:
```

Über die Put-Call Parität kann analog der Wert einer europäischen Put-Option bestimmt werden.[15] Hierzu muß vom Wert der europäischen Call-Option der Term `-S+E*exp(-r*t)` subtrahiert werden. Unter Einbeziehung der Put-Call Parität lautet die Prozedur zum Bewerten von europäischen Put-Optionen:

```
• BlackScholes::PutEurop := proc(S,E,sigma,r,t)
  begin
    float(BlackScholes::CallEurop(S,E,sigma,r,t)-S+E*exp(-r*t));
  end_proc:
```

Das Ergebnis läßt sich in *MuPAD* z. B. mit Hilfe des `plot3d()`-Befehls darstellen. Die nachfolgenden Grafiken verdeutlichen die Abhängigkeit des Optionswertes vom Basiswert S und der Restlaufzeit t für den Fall einer Verkaufs- und Kaufsoption.

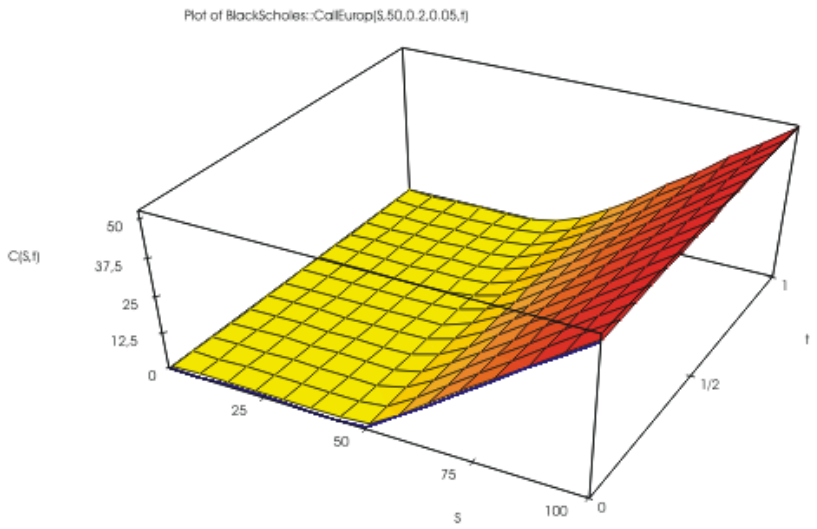


Abbildung 3: Black-Scholes Optionspreis eines europäisches Calls in Abhängigkeit vom Aktienkurs S und der Restlaufzeit t, mit $r = 0.1$, $E = 50$, $\sigma = 0.2$ und $t = 1$

[15] Vgl. [6, S. 174].

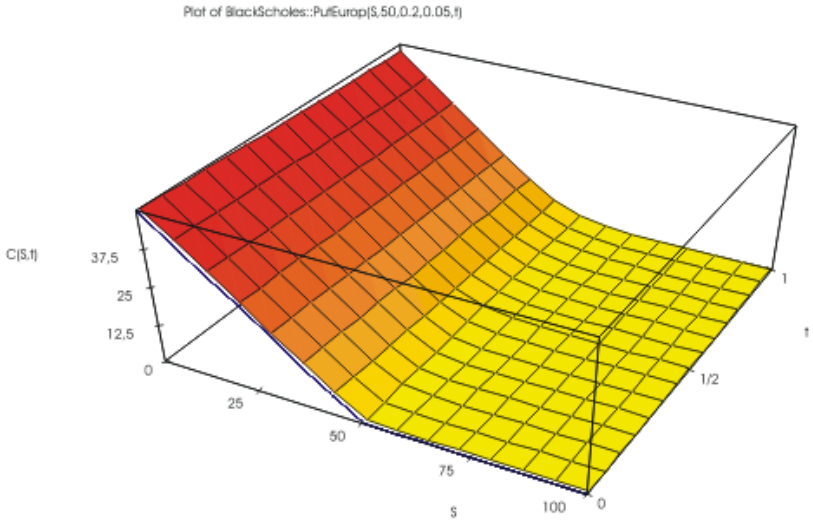


Abbildung 4: Black-Scholes Optionspreis eines europäisches Puts in Abhängigkeit vom Aktienkurs S und der Restlaufzeit t, mit $r = 0.1$, $E = 50$, $\sigma = 0.2$ und $t = 1$

Beispiel

An dieser Stelle soll das Ergebnis anhand eines praktischen Beispiels verdeutlicht werden. Gegeben sind folgende Daten.[16] Ferner sei die Kursvolatitilität der Aktie 29% und der risikolose Zins p.a. 2%.

Deutsche Bank (Bezugsdatum 27.12.2001))					
Art	**Basispreis**	**Bezugsverhältnis**	**Verfall**	**Basiskurs**	**Brief in Euro**
Call	80.00	0.100	26.06.2002	76.20	0.670

Tabelle 1: Call Option auf eine Aktie der Deutschen Bank

Wie der Tabelle zu entnehmen ist, besitzt die Call-Option eine Restlaufzeit von einem halben Jahr $t = 0.5$ und das Basisobjekt kann zu einem Preis von $E = 80.00$ Euro am Ende dieser Laufzeit erworben werden. Das Bezugsverhältnis sagt aus, wieviele Anteile man am Basisobjekt durch den Kauf eines Optionsscheines erwirbt. Für den Fall, dass der Anleger das Recht auf den Erwerb einer Aktie der Deutschen Bank erhält, sind also zehn Optionsscheine zu je 0,670 Euro zu beziehen. Damit ergibt sich der augenblickliche Wert von 6,70 Euro.
Der Anleger hat nun die Möglichkeit zu überprüfen, ob dieser Wert dem "fairen" Preis für dieses Optionsrecht entspricht und kann daraufhin seine Investitionsentscheidung treffen. Bei einem aktuellen Basiskurs von $S = 76.2$ Euro beträgt der "faire" Preis der Kaufoption

- BlackScholes::CallEurop(76.2,80,0.02,0.29,0.5)

  6.99821

[16] Vgl. [5, S.22].

also 6.99 Euro. Das Ergebnis der Black-Scholes Bewertung spiegelt in diesem Fall sehr gut den derzeitigen am Markt angebotenen Preis für den Erwerb des Optionsrechtes wider. Die kleine Abweichung ist auf die restriktiven Modellannahmen zurückzuführen. So bleiben zum Beispiel Tage innerhalb der Restlaufzeit, an denen kein Handel stattfindet, wie an Wochenend- und Feiertagen, unberücksichtigt.

Zusammenfassung

Der Black-Scholes Formel kommt in der Finanzwelt eine große Bedeutung zu. Ziel des Artikels war es, eine Einführung in die Grundlagen der Random-Walk-Hypothese und Optionspreistheorie sowie Beispiele für die jeweilige Implementierung in *MuPAD*, zu geben. Die Ergebnisse zeigen, dass *MuPAD* sehr gut bestückt ist, um Probleme innerhalb der Finanzmathematik von Derivaten aufzugreifen und zu lösen. Der interessierte Leser wird es leicht haben, die hier vorgestellten Ergebnisse, zum Lösen weiterer Aufgabenstellungen innerhalb der Finanzmathematik zu verwenden (z. B. dem Bewerten von amerikanischen Optionen, Compound-Optionen oder kombinierten Strategien wie Straddels etc.). Das hier vorgestellte *MuPAD*-Notebook kann von `www.mupad.de` sowie von `http://wiwi.uni-paderborn.de/vwl6/de/mitarb/lukas/Lukas_Deutsch.htm` heruntergeladen werden.

Neben der Bewertung von europäischen Call- und Put-Optionen ist zudem die Bestimmung der Sensitivitäten des Optionswertes in Bezug auf einzelne Parametern und Variablen möglich, auf die an dieser Stelle nicht näher eingegangen werden konnte.

Literatur

[1] Aristotele (1988): *The Politics*, Cambridge University Press, New York, 1988.

[2] Ayres, H. (1963): Risk aversion in the warrants market, *Industrial Management Review*, 4:497–505.

[3] Bachelier, L. (1900): Theorie de la speculation, *Annales de l'Ecole Normale Superieure*, 17:21–86.

[4] Black, F. und Scholes, M. (1973): The pricing of options and corporate liabilities, *Journal of Political Economy*, 81:637–654.

[5] CitibankAG (2002) Citibank Optionsschein Planer, Januar.

[6] Hull, J. C. (2000): *Options, Futures & Other Derivatives*, Prentice-Hall International Inc., Upper Saddle River, NJ 07458, 4 Ed. edition.

[7] Kloeden, P. E. und Platen, E. (1999): *Numerical Solution of Stochastic Differential Equations*, Springer Verlag, Berlin.

[8] Korn, G. A. und Korn, T. M. (1968): *Mathematical Handbook for Scientists and Engineers*, McGraw-Hill Book Company, New York, 2. Ed.

[9] Merton, R. C. (1973): Theory of rational option pricing, *Bell Journal of Economics and Management Science*, 4:141–183.

[10] Øksendal, B. (2000): *Stochastic Differential Equation: An Introduction with Applications*, Springer Verlag, Berlin, 5 Ed.

[11] Samuelson, P. A. (1965): Rational theory of warrants pricing, *Industrial Management Review*, 6:13–31.

[12] Sandmann, K. (1999): *Einführung in die Stochastik der Finanzmärkte*, Springer Verlag, Hamburg.

[13] Sprenkle, C. (1961): Warrant prices as indications of expactations, *Yale Economic Essays*, 1:179–232.

mαth-kit: **Ein webbasierter multimedialer Baukasten für die Lehre**

Gudrun Oevel, Kathrin Padberg, Bianca Thiere
Universität Paderborn, mailto:mathkit@math.uni-paderborn.de

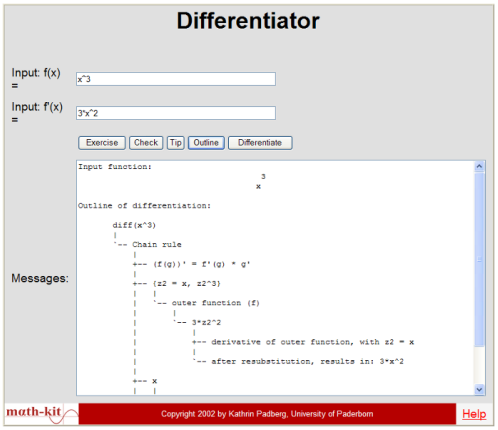
Die Mathematik ist ein unbeliebter aber elementarer Bestandteil vieler Fächer, von den Ingenieur- und Naturwissenschaften über die Informatik bis hin zu den Geistes- und Sozialwissenschaften. Einen Teil der "Schrecken" nehmen soll ihr das Verbundprojekt mαth-kit. *In* mαth-kit *wird ein webbasierter "Baukasten" entwickelt, der Benutzerinnen und Benutzern interaktive Bausteine für grundlegende Inhalte der Mathematik und ihrer Anwendungen zur Verfügung stellt. Angesprochen werden vor allem Lehrende und Lernende an Universitäten, jedoch ist der Einsatz auch in der außeruniversitären Weiterbildung, in der betrieblichen Aus- und Fortbildung und nicht zuletzt in Schulen möglich.*

Als ein Projekt der BMBF-Ausschreibung Neue Medien in der Hochschullehre"wird mαth-kit seit dem 1.3.2001 an den Universitäten Bayreuth, Hagen, Hamburg und Paderborn gemeinsam mit dem industriellen Partner SciFace Software GmbH & Co KG entwickelt. Dem Projektteam bestehend aus Wissenschaftlerinnen und Wissenschaftlern aus den Bereichen Didaktik, Mathematik, Informatik, Elektrotechnik und Maschinenbau unter Leitung von Prof. Dr. Luise Unger (FernUniversität Hagen) geht es besonders darum, Lehrende beim Einsatz von multimedialen Komponenten zu unterstützen, um so die Motivation für Studierende zu erhöhen. Dabei wendet sich mαth-kit in erster Linie an Lehrende und Studierende im Grundstudium, aber auch an (beruflich) Interessierte, die früher Erlerntes auffrischen bzw. wieder entdecken wollen. Wie in einem realen"Baukasten gibt es die verschiedensten Bausteine für die Präsentation durch Lehrende, für die Exploration durch Lernende, Übungen mit Ergebnisauswertungen und Anwendungsbeispiele sowie Werkzeuge für den webbasierten Zugriff auf das Computeralgebrasystem MuPAD. Diese kleinen interaktiven Elemente sollen sich per Mausklick in individuelle Lernumgebungen einbinden lassen, die den jeweiligen spezifischen Wünschen und Bedürfnissen entsprechen. Auch unterschiedliches Vorwissen kann so ausgeglichen werden. Die Flexibilität der Bausteine ermöglicht den Einsatz im Präsenz-, im Fern- und im Selbststudium.

Besonderheiten, die dieses eLearning-Projekt gegenüber anderen auszeichnen, liegen zum einen in der Struktur als Baukasten, dem integrierten Werkzeug MuPAD sowie an dem inhaltlichen Schwerpunkt der Verzahnung von Mathematik mit den Fachwissenschaften.

Über die webbasierte Anbindung an MuPAD sind bereits Bausteine implementiert, mit denen man den Gauss Algorithmus, das Differenzieren und Integrieren oder auch die Berechnung von Reihen- und Folgenwerte üben kann. Der Vorteil dieser Art des Zugriffs auf ein Computeralgebrasystem ist, dass die Benutzerinnen und Benutzer sich weder mit der komplexen Syntax der Software noch mit einer lokalen Installation abmühen müssen. Der Zugriff auf MuPAD und das dort implementierte mathematische Wissen kann unterschiedlich genutzt werden. Zum einen

mathPAD



Differentiator zum Üben des Differenzierens

können beliebig viele Aufgaben generiert und deren Lösungen auch überprüft werden. Zum anderen sind unterschiedliche Wege und Repräsentationen von Lösungen erlaubt, da MuPAD diese leicht auf Richtigkeit untersuchen kann. Außerdem ist es möglich, eine gestufte Hilfe anzubieten, die aus Tipps, dem nächsten Lösungsschritt oder dem kompletten Lösungsweg besteht. Mit einer leichten Modifikation können die webbasierten Bausteine auf unterschiedliches Wissensniveau angepasst werden. So kann z.B. realisiert werden, dass zunächst nur spezielle Klassen von Funktionen betrachtet und geübt werden.

Der Schwerpunkt der Verzahnung von Mathematik und deren Anwendung in den Ingenieur- und Naturwissenschaften ist dem gesamten Projektteam ein weiteres wichtiges Anliegen. Mathematik lernen Studierende eher axiomatisch mit einigen Beispielen; ein wirklicher Praxisbezug, der auf die Modellierung von Problemen und den Einsatz von mathematischen Methoden eingeht, ist oft nicht vorhanden. Häufig sind die Beispiele aus einem rein technischen Umfeld gewählt. Hier fehlt insbesondere Lehramtsstudierenden und auch Frauen der Bezug zur Realität und die Motivation. An dieser Stelle setzt mαth-kit an und versucht, verschiedene Einsatzgebiete von mathematischen Methoden mit Rahmenwissen - wie historische Zusammenhänge - zu verknüpfen. Als ein konkretes Beispiel sei an dieser Stelle das Thema ”Komplexe Zahlen”genannt. Im Mathematikstudium wird üblicherweise der Körper der komplexen Zahlen $\mathbb{C}$ mit seinen Rechenregeln definiert. Es wird darin gerechnet, die Repräsentation in Polarkoordinaten oder per e-Funktion wird meistens nur kurz erwähnt. Diese Sichtweise ist aber genau die, welche in den Ingenieurwissenschaften und in der Informatik maßgeblich ist. Das dort grundlegende Zeigerdiagramm sehen Studierende der Mathematik nicht. So unterscheidet sich schon bei einfachen Inhalten der Zugang und die Sichtweise. Es ist nicht verwunderlich, dass sich dieses fortsetzt und sich die entsprechenden Fachleute bei einer Zusammenarbeit zunächst auf eine gemeinsame Sicht und Sprache einigen müssen. Aber die Verzahnung von Mathematik und deren Anwendungen ist nicht nur wichtig für Studierende in der Mathematik. Auch in den Ingenieurwissenschaften oder der Informatik lernen die Studierenden Mathematik zu Beginn des Studiums und können nicht einschätzen, was daran wirklich wichtig ist und wofür sie es später benötigen. Hier bietet mαth-kit die Unterstützung der Motivation, aber auch die Hilfe, das benötigte mathematische Wissen aufzufrischen und zu vertiefen.

Das Paderborner Projektteam befasst sich schwerpunktmäßig mit der analytischen Seite der Mathematik (Komplexe Zahlen, Folgen, Reihen, Differenzieren und Integrieren sowie Differentialgleichungen) und den entsprechenden Anwendungsgebieten. Gemeinsam arbeiten an diesen Fragestellungen Kathrin Padberg vom Lehrstuhl Angewandte Mathematik (Prof. Dr. M. Dellnitz), Bianca Thiere vom Lehrstuhl Mechatronik und Dynamik (Prof. Dr.-Ing. J. Wallaschek), Dr. Andreas Sorgatz von der SciFace Software GmbH & Co KG sowie als lokale Projektkoordinatorin Dr. Gudrun Oevel.

Weitere Informationen zum mαth-kit Projekt sowie Beispiele gibt es unter `www.math-kit.de` oder per eMail an `mailto:info@math-kit.de`. Informationen über die Aktivitäten des Paderborner Teams gibt es unter `wwwmath.uni-paderborn.de/~mathkit` und via `mailto:mathkit@math.uni-paderborn.de`.

Literatur:

- Unger, L.; Oevel, G. und Mertsching B.: Web-based Teaching and Learning with math-kit. In Proc.: 2nd International Conference on the Teaching and Learning of Mathematics, Crete, 2002
- Padberg, K; Sorgatz, A.: Webbasierte Übungselemente mit MuPAD. In Proc.: Computeralgebra in Lehre, Ausbildung und Weiterbildung III, Bildungshaus Kloster Schöntal, 2002
- Padberg, K; Schiller, S.: Web-Based Drills in Maths Using a Computer Algebra System. In Proc.: ED-MEDIA 2002–World Conference on Educational Multimedia, Hypermedia & Telecommunications, Denver, Colorado, USA, June 24-29, 2002

Interaktive Lehreinheiten im Maschinenbau: “Optische-Spannungsanalyse”

Hans Dietmar Jäger
Bankakademie e.V., Frankfurt am Main, mailto:jagger@orakel.bankakademie.de

Die Vorbereitung der Maschinenbau-Studenten auf das “Maschinenlabor Optische Spannungsanalyse” erfolgte bisher allein auf Basis einer kopierten Blättersammlung. Da bei der Versuchsauswertung komplexe mathematische Verfahren zur Anwendung kommen, wird nun mit Hilfe einer interaktiven MuPAD-Lehreinheit versucht, den Studierenden eine bessere Sicht diese Verfahren zu vermitteln.

Teil I: Konzeption der Lehreinheit und Integration des CA-Systems

Die multimediale Lehreinheit (http://mm-info.upb.de/tutorium) wurde unter didaktischen und gestaltpsychologischen Gesichtspunkten entwickelt. Die reinen Textseiten mit Bildern und Grafiken wurde durch Animationen, eine Volltextsuche und die interaktiven Arbeitsblätter ergänzt.

Einsatz traditioneller Medien in der Lehre

Im Maschinenbaustudium werden praktische Versuchsreihen angeboten, welche grundlegende Techniken behandeln. Dabei sollen Studierende theoretisches Wissen praktisch anwenden. Ein Laborversuch umfaßt die Vorbereitung (Einlesen in den theoretischen Hintergrund), Durchführung und abschließende Auswertung (Darstellung und Interpretation der Ergebnisse).

Bisher werden in der Lehre dabei klassische Medien zur Vorbereitung auf Laborversuche eingesetzt. Das sind in erster Linie Print-Materialien wie photokopierte Texte und Bilder. Zum Teil kommen auch Videos zum Einsatz.

Mit diesen Materialien ausgestattet bereiten sich Studierende auf den Laborversuch vor, indem sie Texte lesen und versuchen, den Ablauf nachzuvollziehen. Die Vorbereitung soll die Teilnehmer in die Lage versetzen, daß sie den Laborversuch selbstständig durchführen, die Wirkungsweisen der behandelten Prinzipien nachvollziehen und den Versuch abschließend auswerten können.

Mathematik im Laborversuch

Bei der Auswertung des Versuchs oder auch zum Verstehen der theoretischen Grundlagen der im Laborversuch zum Einsatz kommenden Techniken liegen oft mathematische Sachverhalte zu Grunde. Einfache Sachverhalte können dabei durch Beispielrechnungen verdeutlicht werden. Sobald aber rekursive oder iterative Verfahren zum Einsatz kommen, wird das Nachvollziehen schwieriger.

