April 2008

Computeralgebra Rundbrief

GI_DMV_GAMM

Sonderheft zum Jahr der Mathematik

- Computeralgebra- und Dynamische Geometriesysteme
- Computeralgebra in Forschung und Lehre
- Computeralgebra in der Praxis

Wissenschaftsjahr 2008
Mathematik
Alles, was zählt

www.fachgruppe–computeralgebra.de

Sonderheft

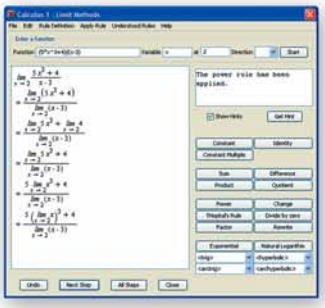
Maple™ 11

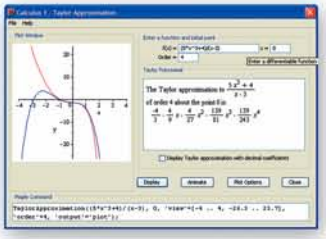
Mathematics • Modeling • Simulation

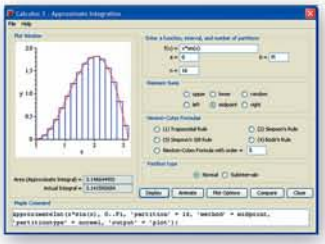
Maple hat die Methoden revolutioniert, mit denen Lehrer und Schüler Mathematik lehren und lernen. Das sogenannte 'Smart Document' Interface bietet mit Point-und-Click eine bisher noch nicht dagewesene Fülle an Möglichkeiten zur Problemlösung und Visualisierung. Diese Vorgehensweise ermöglicht es sowohl Lehrern als auch Schülern, sich auf Konzepte zu konzentrieren statt Befehle und Syntax zu lernen.

Clickable Calculus™

Diese Sammlung interaktiver Applikationen innerhalb von Maple nutzt 'Clickable Calculus' Methoden, um Standardaufgaben aus dem Bereich der Analysis zu lösen:







- Kontext-sensitive Menüs für den sofortigen Zugriff auf Solver und andere Operationen - ohne die lästige Eingabe von Befehlen
- Umfangreiche Auswahl an Paletten zum rein visuellen Editieren mathematischer Ausdrücke
- Komplett interaktive Erstellung von Grafiken und Animationen über die Maus ohne die Notwendigkeit der zusätzlichen Eingabe von Parametern und Attributen
- Drag-und-Drop Operationen für Grafiken, Formeln, Text und mehr
- Interaktive Assistenten, die einfache Mechanismen zur Bearbeitung komplizierter Sachverhalte, wie z.B. das Lösen von Differentialgleichungen und Optimierungsproblemen, zur Verfügung stellen
- Eingebaute Auswahl an interaktiven Tutoren, die visuelle Lernumgebungen zu wichtigen mathematischen Gebieten wie z.B. Analysis, Vektor-Analysis oder Lineare Algebra beinhalten
- Handschrift-Erkennung mathematischer Symbole und einfacher Ausdrücke
- WYSISYG-Features zur Dokumentenverarbeitung die es Ihnen ermöglichen, komplexe mathematische Dokumente schneller und einfacher in ein Textverarbeitungsprogramm oder LaTeX zu übernehmen

Fordern Sie noch heute Ihr kostenloses Exemplar
"Integrating Maple in the Classroom"
an unter www.scientific.de/calculus.html

scientific COMPUTERS Telefon (0241) 40008-0 • www.scientific.de • maple@scientific.de

Computeralgebra–Rundbrief

Sonderheft zum Jahr der Mathematik April 2008

Inhaltsverzeichnis

Impressum

Der Computeralgebra-Rundbrief wird herausgegeben von der Fachgruppe Computeralgebra der GI, DMV und GAMM.
Verantwortlicher Redakteur für dieses Sonderheft: Prof. Dr. Martin Kreuzer (martin.kreuzer@uni-passau.de).

Der Computeralgebra-Rundbrief erscheint halbjährlich, Redaktionsschluss 28.02. und 30.09. ISSN 0933-5994. Mitglieder der Fachgruppe Computeralgebra erhalten je ein Exemplar dieses Rundbriefs im Rahmen ihrer Mitgliedschaft.
Fachgruppe Computeralgebra im Internet: www.fachgruppe-computeralgebra.de.
Die Webseite dieses Sonderhefts: www.fachgruppe-computeralgebra.de/JdM/.

Konferenzankündigungen, Mitteilungen, einzurichtende Links, Manuskripte und Anzeigenwünsche bitte an den verantwortlichen Redakteur Dr. Markus Wessler, (markus.wessler@hm.edu).

Die Geschäftsstellen der drei Trägergesellschaften:

GI (Gesellschaft für
Informatik e. V.)
Wissenschaftszentrum
Ahrstr. 45
53175 Bonn
Telefon 0228-302-145
Telefax 0228-302-167
gs@gi-ev.de
www.gi-ev.de

DMV (Deutsche Mathematiker-
Vereinigung e. V.)
Mohrenstraße 39
10117 Berlin
Telefon 030-20377-306
Telefax 030-20377-307
DMV@wias-berlin.de
www.dmv.mathematik.de

GAMM (Gesellschaft für Angewandte
Mathematik und Mechanik e. V.)
Technische Universität Dresden
Institut für Festkörpermechanik
01062 Dresden
Telefon 0351-463-33448
Telefax 0351-463-37061
GAMM@mailbox.tu-dresden.de
www.gamm-ev.de

Was ist Computeralgebra?

Prof. Dr. Wolfram Koepf
Fachbereich Mathematik
Universität Kassel
Heinrich-Plett-Straße 40
34132 Kassel

Prof. Dr. Elkedagmar Heinrich
Fachbereich Informatik
Hochschule für Technik,
Wirtschaft und Gestaltung Konstanz
78462 Konstanz

koepf@mathematik.uni-kassel.de
heinrich@htwg-konstanz.de

Als Sprecher der Fachgruppe Computeralgebra möchten wir zunächst alle Leserinnnen und Leser dieses Schul-Sonderhefts unseres *Computeralgebra-Rundbriefs* herzlich willkommen heißen! Egal, ob Sie unser Heft als Lehrer oder Schüler, als Mitglied unserer Fachgruppe oder eher durch Zufall in die Hand bekommen haben, lassen Sie sich von den Themen dieses Hefts begeistern, die wir Ihnen hiermit zum Jahr der Mathematik präsentieren möchten!

Bevor wir auf die einzelnen Beiträge dieses Hefts eingehen möchten, wollen wir Ihnen erklären, was Computeralgebra eigentlich ist. Die meisten von Ihnen sind schon mit Computeralgebra in der einen oder anderen Form in Berührung gekommen: Entweder in Form eines der *General Purpose*-Computeralgebrasysteme (CAS) *DERIVE*, *Maple*, *Mathematica* oder *MuPAD* oder aber in Form eines Handheld-Taschenrechners wie dem TI 89/Voyage, dem TI-NSpire oder dem Casio ClassPad.

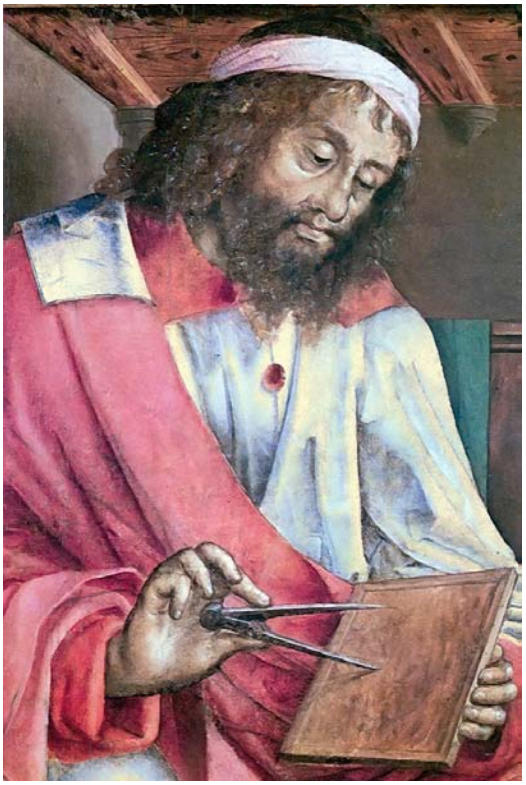
Der Unterschied zu einem „normalen Taschenrechner“ besteht darin, dass diese Systeme und Geräte nicht nur mit Dezimalzahlen (oder ggfs. einfachen Brüchen) arbeiten, sondern mit vielen mathematischen Strukturen rechnen können: Man kann mit beliebig großen ganzen Zahlen, mit ganz-rationalen Funktionen (Polynomen) und auch mit anderen mathematischen Funktionen wie Exponentialfunktionen oder auch trigonometrischen Funktionen symbolisch hantieren. Spezialsysteme arbeiten mit weiteren speziellen mathematischen Strukturen wie Gruppen oder algebraischen Zahlkörpern[1], die im Schulunterricht keine Rolle spielen.

Die größte bisher bekannte Primzahl, die Mersennesche Zahl $2^{32.582.657} - 1$, eine ganze Zahl mit immerhin 9.808.358 Dezimalstellen, s. den Artikel *Primzahltests und Primzahlrekorde* (S. 29), kann ein Computeralgebrasystem genauso mühelos berechnen wie es Funktionsterme differenzieren und integrieren kann. Umformungen von Polynomen und gebrochen-rationalen Funktionen sind ebenso selbstverständlich wie die Vereinfachung trigonometrischer Terme.

Stellen wir uns vor, wir wollen herausfinden, ob die Zahl

$$\frac{302.527.165.409}{1.124.625.805.394}$$

gekürzt werden kann. Dann müssen wir entweder Zähler und Nenner in Faktoren zerlegen — dies ist in unserem Fall relativ „einfach“, aber für große ganze Zahlen im Allgemeinen *sehr langwierig* — oder wir bestimmen den größten gemeinsamen Teiler von Zähler und Nenner.



Euklid von Alexandria, Quelle Wikipedia

Letzteres macht man mit dem *euklidischen Algorithmus*, einer schon vor über 2.000 Jahren bekannten Methode. Dies ist auch für *sehr große ganze Zahlen* gar

[1] oder aber auch mit Molekülstrukturen der Chemie u. ä.

kein Problem und mit einem heutigen Rechner in Sekundenschnelle erledigt. In unserem Fall ergibt sich

$$\frac{302.527.165.409}{1.124.625.805.394} = \frac{2.357}{8.762}.$$

Die meisten Computeralgebrasysteme führen derartige Vereinfachungen völlig ungefragt automatisch durch. Aber sie können ganze Zahlen eben auch in Faktoren zerlegen, falls diese Aufgabe nicht zu viel Zeit in Anspruch nimmt. Oder sie können entscheiden, ob eine Zahl eine Primzahl ist.