Die Idee des Einsatzes eines Computeralgebra-Systems ist, den Studierenden eine andere Sichtweise auf komplexe mathematische Verfahren zu ermöglichen. Dies kann ein demonstrativer Ablauf einer Beispielrechnung sein, der

die Arbeitsschritte filmartig darstellt und mit ergänzenden Texten erklärt. Besser wäre natürlich, dem Studierenden die Möglichkeit zu geben, selbst in die Berechnungen eingreifen zu können und Startwerte zu manipulieren, um so Auswirkungen auf das Resultat zu erkennen.

Vor diesem Hintergrund wurde an der Universität Paderborn für den Fachbereich Maschinenbau in Zusammenarbeit mit den Arbeitsgruppen “Infromatik und Gesellschaft” und der “*MuPAD*-Gruppe” eine interaktive Lehrumgebung unter didaktischen und gestaltpsychologischen Gesichtspunkten entwickelt, die komplexe mathematische Abläufe durch Anbindung eines Computeralgebra-Systems visualisieren kann.

Der dazu ausgewählte Laborversuch “Maschinenlabor Optische Spannungsanalyse” eignet sich besonders gut, da ein iteratives mathematisches Verfahren für die Lösung zu Grunde liegt und gleichzeitig Versuchsergebnisse durch Grauwertbilder visualisiert werden können.

Umsetzung der Vorbereitungsmaterialien in eine multimediale Lehrumgebung

Zuerst wurden alle den Studierenden zur Vorbereitung zur Verfügung gestellten Materialien auf Aktualität hin überprüft. Dieser Vorgang wurde gleichzeitig zur Neuorganisation der Vorbereitung auf den Laborversuchs genutzt. Die einzelnen Kapitel der Blättersammlung wurden in HTML-Dokumente aufgeteilt, die sich in einem Web-Browser darstellen lassen. Grafiken wurden eingescannt und Formlen mittels LaTeX gesetzt und als Bilder zur Verfügung gestellt. Formeln lassen sich in einem eigenen Fenster neben dem Text-Fenster zur besseren Übersicht plazieren. Hierbei ist wichtig, die unterschiedlichen Medienqualitäten zu berücksichtigen: Bei Printmedien stehen bisweilen 600 dpi zur Verfügung während die Darstellung mit 72 dpi am Bildschirm mehr Platz bei gleichem Detailreichtum beansprucht.

Dem spannungsoptischen Verfahren liegt die Verwendung von polarisiertem Licht zu Grunde. Der hierbei wichtige Unterschied zwischen linear und zirkular oder elliptisch polarisiertem Licht wurde in einer Animation deshalb separat aufgearbeitet. Die beiden aufeinander senkrecht stehenden, zeitversetzt schwingenden Lichtwellen führen dabei zu einem - eine Ellipse oder einen Kreis beschreibenden - rotierenden resultierenden Lichtvektor.
Die für die Herstellung der Animation benötigten Bilder wurden wiederum mit *MuPAD* generiert – dazu wurden zwei Graphen gleichzeitig dargestellt – und zu einem Animated-Gif zusammengestellt. Lediglich der rotierende resultierende Vektor wurde von Hand eingezeichnet.

Nach der theoretischne Darstellung der Sachverhalte kommen drei interaktive *MuPAD*-Notebooks zum Einsatz, die die grundlegenden mathematischen Vorgänge behandeln. Hierbei sind zwei Richtungen zu betrachten:

1. Auf Grundlage der analytischen Beziehungen läßt sich die Grauwertverteilung eines Spannungszustandes zu vorgegebenen Parametern berechnen.
2. Zu vorliegendem Grauwertbild können die zum Belastungsfall gehörenden Parameter durch ein iteratives Verfahren bestimmt werden.

Zuerst wird an Hand des spannungsoptischen Ansatzes und der analytischen Beziehungen gezeigt, wie sich die Helligkeit eines Bildpunktes zu den Spannungszustand beschreibenden Parametern berechnen läßt. Dann wird dieses Verfahren angewendet und ein Grauwertbild zu den Parametern erzeugt, wie es sich im praktischen Laborversuch ergeben würde.

Interessanter und mathematisch anspruchsvoller ist die andere Richtung: Aus den Informationen eines im Versuch gewonnen Grauwertbildes lassen sich durch ein iteratives Verfahren die - für die Praxis relevanten - Parameter berechnen.

Die Berechnungen werden per Link aus einer Text-Seite gestartet und laufen dabei in einem separaten *MuPAD*-Fenster ab. Dies gibt die Möglichkeit, erklärenden Text und die ablaufenden Berechnungen gleichzeitig im Blickfeld zu haben. Die *MuPAD*-Notebooks wurden neben den anderen Inhaltsseiten als eigenständige, ergänzende

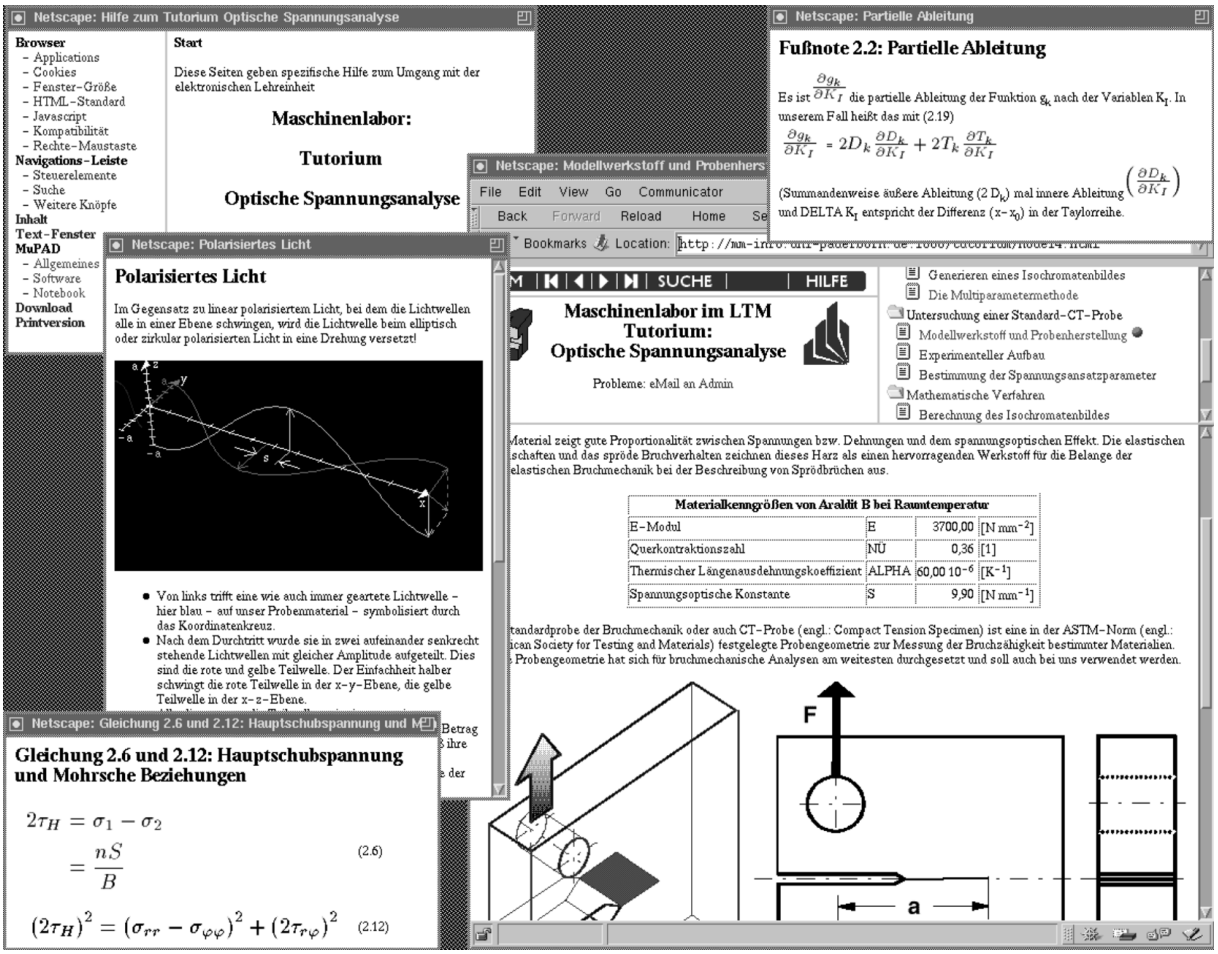


Abbildung 5: Bildschirmkopie des interaktiven Tutoriums mit ausgewählten Funktionen

Lehreinheiten konzipiert. In den Kommentarregionen wird das Vorgehen in den Worksheets erläutert und die Ein- und Ausgaberegionen erklärt.
Die Navigation wird in dieser interaktiven Einheit durch ein Inhaltsverzeichnis, Knöpfe zum Vor- und Rückwärtsblättern sowie einer Volltextsuche ermöglicht.

Umsetzung und Fazit

Die interaktive, multimediale Lehreinheit wurde auf einem Hyperwave Information Server realisiert. Dieser Server – eine Datenbank mit Web Interface und zwischengeschaltetem Gateway – stellt u. a. eine Volltextsuche und Navigationsfunktionalitäten zur Verfügung. Außerdem lassen sich beliebige Objekte in die Datenbank einlagern, in diesem Fall Bilder, HTML-Dokumente und Worksheets, die im *MuPAD*-Notebook geladen und ausgeführt werden können. Ein Inhaltsverzeichnisse wurde – mittlerweile dynamisch mittels Treeview möglich – erstellt.

Da ein Web Interface für den *MuPAD*-Kern, welches auch Grafiken ausgeben kann, noch nicht zur Verfügung stand, wurde der lokalen *MuPAD*-Installation der Vorzug gegeben. Vorteil dieser Lösung ist auch, daß man unabhänhängig von der Netzauslastung ist und dem *MuPAD*-Kern die volle Performanz des Rechners zur Verfügung steht.

Den Studierenden ist mit dieser Lehreinheit die Möglichkeit gegeben worden, sich mittels Web Browser und lokaler *MuPAD* Installation auf den Laborversuch vorzubereiten. Gleichzeitig erreichen sie immer die neueste Version der Unterlagen, da diese zentral vorgehalten und gewartet werden. Veraltete Unterlagen gehören so der Vergangenheit an.
Mit der Anbindung des CA-Systems *MuPAD* wird in drei Schritten – mittles drei aufeinander aufbauenen Worksheets – die Mathematik erarbeitet und visualisiert, die der Ausnutzung der optischen Effekte zu Grunde liegt. Beispielrechnungen können durchgeführt und durch eigene Berechnungen ergänzt werden. Die drei Regionen – farblich getrennte Kommentar-, Eingabe- und Ausgaberegion – gliedern das Worksheet und können ausgedruckt die Arbeitsunterlagen ergänzen.

Teil II: Mathematische Berechnungen und Visualisierung der Ergebnisse mittels MuPAD-Notebooks

Der Laborversuch “Maschinenlabor Optische Spannungsanalyse” (`http://mm-info.upb.de/tutorium`) beschäftigt sich mit spannungsoptischen Untersuchungen an ebenen Bauteilen aus spannungsoptisch aktivem Material. Dabei werden im Polariskop doppelbrechende Eigenschaften des Materials zur Erzeugung eines Grauwertmusters ausgenutzt (siehe Abbildung 6).

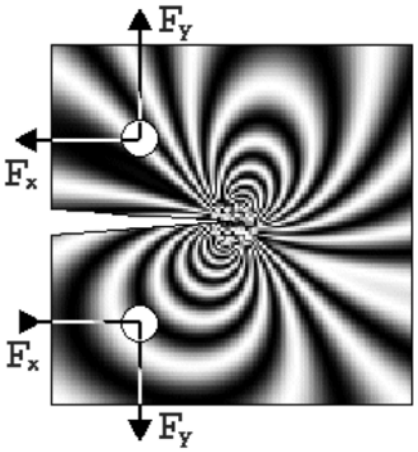


Abbildung 6: Belastete Probe im Polariskop mit resultierendem Isochromatenbild

Bei der Vorbereitung auf den Laborversuch werden die mathematischen Hintergründe dargestellt, die beim Ausnutzen der optischen Effekte benötigt werden. Dies umfaßt die Herleitung der Spannungsansatzfunktionen und die Einbettung der Ansatzfunktionen in die analytischen Zusammenhänge der optischen Spannungsanalyse.

Bei der Untersuchung ergeben sich zwei Wege der Betrachtung: Zum einen kann zu vorgegebenen Parametern, den Spannungsintensitätsfaktoren (SIF), das zugehörige Isochromatenbild[1] berechnet werden. Die Überlagerung mit dem experimentell gewonnenen Isochromatenbild kann eine Beurteilung der gewählten mathematischen Spannungsansatzfunktionen zulassen.
Zum anderen können zu einem experimentell gewonnenen Isochromatenbild die Spannungsintensitätsfaktoren berechnet werden, die eine Aussage über die Entfernung des Bauteils zum kritischen Punkt des Versagens und damit dessen Stabilität selbst zulassen. Dieses Verfahren ist in der Praxis die viel interessantere Möglichkeit und kann zum Beispiel bei Materialprüfverfahren im Flugzeugbau oder der industriellen Flaschenproduktion eingesetzt werden.

[1] Isochromaten stellen Linien gleicher Hauptspannungsdifferenz dar

Generieren eines Isochromatenbildes zu den Spannungsintensitätsfaktoren

Durch die Beziehungen für den Mohrschen Spannungskreis und die sogenannte Hauptgleichung der Spannungsoptik ergibt sich der Zusammenhang zwischen Isochromatenordnung n, der spannungsoptischen Konstante S, der Dicke der Probe B und den Spannungskomponenten allgemein zu

$$\left(\frac{nS}{B}\right)^2 = (\sigma_{rr} - \sigma_{\varphi\varphi})^2 + (2\tau_{r\varphi})^2$$

(wobei $\sigma_{rr}, \sigma_{\varphi\varphi}$ und $\tau_{r\varphi}$ Terme in r (Radius), φ (Winkel) und den Spannungsansatzparametern $K_I, K_{II}, \sigma_{0x}, \dots$ sind)
und mit Lichtintensität I

$$I(n) = I_1 + I_2 \sin^2(\pi n)$$

(wobei I_1 die Lichtintensität für den unbelasteten Fall – bei einer idealen Dunkelfeldanordnung gilt $I_1 = 0$ – und I_2 eine amplitudenmodulierte Lichtintensität ist)
ergibt sich die analytische Beziehung für die Lichtintensität I zu

$$I(K_I, K_{II}, \sigma_{0x}, a_3, b_3, a_4, \dots) = I_1 + I_2 \sin^2\left(\pi \frac{B}{S} \sqrt{(\sigma_{rr} - \sigma_{\varphi\varphi})^2 + (2\tau_{r\varphi})^2}\right).$$

Für die Berechnung eines Isochromatenbildes werden die Reihenentwicklungen der Spannungsansatzfunktionen nach dem siebten Term abgebrochen (7-Parameter-Methode), so daß nur die Parametern $K_I, \dots, b_4$ vorgegebenen werden müssen. Für jeden Bildpunkt $p(x, y)$ müssen die kartesischen Koordinaten in Polarkoordinaten umgewandelt, die Spannungskomponenten berechnet und die Helligkeit bestimmt werden.

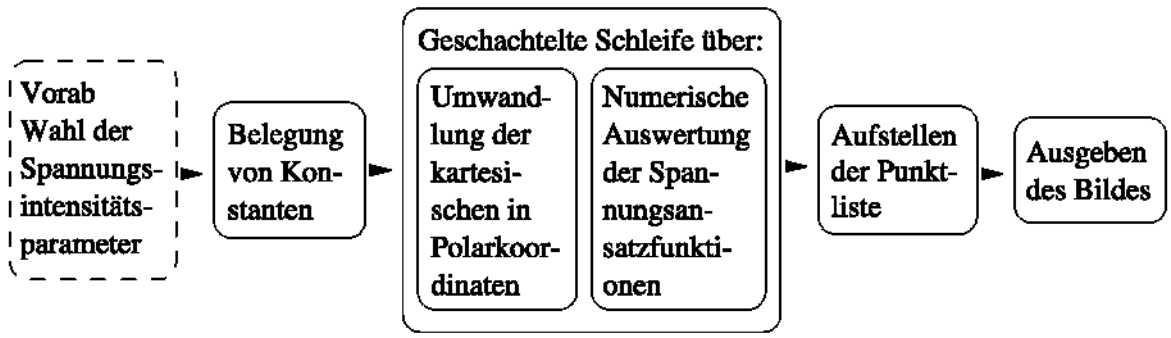


Abbildung 7: Schematischer Programmablauf der *MuPAD*-Implementation des zweiten Notebook

Diese Prozedur wurde in ersten *MuPAD*-Notebook exemplarisch für einen Bildpunkt implementiert. Studierende können sich so bei diesem relativ einfachen Problem mit der Arbeitsweise der *MuPAD*-Notebooks vertraut machen. Im zweiten *MuPAD*-Notebook werden für eine Fläche die Helligkeit der Bildpunkte berechnet und das Bild ausgegeben (siehe *MuPAD*-Notebook in Abbildung 8 auf Seite 32).

Die so generierten Isochromatenbilder dienen aber nicht nur der Veranschaulichung, sondern können in einem anderen Zusammenhang verwendet werden:
In der Praxis wird ein experimentell entstandenes Isochromatenbild untersucht und die Spannungsintensitätsfaktoren berechnet. Mit diesen können dann mathematisch berechnete Isochromatenbilder generiert und mit den ursprünglichen überlagert werden. Auf diese Weise läßt sich der mathematische Ansatz beurteilen und sein Gültigkeitsbereich definieren.

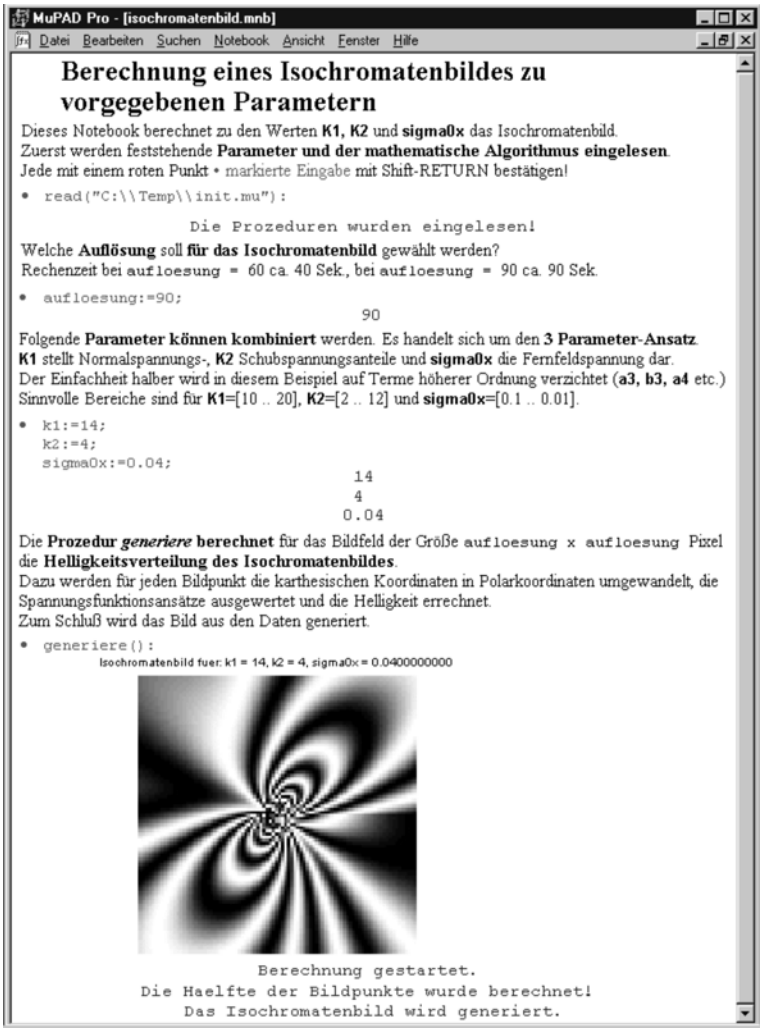


Abbildung 8: *MuPAD*-Notebook zum Berechnen eines Isochromatenbildes zu vorgegebenen Parametern K_I, K_{II}, σ_{0x} (3-Parameter-Methode)

Berechnung der Spannungsintensitätfaktoren aus einem Isochromatenbild

Auf der Grundlage der analytischen Beziehung ist es auch möglich, aus spannungsoptischen Isochromatenbildern die Spannungsintensitätsfaktoren zu berechnen. Dazu werden die Isochromatenschleifen eines spannungsoptischen Grauwertbildes durch eine Menge von Punkten $p_k(r_k, \varphi_k)$ mit zugehörigen Isochromatenordnungen n_k in Polarkoordinaten mit Radius r_k und Winkel φ_k, $k = 1, 2, \ldots$ beschrieben. Ganzzahlige Isochromatenordnungen liegen auf der Mittellinie der dunklen Isochromatenschleifen vor. Jeder Punkt liefert eine Gleichung

$$\begin{aligned} g_k\left(K_I, K_{II}, \sigma_{0x}, a_3, \ldots, b_N\right) &:= \left(\frac{\sigma_{rr}(r_k,\varphi_k) - \sigma_{\varphi\varphi}(r_k,\varphi_k)}{2}\right)^2 + \tau_{r\varphi}^2(r_k,\varphi_k) - \left(\frac{nS}{2B}\right)^2 \\ &= D_k^2 + T_k^2 - \left(\frac{nS}{2B}\right)^2 \\ &= 0 \end{aligned}$$

mit

$$D_k := \left(\frac{\sigma_{rr}(r_k, \varphi_k) - \sigma_{\varphi\varphi}(r_k, \varphi_k)}{2} \right)$$

$$T_k := \tau_{r\varphi}(r_k, \varphi_k).$$

($\sigma_{rr}, \sigma_{\varphi\varphi}$ und $\tau_{r\varphi}$ wie im vorigen Abschnitt)

Die Selektion von M Punkten ergibt M Gleichungen für $2N - 1$ Unbekannte mit $M \gg 2N - 1$. Unter der Voraussetzung werden in diesem Fall die Abweichungen minimiert[2]. Dies führt zum überbestimmten, nichtlinearen Gleichungssystem in den Unbekannten $K_I, K_{II}, \ldots$

Durch Entwickeln der Funktionen g_k in einer Taylorreihe und Abbrechen nach den linearen Termen ergibt sich das Newton-Verfahren für die Unbekannten $\Delta K_I, \Delta K_{II}, \ldots$ im i-ten Iterationsschritt und mit

$$\lim_{i \to \infty} g_k^{i+1} = 0$$

das überbestimmte, lineare Gleichungssystem in den Unbekannten $\Delta K_I, \Delta K_{II}, \ldots$ im i-ten Iterationsschritt

$$\begin{pmatrix} -g_1 \\ \vdots \\ -g_k \\ \vdots \\ -g_M \end{pmatrix}^i = \begin{pmatrix} \frac{\partial g_1}{\partial K_I} & \frac{\partial g_1}{\partial K_{II}} & \frac{\partial g_1}{\partial \sigma_{0x}} & \frac{\partial g_1}{\partial a_3} & \cdots & \frac{\partial g_1}{\partial b_N} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ \frac{\partial g_k}{\partial K_I} & \frac{\partial g_k}{\partial K_{II}} & \frac{\partial g_k}{\partial \sigma_{0x}} & \frac{\partial g_k}{\partial a_3} & \cdots & \frac{\partial g_k}{\partial b_N} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ \frac{\partial g_M}{\partial K_I} & \frac{\partial g_M}{\partial K_{II}} & \frac{\partial g_M}{\partial \sigma_{0x}} & \frac{\partial g_M}{\partial a_3} & \cdots & \frac{\partial g_M}{\partial b_N} \end{pmatrix}^i \begin{pmatrix} \Delta K_I \\ \Delta K_{II} \\ \Delta \sigma_{0x} \\ \Delta a_3 \\ \vdots \\ \Delta b_N \end{pmatrix}^i,$$

kurz

$$g^i = C^i \delta^i.$$

Dieses überbestimmte, inhomogene, lineare Gleichungssystem läßt sich im allgemeinen nicht lösen. Nach der Methode der kleinsten Quadrate ist aber auch die Lösung des bestimmten, linearen Gleichungssystems

$${}^tC^i g^i = {}^tC^i C^i \delta^i$$

– welches mit einem iterativen Verfahren gelöste werden kann – die beste Lösung für das ursprüngliche Ausgangsproblem. In Abbildung 9 auf Seite 34 wird das Lösungsverfahren, welches im dritten *MuPAD*-Notebook implementiert wurde, dargestellt.

Das dritte *MuPAD*-Notebook in Abbildung 10 auf Seite 35 zeigt vier typische Isochromatenbilder im oberen Bereich, auf die das Verfahren exemplarisch angewendet werden kann, d.h. hier liegen Datensätze für Punkte in Polarkoordinaten mit zugehöriger Isochromatenordnung vor. Im unteren Bereich erfolgt die Ausgabe des Programms, welche über Zustand der Iteration und die erreichten Korrekturvektoren und Fehler informiert.

[2] In diesem Fall sind die Fehler in den Werten p_k zufällige Größen, die nach dem Gesetz der Gaußschen Normalverteilung verteilt sind.

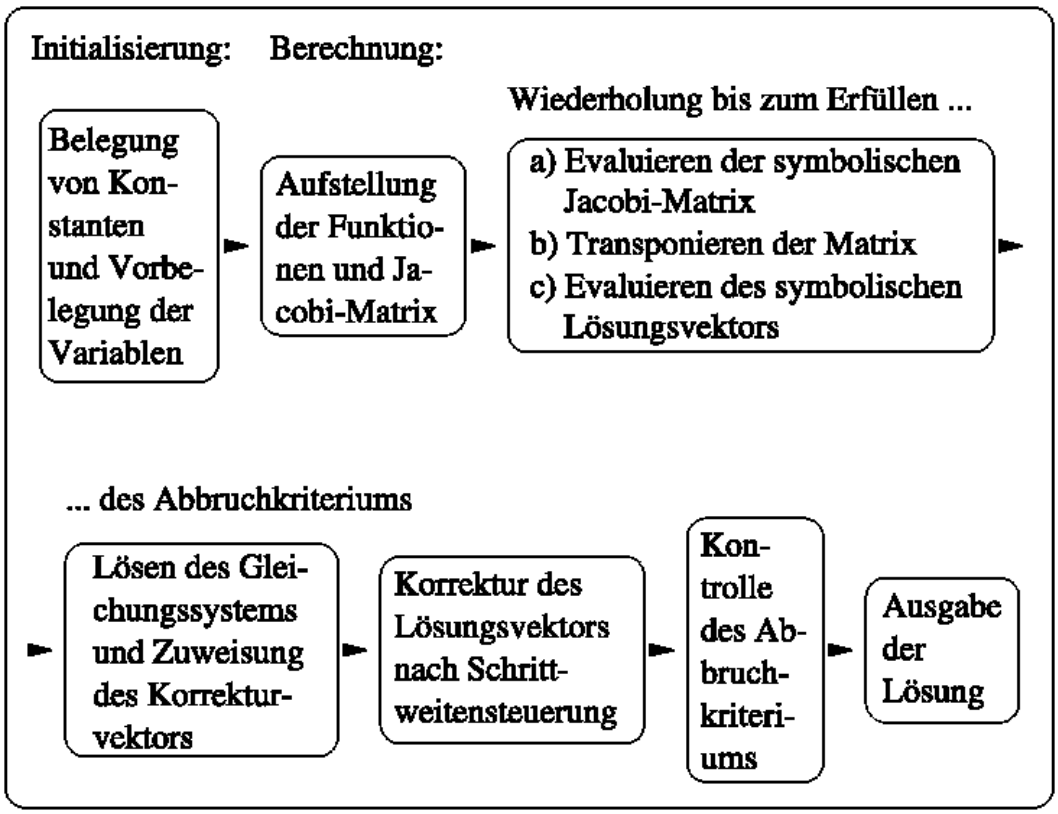
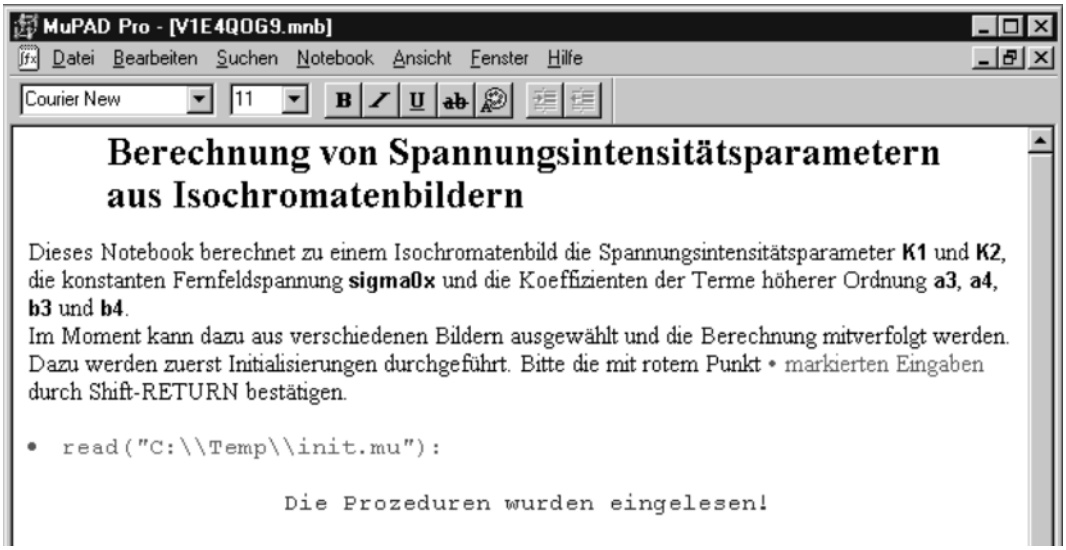
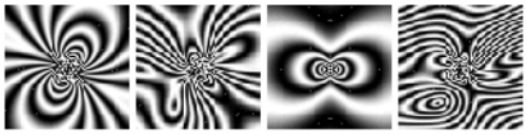


Abbildung 9: Schematischer Programmablauf des dritten *MuPAD*-Notebook



Zu welchem Isochromatenbild sollen die Spannungsintensitätsfaktoren bestimmt werden? Die roten Punkte definieren jeweils die Punktmenge, aus denen das Gleichungssystem generiert wird.
Bitte die dem Bild zugeordnete `waehle`-Zeile mit einem Shift-RETURN bestätigen.



[Bild I] [Bild II] [Bild III] [Bild IV]

- `waehle_I():`