Ähnliche Fragen stellen sich beim Rechnen mit Polynomen und gebrochen-rationalen Funktionen. Die binomische Formel

$$x^2 - 1 = (x+1)(x-1)$$

ist in einem CAS natürlich genauso verfügbar wie die (kompliziertere) Vereinfachung

$$\frac{x^{12}-1}{x^8-1} = \frac{x^8+x^4+1}{x^4+1}$$

(Kürzen) oder die Faktorisierung

$$\frac{x^{12}-1}{x^8-1} = \frac{(x^2-x+1)(x^2+x+1)(x^4-x^2+1)}{x^4+1}.$$

Letzteres führt auf eine Zerlegung des Zählers in Faktoren mit ganzzahligen Koeffizienten, welche algorithmisch behandelt und von allen General-Purpose-CAS durchgeführt werden kann.

Wie wurde oben $\frac{x^{12}-1}{x^8-1}$ gekürzt? Wieder gibt es eine Variante des euklidischen Algorithmus, die den größten gemeinsamen Teiler von Zähler- und Nennerpolynom findet. Allerdings entstehen bei derartigen Rechnungen als Zwischenergebnisse häufig sehr große ganze Zahlen als Koeffizienten oder aber immer komplexere rationale Zahlen, obwohl das Endergebnis wieder „relativ einfach“ aussieht. Dies ist ein typisches Phänomen des symbolischen Rechnens.[2] Damit man den größten gemeinsamen Teiler zweier Polynome also überhaupt finden kann, ist es unbedingt erforderlich, prinzipiell mit *beliebig großen ganzen Zahlen* zu rechnen.

Das effiziente Rechnen mit beliebig großen ganzen Zahlen ist also das A und O eines jeden CAS, dies muss sehr schnell gehen. Daher ist es beispielsweise wichtig, sich zu überlegen, wie man schnell multipliziert. Hierüber erfährt man mehr in dem Artikel *Wie schnell kann man multiplizieren?* (S. 35).

Polynomfaktorisierung ist eine der wirklich ganz großen Stärken von CAS. Während das Ausmultiplizieren des Terms

$$(x^2-x+1)(x^2+x+1)(x^4-x^2+1) \mapsto x^8+x^4+1$$

zwar länglich, aber im Prinzip dennoch nicht schwierig ist und von Hand ausgerechnet werden kann, ist die umgekehrte Operation, nämlich die Faktorisierung

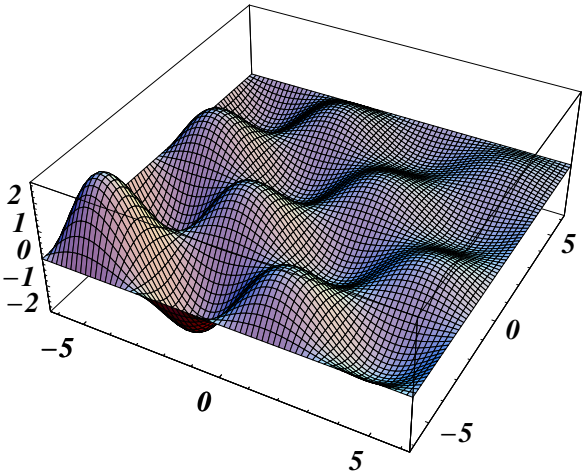
$$x^8+x^4+1 \mapsto (x^2-x+1)(x^2+x+1)(x^4-x^2+1)$$

von Hand praktisch nicht durchführbar. Moderne CAS verwenden hierfür — auch im Fall mehrerer Variablen — sehr effiziente Algorithmen, die sehr schwierige Probleme lösen können.

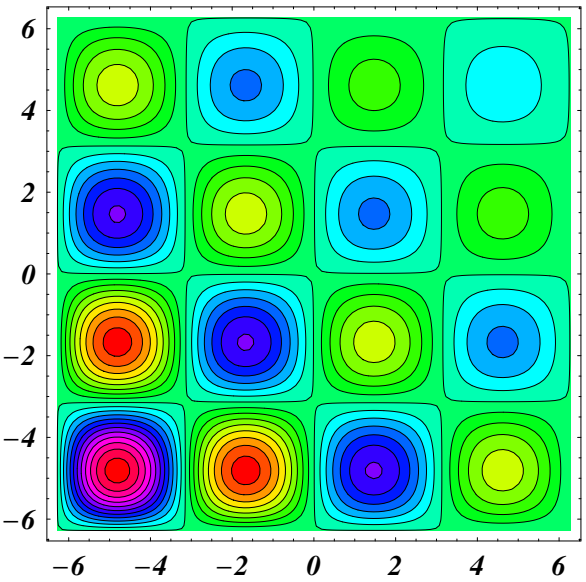
Natürlich können Computeralgebrasysteme nicht nur symbolisch rechnen, obwohl dies ihre besondere Stärke ist. Sie rechnen auch numerisch (mit Dezimalzahlen) wie Taschenrechner und sie können Grafiken erzeugen wie grafische Taschenrechner, in der Regel allerdings in höherer Auflösung. Die nächsten beiden Abbildungen zeigen den Graphen der Funktion

$$f(x,y) = \sin x \sin y \, e^{-\frac{x+y}{10}},$$

einmal in dreidimensionaler Form und zum anderen als Höhenlinien-Graph.



Eine Funktion von zwei Variablen



Dieselbe Funktion als Höhenlinien-Graph

Wie man mit CAS differenzieren kann, wird im Artikel *Warum können CAS differenzieren?* (S. 37) behandelt. Während die Differentiation relativ einfach ist, ist die Integration von Funktionstermen eine weitere

[2] Im wissenschaftlichen Fachjargon wird dieser Effekt *intermediate expression swell* genannt.

große Stärke von CAS. Einiges zu diesem Thema finden Sie im Artikel *Integralrechnung und Computeralgebra* (S. 41). In der Praxis sind CAS in der Lage, dicke Integraltafeln *automatisch* abzuarbeiten.