```
                    Lese die Daten ...
   Das nichtlineare Gleichungssystem besteht aus 19 Gleichungen.
                              ... fertig.
```

- `waehle_II():`
- `waehle_III():`
- `waehle_IV():`

Zum Starten der Berechnung den nächsten Befehlt bestätigen.

- `initialisiere():`
  `berechne():`

```
                    Starte Berechnung.
          Die Gleichungen gk werden aufgestellt ...
   Wir benoetigen die partiellen Ableitungen aller Funktionen
            - die sich aus den Gleichungen ergeben -
    nach allen 7 Unbekannten (k1, k2, sigma0x, a3, ..., b4).
        Diese Matrix wird auch Jacobi-Matrix genannt.
       Startwerte: , 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0
      Iterationsschritt, k1, k2, sigma0x, a3, a4, b3, b4
Schrittweitensteuerung aktiv! ||g(xi)||=24.4 < ||g(xi+1)||=1161.0
                 t=0.500 => ||g(xi+1)||=278.0
                 t=0.250 => ||g(xi+1)||=54.5
                 t=0.125 => ||g(xi+1)||=4.40
       1, 22.0, 11.0, -2.8, -0.047, -0.023, 0.11, 0.002
       2, 23.0, 10.0, -1.3, -0.3, -0.011, 0.052, -0.01
   3, 24.0, 7.4, -0.12, -0.044, -0.00083, 0.0027, -0.00081
   4, 25.0, 6.9, 0.012, 0.0028, 0.000077, -0.00051, 0.00015
5, 25.0, 7.0, 0.00053, 0.00011, 0.0000051, -0.000022, 0.0000037
6, 25.0, 7.0, 0.00048, 0.000098, 0.0000047, -0.00002, 0.0000029
7, 25.0, 7.0, 0.00048, 0.000098, 0.0000047, -0.00002, 0.0000029
8, 25.0, 7.0, 0.00048, 0.000098, 0.0000047, -0.00002, 0.0000029
Die Ergebnisse: k1 = 25.0 und k2 = 7.0, sigma0x = 0.00048 sowie
 a3 = 0.000098, a4 = 0.0000047, b3 = -0.000020, b4 = 0.0000029
```

Bereit | 2353 kB | Eing. 1te | Ze 85 | ÜB

Abbildung 10: *MuPAD*-Notebook zum Berechnen der Spannungsansatzparameter aus Isochromatenbildern

Rule-based Simplification of Expressions

Jürgen Billing and Stefan Wehmeier
MuPAD Research Group, Universitity of Paderborn
mailto:bij@mupad.de, mailto:stefanw@mupad.de

We present a new, rule-based method for the simplification of expressions. This will be the basis for the `simplify` *function in future versions of MuPAD.*

The goal

MuPAD provides many functions that rewrite expressions in an equivalent form, such as `expand`, `rewrite`, `combine`, `normal`, `simplify`, `radsimp`, and many more. While this helps the user to obtain many different expressions equivalent to his input, it still forces the user to try many of these functions manually until some output seems acceptable to him. Some automated search would be nice.

Simplification functions have to find a compromise between strength and running time. It is a pity if *MuPAD* does not find out that a given input is equivalent to zero; but it is also a pity if a call to `simplify` does not return for two days. The user should be able to tell *MuPAD* how much time he wants to spend waiting for a result.

Many users find it hard to foresee what functions like `combine` or `simplify` will do to the input. A good simplification function must provide complete control, letting the user select what rules to apply.

Sometimes users may want to have a list of equivalent expressions, in order to be able to choose on their own. Sometimes users may want a mathematical proof that the expressions are equivalent.

Summing up, some intelligent, highly configurable search is needed.

Some theory

We may consider different–looking expressions equivalent if we find a proof that they are equivalent. An equivalence proof consists of finitely many steps; each step is the application of a rule (axiom, theorem, ...) to the result of the previous step. Two expressions are equivalent if one is the input to the first step and the other is the output of the last step of an equivalence proof.

Finding an equivalence proof is difficult (in fact, undecidable). We might apply all rules we know to the input to obtain all expressions the equivalence of which can be proved in just one step; then apply all rules to all of these to obtain all expressions that have an equivalence proof of length two; and so on. In fact, this is how to prove that the problem is recursively enumerable. If we have a definition of complexity, the same argument works in order to show that we are able to find the simplest expression that can be proved to be equivalent to the input in at most n steps, with n fixed.

We may visualize the problem as a graph where the vertices represent expressions, and an edge between two vertices indicates that one expression may be derived from the other in one step.

All of these statements are true in a well-known context, the theory of formal languages and grammars. Formal grammars are a standard concept of computer science: they consist of an initial word and a set of replacement rules.

Sequences of symbols that can be obtained, starting from the initial word, by successive application of replacement rules, are called words of the language defined by the grammar. Our problem can then be rephrased as searching for the shortest word in a formal language defined by a grammar.

Some ideas for heuristics

So far, we have learnt that we are searching a vertex in an infinite graph, but no idea how to do it. This is no surprise since we have talked about general grammars, using no knowledge about arithmetical expressions.

Let us start with symmetry: we have not used it, should we try to use it somehow? In principle, the converse of every rule will hold: we will rewrite $\sin(x)^2 + \cos(x)^2$ as 1, and we would be allowed to do it the other way around as well. Intuitively, we feel that this is a bad idea: we should not use all rules that are mathematically valid, and the set of rules we actually use will not be closed under exchanging left and right hand side. This means that we are walking in a directed graph.

Now suppose that we can assign a “valuation” to every expression which we want to minimize. How can we search?

One idea would be to go from a vertex to its “best” neighbour, and continue this until some vertex is reached that is a local minimum. However, good results in mathematics are often achieved through complicated intermediate results.

Should we then select a neighbour vertex at random, with the probabilty weighted by valuation, and allow increase in valuation during the early steps, in a simulated–annealing manner? But since we are in a directed graph, we can never correct a false decision.

Just the contrary idea is to follow our theoretical proof from above (width-first search), stop the enumeration after a while, and return the best result found so far. The drawback to this approach is (of course) that only shallow depths can be reached within a reasonable amount of time.

We have chosen to do a best-first search instead, as we will describe below.

Local vs. global simplification

We have not yet discussed another point that is important both in general grammars as well as in our special case of mathematical expressions. Grammar rules (or mathematical rules) say that a given substring (or subexpression, respectively) may be replaced by another one.

If the left/right hand sides of the rules are small compared to the word size, much of the previous word remains. If some other substitution step applying to another part of the word was possible before, it might still be possible now.

On the other hand, we might try to find a rule that only replaces new parts of the word. In this way, we may hope to be able to manipulate different parts of the word in a nearly context-free way.

A good algorithm should try to do this, and collect information how every subword can be locally transformed. We do this, storing all derivations we have made in a large table.

The algorithm

Let an expression A be given. The main part of our algorithm can be described in a very simple way:

- create a toDo-list of simplification steps for the simplification of A
- while no stopping criterion is reached
  - do the next simplification - step for A
- return the simplest expression found so far

Now this tells almost nothing, we have to explain what a simplification step is, how the *next* step is determined, what our stopping criteria are, and what the simplicity of an expression is.

A simplification step produces an equivalent expression, and/or it produces new tasks that have to be done. Possible steps are:

`findRules`. Given an expression, find out which rules can be applied. This is the only item in every newly created toDo-list. It returns nothing, and creates one `applyRule` task for every rule found. It also initializes to-do lists for every operand, and creates one `recurseOnOperand` task for every operand of the expression.

`applyRule`. Given an expression and a rule, apply the rule to that expression. This returns an equivalent expression, and creates a `findRules` task for that new expression.

`recurseOnOperand` do a local simplification step on a given operand; replace the operand by the equivalent expression found. We explain this on page 38.

We will explain below on page 39 how rules are organized and what applying a rule exactly means.

The tasks are sorted according to the expected complexity of the output they are going to produce. On page 39, we will explain how the priority of tasks is computed. Stopping criteria are explained on page 40. We will describe on page 40 how the complexity of an expression is measured.

The recursion step

Suppose we want to simplify an expression with many operands (for simplicity, let us consider a sum) $a_1+\ldots+a_n$. It is clear that we may apply rules with n-element sums on their left hand side. On the other hand, we may substitute some a_i by an equivalent one; or apply some rule for sums with two summands to $a_i + a_j$, for some i, j.

Some optimization must be done here: since simplified versions of the same subexpression are needed many times, we have to store them. Now suppose that for each a_i, some equivalent expressions $a_{i,j}$, $1 \leq j \leq m_i$, are already available. One might suspect that trying all $m_1 \cdots m_n$ possible substitutions was too much effort. Experiments show that this is not the case; therefore we form all combinations involving a particular a_{ij} and a's obtained earlier as soon as we compute $a_{i,j}$.

Technically, a recursion step for simplifying one a_i looks as follows:

- Do one simplification step for a_i
- Combine the expression obtained thereby with all a_k, $k \neq i$, and expressions equivalent to them, in all possible ways.

- Store the results as expressions equivalent to the original sum of a_i's.
- For each newly obtained result, put the task to simplify it on the to-do list. Put the task to simplify a_i another time on the to-do list.

A recursion step for a partial sum $a_{i_1} + \ldots + a_{i_k}$ looks similarly.

The first step means that our simplifier must be able to call itself in that way that only one step is done, but the to-do list is not lost afterwards. However, *MuPAD* has no static variables; we discuss this problem in a separate article.

Rules

A rule is a prescription how to rewrite an expression into another expression and consists of three parts:

1. the left hand side: a pattern
2. the goal or right hand side: an expression to substitute
3. some conditions for the pattern variables (not necessary)

If the pattern matches a given expression, then we say that the rule is applicable to that expression.

An example for a rule is: `Rule(sin(X)^2 + cos(X)^2, 1)`

The pattern (part 1) is the expression `Rule(sin(X)^2 + cos(X)^2` where `X` matches every *MuPAD* expression (because no conditions for `X` are given), the second part `1` is the goal of this rule. In this case no conditions for the pattern variable `X` are necessary.

A more complex rule is:
```
Rule(X^p, hold(I^p * op(X, 2)^p),
     {type(X) = DOM_COMPLEX and op(X, 1) = 0 and op(X, 2) > 0})
```

The rule rewrites powers when the base is an imaginary number with positive imaginary part. To prevent evaluating the right hand side of a rule, one should use `hold`.

It's also possible to use a special function to simplify an expression:

```
Rule(X, hold(rewrite(X, ln)), {hold(has(X, exp))})
```

A rule base is a collection of rules, e.g., a list. For large collections of rules, it is useful to split the list into several smaller ones, one per pattern type. The selection of rules is done by the function `Simplify::selectRules` which is called with an expression and returns all rules that could be applicable the given expression.

This select function can be replaced by a user defined function to consider user defined rule bases (see p. 41).

The priority of tasks

We use the convention that small numbers stand for high priorities. The priority of a step equals the expected complexity of its output. In principle, one should multiply this by the exponential of some constant times the expected running time. Since it is difficult to foresee both complexity of the output and running time, we have to enter some arbitrary estimates anyway. In detail, we assign the following priorities:

- To find applicable rules for a given expression has the priority given by the complexity of the expression, times a constant factor smaller than one which represents the self-estimate of the simplifier (the average factor by which an expression gets better per simplification step).
- The priority of applying a given rule to a given expression equals the complexity of the expression times a factor specified inside the rule; by default, the ratio between the complexity of the right hand side and the complexity of the left hand side is used.
- The priority of doing a simplification step for a subexpression is determined based on a guess how much the given subexpression will be simplified in that step, and how much the valuation of the whole expression will change thereby. Here, the expected valuation of the result of simplifying the subexpression can be read off from the priority of the first element of the to-do list for that subexpression. It is assumed that the decrease of the complexity of the subexpression will cause the same amount of decrease of complexity of the whole expression.

Stopping criteria

It may happen that no tasks are left at some point, such that all equivalent expressions have been found. Mathematically, it cannot happen that there are only finitely many expressions equivalent to a given one; but a sensible rule base will have no rule e.g. with left hand side 0, such that trying to simplify 0 will immediately return.

Except for this case, several stopping criteria can be imagined. We may do a fixed number of simplification steps; or stop if the next step has a too bad priority; or stop if a given goal has been reached.

In our current implementation, we use 150 simplification steps. The user may customize this value using the option `MaxSteps = n`, where `n` is a positive integer. The user may define a particular expression *as goal* using the option `Goal = expression`. The simplification stops until the goal is reached and the number of steps is shown. We did not implement a priority-based stopping criterion yet.

Complexity

One important point is to find a measure to compare expressions and find out the simplest expression.

Unfortunately, there exists no generally useful solution, although most users have a certain opinion what simplicity means. Therefore, we allow the user to set his valuation (measure of complexity) by an option.

We define a default valuation recursively. Atomic *MuPAD* objects get a constant valuation only depending on their type, e.g., rational numbers are more complicated than integers, and complex numbers are even more complicated. An expression gets the sum of the valuation of its operands plus a value for the operation as valuation. Thus operations gets different values for their complexity, e.g., `+` and `*` are simple operations, and `cot` and `arcsin` are more complex.

Let us look at some examples:

Expression	**valuation**
`0`	1
`-1523`	1
`1/4`	2
`x`	2

Expression	**valuation**
`x + 1`	5
`x + y`	6
`2 + I*7.1`	10
`x^2`	10

Expression	**valuation**
`PI^(1/2)*(2*x)^(1/2)`	29
`sin(x)`	32
`sin(2*x)`	36
`int(sin(2*x)^2, x)`	146

With the recursive definition it cannot happen, that parts of expressions get a greater valuation than the expression itself.

Configurability

One of our goals is to give the user maximum control over the simplification. Our implementation has many optional arguments to achieve this:

- `selectRules`: the user may provide a function that returns rules possibly matched by a given expression
- `Valuation`: the user may provide a function that assigns a complexity to each expression
- `MaxSteps`: the user may limit the number of steps and thereby the running time
- `Goal`: the user may choose to stop when a given goal expression is found

The user may also specify the kind of output he wants; currently, there are three kinds of output.

- By default, one expression equivalent to the input is returned.
- If the option `OutputType = ''Proof''` is given, the return value is a proof of the equivalence between the input and the expression that it would return by default.
- With option `All`, a list of all equivalent expressions and their valuations is returned.

Examples

The new mechanism maintains most results of the standard `simplify` function. Differences are due to another normal form and notion of simplicity. We do not give a list of expressions that can be simplified; rather, we want to show some ways to influence simplification.

Strength vs. speed

Our current version happens to find out that $\cos(x)\tan(x) = \sin(x)$ in the 12th simplification step (expanding $\tan$ makes an expression more complicated, therefore it is not chosen as the very first option). 11 steps do not suffice:

MuPAD

```
>> Simplify(cos(x)*tan(x), MaxSteps = 11)
```

Output

$$\cos(x)\tan(x)$$

but 19 steps do:

MuPAD

```
>> Simplify(cos(x)*tan(x), MaxSteps = 12)
```

Output

$$\sin(x)$$

Hence the default (150 steps) would suffice but take more time than necessary.

Controlling the valuation

To many people, expanding means to minimize the number of brackets in the output. Hence, let us define

```
NumberOfBrackets:=
  proc(ex)
  begin
    nops(stringlib::contains(expr2text(ex), "(", IndexList))
  end_proc
```

Suppose we want to use the default valuation, but impose an additional penalty of 2 for each bracket in the output:

```
valuation:= Simplify::valuation + 2*NumberOfBrackets:
```

To look at the difference, it is best to choose the output type `All`:

```
MuPAD
>> Simplify(x*(y + z), All)
   Output
                        [[x (y + z), 13], [x y + x z, 14]]
```

such that by default, the expression is not expanded because the input is simpler. With the penalty, the order is reversed:

```
MuPAD
>> Simplify(x*(y + z), Valuation = valuation, All)
   Output
                        [[x y + x z, 14], [x (y + z), 15]]
```

It seems a matter of personal taste indeed which result is better.

Using the new simplify to write special simplification modules

One could wish to simplify powers “unsophisticated” as described in school books, however, none of the many rewriting procedures of *MuPAD* can do it.

Here the new simplification project used with an own rule base leads to the desired “simplification modul”:

```
PowerRules:=
  [Rule(A^m*A^n, hold(A^(m + n))),    Rule(A^m/A^n, hold(A^(m - n))),
   Rule(A^n*B^n, hold((A*B)^n)),      Rule(A^n/B^n, hold((A/B)^n)),
   Rule(A^n/B^n, hold((B/A)^-n)),     Rule(A^m^n,   hold(A^n^m))]
```

We have defined a small rule base to simplify powers as learned in school. We need also a selection procedure; it simply returns the whole rulebase, independently of expression.

```
powerRules:= () -> PowerRules:
```

Now we can use the new `Simplify` with our own rulebase to simplify powers as expected:

MuPAD

```
>> a := T^(1/2)*(R*T)^(-1/2)
```

Output

$$\frac{\sqrt{T}}{\sqrt{RT}}$$

```
>> Simplify(a, SelectRules = powerRules)
```

Output

$$\frac{1}{\sqrt{R}}$$

Equivalence proofs

For many reasons, it may be interesting to get the result together with a proof that it is correct.

In the example before, one obtains

MuPAD

```
>> Simplify(a, SelectRules = powerRules, OutputType = Proof")
```

Output

```
Input was T^(1/2)/(R*T)^(1/2).
Applying the rule A^n/B^n  ->  (A/B)^n
  to T^(1/2)/(R*T)^(1/2) gives (1/R)^(1/2)
Applying the rule (A^m)^n  ->  (A^n)^m
  to (1/R)^(1/2) gives 1/R^(1/2)
END OF PROOF
```

Future perspectives

The new simplification package is still under development; because of its modular structure, many parts of it can be fine-tuned independently of each other, and this fine-tuning has still to be done. We did not talk about running times in this article; in fact, analysing the connection between the different input parameters and the resulting running time has still to be done, too. We are going to present the final version in a future version of *MuPAD*.

A Note on Volume Integrals

Christopher Creutzig
MuPAD Research Group, University of Paderborn, mailto:ccr@mupad.de

Currently, no CAS we know of offers symbolic volume integrals. Instead, users have to rely on nested one-dimensional integrals. The current article discusses this situation and some of the limitation caused by the lack of algorithms for volume integrals.

What are Volume Integrals?

Most of us are used to definite integrals over the real axis or to contour integrals in the complex plane, which are really just definite integrals over the reals combined with a suitable transformation of the coordinate system.

Whether integrals are defined as Riemann integrals or via Lebesgue theory, very much the same way of defining definite integrals can be performed not only over an interval of real numbers, but over much more general subsets V of $\mathbb{R}^n$.

We will not go into the details of conditions on V and f in this short essay. Let it suffice to say that most "simple" sets and functions are sufficiently smooth for the theory to work. We will denote the integral of

$$f(x_1,\ldots,x_n) \text{ over } V \subset \mathbb{R}^n \text{ by } \int_V f \,\mathrm{d}x_1 \cdots \mathrm{d}x_n.$$

Nested Integrals as Volume Integrals

Unfortunately, while there is a large volume of theory on (symbolic) integration over the reals, it seems that the more general problem of volume integrals has hardly been worked on at all. We know of no computer algebra system (CAS) providing volume integrals directly.

For a very restricted type of sets V, it is possible to use nested one-dimensional integrals to express volume integrals. For example, if

$$V = \{(x,y) \mid 0 \le x+y \le 1, x \ge 0, y \ge 0\},$$

we have

$$\int_V f(x,y)\,\mathrm{d}x\,\mathrm{d}y = \int_0^1 \left(\int_0^y f(x,y)\,\mathrm{d}x\right)\mathrm{d}y.$$

For sufficiently "simple" f, you can get the required integral from this representation:

MuPAD

```
>> int(int(x^2+sin(y), x=0..y), y=0..1)
```

Output

$$\sin(1) - \cos(1) + \frac{1}{12}$$

Divergent Integrals

An important question on (real or volume) integrals is whether they (that is, the defining series) converge or diverge, and if they diverge, whether they are properly divergent to plus or minus infinity.

Traditionally, CAS return some representation of ∞ or $-\infty$ in this case and some "undefined" value for improperly divergent integrals:

MuPAD

```
>> int(1/x, x=0..infinity), int(1/x, x=-1..infinity)
```

Output

Warning: Antiderivative is unbounded at lower limit [intlib::antiderivative]

$$\infty, \text{undefined}$$

This raises the question of divergent volume integrals. The treatment of these differ in different CA-systems. Before we list the different approaches and discuss their merits and drawbacks, let us first think about the goals:

- Integrals should in some way behave "reasonably". For example, the integral of a continuous strictly positive function should either be finite or ∞.
- Fubini's lemma should hold, that is, the value of an integral should not depend on the order of integration. Notice that, strictly speaking, this lemma has the condition if integrability, so it does not directly apply to divergent integrals. However, using nested integrals as volume integrals makes this requirement absolutely reasonable.

As we shall see from two examples, with the current approach these two goals are not compatible. The current approach taken is to evaluate the nested integrals from the innermost outwards, possibly after noting the integration interval in some sort of "property".

The two examples we shall look at are the following:

$$\int_0^\infty \int_0^\infty \frac{1}{1+x+y}\,\mathrm{d}x\,\mathrm{d}y \stackrel{!}{=} \infty$$

$$\int_0^\infty \int_0^\infty (x-y)\,\mathrm{d}x\,\mathrm{d}y \stackrel{!}{=} \int_0^\infty \int_0^\infty (x-y)\,\mathrm{d}y\,\mathrm{d}x$$

We have conducted tests with four computer algebra systems: *MuPAD* 2.5, Mathematica 4.1.5, Maple 7, and Derive 5.05.

For the first integral, *MuPAD* returns "undefined", Mathematica yields a couple of warnings and returns the integral as typed in, while Maple and Derive return ∞. Mathematica seems to be unable to compute the inner integral, which "is" ∞ for any real value of y. The other three systems evaluate this inner integral to ∞. Of the four systems, only *MuPAD* refuses to return ∞ as the value of $\int_0^\infty \infty \,\mathrm{d}y$, as users would expect.

For the second example, *MuPAD* returns "undefined" for both integrals. The other three systems (apparently) evaluate the inner integrals first, getting ∞ and $-\infty$ and return ∞ for the left hand side of the equation and $-\infty$ for the right hand side. Obviously, the *volume* integral of this function is not defined—the integral over any bounded *square* with the origin as a corner is zero, while the integral over general rectangles may take any value whatsoever.

Note that nothing of the above relies on using an unbounded integration interval. By a simple transformation such as $(x, y) \mapsto (1/(x+1), 1/(y+1))$, the examples can be modified to use an unbounded function over a bounded set. The presentation above has been chosen for clarity.

Summary

Computer algebra systems use quantities like ∞ to represent properly divergent integrals. At the same time, users are forced to write nested (real or contour) integrals to denote volume integrals, which in itself is a strong limitation on the volumes over which one can integrate.

Unfortunately, the combination of these two facts leads straight into conflicting design goals. No system is known yet which yields the expected answers for two very simple examples.

Making MuPAD fit for MathML

Ralf Hillebrand and Andreas Sorgatz
MuPAD Research Group, University of Paderborn, mailto:tonner@mupad.de
SciFace Software, mailto:sorgatz@sciface.com

MathML gets more and more important for displaying mathematical formulas in web-based applications as well as for exchanging mathematical expressions between software components. Taking this into account and, last not least, due to the involvement in BMBF projects like mαth-kit [10] and in²Math [11] and Uni-Mobilis at the University of Paderborn where MuPAD is used in teaching materials for eLearning or Blended Learning, respectively, we started to make the forthcoming version of MuPAD fit for MathML.

MathML defines a standard for describing mathematical expressions in formula layout and mathematical meaning in XML [1] for the presentation in the World-Wide-Web but also for the exchange of mathematical data between different tools like computer algebra systems and formula editors. More detailed information about MathML are available on the *MathML home page* [2] hosted at *World-Wide-Web Consortium (W3C)* [3].

The W3C released MathML 1.0 as a recommendation in April 1998. The current version of this standard is MathML 2.0 [4], a W3C recommendation released on February 21, 2001.

MathML Presentation vs. MathML Content

The MathML specification includes two different parts: On the one hand, it specifies how to represent the layout of a mathematical formula. This is called *MathML Presentation*. Like, e.g., LaTeX [5], this concept can be used for displaying mathematical expressions. MathML Presentation can be used within documents based on HTML and XML or XHTML, respectively. The following example demonstrates how the formula layout of the expression $x \in \{A, B, C\}$ can be described in MathML Presentation, where `&Element;` denotes the "element sign" $\in$, the parentheses `{` and `}` are explicitly specified and `<!-- ... -->` are comment regions:

```
<mrow>                              <!-- opens a new row element -->
  <mi>x</mi>                        <!-- an identifier element -->
  <mo>&Element;</mo>                <!-- an operator -->
  <mfenced open='{' close='}'>      <!-- opens a fence element -->
    <mi>A</mi>                      <!-- an identifier element -->
    <mi>B</mi>                      <!-- an identifier element -->
    <mi>C</mi>                      <!-- an identifier element -->
  </mfenced>                        <!-- closes the fence -->
</mrow>                             <!-- closes the row -->
```

The advantage of this presentation is: (a) due to its strict XML structure, this specifications can easily be parsed in a software components and (b) because of the grid layout (rows and cols) it is relative easy to render and display the given expression.

However, no information about the mathematical semantics of the expression is included in this representation. For example, the element `<mfenced...>` is not limited to a set (in a mathematical sense) but can also be a list, a table or whatever.

That is where the second concept of MathML, *MathML Content*, comes into the game. It specifies how to represent the mathematical meaning of a formula. With respect to this, MathML compares to OpenMath [6] but does not support the broad bandwidth of mathematical domains of OpenMATH. The next example specifies the above expression in MathML Content:

```
<math xmlns="http://www.w3.org/1998/Math/MathML">
  <apply>              <!-- apply operator #1 to element #2 and #3 -->
    <in/>              <!-- operator 'is element of' -->
    <ci>x</ci>         <!-- element #2 -->
    <set>              <!-- element #3 is a 'set' -->
      <ci>A</ci>       <!-- entries of the set: 3 identifiers -->
      <ci>B</ci>
      <ci>C</ci>
    </set>
  </apply>
</math>
```

Here the mathematical operator *in* is applied to the *identifier* x, represented by the tag `ci`, and a *set*, represented by the tag `set` and consisting of the three *identifiers* A, B and C.

In the current developers version of MuPAD, an output like this can simply be generated using the command `generate::MathML` which works analog to the well known MuPAD command `generate::TeX` for generating LaTeX output. The example below demonstrates this:

MuPAD

```
>> generate::MathML(x in {A, B, C})
```

Output

```
<math xmlns='http://www.w3.org/1998/Math/MathML'>
  <semantics>
    <apply>
      <in/>
      <ci>x</ci>
      <set>
        <ci>A</ci>
        <ci>B</ci>
        <ci>C</ci>
      </set>
    </apply>
    <annotation encoding='MuPAD'>
      x in {A, B, C}
    </annotation>
  </semantics>
</math>
```

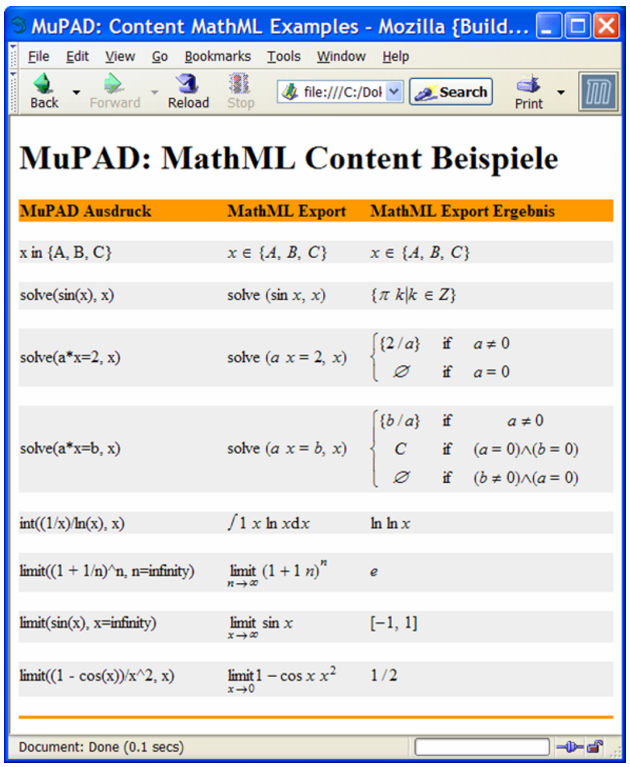
Note that MuPAD adds the representation of the mathematical expression in MuPAD as an annotation. The tag `<semantics>` declares the MathML Content part and the MuPAD annotation parts as a semantical unit. Otherwise a renderer would try to print the annotation.

From MathML Content to MathML Presentation

In spring 2002 the W3C consortium published a first draft version of XSL [7] style sheets for transforming MathML Content into MathML Presentation [8]. Modern web browsers are able to use such styles to transform XML or XHTML web pages on the fly before they display them. With this, the W3C style sheets can be used to display MathML Content in web browsers "directly", like with Mozilla 1.0 [9], which is able to natively render MathML Presentation. The following example demonstrates the use of the W3C style [1] within an XHTML (`.xhtml`) file:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="mathml.xsl"?>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>MuPAD: Content MathML Beispiele</title>
  </head>
  <body>
    ...
    <math xmlns='http://www.w3.org/1998/Math/MathML'>
      ...
    </math>
    ...
  </body>
</html>
```

This finally results in web pages like the following one, which contains some formulas generated with MuPAD using the command `generate::MathML`:



[1] In this example, a local copy of the W3C style sheet `mathml.xsl` is used. Refer to [8] for information about using a link to the W3C style sheet in the web and for downloading it for local use.

Summary and Perspectives

The MuPAD command `generate::MathML` is available with the current MuPAD developers version. Most MuPAD data types can already be exported in MathML, but some non-trivial are still missing. Ergo, this feature is still under development and needs to be refined and optimized within the next weeks.

The same is true for the W3C style sheets as well as for MathML rendering in Mozilla. Both still contain smaller bugs which lead to incorrect displaying of some kind of formulas. To be more independent from those developments, the future version of the MuPAD command `generate::MathML` will generate both MathML Content and MathML Presentation as well.

Supporting the MathML output for MuPAD formulas is one aspect of a more global strategy to support XML based communication, like XML output for 2D and 3D graphics, and HTML, like the HTML-Export of MuPAD Notebooks under Windows, in MuPAD.

Furthermore, MathML output will be available for the MuPAD Computing Server[12] and thus can be used for displaying MuPAD results within eLearning materials in the Web in a much nicer way.

References

[1] *Extensible Markup Language (XML)*, http://www.w3.org/XML

[2] *W3C Math Home*, http://www.w3.org/Math

[3] *W3C, World-Wide-Web Consortium Home Page*, http://www.w3.org

[4] *Mathematical Markup Language (MathML) Version 2.0*, http://www.w3.org/TR/MathML2

[5] *TeX Users Group Home Page*, http://www.tug.org/

[6] *The OpenMath Home Page*, http://www.openmath.org

[7] *The Extensible Stylesheet Language (XSL)*, http://www.w3.org/Style/XSL

[8] *Putting mathematics on the Web with MathML*, http://www.w3.org/Math/XSL

[9] *MathML in Mozilla 1.0*, http://www.mozilla.org/projects/mathml

[10] mαth-kit *Refer to the article on page 25 in this volume as well*, http://www.math-kit.de

[11] *in²Math*, http://www.uni-koblenz.de/ag-ki/PROJECTS/in2math

[12] *Mathematik im Web – Der MuPAD Computing Server*, mathPAD 10, Ausgabe 1, August 2001, Sonderausgabe "Schule und Studium", ISSN 0941-9187, http://www.mupad.de/mathpad.shtml

Involutive Bases in *MuPAD* – Part I: Involutive Divisions

Marcus Hausdorf and Werner M. Seiler
Lehrstuhl für Mathematik I, Universität Mannheim,
68131 Mannheim, Germany,
mailto:hausdorf@math.uni-mannheim.de
mailto:werner.seiler@math.uni-mannheim.de
http://www.math.uni-mannheim.de/~wms

This is the first of two articles on an implementation of involutive bases techniques in MuPAD. Whereas the computation of involutive bases in polynomial algebras of solvable type is treated in the second part, we introduce here the concept of an involutive division on multi indices, show how to complete a given set to involution and present MuPAD domains for the two most important involutive divisions, the Janet and the Pommaret division.

Introduction

Involutive bases are a special kind of non-reduced Gröbner bases. They have been introduced for polynomial ideals by Gerdt and collaborators (see e.g. [3, 4]) based on ideas from the Janet-Riquier theory of differential equations. Involutive bases are distinguished by special combinatorial properties and allow for a structural analysis of the ideals they span; in particular, they define Stanley decompositions. For more details, the reader is referred to [1, 9] and references therein.

The Janet-Riquier theory also motivated an explicit algorithm for the determination of involutive bases. It was first implemented in REDUCE and C by Blinkov and Gerdt; there followed packages for MAPLE [8] and MATHEMATICA [2]. A highly competitive implementation of Janet bases in C++ using intricate data structures by Gerdt and Yanovich [5, 6] could in many instances produce a Gröbner basis faster than traditional algorithms.

While not making full use of the above mentioned optimisation, the here presented implementation is the most general so far. Utilising *MuPAD*'s categories and domains capabilities, we can provide a completion algorithm for so-called polynomial algebras of solvable type. This comprises, for example, rings of linear differential or difference operators, the Weyl algebra or universal enveloping algebras of Lie algebras. We are going to look at the definition of a algebra of solvable type (the "polynomial" case) and the completion of ideal bases to involutive bases in the second part of this article; first, we want to explain the combinatorial ideas lying underneath by studying involutive divisions on multi indices (the "monomial" case). The transfer will then be rather straightforward.

The Idea behind Involutive Divisions

Involutive division were originally introduced for monomials, i. e. elements of the ring $\mathbb{K}[x_1, \ldots, x_n]$. But it suffices, at first, to work only with the exponent vectors. Therefore, we consider multi indices of a fixed length n, being elements of the Abelian monoid $(\mathbb{N}_0^n, +)$. Unfortunately, this leads to a slight confusion in terminology: instead of *multiplying* and *dividing* terms, we are now actually *adding* and *subtracting* multi indices.

For two multi indices $\mu = (\mu^1, \ldots, \mu^n)$ and $\nu = (\nu^1, \ldots, \nu^n)$, we say that $\mu|\nu$ (μ *divides* ν), if $\mu^i \leq \nu^i$ for all i. The set $C(\mu) = \mu + \mathbb{N}_0^n$ of all *multiples* of a given multi index is called its *cone*. For a finite set $\mathcal{N} = \{\mu_1, \ldots, \mu_r\}$ its *span* is the union $\langle \mathcal{N} \rangle = \bigcup_{\mu_i \in \mathcal{N}} C(\mu_i)$ of all the cones of its elements. Of course, the cones of $\mathcal{N}$ will always overlap (Fig. 11 on the left). If we desire a disjoint union of the span of $\mathcal{N}$, we have to restrict the directions in which the cones stretch; this is exactly the task of an involutive division. In the following definition, let $C_N(\mu)$, the *restricted cone* of μ with respect to $N \subseteq \{1, \ldots, n\}$, denote the set $\mu + \{\nu \in \mathbb{N}_0^n : \nu^i = 0 \text{ for } i \notin N\}$.

Definition 1. *An* involutive division L *on* $\mathbb{N}_0^n$ *is given by prescribing for each finite subset* $\mathcal{N} \subset \mathbb{N}_0^n$ *and for each multi index* $\mu \in \mathcal{N}$ *a set* $N_{L,\mathcal{N}}(\mu)$ *of* multiplicative indices *such that the following holds: (i) If* $C_{L,\mathcal{N}}(\mu)$ *is used as a shorthand for* $C_{N_{L,\mathcal{N}}(\mu)}(\mu)$*, then for all* $\mu, \nu \in \mathcal{N}$ *with* $C_{L,\mathcal{N}}(\mu) \cap C_{L,\mathcal{N}}(\nu) \neq \emptyset$ *either* $C_{L,\mathcal{N}}(\mu) \subseteq C_{L,\mathcal{N}}(\nu)$ *or* $C_{L,\mathcal{N}}(\nu) \subseteq C_{L,\mathcal{N}}(\mu)$*; (ii) if* $\mathcal{M} \subset \mathcal{N}$*, then* $\forall \mu \in \mathcal{M} : N_{L,\mathcal{N}}(\mu) \subseteq N_{L,\mathcal{M}}(\mu)$.

$C_{L,\mathcal{N}}(\mu)$ *is called the* involutive cone *of* μ *with respect to* L *and* $\mathcal{N}$*. We denote the complement of* $N_{L,\mathcal{N}}(\mu)$ *in* $\{1, \ldots, n\}$*, the* non-multiplicative indices *of* μ*, by* $\bar{N}_{L,\mathcal{N}}(\mu)$*. Finally, for* $\mu \in \mathcal{N}$ *and* $\nu \in \mathbb{N}_0^n$ *we write* $\mu \,|_{L,\mathcal{N}}\, \nu$ *(*μ involutively divides *or is an* involutive divisor *of* ν*), if and only if* $\nu \in C_{L,\mathcal{N}}(\mu)$.

Let us rephrase the two conditions for an involutive division L in the definition above: The first one says that if the involutive cones of two multi indices intersect, one cone must lie completely in the other one. The second condition requires that if we remove a multi index from $\mathcal{N}$, there must be at least the same multiplicative indices with respect to L for the remaining elements.

The right half of Fig. 11 shows the situation for an involutive division where the multiplicative indices are $\{1\}$ for $[2, 0]$ and $\{1, 2\}$ for $[0, 2]$. Actually, these are exactly the multiplicative indices for both the Janet and the Pommaret division, which will be defined below. Obviously, the two involutive cones do not intersect, but instead now we are missing the half-line starting at $[1, 2]$. We will thus have to *complete* the set $\{[0, 2], [2, 0]\}$.