Methoden der Computeralgebra haben vielfältige Anwendungen im täglichen Leben, derer wir uns häufig gar nicht bewusst sind: *Fehler korrigierende Codes* (S. 32) spielen bei der elektronischen Datenübertragung eine wichtige Rolle. Der Artikel *Die Mathematik der Compact Disk* (S. 53) stellt vor, wie beim Abspielen von CDs trotz vieler Lesefehler dennoch der Hörgenuss sichergestellt wird.

Sichere Kommunikation über unsichere Leitungen (S. 48) ist möglich durch den Einsatz großer ganzer Zahlen (welche Überraschung!) und spezieller kryptographischer Codes wie dem RSA-Verfahren. Solche Verfahren machen Bankgeschäfte im Internet erst sicher. Gelingt es in der Zukunft, einen *Quantencomputer* zu bauen, so wird das RSA-Verfahren unsicher. Dies wird in *Wie rechnen Quanten?* (S. 57) erläutert. Auch in der Logik, der Theorie korrekter Schlussfolgerungen, kann Computeralgebra eingesetzt werden, wie in *Alles logisch, oder was?* (S. 44) an Hand von Logeleien vorgestellt wird.

Anwendungen in der Industrie werden in *Simulation der Erwärmung von Zugbremsscheiben* (S. 67) und in *Algebraisches Erdöl* (S. 70) vorgestellt. Eine weitere Anwendung findet man in *Computeralgebra in der Systembiologie* (S. 62).

In der letzten Zeit wird verstärkt versucht, symbolische Komponenten auch in *dynamischen Geometrieprogrammen* unterzubringen, die ja vielfach auch im Schulunterricht eingesetzt werden. Davon zeugen Programme wie GeoGebra, Cinderella und FeliX, deren Autoren in den Artikeln *GeoGebra: Vom Autodesign zur Computerschriftart* (S. 9), *Cinderella.2 — Geometrie und Physik im Dialog* (S. 12) und *FeliX — Mit Algebra Geometrie machen* (S. 15) ihre Programme vorstellen.

Die Artikel *Enthüllt: Schüler schummelten in Klausuren* (S. 26), *Wellen als Vektoren* (S. 22) und *Enrichment, Computermathematik & Maple* (S. 18) runden zusammen mit einem historischen Beitrag zur Fachgruppe (S. 7) und einer Literaturliste (S. 75) unser Heft ab.

Schließlich möchten wir auf unseren Wettbewerb hinweisen (das Wettbewerbsposter finden Sie auf den nächsten Seite): Schüler, die eine Facharbeit oder Ähnliches zu einem Thema der Computeralgebra erstellt haben, können diese bei uns einreichen und Geld- und Sachpreise gewinnen! Einsendeschluss ist der 31. Oktober 2008, aber wir bitten um Anmeldung zum Wettbewerb bis zum 15. September 2008 unter

WCA@mathematik.uni-kassel.de.

Die Siegerehrung findet Anfang Dezember 2008 statt. Wir bitten um zahlreiche Beteilung und wünschen viel Erfolg!

Auf der Webseite

www.fachgruppe-computeralgebra.de/JdM/

finden Sie dieses Heft auch zum Download. Ferner gibt es zu einigen Artikeln und schließlich auch zum Wettbewerb viele weitere nützliche Informationen.

Wir wünschen Ihnen viel Spaß beim Durchlesen unseres Sonderhefts!

Wolfram Koepf (Kassel)
Elkedagmar Heinrich (Konstanz)

GI (Gesellschaft für Informatik e.V.)
Wissenschaftszentrum
Ahrstraße 45
53175 Bonn
http://www.gi-ev.de

DMV (Deutsche Mathematiker-Vereinigung e.V.)
Mohrenstraße 39
10117 Berlin
http://www.dmv.mathematik.de

Gamm (Gesellschaft für angewandte Mathematik und Mechanik e.V.)
Technische Universität Dresden
Institut für Festkörpermechanik
01062 Dresden
http://www.gamm-ev.de

Schreiben Sie an einer Facharbeit oder Studienarbeit zum Thema Computeralgebra? Haben Sie Lust, sich mit computerorientierter Mathematik zu beschäftigen? Dann nehmen Sie doch teil am

Computeralgebra-Wettbewerb

Anmeldung bis zum 15.9.2008 unter
WCA@mathematik.uni-kassel.de

Einsendeschluss: 31.10.2008

Siegerehrung: Anfang Dezember 2008

Teilnahmeformular, Wettbewerbsbedingungen und Themenvorschläge auf der Webseite

www.fachgruppe-computeralgebra.de/JdM/

1. Preis: 2^{10} €

2. Preis: 2^{9} €

3. Preis: 2^{8} €

4. Preis: 2^{7} €

5. Preis: 2^{6} €

Wissenschaftsjahr 2008
Mathematik
Alles, was zählt

Fachgruppe
Computeralgebra

20 Jahre Fachgruppe Computeralgebra

Prof. Dr. Johannes Grabmeier
Fakultät Betriebswirtschaft und Wirtschaftsinformatik
Hochschule für angewandte Wissenschaften — FH Deggendorf
Edlmairstraße 6 + 8
94469 Deggendorf