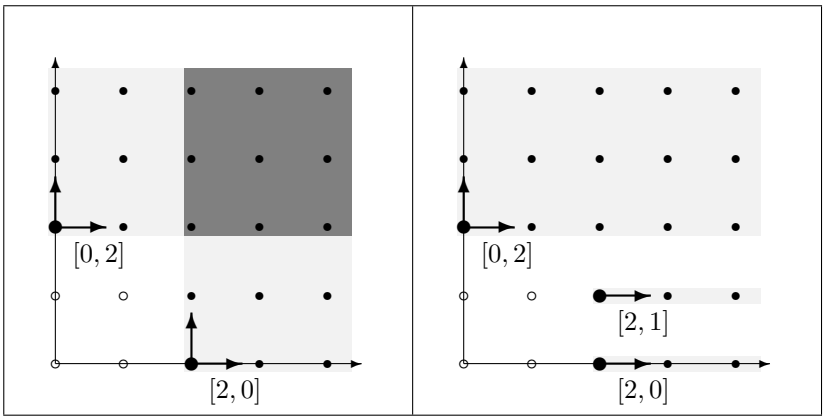


Figure 11: *Left:* intersecting cones. *Right:* involutive cones.

Definition 2. *Let* L *be an involutive division on* $\mathbb{N}_0^n$ *and* $\mathcal{N} \subset \mathbb{N}_0^n$ *a finite set.*

1. *The* involutive span *of $\mathcal{N}$ is the union of the involutive cones of its elements:*
$$\langle \mathcal{N} \rangle_L = \bigcup_{\mu \in \mathcal{N}} C_{L,\mathcal{N}}(\mu)\,. \tag{19}$$
2. *The set $\mathcal{N}$ is* (L-)involutively autoreduced, *if $C_{L,\mathcal{N}}(\mu) \cap C_{L,\mathcal{N}}(\nu) = \emptyset$ for all $\mu \neq \nu \in \mathcal{N}$.*
3. *$\mathcal{N}$ is called* weakly (L)-involutive, *if $\langle \mathcal{N} \rangle_L = \langle \mathcal{N} \rangle$ where $\langle \mathcal{N} \rangle$ denotes the ordinary span of $\mathcal{N}$, i. e. the monoid ideal $\langle \mathcal{N} \rangle = \mathcal{N} + \mathbb{N}_0^n$.*
4. *A finite subset $\hat{\mathcal{N}} \subset \langle \mathcal{N} \rangle$ is called a* weak involutive basis *of $\langle \mathcal{N} \rangle$, if $\langle \hat{\mathcal{N}} \rangle_L = \langle \mathcal{N} \rangle$. If $\hat{\mathcal{N}}$ contains $\mathcal{N}$, it is a* weak (L)-completion *of $\mathcal{N}$. If the set $\hat{\mathcal{N}}$ is furthermore autoreduced (i. e., no multi index is a multiple of another one in the set), it is a* (strong) involutive basis *of $\langle \mathcal{N} \rangle$ or a* (strong) completion *of $\mathcal{N}$, respectively.*
5. *An involutive division L is called* Noetherian, *if every finite set $\mathcal{N} \subset \mathbb{N}_0^n$ of multi indices has a completion.*

For an involutively autoreduced set the union in (19) is disjoint. It is a nice exercise to show that any weak involutive basis contains a strong involutive basis as a subset. Only the two properties in Def. 1 are needed for this. The three multi indices $\{[0,2],[1,2],[2,0]\}$ are a strong involutive basis and a completion of the initial set $\{[0,2],[2,0]\}$ with respect to both the Janet and the Pommaret division.

The actual algorithm for the completion of a set of multi indices to involution is very similar to the Buchberger algorithm for the construction of Gröbner bases. Buchberger's criterion for S-polynomials is replaced by the criterion of *local involution*: A set $\mathcal{N}$ of multi indices is called *locally involutive* with respect to the involutive division L, if for every $\mu \in \mathcal{N}$ and $i \in \bar{N}_{L,\mathcal{N}}(\mu)$, there exists some $\nu \in \mathcal{N}$ with $\mu + 1_i \in C_{L,\mathcal{N}}(\nu)$. That means, if we go one step from a multi index $\mu \in \mathcal{N}$ into a non-multiplicative direction, we always arrive in the multiplicative cone of another multi index. In [3], it is proved that local involution of a set implies its involution, provided the involutive division L satisfies some rather technical property called continuity. For almost all involutive divisions used in practice (especially the Janet and Pommaret divisions), this is the case.

Algorithm: *Involutive completion in $\mathbb{N}_0^n$*
Input: Finite subset $\mathcal{N} \subset \mathbb{N}_0^n$, involutive division L, term order $\preceq$ on $\mathbb{N}_0^n$
Output: Involutive basis of $\langle \mathcal{N} \rangle$

/1/ `repeat`
/2/ $\quad \mathcal{S} \leftarrow \{\mu + 1_i \,:\, \mu \in \mathcal{N},\, i \in \bar{N}_{L,\mathcal{N}}(\mu),\, \mu + 1_i \notin \langle \mathcal{N} \rangle_L\}$
/3/ $\quad \mathcal{N} \leftarrow \mathcal{N} \cup \{\min_{\preceq} \mathcal{S}\}$
/4/ `until` $\mathcal{S} = \emptyset$
/5/ `return` `InvAutoReduce`$_L(\mathcal{N})$

Figure 12: Completion in $\mathbb{N}_0^n$

The algorithm for the completion of a set of multi indices to completion is shown in Fig. 12. For it to work, in addition to continuity the involutive division L is required to be constructive (an even more complicated condition, which can be read about in [3] but nevertheless is valid for all divisions we consider here). Notice the parallels between this and the Buchberger algorithm: Instead of examining S-polynomials of critical pairs, we investigate all non-multiplicative multiples; instead of computing normal forms, we check whether the multiples can be obtained multiplicatively. The polynomial algorithm, which will be formulated in the second part, will work in the same way; then actually involutive normal forms will be computed. The choice of the next element to be treated in line /3/ with respect to a term order is necessary for the proof of termination. Of course, the algorithm does in general not terminate for a non-Noetherian involutive division.

The Category of Involutive Divisions

`Cat::InvolutiveDivision(n)` contains the basic operations with involutive divisions. The parameter `n` is the length of the multi indices the involutive divisions is defined on. The category is only a member of `Cat::BaseCategory`. The following undefined entries must be provided by the domains representing the different involutive divisions:

`sepListMult(mu_l):` returns a list of lists of the form `[mu, mv_s]` giving for each multi index `mu` in `mu_l` its multiplicative indices `mv_s` with respect to the involutive division and the multi indices in `mu_l`.

`sepListMultAdd(sep_l,mu):` given a separation list (as returned by `sepListMult`), the multi index `mu` is inserted into `sep_l`; the multiplicative indices of the multi indices in the new list are recomputed (with respect to all elements).

`sepListMultRem(sep_l,mu):` same as `sepListMultAdd(sep_l,mu)`, only that `mu` is removed from `sep_l`.

`multDirs(mu_l,mu):` returns the set of multiplicative directions for the multi index `mu` with respect to the multi indices in `mu_l`.

For the following entries, default implementations are present in the category:

`sepListNonMult`, `sepListNonMultAdd`, `sepListNonMultRem`, `nonMultDirs`: same as the respective methods above, only for non-multiplicative indices.

`isInvDivisor(mu,nu,mv_s):` returns `nu`$-$`mu` if `nu` lies in the cone restricted by the directions in the set `mv_s` and 0 otherwise.

`autoreduce(mu_l):` returns an autoreduced list of multi indices with the same span as the multi indices in the original list `mu_l`.

`invAutoreduce(mu_l):` returns an involutively autoreduced list of multi indices with the same involutive span as the multi indices in the original list `mu_l`.

`complete(mu_l):` returns an involutively autoreduced list of multi indices with the same span as the multi indices in the original list `mu_l`, i. e., a (strong) involutive basis.

The Janet Division

At long last, we give the definition of the first of the two important involutive divisions most frequently encountered. As before, let $\mathcal{N} = \{\mu_1, \ldots, \mu_r\}$ denote a set of multi indices of length n; furthermore, we write $\mathcal{T}_{\mathcal{N},k}(\nu) = \{\mu \in \mathcal{N} : \mu^i = \nu^i, k \leq i \leq n\}$ for the subset of $\mathcal{N}$ consisting of those multi indices of $\mathcal{N}$ the last k indices of which coincide with those of ν. For the Janet division $\mathcal{J}$, we then have:

- $n \in N_{\mathcal{J},\mathcal{N}}(\nu)$, if $\nu^n = \max_{\mu\in\mathcal{N}}\{\mu^n\}$;
- $n > m \in N_{\mathcal{J},\mathcal{N}}(\nu)$, if $\nu^m = \max_{\mu\in\mathcal{T}_{\mathcal{N},m+1}(\nu)}\{\mu^m\}$

This seemingly convoluted definition unfolds into a rather straightforward algorithm for computing the multiplicative indices of $\mathcal{N}$ with respect to the Janet division.

Algorithm: *Multiplicative Indices for the Janet Division*
Input: list $\mathcal{N} = [\mu_1, \ldots, \mu_r]$ of pairwise different multi indices of length n,
Output: list $\mathcal{M} = [N_{\mathcal{N},\mathcal{J}}(\mu_1), \ldots, N_{\mathcal{N},\mathcal{J}}(\mu_r)]$ of multiplicative indices

/1/ $\mathcal{N} \leftarrow$ `sort`$(\mathcal{N}, \succeq_{\text{ilex}})$; $\nu \leftarrow \mathcal{N}[1]$; $p_1 \leftarrow n$; $N \leftarrow \{1, \ldots, n\}$; $\mathcal{M}[1] \leftarrow N$;
/2/ `for` k `from` 2 `to` r `do`
/3/ $p_2 \leftarrow \max\{i : (\nu - \mathcal{N}[k])^i \neq 0\}$; $N \leftarrow N \setminus \{p_2\}$;
/4/ `if` $p_1 < p_2$ `then`
/5/ $N \leftarrow N \cup \{p_1, \ldots, p_2 - 1\}$;
/6/ `end_if`;
/7/ $\mathcal{M}[k] \leftarrow N$; $\nu \leftarrow \mathcal{N}[k]$; $p_1 \leftarrow p_2$;
/8/ `end_for`;
/9/ `return`$(\mathcal{M})$;

Figure 13: *Multiplicative Indices for the Janet Division*

We start by ordering the list $\mathcal{N}$ decreasingly inverse lexicographically. For the first multi index in $\mathcal{N}$, obviously all directions are multiplicative. We now proceed by comparing each element of $\mathcal{N}$ with its successor. The position at which they differ is p_2, while p_1 holds the appropriate value from the last iteration. Clearly, $\mathcal{N}[k-1]$ and $\mathcal{N}[k]$ both lie in $\mathcal{T}_{\mathcal{N},p_2-1}$, and because of the ordering and the definition of the Janet division, p_2 must be non-multiplicative for $\mathcal{N}[k]$. If $p_2 > p_1$, the determination whether the indices $p_1, \ldots, p_2 + 1$ are multiplicative or not is not affected by $\mathcal{N}[k-1]$, so they must become multiplicative (if they have not been before).

The domain `Dom::JanetDivision(n)` takes as optional argument a multi index of length n representing a permutation exerted on all multi indices when computing the multiplicative indices. So if the permutation is given by $[\pi_1, \ldots, \pi_n]$, a multi index μ is transformed into $[\mu^{\pi_1}, \ldots, \mu^{\pi_n}]$. This is especially useful since different authors define the Janet division differently (some start at the front, some at the back). The same holds for the Pommaret division, and so this optional parameter is also valid there.

The algorithm of Fig. 13 has the advantage that if a multi index is inserted (resp. removed) from a list of multi indices for which the multiplicative indices are already known, a recomputation is only required from the position of the new (resp. the removed) multi index in the list. The starting values for N and p_1 are readily computed from the preceding two multi indices.

The Pommaret Division

As a second domain for an involutive division, `Dom::PommaretDivision(n)` is implemented. The definition of the Pommaret division is rather simple: For a multi index μ, the class $cls(\mu) = \min\{i : \mu^i \neq 0\}$ is its leftmost non-vanishing entry; we take as multiplicative indices for μ all those smaller than or equal to $cls(\mu)$. An important difference to the Janet division is that the multiplicative directions of a multi index are fixed *a priori* and thus independent of the set $\mathcal{N}$ currently considered. Such an involutive division is called *globally defined*. While this property allows for a much simpler computation of multiplicative indices, the Pommaret division is unfortunately not Noetherian. Completions exist only in special coordinate systems called δ-regular (see [7, 9] for more details and a constructive solution of this problem). The Pommaret division is of great theoretical importance for the structure analysis of polynomial modules and also used in the hybrid geometric-algebraic completion algorithm described in [7] and implemented as part of the *MuPAD* DETools-library.

Let us finish with a simple example of how all these things work. We create domains for the Janet and the Pommaret division and consider the situation of Fig. 11. The completion is the same for both divisions.

MuPAD

```
>> JD:= Dom::JanetDivision(2): PD:= Dom::PommaretDivision(2): mu_l:= [[0,2],[2,0]]:
>> PD::complete(mu_l);
```

Output

```
                                [[2, 0], [0, 2], [2, 1]]
```

If we translate the multi indices of the starting set one step to the right, the multiplicative indices differ. Now there does not exist a finite completion with respect to the Pommaret division anymore: infinitely many multi indices would have to be added ($[3,1],[1,3],[1,4],[1,5],\ldots$). The Janet division does not have any problems.

MuPAD

```
>> mu_l:= [[1,2],[3,0]]: PD::sepListMult(mu_l); JD::sepListMult(mu_l);
>> JD::complete(mu_l);
```

Output

```
                              [[[1, 2], {1}], [[3, 0], {1}]]
                            [[[1, 2], {1, 2}], [[3, 0], {1}]]
                                [[1, 2], [3, 0], [3, 1]]
```

References

[1] J. Calmet, M. Hausdorf, and W.M. Seiler. A constructive introduction to involution. In R. Akerkar, editor, *Proc. Int. Symp. Applications of Computer Algebra — ISACA 2000*, pages 33–50. Allied Publishers, New Delhi, 2001.

[2] V.P. Gerdt, M. Berth. Computation of Involutive Bases with Mathematica. *Proc. Workshop on Mathematica System in Teaching and Research*, pages 29–34. University of Podlasie, Sieldce, 2001.

[3] V.P. Gerdt and Yu.A. Blinkov. Involutive bases of polynomial ideals. *Math. Comp. Simul.*, 45:519–542, 1998.

[4] V.P. Gerdt and Yu.A. Blinkov. Minimal involutive bases. *Math. Comp. Simul.*, 45:543–560, 1998.

[5] V.P. Gerdt and D.A Yanovich. Construction of Janet Bases I. Monomial Bases. In V.G. Ganzha, E.W. Mayr, E.V.Voroszhtsov, editors, *Computer Algebra in Scientifing Computing – CASC 2001*, pages 233–247. Springer, Berlin, 2001.

[6] V.P. Gerdt and D.A Yanovich. Construction of Janet Bases II. Polynomial Bases. In V.G. Ganzha, E.W. Mayr, E.V.Voroszhtsov, editors, *Computer Algebra in Scientifing Computing – CASC 2001*, pages 249–263. Springer, Berlin, 2001.

[7] M. Hausdorf and W.M. Seiler. An Efficient Algebraic Algorithm for the Geometric Completion to Involution. *Appl. Alg. Eng. Comp. Comm.*, 2002, to appear.

[8] V.A. Mityunin. Implementation of Differential Involutive Algorithm in Maple V5. *Proc. IMACS Conference on Applications of Computer Algebra ACA 2000*, pages 38–39, St. Petersburg, 2000.

[9] W.M. Seiler. Involution — The Formal Theory of Differential Equations and Its Applications in Computer Algebra and Numerical Analysis. Habilitation thesis, Dept. of Mathematics, Universität Mannheim, 2001.

The new Statistics Library in MuPAD 2.5

Walter Oevel
MuPAD Research Group, University of Paderborn, mailto:walter@mupad.de

With MuPAD Release 2.5, the `stats` *library for statistical computations was extended significantly to almost 100 routines. Thus, it became one of the largest specialized application packages in the MuPAD system. We give a brief survey of its functionality.*

Overview

The `stats` package provides methods for statistical analysis. In previous MuPAD versions it used to be a rather small library containing only a few routines for handling and analyzing statistical data. With MuPAD version 2.5, a variety of new functions were introduced to the package. It now features 17 statistical distributions, routines for data analysis, statistical tests, regression analysis, data structures for statistical samples and utility functions for handling and manipulating data samples. Import of statistical data coming from external sources is provided by `import::readdata`. Statistical data can also be analyzed visually via "boxplots" provided by the specialized graphical routine `plot::boxplot`.

Various standard distributions were implemented. There are four routines associated with each distribution "`xxx`":

- a cumulative distribution function "`xxxCDF`";
- a probability density function "`xxxPDF`" (continuous case) or a probability function "`xxxPF`" (discrete case);
- a quantile function "`xxxQuantile`" (the functional inverse of "`xxxCDF`"),
- a random generator "`xxxRandom`".

The continuous distributions β, Cauchy, χ^2, Erlang, Exponential, Fisher's f-distribution, γ, Logistic, Normal, Student's t-distribution, Uniform, and Weibull are implemented:

CDF	PDF	Quantile	Random Generator
`betaCDF`	`betaPDF`	`betaQuantile`	`betaRandom`
`cauchyCDF`	`cauchyPDF`	`cauchyQuantile`	`cauchyRandom`
`chisquareCDF`	`chisquarePDF`	`chisquareQuantile`	`chisquareRandom`
`erlangCDF`	`erlangPDF`	`erlangQuantile`	`erlangRandom`
`exponentialCDF`	`exponentialPDF`	`exponentialQuantile`	`exponentialRandom`
`fCDF`	`fPDF`	`fQuantile`	`fRandom`
`gammaCDF`	`gammaPDF`	`gammaQuantile`	`gammaRandom`
`logisticCDF`	`logisticPDF`	`logisticQuantile`	`logisticRandom`
`normalCDF`	`normalPDF`	`normalQuantile`	`normalRandom`
`tCDF`	`tPDF`	`tQuantile`	`tRandom`
`uniformCDF`	`uniformPDF`	`uniformQuantile`	`uniformRandom`
`weibullCDF`	`weibullPDF`	`weibullQuantile`	`weibullRandom`

The following discrete distributions are implemented: Binomial, Geometric, Hypergeometric and Poisson. Further, for the statistical analysis of given data, the cumulative empirical distribution function and its inverse (the quantile function) is available:

CDF	PF	Quantile	Random Generator
`binomialCDF`	`binomialPF`	`binomialQuantile`	`binomialRandom`
`geometricCDF`	`geometricPF`	`geometricQuantile`	`geometricRandom`
`hypergeometricCDF`	`hypergeometricPF`	`hypergeometricQuantile`	`hypergeometricRandom`
`poissonCDF`	`poissonPF`	`poissonQuantile`	`poissonRandom`
`empiricalCDF`		`empiricalQuantile`	

The following procedures serve for analyzing statistical data (descriptive statistics):

`stats::covariance`	the covariance of data samples
`stats::geometricMean`	the geometric mean of a data sample
`stats::harmonicMean`	the harmonic mean of a data sample
`stats::kurtosis`	the kurtosis (excess) of a data sample
`stats::mean`	the (arithmetic) mean of a data sample
`stats::meandev`	the mean deviation of a data sample
`stats::median`	the median value of a data sample
`stats::modal`	the modal (most frequent) value(s) in a data sample
`stats::moment`	the k-th moment of a data sample
`stats::obliquity`	the obliquity (skewness) of a data sample
`stats::quadraticMean`	the quadratic mean of a data sample
`stats::stdev`	the standard deviation of a data sample
`stats::variance`	the variance of a data sample

The following statistical standard tests are implemented:

`stats::csGOFT`	the classical χ^2 goodness-of-fit test
`stats::ksGOFT`	the Kolmogorov-Smirnov goodness-of-fit test
`stats::swGOFT`	the Shapiro-Wilk goodness-of-fit test for normality
`stats::tTest`	the t-test for a mean

A utility function `stats::equiprobableCells` was implemented to create equiprobable cells for the χ^2 goodness-of-fit test.

Both linear as well as nonlinear regression analysis is available:

`stats::linReg`	linear regression (least squares fit)
`stats::reg`	general linear and nonlinear regression (general least squares fit)

The following specialized data structure is available to hold statistical data:

`stats::sample` : the domain of statistical samples

The following utility functions are available for the manipulation of statistical data contained in a sample of type `stats::sample`:

`stats::calc`	apply functions to a sample

`stats::col`	select and re-arrange columns of a sample
`stats::concatCol`	concatenate samples column-wise
`stats::concatRow`	concatenate samples row-wise
`stats::row`	select and re-arrange rows of a sample
`stats::sample`	generate a sample of type `stats::sample`
`stats::sample2list`	convert a sample to a list of lists
`stats::selectRow`	select rows of a sample
`stats::sortSample`	sort the rows of a sample
`stats::tabulate`	statistics of duplicate rows in a sample
`stats::unzipCol`	extract columns from a list of lists
`stats::zipCol`	convert a sequence of columns to a list of lists

MuPAD provides a variety of utilities for importing data. See the help page `?fileIO` for an overview of all routines that are available. In particular, statistical data contained in an ASCII file can be read into a MuPAD session via the function `import::readdata`.

MuPAD provides a variety of plotting utilities for discrete data such as `plot::data, plot::Pointlist` etc. A special type of plot often used in a statistical context is provided by the function `plot::boxplot`. It helps to analyze and compare statistical data samples visually via 'boxplots'.

Working with the `stats` Package: an Example

We demonstrate the capabilities of the `stats` package by performing a data analysis as might be required in a realistic application. First, we generate two statistical samples `data1` and `data2` via random generators provided by the library:

MuPAD
```
>> rand1 := stats::uniformRandom(-1, 2):
>> data1 := [rand1() $ k = 1.. 1000]:
>> rand2 := stats::normalRandom(5, 2):
>> data2 := [rand2() $ k = 1.. 1000]:
```

The data are written to an ASCII file "`data.txt`":

MuPAD
```
>> file := fopen("data.txt", Append, Text):
>> for i from 1 to 1000 do
>>   fprint(file, data1[i], data2[i]);
>> end_for:
>> fclose(file):
```

The file now contains 1000 lines of text, each consisting of two floats separated by a colon:

```
-0.1889298903:4.998222155:
1.442803572:4.448286266:
-0.6562067682:4.331347777:
...
```

We treat the data as being given by some external source and demonstrate how to import ASCII data into a MuPAD session. The routine `import::readdata` is the tool to read such data from a file. Specifying the separator ":", each line is imported as a list with two elements:

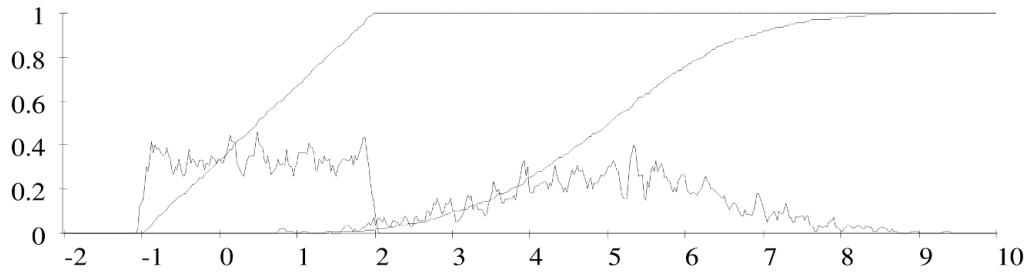
```
MuPAD
>> data:= import::readdata("data.txt", ":"):
```

The data now consist of a list with 1000 entries, each entry being a sublist of two elements representing one line in the file. This nested list may be regarded as a matrix-like structure of dimension 1000×2. Each of the two columns shall be interpreted as a separate sample. We convert the data to a `stats::sample` object and use the method `col2list` to convert the columns to lists (they coincide with the original lists `data1`, `data2` we started with):

```
MuPAD
>> s:= stats::sample(data):
>> data1:= s::dom::col2list(s, 1):
>> data2:= s::dom::col2list(s, 2):
```

Pretending that we do not know the distribution of the data, we wish to analyze the data with the aim of identifying their distribution. First, we start with a visual analysis by plotting the empirical cumulative distribution functions (CDFs) of the two samples. The plot also includes numerical derivatives of the CDFs (a finite difference approximation) to get a rough idea of the form of the corresponding probability densities functions (PDFs):

```
MuPAD
>> CDF1 := stats::empiricalCDF(data1):
>> CDF2 := stats::empiricalCDF(data2):
>> h:= 0.1:
>> PDF1(x) := (CDF1(x + h/2) - CDF1(x-h/2))/h:
>> PDF2(x) := (CDF2(x + h/2) - CDF2(x-h/2))/h:
>> plotfunc2d(CDF1(x), PDF1(x), CDF2(x), PDF2(x) , x = -2..10,
>>            Title = "", Labels = ["", ""], FontSize = 16, Grid = 500,
>>            Ticks = [Steps = 1, Steps = 0.2], Axes = Corner)
```



The CDF1 looks like a linear ramp, indicating a uniform distribution of the first sample supported on the interval $[-1, 2]$ (approximately). The PDF2 resembles a Gaussian function, indication a normal distribution. We apply the non-parametric Shapiro-Wilk goodness-of-fit test for normality implemented by `stats::swGOFT`:

```
MuPAD
>> stats::swGOFT(data1), stats::swGOFT(data2)
Output
          [0.9589770221, 3.93240128e-16], [0.9980432669, 0.3012263123]
```

The attained significance levels are the second entries of the lists. The first sample attains a tiny significance level, i.e., the hypothesis of a normal distribution for the first sample can clearly be rejected. The second sample passes the test well with a significance level of $0.30...$.

We have a closer look at the first sample. The visual analysis indicated a uniform distribution on the interval $[-1, 2]$. We do a regression analysis on the CDF1 by looking for parameters a, b such that the linear model function $a \cdot x + b$ approximates the CDF1 on this interval in the least squares sense. We discretize the interval by 101 equidistant sample points:

```
MuPAD
>> samplepoints := [-1 + 3*i/100 $ i = 0..100]:
>> [a, b] := stats::linReg([[x , CDF1(x)] $ x in samplepoints]):
>> [a, b] := float([a, b])
Output
                              [0.32762318, 0.3374269074]
```

From these optimized fit parameters we get estimates of the interval supporting the distribution by looking at the points where the model CDF $a \cdot x + b$ attains the values 0 and 1, respectively:

```
MuPAD
>> solve(a + b*x = 0, x), solve(a + b*x = 1, x)
Output
                           {-0.9709456264}, {1.992659166}
```

Summarizing the analysis for the first data sample, we found that the data seem to be uniformly distributed on the interval $[a, b]$ with $a \approx -1$ and $b \approx 2$.

Now we turn to the second sample. The visual analysis indicated a normal distribution, which was supported by the Shapiro-Wilk test above. Hypothesizing a normal distribution, we need estimates of the mean and the variance to proceed with further (parametric) tests:

```
MuPAD
>>  [m, v] := [stats::mean(data2), stats::variance(data2)]
Output
                             [5.01946907, 2.060468717]
```

We do a regression test: search for values of the mean `M` and the variance `V`, such that the quadratic deviation between the data encoded in `CDF2` and the cumulative distribution function of the normal distribution with

mean M and variance V is minimized. The empirical distribution CDF2(x) of the data and the model function stats::normalCDF(M, V)(x) are sampled at 101 equidistant points between 0 and 10:

```
MuPAD
>> samplepoints := [i/10 $ i = 0..100]:
>> stats::reg([[x , CDF2(x)] $ x in samplepoints],
>>            stats::normalCDF(M, V)(x), [x], [M, V],
>>            StartingValues = [5.02, 2.06])
Output
                  [[5.014889699, 1.968364751], 0.003212084222]
```

The optimized fit parameters $M \approx 5.01$ and $V \approx 1.97$ are close to the empirical values for the mean and the variance found before. The last number returned by stats::reg is the quadratic deviation between the data in CDF2 and the model function with the optimized mean M and the optimized variance V. This deviation is rather small indicating a good fit. This suggests that the data are indeed normally distributed with mean ≈ 5 and variance ≈ 2. Next, we apply the classical χ^2 goodness-of-fit test implemented by stats::csGOFT to support this conjecture. We use the empirical estimates m and v computed above and test whether the data are consistent with the hypothesis of a normal distribution with the estimated parameters. For the test, we partition the real line into 32 cells that are equiprobable w.r.t. the normal distribution:

```
MuPAD
>> cells:= stats::equiprobableCells(32, stats::normalQuantile(m, v)):
>> stats::csGOFT(data2, cells, CDF = stats::normalCDF(m, v))
Output
                    [36.28800005, 0.7643756777, 31.24999998]
```

The second number $0.76...$ of the returned list is the significance level attained by the sample. This level is not small, i.e., there is no indication that the hypothesis should be rejected. Finally, we apply the Kolmogorov-Smirnov test implemented by stats::csGOFT to test the hypothesis of a normal distribution with the parameters m and v:

```
MuPAD
>> stats::ksGOFT(data2, CDF = stats::normalCDF(m, v))
Output
            [0.6250692939, 0.5481898803, 0.7179225561, 0.6486060967]
```

The two significance levels $0.548...$ and $0.648...$ are not close to 1, i.e., the data pass the test well (see the help page of stats::ksGOFT for an interpretation of the returned data). There is no indication that the hypothesis of the data being normally distributed should be rejected.

Summarizing the analysis for the second sample, we found that the data seem to be normally distributed with mean $m \approx 5$ and variance $v \approx 2$.

Numerics with Hardware Floats in MuPAD 2.5

Walter Oevel
MuPAD Research Group, University of Paderborn, mailto:walter@mupad.de

Numerical computations in MuPAD use multiprecision software floats. Therefore the software arithmetic in MuPAD is much slower than the arithmetic in special purpose numerical tools that are based on compiled C or Fortran code supported by hardware floats. With MuPAD Release 2.5, the numerical Scilab package was linked to the MuPAD system. This allows to send numerical tasks to the external Scilab tool for fast processing using hardware floats.

What is Scilab?

Scilab is a numerical package developed at INRIA Rocquencourt (`http://www-rocq.inria.fr/scilab`). As a standalone tool, it provides an interactive user interface to communicate with the Scilab interpreter. Numerous functions for numerical analysis are available, some of which are written in the Scilab language. Others are mere interfaces to C and Fortran routines from standard numerical packages such as LINPACK or, in the current versions of Scilab, LAPACK.

Scilab provides basic functionality such as matrix data types and their manipulation as well as various special libraries and toolboxes. In these libraries, numerous routines exist for various applications areas such as control theory, linear algebra, optimization, statistics etc.

The Generic MuPAD–Scilab Link

Overview

MuPAD 2.5 is available with or without the Scilab package as an add-on. If Scilab is available, MuPAD establishes a link to the Scilab system by starting an (invisible) external Scilab process without a window and sending/receiving data to/from the Scilab kernel. Technically, this is realized by a *dynamic module* providing a domain `scilab` to the MuPAD user. A Scilab routine `xyz`, say, is available in MuPAD as a method `scilab::xyz`. It can be called like any other MuPAD function.

The link is *generic*: Apart from the Scilab graphics, *all* functions in the Scilab installation are available and can be called from MuPAD. The idea is that the input data for Scilab routines are created as MuPAD objects. Calling a function `scilab::xyz(data)` converts the MuPAD data to a suitable collection of floating point data which are sent to the external Scilab process together with the information which Scilab routine is to be called. Scilab processes the data with the requested Scilab routine and sends the result back to MuPAD. There, the data are received, converted to MuPAD objects and handed to the current MuPAD session in which the result can be further processed by MuPAD commands.

The documentation of the MuPAD–Scilab link (call `?scilab` in the MuPAD session) provides a conversion table between MuPAD data types and corresponding Scilab data types. E.g., MuPAD matrices of type `DOM_ARRAY` or `Dom::Matrix()` are converted to (dense) Scilab matrices; MuPAD matrices of type `Dom::SparseMatrix()` are converted to sparse Scilab matrices.

The generic link between MuPAD and Scilab boosts MuPAD's functionality dramatically: *All* Scilab functions have become part of the MuPAD system and can by used like the functions of any other native MuPAD library.

The only drawback of this approach is that the user has to know the name and the calling syntax (number and types of input parameters) of the Scilab function that he wishes to use. *Some acquaintance with Scilab is required!*

Overhead

There certainly is some overhead using MuPAD as an interface to Scilab:

1) Before sending a MuPAD object to Scilab, it has to be checked for suitability. All operands in the MuPAD object are converted to floats. It is checked that this was successful and that all floats are of a suitable size such that they can be represented in Scilab.
2) Some time is needed for sending and receiving the data to/from Scilab.
3) The result computed by Scilab has to be converted to a MuPAD object.

Whilst the communication overhead 2) is almost negligible, the time needed for the conversions 1) and 3) can be observed and measured. Experiments have shown that for certain problems such as operations on large $n \times n$ matrices, the overhead typically amounts to the same order of magnitude as the computational costs in Scilab. In particular, for algorithms with costs $O(n^3)$ such as matrix multiplication, inversion, eigenvalue computations etc., the computing costs in Scilab exceed the overhead of the conversions if the data are sufficiently large.

In any case, when comparing the timings needed by Scilab together with the overhead with the timings needed by MuPAD's software floats, it is worthwhile to use the MuPAD–Scilab link even for small numerical problems. E.g., for the multiplication of $n \times n$ float matrices, the MuPAD–Scilab link beats MuPAD's software floats for matrices as small as $n = 5$ (on LINUX systems).

Demonstration

Let us give some demonstration of the generic link. First, we let Scilab generate a random float matrix of dimension 200×200. The matrix can be processed by functions of the MuPAD libraries. We compute the trace of A via `linalg::tr`:

MuPAD

```
>> A := scilab::rand(200, 200):
>> linalg::tr(A)
```

Output

```
                                  98.62451434
```

For computing the inverse of A, we may use MuPADs matrix arithmetic by calling `A^(-1)`. Note, however, that matrix inversion is a rather expensive algorithm ($O(n^3)$ for $n \times n$ matrices):

```
MuPAD
>> rtime(A^(-1))*msec
    Output
                                192999 msec
```

It pays off to let Scilab do the work. The Scilab function for inverting a matrix is `inv`:

```
MuPAD
>> rtime(scilab::inv(A))*msec
    Output
                                 753 msec
```

In Scilab, some functions need to assign their return values to a special "left hand side". An example is the QR factorization of a matrix, which has to be called in the form `[Q, R] = qr(A)` in Scilab. Without an assignment to a list `[Q, R]` of the matrix factors, the Scilab function `qr` raises an error. The generic MuPAD–Scilab link assumes simple function calls without assignments to special "left hand sides" and thus calls `qr` in an illegal way:

```
MuPAD
>> scilab::qr(A)
    Output
                          [FAIL, "Scilab Error", 41.0]
```

In such a case, a small Scilab program has to be sent. A Scilab program can be generated in MuPAD via the method `scilab::func`:

```
MuPAD
>> f := scilab::func(["A"], ["Q", "R"], ["[Q, R] = qr(A)"]):
```

In this declaration, the input parameter is the matrix A, the return values are the matrices Q and R. The program itself is a single Scilab command calling the `qr` function and assigning the result to the Scilab list `[Q, R]`.
The MuPAD function `f` can now be called in the MuPAD session, returning the sequence of return values Q and R specified in `scilab::func`:

```
MuPAD
>>  [Q, R] := [f(A)]:
```

We use Scilab to compute the determinants of Q, R and A:

```
MuPAD
>> scilab::det(Q), scilab::det(R), scilab::det(A)
    Output
                      -1.0, -2.364152509e80, 2.364152509e80
```

With `scilab::func`, arbitrarily complex Scilab routines may be defined. In the following example, we consider the numerical integration of an ordinary differential equation. The initial value problem to be solved is:

$$\frac{d}{dt}\, y(t) = f(t, y(t)) = t \cdot \sin(y(t)) + \cos(y(t)), \quad y(t_0) = y_0.$$

In Scilab, the numerical integrator is the routine `ode`. It requires the right hand side $f(t, y)$ of the differential equation as a procedure. Thus, the Scilab code needs to consist of two steps: i) declaring $f(t, y)$ as a scilab procedure, ii) passing this procedure together with the initial conditions to `ode`.

Correspondingly, the following code generated via `scilab::func` consists of two command lines:

i) The Scilab routine `deff` is used for the on-line definition of a suitable Scilab function; `deff` has to be called with Scilab strings (enclosed by single quotes).
ii) The numerical integrator `ode` is called.

MuPAD

```
>> Sci := scilab::func(["t0", "y0", "t"], ["y"],
>>        ["
>>          deff('[ydot] = f(t, y)', 'ydot = t*sin(y) + cos(y)');
>>          y = ode(y0, t0, t, f)
>>         "]):
```

In MuPAD, one may declare a corresponding integrator by the following steps:

MuPAD

```
>> f := (t, y) -> [t*sin(y[1]) + cos(y[1])]:
>> Mup := (t0, y0, t) -> numeric::odesolve(f, t0..t, [y0]):
```

We call both the Scilab integrator and MuPAD's ODE solver `numeric::odesolve` and compare the results:

MuPAD

```
>> t0 := 0: y0 := 0:
>> for t in [1, 10, 100] do
>>   ts := rtime((ys := Sci(t0, y0, t)))*msec:
>>   tm := rtime((ym := Mup(t0, y0, t)))*msec:
>>   print(Unquoted, "t" = t, Scilab = ys, ts, MuPAD = ym, tm);
>> end_for:
```

Output

```
      t = 1, Scilab = 1.134477454,  5 msec, MuPAD = [1.134477255], 63 msec

     t = 10, Scilab = 3.040908148, 11 msec, MuPAD = [3.040908158], 719 msec

    t = 100, Scilab = 3.131592018, 24 msec, MuPAD = [3.131591987], 14848 msec
```

Summary: The generic MuPAD–Scilab link allows to call arbitrary Scilab functions from a MuPAD session. Also arbitrary Scilab procedures can be created and launched from a MuPAD session. Despite the communication overhead between MuPAD and Scilab, the use of the link is worthwhile even for small numerical tasks.

Hardware Floats hidden in MuPAD's `numeric` library

The generic MuPAD–Scilab link described before requires some acquaintance with Scilab: the user has to know the name and the calling syntax of the Scilab function appropriate for the task.

In order to make the use of fast numerics easier for devoted MuPAD users, Scilab was hidden in some routines of MuPAD's `numeric` library in such a way that no knowledge of Scilab functions and their calling syntax is required.

Scilab uses double precision floats which corresponds to an arithmetic with about 16 decimal digits. If the MuPAD environment variable `DIGITS` determining the numerical working precision has a value smaller than 16, the following `numeric` routines call Scilab internally if it is available:

`numeric::det`	determinant of a matrix
`numeric::eigenvalues`	eigenvalues of a matrix
`numeric::eigenvectors`	eigenvectors of a matrix
`numeric::expMatrix`	exponential function for matrices
`numeric::factorCholesky`	Cholesky factorization of a matrix
`numeric::factorLU`	LU factorization of a matrix
`numeric::factorQR`	QR factorization of a matrix
`numeric::fft`	fast Fourier transform
`numeric::fMatrix`	functional calculus for matrices
`numeric::invfft`	inverse fast Fourier transform
`numeric::inverse`	inverse of a matrix
`numeric::leastSquares`	least squares solution of an overdetermined linear system
`numeric::linsolve`	solution of a linear system given by equations
`numeric::matlinsolve`	solution of a linear system given by a coefficient matrix
`numeric::singularvalues`	singular values of a matrix
`numeric::singularvectors`	singular value decomposition of a matrix

If the precision goal set by `DIGITS` is 16 or more decimal places, or if Scilab is not available, these functions use the software arithmetic of the MuPAD kernel to compute their result. The user notices the difference only by the time needed to get the answer.

Supported by hardware floats, computing the eigenvalues of a 300×300 matrix is a matter of seconds:

MuPAD

```
>> A := linalg::hilbert(300):
>> t := rtime():
>> eigenvals := numeric::eigenvalues(A);
>> (rtime() - t)*msec
```

Output

```
        [2.322019937, 1.033217168, 0.3433362682, ... , -3.147852847e-16]

                                    3314 msec
```

The use of hardware floats can be suppressed via the option `SoftwareFloats`. The hardware floats used internally in the previous call turn out to be about 100 times faster than MuPAD's software floats:

```
MuPAD
>> t := rtime():
>> eigenvals := numeric::eigenvalues(A, SoftwareFloats);
>> (rtime() - t)*msec
```

Output

```
[2.322019937, 1.033217168, 0.3433362682, ... , -4.387637692e-48]

                                  324225 msec
```

With Scilab in the background, the routine `numeric::matlinsolve` is the fastest numerical linear solver available in MuPAD 2.5. Using MuPAD's new domain for sparse matrices, there is no problem to generate and solve huge sparse systems $A \cdot x = b$. We consider the tridiagonal system

$$\begin{pmatrix} 2 & -1 & & & 0 \\ -1 & 2 & -1 & & \\ & -1 & \ddots & \ddots & \\ & & \ddots & \ddots & -1 \\ 0 & & & -1 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ \vdots \\ 1 \end{pmatrix}$$

which arises in the discretization of second order differential equations. We solve 10^4 equations for 10^4 unknowns:

```
MuPAD
>> n := 10000:
>> A := sparsematrix(n, n, [-1, 2, -1], Banded):
>> b := matrix([1 $ n]):
>> HardwareFloats = rtime(numeric::matlinsolve(A, b))*msec;
```

Output

```
                        HardwareFloats = 5309 msec
```

Also MuPAD's software floats can handle the task, needing more than 50 times the runtime of the previous call:

```
MuPAD
>> SoftwareFloats = rtime(numeric::matlinsolve(A, b, SoftwareFloats))*msec
```

Output

```
                        SoftwareFloats = 286005 msec
```

See the article *Dom::SparseMatrix – A domain for sparse matrices in MuPAD* by Kai Gehrs on page 69 in this volume for more information about sparse matrices in MuPAD 2.5.

Dom::SparseMatrix – A Domain for Sparse Matrices in MuPAD

Kai Gehrs
MuPAD Research Group, University of Paderborn, mailto:acrowley@mupad.de

Matrices arising from applications are often large and sparse. Only the non-zero components of such matrices should be stored. Algorithms such as matrix multiplication or linear solvers have to be optimized to benefit from the sparse structure.

The internal representation of the traditional (dense) MuPAD matrices created by `Dom::Matrix` is based on two dimensional arrays – a data structure in which every component of a matrix is stored regardless, whether this component is zero or not. The communication between MuPAD and the recently integrated numerical package Scilab is realized via matrices that are used as containers to transfer data between the two systems. In order to represent large matrix objects in MuPAD, it was necessary to develop a new MuPAD domain `Dom::SparseMatrix`. Since arrays are unsuitable for this purpose, the implementation of the new MuPAD domain is based on a different data structure.

The data structure

The data structure for sparse matrices has to satisfy the following requirements:

- It should be easy and cheap to store and access the non-zero components of a given matrix.
- The arithmetical operations such as addition and multiplication of matrices should be performed as fast as possible.

It turned out, that a combination of lists and univariante polynomials serves best to satisfy the above conditions. In the internal representation, each column of the matrix is stored as a univariate polynomial, whose degree equals the number of rows of the matrix. A list is used, in which the j-th component is a polynomial representing the j-th column of the matrix. In particular, the matrix

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix} \in R^{m \times n}$$

over some ring R is stored by the list $[p_1, \ldots, p_n]$, where $p_1, \ldots, p_n \in R[x]$ are polynomials in one variable x over R and

$$p_1 = a_{11} \cdot x + \ldots + a_{m1} \cdot x^m, \quad \ldots \quad , p_n = a_{1n} \cdot x + \ldots + a_{mn} \cdot x^m.$$

Since univariate polynomials of type `DOM_POLY` have a natural sparse structure in MuPAD (only the non-zero coefficients of a polynomial are stored in the kernel), our first demand is satisfied. The creation of polynomials as well as the extraction of coefficients via the kernel function `coeff` is quite fast indeed. The other request of

being able to perform fast standard arithmetical operations with matrices is also satisfied: e.g., the addition of two matrices represented by the above data structure can be reduced to m additions of pairs of polynomials.

Another advantage of using polynomials is the fact, that MuPAD can efficiently convert polynomials into lists and vice versa. The representation of ploynomials via lists keeps the sparsity. Hence, it is possible to switch between this two representations whenever necessary without significant loss of efficiency.