johannes.grabmeier@fh-deggendorf.de

Gründung

Am 7. November 1987 wurde in Deutschland die Fachgruppe Computeralgebra gegründet. Es galt der stürmischen Entwicklung dieses Gebietes im Überlappungsbereich von Informatik, Mathematik und Anwendungsgebieten in den Natur-, Ingenieur- und Wirtschaftswissenschaften eine gemeinsame Plattform zu geben. Insofern war es naheliegend, aber keinesfalls selbstverständlich, dass Vertreter der verschiedenen Gesellschaften für Mathematik und Informatik mitwirkten. Es entstand daher eine gemeinsamen Fachgruppe Computeralgebra der DMV (Deutsche Mathematiker-Vereinigung), der GAMM (Gesellschaft für angewandte Mathematik und Mechanik) und der GI (Gesellschaft für Informatik). Diese Struktur hat sich bewährt und ist bis heute Fundament und Ausgangspunkt der Aktivitäten der Fachgruppe Computeralgebra. Nur in dem gemeinsamen Zusammenwirken der Vertreter der vielfältigen Aspekte der Computeralgebra konnte eine Erfolgsgeschichte geschrieben werden, die die Mathematik selbst, den Unterricht und die Anwendungen von Mathematik durch symbolisches Rechnen nachhaltig verändert hat.

Ziele

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

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

Im Report Computeralgebra von 1993 wird das Gebiet weiter präzisiert:

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

Fachgruppenleitungen, Sprecher und Aktivitäten

Fritz Schwarz 1987 – 1990

Zum ersten Sprecher der Fachgruppenleitung wurde Fritz Schwarz von der Gesellschaft für Mathematik und Datenverarbeitung (GMD) in St. Augustin gewählt. Von Anfang an bis heute ist der Computeralgebra-Rundbrief das Rückgrat aller Aktivitäten der Fachgruppe, der zweimal im Jahr an alle Mitglieder gesandt wird. Der Rundbrief wurde über die Jahre immer wieder zeitgemäß in ein neues Gewand gebracht und erweitert. 1988 fand erstmals bei der DMV-Jahrestagung eine eigene Sektion „Computational Algebra“ statt.

Volker Weispfenning 1990 – 1993

1990 – 1993 nach der ersten Wahl der Fachgruppenleitung durch die Mitglieder leitete Volker Weispfenning, Universität Passau, die Fachgruppe. In dieser Zeit wurde der Report „Computeralgebra in Deutschland — Bestandsaufnahme, Möglichkeiten, Perspektiven“ erstellt. Themen und Schwerpunkte, Anwendungen und Systeme der Computeralgebra sowie alle Arbeitsgruppen in Deutschland wurden aufgenommen und breit verteilt. Damit entstand ein Referenzpunkt, der zur weiteren Identitätsfindung des Gebiets Computeralgebra maßgeblich beitrug. Später entstand dann daraus ein weltweit organisiertes Projekt für ein „Handbook of Computer Algebra“, siehe auch den Artikel „Literatur zur Computeralgebra“ auf Seite 75. F. Schwarz organisierte 1991 in Bonn zum zweiten Mal in Deutschland nach 1987 (Leipzig) die internationale Tagung ISSAC’91.

Johannes Grabmeier 1993 – 1999

In der Zeit von 1993 – 1999 hatte dann Johannes Grabmeier, IBM Heidelberg, das Sprecheramt inne. Mit dem CAIS — Abkürzung für Computeralgebra-Informationssystem — wurde sehr früh ein Internetauftritt realisiert. Im Jahr 1994 bestritt die Fachgruppe in Bonn mit ihrem Thema eine Wissenschaftspressekonferenz mit Berichten in Printmedien und im Hörfunk. Anwesend waren neben den Referenten der Computeralgebra alle Präsidenten der drei Trägergesellschaften.

Die Fachgruppe organisierte 1994 auch eine Konferenz „Computer Algebra in Science and Engineering“ am Zentrum für interdisziplinäre Forschung (ZiF) in Bielefeld. Die Intensivierung der Aktivitäten machte es 1994 auch notwendig, einen bescheidenen Jahresbeitrag zur Finanzierung der Fachgruppe von den Mitgliedern zu erheben. Neu eingebunden in die Fachgruppenleitung in dieser Periode wurden Vertreter der Physik — dem ältesten Anwendungsgebiet der Computeralgebra — sowie der Fachhochschulen. 1996 war erstmals ein Workshop Computeralgebra und ein Hauptvortrag auf der GI-Jahrestagung in Klagenfurt. Unter der Federführung der Fachgruppe wurde von Karl Hantzschmann die ISSAC'98 in Rostock organisiert. Ein Impuls zum Thema „Computeralgebra in Lehre, Ausbildung und Weiterbildung“ wurde durch den Beginn einer Tagungsreihe mit diesem Namen 1998 in Schloss Thurnau bei Bayreuth gesetzt. Seitdem wird diese Reihe alle zwei Jahre an wechselnden Orten von der Fachgruppe veranstaltet.

H. Michael Möller 1999 – 2002

In dieser Periode war H. Michael Möller von der Universität Dortmund der Sprecher. Das wegen der Breite der Computeralgebra schwierige Thema Benchmarking wurde in dieser Periode angegangen und die bewährten Aktivitäten der Fachgruppe fortgesetzt. 2000 gab es beispielsweise eine eigene Sektion Computeralgebra bei der Jahrestagung der GAMM in Göttingen.

Wolfram Koepf, seit 2002

Seit 2002 ist Wolfram Koepf von der Universität Kassel nun in der dritten Periode Sprecher der Fachgruppe Computeralgebra.