The idea of representing matrix data by polynomials had already been realized within the linear solvers in the `numeric` library of previous MuPAD versions. This turned out to be quite efficient and successful.

Why do we use polynomials to represent columns and not rows of the matrix?

Both alternatives would have been possible. Representing rows by polynomials is more efficient when performing gaussian elimination since row operations are ubiquitous in this algorithm. On the other hand, the multiplication $A \cdot v$ of a matrix A by a column vector v is implemented more efficiently when the columns of a matrix are encoded by polynomials, since matrix-vector multiplication boils down to computing linear combinations of column polynomials. It was decided to put emphasis on matrix-vector multiplication as the most common operation. For this reason, a column representation was chosen.

Functionalties of the domain

`Dom::SparseMatrix` offers all mathematical functions, which are available for elements of the traditional matrix type `Dom::Matrix`. These include matrix arithmetic, (standard and fraction free) gaussian elimination, transposition, creation of random and special banded matrices and a lot more. Additionally, the whole functionality of the libraries `numeric` for numerical computations and `linalg` for linear algebra can be used for sparse matrices.

In contrast to `Dom::Matrix`, some additonal features were included:

- The `new`-method was extended: it is possible to create matrices from tables (note that MuPAD tables also represent a sparse data container).
- The `random`-method was extended: it is now easy to generate large and sparse random matrices.
  - `random(m, n)` – generates a dense random matrix with m rows and n columns.
  - `random(m, n, p)` – generates a random $m \times n$ matrix with at most p non-zero entries that are placed randomly.
  - `random(m, n, f)` – generates a dense $m \times n$ matrix with entries created by the random generator f.
  - `random(m, n, p, f)` – generates an $m \times n$ matrix with at most p non-zero entries created by the random generator f.

  The last two calls of `random` is supported by the new statistic library in MuPAD 2.5, which offers various new random generators.
- A new method `doprint` to print large and sparse matrices was included. This method prints only non-zero components of a sparse matrix A in the form $(i, j) =$ value.
- A new method `nonZeroes` returns the number of non-zero components of a sparse matrix.
- A new method `nonZeroOperands` returns a sequence of the non-zero components of a sparse matrix.
- A new method `mapNonZeroes`, which corresponds to the MuPAD function `map`. In contrast to `map`, the routine `mapNonZeroes` can be used to a map a function only to the non-trivial components of a sparse matrix. Note that mapping a function f with $f(0) \neq 0$ turns a sparse matrix into a dense on, when `map` is used.

Many of the standard routines of `Dom::SparseMatrix`, e.g. the transposition routine, use special heuristics taking into account the number of non-zero components of the input matrix. For example, if the matrix is truely sparse (say fewer than half of all components are non-zero), it turned out that it is more efficient to convert the polynomials into lists, operate on the lists and then reconvert these lists into the internal representation via polynomials.

In the dense case, this conversion process is avoided, since direct work with polynomials is more efficient in this case.

Dom::Matrix vs. Dom::SparseMatrix

Tests indicate that nearly all matrix operations are faster and more efficient when using `Dom::SparseMatrix` instead of `Dom::Matrix` if the coefficient domain is `Dom::ExpressionField()` or any other MuPAD domain with a system representation ("facade domain"). Remarkably, this is also the case for dense matrices indicating that the overhead of converting data to polynomials is rather small. In the following, we consider two examples: First, we compute the square of a matrix. Second, linear equations with a random sparse matrix structure are solved numerically. In all examples we used a Windows-PC with a 950 MHz Pentium CPU and 256 MB RAM.

Computing squares of matrices

For the values $i = 1, 2, \ldots, 10$ we generate random $(10 \cdot i) \times (10 \cdot i)$ matrices with components in the coefficient domain `Dom::ExpressionField()`. All non-zero components are random elements taken from the set $\{1, 2, \ldots, 10\}$ and placed in the matrix according to a uniform random distribution. In the sparse case, for each i we consider matrices with at most $10 \cdot i^2$ non-zero components (i.e., only 10% of the matrix is filled). First, we generate these matrices with `Dom::SparseMatrix()` and then convert them to `Dom::Matrix()`. Note, that we can use the short forms `sparsematrix` and `matrix` to generate matrices over the coefficient domain `Dom::ExpressionField()`. Then we measure the time used to compute the square of each of the matrices.

MuPAD

```
>> f:= random(1..10):
>> for i in [5, 10, 20, 40] do
>>   A1:= sparsematrix::random(10*i, 10*i, 10*i^2, f):
>>   A2:= matrix(A1):
>>   print(dimension = [10*i, 10*i], time(A1^2)*msec, time(A2^2)*msec):
>> end_for:
```

Output

```
                  dimension = [50, 50], 10 msec, 842 msec

                 dimension = [100, 100], 60 msec, 6740 msec

                dimension = [200, 200], 511 msec, 54248 msec

               dimension = [400, 400], 4707 msec, 437058 msec
```

These timings indicate that the squarings performed with `Dom::SparseMatrix` are significantly faster than those performed with `Dom::Matrix`. Hence, in the sparse situation the new domain works fine.

Now we have a look at the dense case. We fill up the matrices with random numbers:

MuPAD

```
>> for i in [5, 10, 20] do
>>   A1:= sparsematrix(10*i, 10*i, f):
>>   A2:= matrix(A1):
>>   print(dimension = [10*i, 10*i], time(A1^2)*msec, time(A2^2)*msec):
>> end_for:
```

Output

```
                  dimension = [50, 50], 831 msec, 1012 msec

                 dimension = [100, 100], 6669 msec, 8302 msec

                dimension = [200, 200], 52786 msec, 70441 msec
```

The squarings performed with `Dom::SparseMatrix` are still a bit faster than those with `Dom::Matrix`. Thus, even in the dense case, when computing over the coefficient domain `Dom::ExpressionField()`, the domain for sparse matrices does not show any significant overhead when compared with MuPAD's traditional matrix domain.

The obvious question is: why not replace `Dom::Matrix` by `Dom::SparseMatrix`?

The problem is, that we cannot expect this positive runtine behaviour over *all possible* coefficient domains in MuPAD. In general, the new domain will be faster than `Dom::Matrix` whenever the underlying coeffcient domain is a facade for a domain of the MuPAD kernel (such as `Dom::ExpressionField()` or `Dom::Float`). Over some of the library domains such as `Dom::DistributedPolynomial`, the domain for sparse matrices turns out to be somewhat less effcient than `Dom::Matrix` when the matrices are dense:

MuPAD

```
>> R:=  Dom::DistributedPolynomial([x], Dom::Integer):
>> A1:= Dom::SparseMatrix(R)(30, 30, R::random):
>> A2:= Dom::Matrix(R)(A1):
>> time(A1^2)*msec, time(A2^2)*msec;
```

Output

```
                              25557 msec, 17505 msec
```

In this (dense) situation, the traditional `Dom::Matrix(R)` clearly is the better choice.

Of course, in a sparse situation, the sparse multiplication algorithm makes `Dom::SparseMatrix` the more appropriate data structure.

In the special case of the library domain `R = Dom::IntegerMod(p)`, `Dom::SparseMatrix(R)` converts the components of the input matrix to polynomials that use modular arithmetic of the MuPAD kernel. In the end, the results are reconverted to elements of the library domain `Dom::IntegerMod(p)`. Thanks to this trick, `Dom::SparseMatrix` is much faster than `Dom::Matrix` even in the dense case, because `Dom::Matrix(R)` uses the slower arithmetic of the library domain `Dom::IntegerMod(p)`:

MuPAD

```
>> R:= Dom::IntegerMod(263):
>> A1:= Dom::SparseMatrix(R)(50, 50, R::random):
>> A2:= Dom::Matrix(R)(A1):
```

```
>> time(A1^2)*msec, time(A2^2)*msec
```

Output

```
                                 972 msec, 5267 msec
```

Computing solutions of sparse systems of linear equations

In practical applications it is often neccessary to compute numerical solutions to large systems of linear equations, whose coefficient matrices have a special banded structure. In the following we consider the runtime behaviour for the computation of numerical solutions to a sequence of banded linear systems.

Again, we define matrices with `Dom::SparseMatrix` and convert them to `Dom::Matrix`. The matrices considered are $(10 \cdot i) \times (10 \cdot i)$ matrices, with main diagonal entries 2 and -1 above and below the main diagonal. The right-hand side of the system of linear equations is always given by the column vector with components $1, 2, \ldots, 10 \cdot i$.

For illustration, in the case of (6×6) matrices, we consider the system

$$\begin{pmatrix} 2 & -1 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & -1 & 2 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{pmatrix}.$$

The dimension of the considered matrix together with the time (in milliseconds) needed to compute a solution of the linear system is stored within the lists `TimesA1`, `TimesA2` for matrices created with `Dom::SparseMatrix`, and in the lists `TimesB1`, `TimesB2` for matrices defined with `Dom::Matrix`, respectively.

We interpret the pairs

```
(  number of rows respectively columns,
   time in milliseconds                 )
```

as points, connect them by a line and then plot the polygon to visualize the runtime behaviour.

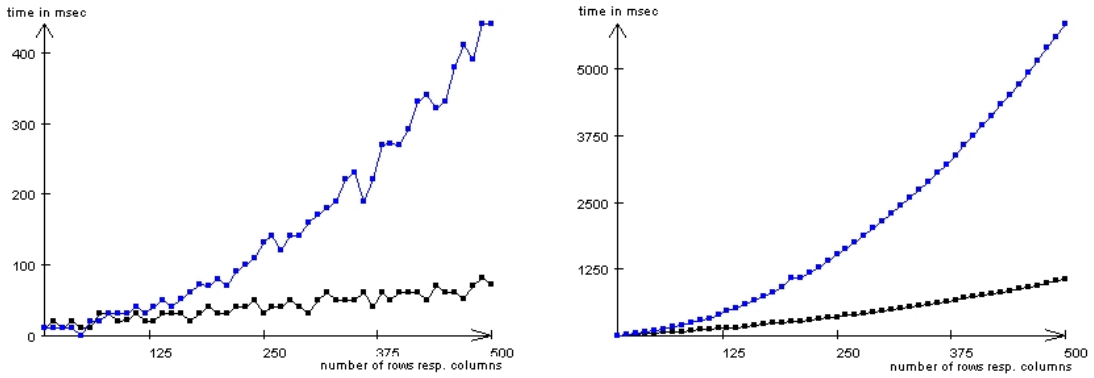
We use the function `numeric::matlinsolve` to compute the solutions. First, the option `HardwareFloats` is specified, such that MuPAD sends the task to the external hardware floating point tool Scilab. Scilab computes the result and returns it to MuPAD. Then, using the option `SoftwareFloats`, the solution of the linear system is computed by MuPAD's `numeric` library using MuPAD's software floats.

Hence, the first case allows us to check if `Dom::SparseMatrix` really serves as an efficient container for the communication between MuPAD and Scilab. The latter case tests the efficiency of `Dom::SparseMatrix` for computations within MuPAD itself.

The complete code is listed on the next page:

MuPAD

```
>> TimesA1:= [0 $ 50]:
>> TimesA2:= [0 $ 50]:
>> TimesB1:= [0 $ 50]:
>> TimesB2:= [0 $ 50]:
>> for i from 1 to 50 do
>>   A:= Dom::SparseMatrix()(10*i, 10*i, [-2, 3, -2], Banded):
>>   v:= Dom::SparseMatrix()([j $ j=1..10*i]):
>>   TimesA1[i]:= plot::Point(10*i, time(numeric::matlinsolve(A,v,HardwareFloats)));
>>   TimesA2[i]:= plot::Point(10*i, time(numeric::matlinsolve(A,v,SoftwareFloats)));
>>   B:= Dom::Matrix()(10*i, 10*i, [-2, 3, -2], Banded):
>>   v:= Dom::Matrix()([j $ j=1..10*i]):
>>   TimesB1[i]:= plot::Point(10*i, time(numeric::matlinsolve(B,v,HardwareFloats)));
>>   TimesB2[i]:= plot::Point(10*i, time(numeric::matlinsolve(B,v,SoftwareFloats)));
>> end_for:
>> p1:= plot::Pointlist(op(TimesA1), DrawMode = Connected, Color = RGB::Black):
>> p2:= plot::Pointlist(op(TimesB1), DrawMode = Connected, Color = RGB::Grey):
>> p3:= plot::Pointlist(op(TimesA2), DrawMode = Connected, Color = RGB::Black):
>> p4:= plot::Pointlist(op(TimesB2), DrawMode = Connected, Color = RGB::Grey):
>> plot(p1, p2);
>> plot(p3, p4);
```



In the left plot, the blue graph (the upper one) corresponds to `Dom::Matrix`, whereas the black graph corresponds to `Dom::SparseMatrix` (the graph below). Indeed, the new `Dom::SparseMatrix` serves as an efficient container for the communication between MuPAD and Scilab. In contrast to the blue graph, which grows quadratically, the black graph suggests linear growth. The deviation of the data from a smooth curve is explained by the granularity of the time measurement (the smallest time difference that can be measured is 10 milliseconds).

The right plot demonstrates the runtime behaviour, when computing the same solutions by the MuPAD library function `numeric::matlinsolve` without any help from Scilab. As one might expect, the time used to perform the same computations as before (using Scilab) is significantly higher now. Again, the blue graph (the upper one) represents the computation with `Dom::Matrix`, the black graph (the one below) corresponds to `Dom::SparseMatrix`. The domain `Dom::SparseMatrix` exhibits an almost linear growth whereas the runtime behaviour with `Dom::Matrix` is clearly nonlinear.

The maximal time needed for a computation with `Dom::Matrix` is approximately 5000 milliseconds. The same computation performed by Scilab needs only approximately 400 milliseconds. Note, that with Scilab the timings in the plot above only refer to the time MuPAD needs for type checking and conversion of the matrix components to floating point numbers. The Scilab time is not included.

Storage and Memory Considerations

We mentioned, that a matrix of type `Dom::SparseMatrix` is stored by a list of polynomials. The internal representation of matrices of the traditional matrix type `Dom::Matrix` is based on two dimensional arrays. Hence, one might suggest, that in the case of large and dense matrices, the definition of a matrix over `Dom::Matrix` does not allocate as much memory as the same matrix defined over `Dom::SparseMatrix`. Indeed, this can be the case, since arrays are a simpler data structure than lists of polynomials. We will consider a few examples to illustrate the memory allocation behaviour.

In each of the computations stated below we started a completely new MuPAD session (this is necessary if we want observe the memory allocation behaviour in an appropriate way).

In the first example we generate two dense matrices over the coeffcient domain `Dom::ExpressionField` – once with `Dom::Matrix` and once with `Dom::SparseMatrix`.

MuPAD

```
>> M:= matrix:
>> bytes();
>> A:= M(500, 500, (i,j) -> (i+j)^10):
>> bytes();
```

Output

```
                          1218600, 1356288, 2147483647

                         29343436, 30585952, 2147483647
```

The numbers of our interest are 1356288 and 30585952. MuPAD has to load the domain `Dom::Matrix` first before we can perform any computations. The above lines tell us, that approximately 1.4 MB memory are used for this. Hence, we have to subtract 1.4 MB from the second number, which is approximately 30.6 MB. Consequently, MuPAD uses about 29.2 MB memory to store the above matrix A. We consider the same matrix defined with `Dom::SparseMatrix`:

MuPAD

```
>> M:= sparsematrix:
>> bytes();
>> A:= M(500, 500, (i,j) -> (i+j)^10):
>> bytes();
```

Output

```
                          1374072, 1487360, 2147483647

                         29713280, 30962784, 2147483647
```

Approximately 1.5 MB are spent for loading `Dom::SparseMatrix`. Thus, we have to subtract this number from the above 31 MB, which yields about 29.5 MB used to store if we generate the matrix with `Dom::SparseMatrix`. The deviation between the two values is rather small, which shows that it is not a problem to define large and dense matrices with `Dom::SparseMatrix` rather than with `Dom::Matrix`. This behaviour does not depend on the choice of the coefficient domain `Dom::ExpressionField`. Even over some more complicated coeffcient domains, such as the library `Dom::DistributedPolynomial([x], Dom::Integer)`, `Dom::SparseMatrix` does not allocate much more memory than `Dom::Matrix`. For the storage of the following 500×500 matrix MuPAD uses approximately 53 MB, if we define the matrix via `Dom::Matrix`.

MuPAD

```
>> R:= Dom::DistributedPolynomial([x], Dom::Integer):
>> M:= Dom::Matrix(R):
>> bytes();
>> A:= M(500, 500, (i,j) -> R(i*x^j + j*x^i)):
>> bytes();
```

Output

```
                          1390124, 1536512, 2147483647

                         66417408, 67712096, 2147483647
```

Definition of the same dense matrix with `Dom::SparseMatrix` needs about 51 MB, which is even less than the memory allocated for the matrix defined via `Dom::Matrix`:

MuPAD

```
>> R:= Dom::DistributedPolynomial([x], Dom::Integer):
>> M:= Dom::SparseMatrix(R):
>> bytes();
>> A:= M(500, 500, (i,j) -> R(i*x^j + j*x^i)):
>> bytes();
```

Output

```
                          1575116, 1716736, 2147483647

                         67682300, 67973632, 2147483647
```

Consequently, in the dense case both alternatives are a good choice (viewed from the standpoint of memory allocation). In the sparse case, `Dom::SparseMatrix` is definitely the better choice:

MuPAD

```
>> M:= matrix:
>> bytes();
>> A:= M(1000, 1000, [-1,2,-1], Banded):
>> bytes();
```

Output

```
                          1218548, 1356288, 2147483647

                          5222028, 9452320, 2147483647
```

As the example demonstrates, `Dom::Matrix` uses about 8.1 MB free memory to store the 400×400 banded matrix. In contrast to this, `Dom::SparseMatrix` only uses approximatetly 0.12 MB:

MuPAD

```
>> M:= sparsematrix:
>> bytes();
>> A:= M(1000, 1000, [-1,2,-1], Banded):
>> bytes();
```

Output

```
1374072, 1487360, 2147483647

1492160, 1602048, 2147483647
```

We did not measure the time used to generate any of the above matrices, but e.g. in the case of the upper banded matrices, `Dom::Matrix` does not only allocate much more memory than `Dom::SparseMatrix`, the time to generate the matrices is even much shorter if one uses `Dom::SparseMatrix`.

Conclusions

In short, all library functions, which could be used for matrices defined over `Dom::Matrix` in previous versions of MuPAD can also be applied to matrices defined over `Dom::SparseMatrix`.

The domain works well together with the numerical package Scilab. Whenever one has to deal with large MuPAD matrices over the coefficient domains `Dom::ExpressionField`, `Dom::IntegerMod`, `Dom::Rational`, `Dom::Real`, `Dom::Complex` or `Dom::Float`, the use of `Dom::SparseMatrix` is a good choice even if the considered matrices are rather dense.

Nevertheless, we saw a few examples where the traditional matrix domain `Dom::Matrix` is more efficient than `Dom::SparseMatrix`. These examples demonstrate, that, at this point, it is not a good idea to remove the traditional matrix domain and replace it by `Dom::SparseMatrix`.

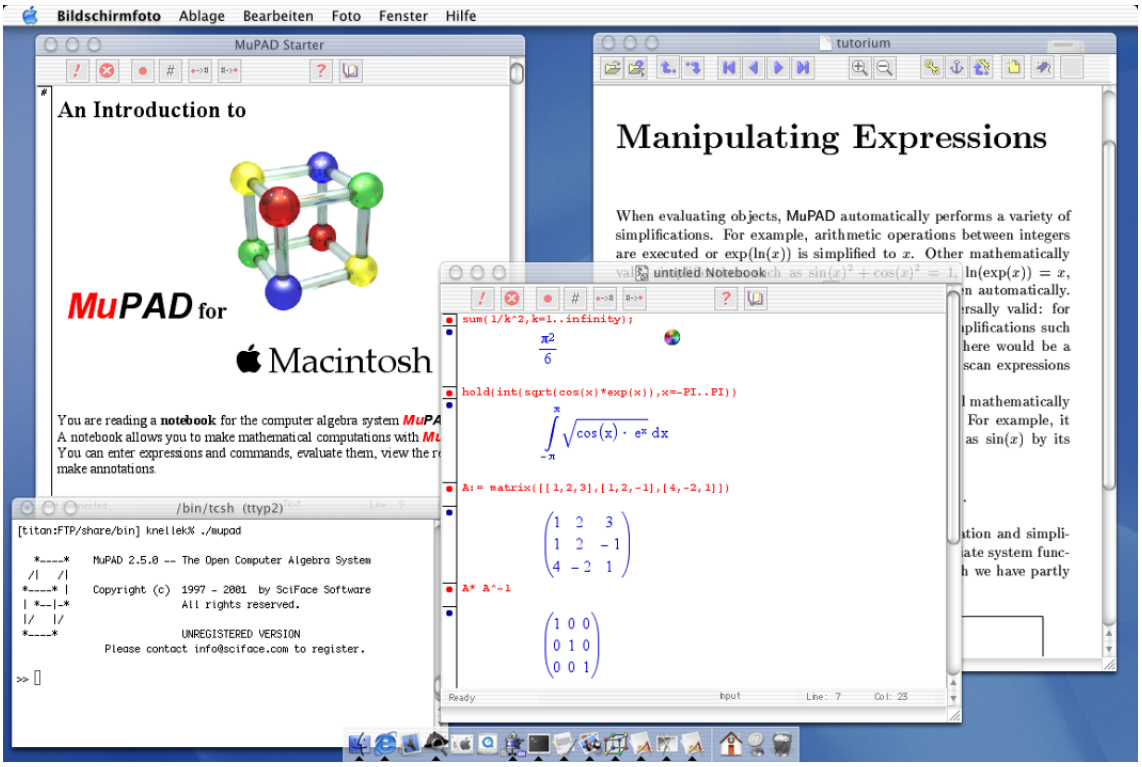
One of the perspectives for the future development is to do some more fine tuning of the algorithms provided by `Dom::SparseMatrix`, until they work nearly as effcient as the algorithms provided by `Dom::Matrix` over any possible coeffcient domain in MuPAD.

For more information about Scilab integrated with MuPAD 2.5 refer to the article *Numerics with Hardware Floats in MuPAD 2.5* by Walter Oevel on page 63 of this volume.

MuPAD Pro 2.5 for Apple Macintosh

Peter Horn and Martin Knelleken
SciFace Software, mailto:knellek@sciface.com
University of Paderborn, mailto:hornp@mupad.de

After two years of waiting, Apple Macintosh users can look forward to a new version of MuPAD for their favorite computer – MacMuPAD 2.5.