Organisatorisch neu war die Idee die Rubriken des Rundbriefs in feste Verantwortungen zu geben. Das Layout der Rundbriefs wurde weiter verbessert und auch Farbe ins Spiel gebracht. Die neue und bis heute gültige Internetadresse `www.fachgruppe-computeralgebra.de` wurde aktiviert und die Inhalte neu strukturiert. 2006 wurde dafür ein Content-Mangementsystem eingerichtet. Erstmals 2003 in Kassel organisierte die Fachgruppe eine wissenschaftliche Tagung mit eingeladenen Hauptvorträgen und Kurzvorträgen. Seitdem findet diese Tagung alle zwei Jahre statt. Neu eingeführt zu Werbezwecken bei Tagungen und anderen Gelegenheiten wurde ein CA-Flyer. Für 2010 plant die Fachgruppe zusammen mit Ernst Mayr von der TU München, sich um die erneute Ausrichtung der ISSAC-Konferenz zu bewerben. In Haus Schönenberg wurde 2006 die mittlerweile 5. Konferenz zur Computeralgebra in der Lehre organisiert. Aktuelle Aktivitäten sind — siehe dieses Heft — Beiträge der Fachgruppe Computeralgebra zum Jahr der Mathematik 2008.

GeoGebra: Vom Autodesign zur Computerschriftart

Dr. Markus Hohenwarter
Mathematical Sciences
Florida Atlantic University
777 Glades Road
Boca Raton, FL 33431, USA

mhohen@math.fau.edu

Zusammenfassung

Wie werden eigentlich Computerschriftarten in verschiedenen Formen und Größen erstellt und gespeichert? Bei der Untersuchung dieser Frage zeigt sich, wie viel an spannender Mathematik in grundlegenden Dingen unseres Alltags steckt. Auf den Spuren des Autodesigners und Mathematikers Pierre Bézier entdecken wir dabei mit Hilfe der dynamischen Mathematiksoftware GeoGebra grundlegende Bausteine von Computerdesign und Vektorschriftarten.

Schriftarten am Computer

Wir lesen und schreiben tag-täglich elektronische Texte, sei es die Kurznachricht am Mobiltelefon, der Werbetext im Fernsehen, der Musiktitel am iPod oder der Artikel am Computer. In einem modernen Textverarbeitungsprogramm haben wir die Qual der Wahl unter hunderten von Schriftarten und können auch noch die Größe jedes Schrifttyps frei wählen. Wie funktioniert das eigentlich? Wie werden diese Schriftarten und -größen am Handy oder Computer gespeichert und dargestellt?

Ein Gedanke, der einem als mögliche Antwort durch den Kopf schießen mag, sind „Bilder“: man braucht doch nur jeden Buchstaben jeder Schriftart in jeder gewünschten Größe als kleines Bildchen zu speichern und kann daraus dann seinen gewünschten Text im Lego-Baukastensystem zusammensetzen. Teilweise wurde und wird das auch tatsächlich so gemacht, in den meisten Fällen macht man es heute aber cleverer: mit Hilfe von Geometrie und Algebra. Sehen wir uns zunächst einmal an, wie Bilder auf einem Computer gespeichert werden. Dabei unterscheidet man zwischen Rastergrafiken und Vektorgrafiken.

Raster- und Vektorgrafiken

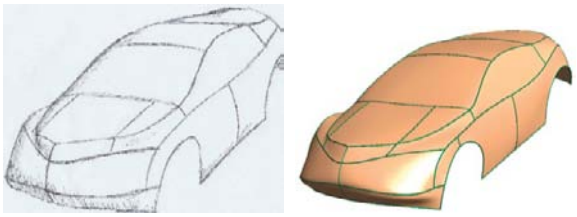
Rastergrafiken bestehen aus einer rasterförmigen Anordnung von so genannten Pixeln (Bildpunkten), denen jeweils eine Farbe zugeordnet ist. Die Hauptmerkmale einer Rastergrafik sind daher die Breite und die Höhe in Pixeln, auch Auflösung genannt.

Vektorgrafiken basieren anders als Rastergrafiken nicht auf einem Pixelraster, in dem jedem Bildpunkt ein Farbwert zugeordnet ist, sondern definieren sich über eine Bildbeschreibung mittels mathematischer Objekte wie Punkte, Kreise und Kurven.

Mit einer Digitalkamera erzeugte Fotos sind Rastergrafiken: je mehr Bildpunkte (Stichwort „Megapixel“) ein Bild hat, umso höher ist seine Qualität. Ein Nachteil von Rastergrafiken ist, dass sie sich nur mit Qualitätsverlust vergrößern lassen. Bei einer Vektorgrafik werden die Bildinformationen als mathematische Objekte in einem Koordinatensystem gespeichert. So kann beispielsweise ein Kreis in einer Vektorgrafik über die Lage des Mittelpunktes, Radius, Linienstärke und Farbe vollständig beschrieben und ohne Qualitätsverlust beliebig vergrößert werden.

Für elektronische Schriftarten, die auf verschiedenen Bildschirmen und Druckern mit unterschiedlichen Auflösungen in bestmöglicher Qualität reproduziert werden sollen, sind Vektorgrafiken daher die erste Wahl. Dabei werden die Umrisse jedes einzelnen Buchstaben durch mathematische Kurven beschrieben. Praktisch alle Schriftarten, die wir heute auf Computern verwenden, sind Vektorschriftarten (z.B. TrueType, MetaFont) und integraler Bestandteil von Dokumentformaten wie PostScript oder PDF (portable document format). Bei der mathematischen Beschreibung von Vektorschriftarten haben die sogenannten *Bézierkurven* eine besondere Bedeutung, die ursprünglich für das Design von Autos entwickelt wurden.

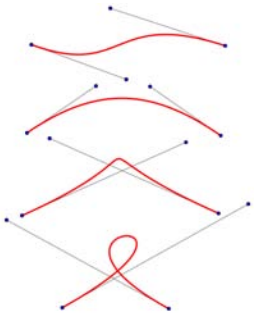
Autodesign und Bézierkurven



Händischer Entwurf und Computerdesign eines Autos [6]

Als Paul De Casteljau 1959 eine Methode zur einfachen Beschreibung von Karosserieteilen für den französichen Autohersteller Citroen ersann, wusste er wahrscheinlich nicht, dass sein Kollege Pierre Bézier bei Renault an einer ähnlichen Idee arbeitete. Die beiden Ingenieure und Mathematiker entwickelten damals mit Hilfe von Geometrie und Algebra die aus der heutigen Computergrafik nicht mehr wegzudenkenden Bézierkurven.

Vor der Verwendung von Computern basierte Autodesign auf einem langwierigen Prozess von händischen Konstruktionen und Tonmodellen. Durch die Erfindung von De Casteljau und Bézier konnten Autoteile ab Mitte der 1960er Jahre dagegen sehr viel einfacher und schneller mit Computergrafik entworfen werden. Das besondere an den neuen Bézierkurven war, dass sie auf intuitive und interaktive Art und Weise am Computer verändert werden konnten.

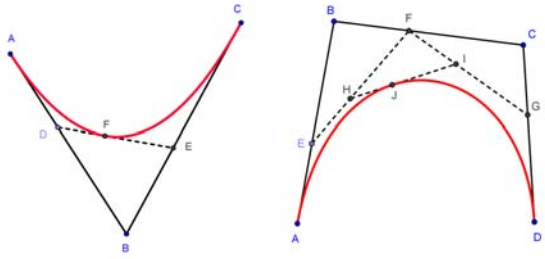


Pierre Bézier (1910-1999) [7] und kubische Bézierkurven

Eine quadratische Bézierkurve hat drei sogenannte Kontrollpunkte, mit denen sich ihre Form und Lage beeinflussen lässt. Noch mehr Flexibilität bietet eine kubische Bézierkurve mit vier Kontrollpunkten. Auf der Webseite [5] können Sie mit der kostenlosen dynamischen Mathematiksoftware GeoGebra [3] beide Typen von Kurven interaktiv verändern und untersuchen. Der Algorithmus von De Casteljau (siehe Kasten) beschreibt, wie solche Bézierkurven geometrisch konstruiert werden. Die entsprechenden interaktiven Konstruktionen für quadratische und kubische Bézierkurven sind ebenfalls unter [5] im Internet zu finden. In De Casteljaus Konstruktion wird ersichtlich, wie geometrische Konstruktionen und analytische Geometrie, sprich die Vektorrechnung, hier zusammenspielen.

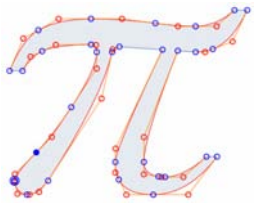
Algorithmus von De Casteljau

Um eine quadratische Bézierkurve zu konstruieren, erzeugen wir zunächst drei beliebig gewählte Kontrollpunkte A, B und C und zeichnen die Verbindungsstrecken $\overline{AB}$ und $\overline{BC}$. Als nächstes setzen wir einen Punkt D auf die erste Strecke $\overline{AB}$ und sehen uns jenes Verhältnis t an, in dem D die Strecke $\overline{AB}$ teilt. Damit erhalten wir den Teilungsparameter $t = \overline{AD}/\overline{AB}$, also eine Zahl zwischen 0 und 1. Nun übertragen wir dieses Teilverhältnis auf die zweite Strecke $\overline{BC}$. Dazu suchen wir jenen Punkt E, für den $\overline{BE}/\overline{BC} = t$ gilt. Mit Hilfe der Vektorrechnung erhalten wir diesen Teilungspunkt als $E = B + t \cdot \overrightarrow{BC}$ bzw. $E = B + t \cdot (C - B)$. Die beiden neuen Punkte D und E verbinden wir ebenfalls durch eine Strecke $\overline{DE}$, auf welche wir nochmals unser Teilverhältnis anwenden. Damit bekommen wir schließlich einen Punkt F auf $\overline{DE}$ mit $F = D + t \cdot (E - D)$. Bewegen wir nun den Punkt D entlang der Strecke $\overline{AB}$, so wandert F entlang einer quadratischen Bézierkurve. Oder mit anderen Worten: Die Ortslinie von F in Abhängigkeit von D beschreibt eine quadratische Bézierkurve.



Quadratische und kubische Bézierkurven in GeoGebra

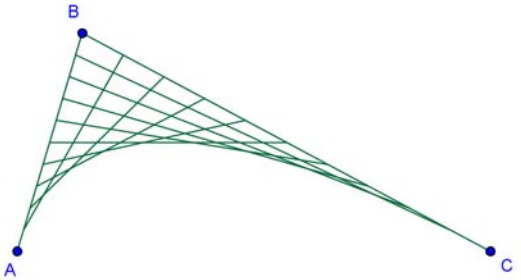
In GeoGebra lassen sich basierend auf diesen Konstruktionen auch benutzerdefinierte Werkzeuge für Bézierkurven erstellen. Damit können wir dann versuchen, durch Aneinanderfügen von Bézierkurven den Umriss eines Buchstabens zu designen (siehe [5]). Sogenannte „TrueType“ Schriftarten wie etwa „Times New Roman“ oder „Arial“ auf Windows Computern verwenden quadratische Bézierkurven mit drei Kontrollpunkten, um die Konturen von Buchstaben zu beschreiben. „PostScript“ Schriftarten verwenden zusätzlich auch kubische Bézierkurven mit vier Kontrollpunkten. Im Prinzip könnte man auch Bézierkurven noch höherer Ordung verwenden, in der Praxis ist es aber einfacher, kompliziertere Kurven durch Aneinanderfügen dieser beiden einfachen Typen zu bauen.