Introduction

The last official release of Mac*MuPAD* was 1.4.2. This is indeed pretty outdated. In the meantime both, *MuPAD* and the MacOS, did greatly evolve. *MuPAD* 2.0 and 2.5 were introduced, and Apple released the Unix-based MacOS X. Both offer a lot of new features and technologies.

This forced us to rewrite Mac*MuPAD* from scratch. The result is a completely new program which matches the look and feel and takes advantage of modern features of the most recent MacOS versions. It comes as a *Carbon* application so that it will run natively under MacOS X as well as under the *Classic* MacOS 8/9.

New features

The main goal for the new Mac*MuPAD* was bringing the functionality close to that of *MuPAD* Pro for Windows preserving the Macintosh look and feel.

Mac*MuPAD* 1.4.2 already had some features like a graphical debugger, a source code editor, the support of dynamic modules and a so called *Session* document which allows to edit and recalculate *MuPAD* commands subsequently. Of course, you will find all these things in Mac*MuPAD* 2.5. Additionally, several new help- and powerful features are introduced:

Notebook concept Former sessions are now called *Notebooks* and combine *text*, *computations* and *inline graphics*. One can open more then one notebook at a time. You can choose to connect each notebook to its own *MuPAD* kernel so that the calculations are completely independent. On the other hand, it is possible to connect more than one notebook to a kernel, so that different computations can be made in the same environment.

Text formatting features The new notebook allows you to change lots of text attributes like font, size, face and color for each single character. Furthermore you can set the alignment for lines or paragraphs to right or center.

Inline Graphics Graphics can be placed in the notebook. This is true for external pictures, logos and so on, and for graphics produced by *MuPAD* itself. Embedded *MuPAD* graphics can later be edited in a separate window.

Typesetting Output The *MuPAD* outputs are now typesetted. That is, you get the results of your computations in the form that you would expect from a book. This makes it much easier to read nested or compound expressions. Additionally, it allows you to generate formula graphics very quickly without using the fabulous but tricky TEX.

Command palette and Extras Menu Mac*MuPAD* offers a toolbar for fast access to some commonly used *MuPAD* commands, e.g., `expand`, `factor` and `approximate`. Besides, there is an *Extras* menu with a collection of *MuPAD* commands sorted by topic submenus. This menu can easily be modified at runtime.

New Help tool The viewer for the *MuPAD* manual is highly improved, too. It allows the creation of bookmarks, user-defined hyperlinks and notes on each page.

Import of *MuPAD* Pro Notebooks There are lots of useful notebooks available on the web which are created by *MuPAD* Pro for Windows. Mac*MuPAD* 2.5 can import these notebooks. Even though for technical reasons not all details can be interpreted, the main structure is preserved.

Moreover, the automatic command completion of *MuPAD* is supported by the frontend. This function tries to complete an incomplete input phrase to a valid *MuPAD* command. If there are several matches, you can make your choice in a pop-up menu.

Of course there are lots of other improvements that make working with Mac*MuPAD* an enjoyable experience.

Where it is?

Mac*MuPAD* 2.5 is in a final test phase and will be released soon. If you have specific questions or you are interested in becoming a betatester you should directly contact the author `mailto:knellek@mupad.de`.

Beside of this main product there is a text based terminal version for MacOS X (strictly speaking: for Darwin) available. It does not have a graphical user interface but has got some advantages in automation and scripting when using it in the UNIX enviroment offered by MacOS X.

With the release of Mac*MuPAD* 2.5 there is a modern, beautyfull version of *MuPAD* available on the Macintosh again. It will offer most major features of the Windows-version making Mac-users first class citizens of the *MuPAD* world.

Programming in MuPAD: option escape

Stefan Wehmeier
MuPAD Research Group, University of Paderborn, mailto:stefanw@mupad.de

We present some programming tricks for procedure-valued procedures, using `option escape` *in MuPAD procedures.*

What is it?

MuPAD is one of the few languages that handle procedures as data objects: procedures may be assigned to identifiers and local variables, passed as argument, or returned as result. On the other hand, *MuPAD* (in version from 2.0 upwards) follows the rules of lexical scoping: local variables of a procedure `f` may be used inside a procedure `g` that has been *defined* inside `f`, no matter when `g` is executed. In most cases, `g` will be called by `f`, or by a procedure called by `f` (through several stages, maybe). However, it is possible to call `g` even after the execution of `f` has been finished: in this case, the local variables of `f` have already been removed from memory: in spite of the scoping rules, `g` must not use them. To avoid this, *MuPAD* has an exception to this exception to the scoping rules:

> If a procedure has `option escape`, its local variables may be used by inner procedures at any time, even if the execution of the outer procedure has already been finished. Inner procedures created during different calls of the outer procedure use separate sets of values for the local variables.

Counting operations

Now, how to use this feature? A classical example known from other languages are static variables. *MuPAD* does not have them, but they are easily emulated. We start with a classical example: procedures that count how often they are called.

MuPAD

```
>> outer:=
>> proc()
>>   option escape;
>>   local counter;
>> begin
>>   counter:= 0;
>>
>>   // inner procedure
>>   proc()
>>   begin
>>   counter:= counter + 1;
```

```
>>   print("This procedure has been called for the ".
>>         output::ordinal(counter)." time")
>>   end_proc
>>
>> end_proc;
>>
>> inner1:= outer();
>> inner2:= outer()
```

The procedure `outer` returns a procedure and terminates. Nevertheless, its local variable `counter` may still be used, both for read and write access. The state of `outer` has been stored in a *procedure environment*; that state has not been destroyed when `outer` was left since it is still in use (as `op(inner1, 12)`, the enclosing procedure environment of `inner1`). You should not try to access the enclosing procedure environment via `op` — the numbering of operands of procedures may change in future versions. Each call to a procedure creates its own procedure environment: hence `inner2` points to another instance of `counter`, such that calls to `inner1` and `inner2` are counted separately:

MuPAD

```
>> inner1(), inner1(), inner2()
```

Output

```
              "This procedure has been called for the 1st time"

              "This procedure has been called for the 2nd time"

              "This procedure has been called for the 1st time"
```

It is clear that we could also return the current counter value instead of printing it. In fact, this is just what most methods from `combinat::generator` do: they return procedures that iterate over some collection of objects at a rate of one per call, by storing their current position in a static variable.

In the same way, you can easily translate `C` programs involving static variables:

```
int func(int argument1, int argument2)
{
     int a;           /* auto   variable */
     static int b;    /* static variable */


     /* body */
     ....
}
```

becomes

MuPAD

```
>> func:=
>> proc()
>>   option escape;
>>   local b: DOM_INT;
```

```
>> begin
>>   proc(argument1: DOM_INT, argument2: DOM_INT): DOM_INT
>>     local a: DOM_INT;
>>   begin
>>     /* body */
>>     ....
>>   end_proc
>> end_proc()
```

where we immediately called the outer procedure instead of assigning it to an identifier, since we won't use it twice.

Sharing data

Now what if we want several procedures to manipulate the same set of data? Writing a domain might be the first idea, but we can also make our outer procedure return a sequence or list of several inner procedures. For example, we might want to read out the static variables of a procedure, without using a crude hack accessing its 12th operand. As an example, we write a procedure `count` that takes a procedure `f` as argument and returns an equivalent procedure that counts how often it is called. No messages are printed, but another procedure is returned that reads off the counter, and a third one that resets the counter.

MuPAD

```
>> count :=
>> proc(f)
>>   option escape;
>>   local counter;
>> begin
>>   counter:= 0;
>>
>>   [
>>    // inner procedure equivalent to f
>>    proc()
>>    begin
>>      counter:= counter + 1;
>>      f(args())
>>    end_proc,
>>    // procedure to read off counter
>>    () -> counter,
>>    // procedure to reset counter
>>    () -> (counter:= 0)
>>   ]
>> end_proc
```

For example, with

MuPAD

```
>> fib:= n -> (if n=1 or n=2 then 1 else fib(n-1)+fib(n-2) end);
>> [fib, readcounter, resetcounter]:= count(fib)
```

we are able to find out that computing the seventh Fibonacci number in the naive way takes 25 calls:

MuPAD
```
>> fib(7)
```
Output
```
                                    13
```

MuPAD
```
>> readcounter()
```
Output
```
                                    25
```

We reset our counter:

MuPAD
```
>> resetcounter()
```
Output
```
                                    0
```

Now the counter is zero agian, such that we could start the next experiment:

MuPAD
```
>> readcounter()
```
Output
```
                                    0
```

Tracing a function

Instead of increasing a counter, a function could also be made to output a message. This is what `prog::trace` does. Instead of discussing the rather complicated source code of `prog::trace`, we give a simplified version here. It does not modify its input (like `prog::trace`), but returns a modified version the user has to assign himself.

MuPAD
```
>> trace:=
>> proc(f: DOM_PROC)
>>   option escape;
>>   local namef;
>> begin
```

```
>>   if op(f, 6) = NIL then
>>      namef:= "Unnamed procedure"
>>   else
>>      namef:= expr2text(op(f, 6));
>>   end_if;
>>
>>   // inner procedure
>>   proc()
>>     local result;
>>   begin
>>     print(namef." called with args ".expr2text(args()));
>>     result:= f(args());
>>     print("Leaving ".namef." with result ".expr2text(result));
>>     result
>>   end_proc
>>
>> end_proc
```

One could imagine a procedure that does not output messages, but writes its name and arguments into a list `callStack` and removes them after the call is finished. The *MuPAD* library has an undocumented function `prog::remember` that uses this technique to detect when a function calls itself recursively with identical arguments.

Modifying function environments

Up to here, we have only manipulated procedures. If you apply `count` to a function environment, you will lose all of its slots.

Therefore, `count` and other procedure-valued procedures should be applied to the first operand of a function environment, and the result should be substituted into the function environment without changing the slots. We could present modified versions of our examples, but there is a generic solution that fits into our subject.

We write a function `modifyUtilityFunction`. It expects one argument, a procedure `utility` that maps procedures to procedures, and returns a procedure that maps procedures to procedures and function environments to function environments.

MuPAD

```
>> modifyUtilityFunction:=
>> proc(utility)
>>   option escape;
>>   begin
>>
>>   // the modified utility function:
>>   proc(f)
>>   begin
>>     if type(f) = DOM_PROC then
>>     // by supposition, utility can handle this case
>>        utility(f)
>>     elif type(f) = DOM_FUNC_ENV then
>>        if type(op(f, 1)) <> DOM_PROC then
>>          error("Cannot handle this kind of function environment")
```

```
>>        end_if;
>>        subsop(f, 1 = utility(op(f, 1)))
>>     else
>>        return("Argument must be a procedure or function environment")
>>     end_if
>>   end_proc
>> end_proc
```

Now, by entering

```
trace:= modifyUtilityFunction(trace)
```

or

```
count:= modifyUtilityFunction(count),
```

we can obtain improved versions of our functions above.

We have chosen to raise an error if the input is a function environment the first operand of which is a kernel function (`DOM_EXEC`). It is a bit risky to replace a kernel function by a procedure; and some utility functions will try to access the operands of their input directly (as our `trace` does with the name of the procedure). In some cases (as with our example `count`), it should work though.

Some caveats

In order to make the examples more readable, we left out some necessary code to handle some minor problems.

- Often a modified version of a procedure consists of some initial code, a call to the original version, and some exit code. It is clear that the exit code cannot be executed if an error happened before. Using `traperror` may help here; after the exit code has cleaned up everything, the error can be raised again using `lasterror`.
- If a procedure is replaced by a modified version, other procedures interacting with it might get confused. In particular, it may be safer to substitute the name (sixth operand) by the name of the original procedure.

Further ideas

Procedures that manipulate other procedures are a major feature of *MuPAD*. We could only give few examples. A utility function could return an overloadable version of a given procedure, or a procedure that implements functional composition (as `_fconcat` does). To end with a slightly advanced example, imagine a procedure `SimulateOneStepOnATuringMachine` that stores the state of a given simulated Turing machine in a static variable, and that may be called over and over in order to “debug” a Turing machine.

We hope we have encouraged you to implement functions of these kind in *MuPAD*.

Kalender

Refer to the web site `www-sop.inria.fr/galaad/conf.html` *to read further announcements of conferences related to symbolic computing.*

MNU-Tagung Hessen – Darmstadt, Deutschland, August 26.,2002
Home page: `http://www.physik.uni-marburg.de/didact/mnu/mnuhess.htm`

MNU-Tagung Berlin – Humboldt-Universität, Berlin, Deutschland, September 5.-6., 2002
Home page: `http://www.inf.fu-berlin.de/mnu-berlin`

DMV Jahrestagung 2002 – Halle, Martin-Luther-Universität Halle-Wittenberg , September 15.-21., 2002
Home page: `http://w3.mathematik.uni-halle.de/dmv2002`

Studentenkonferenz Mathematik 2002 – Halle (Saale), Melanchthonianum, September 17., 2002
Home page: `http://w3.mathematik.uni-halle.de/dmv2002/studkonferenz/`

MNU-Tagung Westfalen – Universität Dortmund, Deutschland, September 24., 2002
Home page: `http://www.mnu-westfalen.de`

MNU-Tagung Niedersachsen – Lutherschule, Hannover, Deutschland, September 25., 2002
Home page: `http://www.gbg-seelze.de/MNU-Nds`

MNU-Tagung Nordrhein – Physikalisches Institut der Universität Köln, Deutschland, Oktober 10., 2002
Home page: `http://www.mnu-nordrhein.de`

Kolloquium über Kombinatorik – Universität Magdeburg, November 15. -16., 2002
Home page: `http://www.math.uni-magdeburg.de/kolkom`

MNU-Tagung Bremen – Bremerhaven, Deutschland, November 18.-19., 2002
Home page: `http://didaktik.physik.uni-bremen.de/mnu`

ISAAC 2002 – The 13th Annual International Symposium on Algorithms and Computation Vancouver, Canada, Vancouver, Canada, November 20. -23., 2002
Home page: `http://www.gwdg.de/~cais/konfhinw/node13.html`

CASK 2003 – Computeralgebra-Symposium Konstanz, Konstanz, March 13. -14., 2003
Home page: `http://www.gwdg.de/~cais/konfhinw/node14.html`

Bildungsmesse 2003 – Nürnberg, Deutschland, März 31.-April 4., 2003
Home page: `http://www.didacta.de`

94. MNU-Kongress 2003 – Frankfurt am Main, Deutschland, April 14.-17., 2003
Home page: `http://www.mnu.de`

MEGA 2003 – Universität Kaiserslautern, Juni 10.-14., 2003
Home page: `http://www.mathematik.uni-kl.de/~mega2003`

ISSAC 2003 – Drexel University Philadelphia, Pennsylvania, USA, August 3.-6., 2003
Home page: `http://knave3.mcs.drexel.edu/~issac2003/index.html`

Surfin' the Web

Und noch einige interessante, nützliche und kuriose Plätze im „global village“.

Mathematik / Forschung

`http://www.math-atlas.org`
Diese Web-Site bietet einen knappen Überblick über alle existierenden Teilgebiete der Mathematik, geordnet nach der *Mathematical Subject Classification*.

`http://mathworld.wolfram.com`
“Eric Weisstein's World of Mathematics” liefert interessantes Informationsmaterial rund um die Mathematik.

`http://www.utm.edu/research/primes`
The prime pages - eine Sammlung von Informationen zu Primzahlen.

`http://www.w3.org/Math/XSL`
“Putting mathematics on the Web with MathML” zeigt wie mit welchen Web-Browsern und mit welchen Plugins MathML-Formeln im Web dargestellt werden können. Es lohnt sich, hier ab und zu mal wieder vorbei zu schauen, da das Thema brandaktuell ist und die meisten Werkzeuge und Dokumente in diesem Bereich noch in der Entwicklung sind.

`http://www.medien-bildung.net`
Die Service Seiten vom Projektträger “Neue Medien in der Bildung + Fachinformation” informieren über aktuelle Projekte und Veranstaltungen und halten Links zu anderen Informationsdiensten bereit.

`http://www-nbp.upb.de`
Web-Site des Projektes *Neue Bahntechnik Paderborn*.

Schule

`http://www.bildungslinks.de/index1.htm`
Sie benötigen Informationen zum Thema Bildung in Deutschland und Europa? Diese Web-Site bietet eine große Sammlung von Verweisen auf Bildungsserver, Kataloge, Datenbanken und mehr.

`http://www.nw.schule.de`
Der “ODS-Server Nordrhein-Westfalen” (Offenes Deutsches Schulnetz) informiert zum Thema Schule und Bildung in Nordrhein Westfalen und darüber hinaus. Sensible Naturen sollten vor dem Öffnen dieses Links allerdings den Lautsprecher ihres PC abschalten oder zumindest leise schalten :-).

Weitere Services

`http://portal.acm.org`
Das ACM-Portal: Von hier bekommt man Zugriff auf tausende von Artikeln aus dem Bereich Informatik und verwandten Gebieten.

`http://www.linguatec.net/online`
Der Online Service der linguatec Entwicklung und Services GmbH stellt ein sehr gutes Wörterbuch für Deutsch-Englisch Übersetzungen zur Verfügung.

http://dict.leo.org
Eine andere Alternative stellt das Deutsch-Englisch Wörterbuch LEO dar. Über 2.000.000 Anfragen täglich weisen auf eine zunehmend wachsende Fan-Gemeinde hin.

Software

http://wwwvis.informatik.uni-stuttgart.de/~kraus/LiveGraphics3D
Die "LiveGraphics3D Homepage" von Martin Kraus bietet ein interessantes, interaktives 3D-Viewer Applet zum Download an, sowie viele Grafik-Beispiele.

http://www.meybohm.de/proton.html
Ein praktischer Freeware-Texteditor mit Syntaxfärbung und weiteren nützlichen Funktionalitäten für Windows.

http://www.povray.org
POV-Ray – The Persistence of Vision Raytracer is a high-quality, totally free tool for creating stunning three-dimensional graphics. It is available in official versions for Windows 95/98/NT, DOS, the Macintosh, i86 Linux, SunOS, and Amiga. The source code is available for those wanting to do their own ports.

http://home.germany.net/100-122054/texwin.htm
Textsatz mit LaTex unter Windows: Eine Übersicht zu verschiedenen Editoren und Oberflächen aus dem Bereich der Freeware und Shareware.

In Eigener Sache

http://www.mupad.de/schule+studium
Das deutschsprachige Web-Portal zum Thema MuPAD in Schule und Studium.

http://www.sciface.com/doc
MuPAD 2.5 Documentation Online: The documentation of the MuPAD 2.5 libraries is available in English and German as PDF or HTML files, respectively.

Allerlei

http://www-nbp.upb.de/de/webcam/
WebCam des Projektes *Neue Bahntechnik Paderborn*. Hier kann man zusehen, wie die neue Bahn-Teststrecke hinter dem Wirtschaftsgebäude der Universitt Paderborn im Maßstab 1:2,5 aufgebaut wird.

http://www.hamburg.de/fhh/behoerden/datenschutzbeauftragter
Informationen des Hamburgischen Datenschutzbeauftragten – auch über Hamburg hinaus.

http://www.bod.de/produkte/kalkulator.html
"BOD – Books on Demand" bietet Autoren die Möglichkeit Bücher auch in Kleinserien aufzulegen (mit oder ohne ISBN) und über das Internet und den Handel zu vertreiben. Der "Kalkulator" berechnet dabei Kosten und Verdienstmöglichkeiten.

http://www.physik.tu-muenchen.de/~rwagner/physik/mathewitze.html
Witze von und für und/oder über Mathematiker – oder so.

http://www.zompist.com/numbers.shtml
Kein Witz: Die Zahlwörter von 1 bis 10 in über 4500 Sprachen.

Der DUBBEL auf CD-ROM - topaktuell in der Version 2.0!

W. Beitz, K.-H. Grote (Hrsg.)

DUBBEL interaktiv 2.0

Das elektronische Taschenbuch
für den Maschinenbau

Vollversion für Privatnutzer

Systemanforderung: Windows 95/98/ME 32 MB RAM (64 MB RAM empfohlen); Windows NT 4/2000/XP 64 MB RAM (128 MB RAM empfohlen).

Für alle Betriebssysteme: Pentium I mit 166 MHz (Pentium II mit 400 MHz empfohlen); 20 MB freier Speicher auf der Festplatte. CD-ROM oder DVD-ROM Laufwerk

Version 2.0; 2003. CD-ROM. **€ 99,95; sFr 155,-
ISBN 3-540-14944-9

Die Bestelldaten der Industrielizenzen der Version 2.0!

Industrielizenz - Einzelplatzversion 2.0
**€ 349,-; sFr 527,-
ISBN 3-540-14942-2

Industrielizenz - Mehrplatzversion 2.0
**€ 698,-; sFr 1054,-
ISBN 3-540-14943-0

Wesentlich verbessert und benutzerfreundlicher:

In der Version 2.0 sind die Oberflächenhandhabung und die Eingabedialoge für Variablen wesentlich verbessert worden und zusätzliche Gleichungen - auch in Tabellen - interaktiv.

Insgesamt bietet die Dubbel CD-ROM:

- mehr als 1600 interaktive Gleichungen, deren Rechenergebnisse graphisch unmittelbar und dreidimensional dargestellt werden können
- Interpolationsfunktion in Graphen
- mehr als 5000 Variablenerläuterungen per Mausklick
- hochaufgelöste, ausdruckbare technische Zeichnungen
- Annotationsmöglichkeiten
- Referenz-Hyperlinks
- Tabellenkalkulation

Zusatz-Information; jetzt neu, der Upgrade-Preis:
Sie können bei Einsenden der CD-ROM Dubbel interaktiv 1.0, Privatnutzerlizenz (ISBN 3-540-14779-9) die neue CD-ROM Dubbel interaktiv 2.0, Privatnutzerlizenz (ISBN 3-540-14944-9) zu einem **Upgrade-Preis** von Brutto € 50,- bestellen.
Die Studentenversion (ISBN 3-540-14937-6) ist von dieser Regelung ausgenommen.

Springer · Kundenservice
Haberstr. 7 · 69126 Heidelberg
Tel.: (0 62 21) 345 - 217/-218
Fax: (0 62 21) 345 - 229
e-mail: orders@springer.de

**Die Preise für elektronische Produkte sind unverbindliche Preisempfehlungen inkl. 16% MwSt. in Deutschland. In anderen Ländern zzgl. landesüblicher MwSt.

Preisänderungen und Irrtümer vorbehalten. d&p · 00874/MNT/SP

Springer