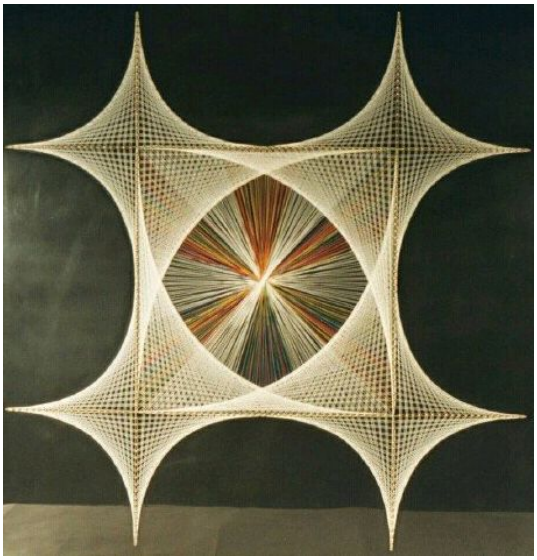
Buchstabe als Vektorgrafik mit quadratischen Bézierkurven

Fadenkunst und Tangenten

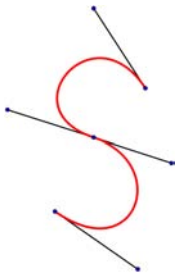
Die Konstruktionsmethode von De Casteljau für quadratische Bézierkurven findet sich übrigens auch in der sogenannten „Fadenkunst“. Dabei unterteilt man die beiden aneinanderstoßenden Kontrollstrecken $\overline{AB}$ und $\overline{BC}$ in eine gleiche Anzahl von Abschnitten und verbindet entsprechende Teilungspunkte (oft Stecknadeln oder Nägelchen) mit Fäden. Auf diese Art entsteht eine quadratische Bézierkurve als sogenannte Hüllkurve, bei der alle Fäden Tangenten an die Bézierkurve sind.



Quadratische Bézierkurve mittels „Fadenkunst“



„Fadenkunst“. Fotografie: Condon Kay Collection [2]



Insbesondere sind auch die Kontrollstrecken selbst Tangenten an die Bézierkurve. Diese Eigenschaft lässt sich gut dazu benutzen, zwei Bézierkurven „glatt“ aneinander zu fügen, indem man die beiden entsprechenden Kontrollstrecken auf einer Geraden platziert, wie hier für den Buchstaben S mit kubischen Bézierkurven zu sehen ist.

Mit GeoGebra können wir eine Bézierkurve einerseits mittels geometrischer Konstruktion als Ortslinie und andererseits mit Hilfe ihrer algebraischen Darstellung als Parameterkurve (siehe Kasten) erzeugen und für interaktive Experimente nutzen. So können wir etwa fünf beliebige Punkte auf einer quadratischen Bézierkurve wählen und untersuchen, welche Art von Kegelschnitt wir durch diese Punkte legen können. GeoGebra gibt uns dabei sowohl Auskunft über die Gleichung als auch den Typ des entsprechenden Kegelschnitts. Im interaktiven Experiment (siehe [5]) sehen wir dabei, dass eine quadratische Bézierkurve offenbar immer ein Parabelbogen ist. Auch die Tangenteneigenschaft der Kontrollstrecken können wir mit GeoGebra experimentell untersuchen, indem wir mittels Tangentenwerkzeug eine Tangente in einem Punkt einer Bézierkurve erzeugen, die dann interaktiv und dynamisch verschoben werden kann.

Dynamische Mathematik

Dynamische Mathematiksoftware wie GeoGebra ermöglicht die interaktive Untersuchung von geometrischen und algebraischen Zusammenhängen. Damit lassen sich interessante mathematische Zusammenhänge und Anwendungen untersuchen — wie hier Computerschriften und Bézierkurven. Zahlreiche weitere interaktive Konstruktionen sind im Materialienpool GeoGebra-Wiki [4] kostenlos verfügbar.

Parameterdarstellung einer Bézierkurve

Mit Hilfe des Algorithmus von De Casteljau können wir auch eine algebraische Darstellung von Bézierkurven angeben und die angesprochene Tangenteneigenschaft der Kontrollstrecken überprüfen. Sehen wir uns wieder den Fall einer quadratischen Bézierkurve mit den Kontrollpunkten A, B und C an. In der Konstruktionsmethode von De Casteljau haben wir Strecken wiederholt im gleichen Verhältnis geteilt und folgende Zusammenhänge erhalten: (1) $D = A + t \cdot (B - A)$, (2) $E = B + t \cdot (C - B)$ und (3) $F = D + t \cdot (E - D)$, wobei t zwischen 0 und 1 läuft. Daraus bekommen wir eine algebraische Darstellung $B(t)$ der Kurve, die nur von t und den Kontrollpunkten A, B und C abhängt, indem wir (1) und (2) in (3) einsetzen:

$$B(t) = t^2 \cdot (A - 2B + C) + 2t \cdot (B - A) + A.$$

Wir erhalten also eine quadratische Parameterdarstellung in t, womit auch klar wird, woher der Name „quadratische“ Bézierkurve stammt. Mit Hilfe dieser Parameterdarstellung lässt sich nun auch nachprüfen, dass die Kontrollstrecken Tangenten an die Kurve sind. Dazu betrachten wir die Ableitung der Kurve, welche uns für jeden Parameterwert t die Richtung des Tangentenvektors der Kurve angibt:

$$B'(t) = 2t(A - 2B + C) + 2(B - A).$$

Sehen wir uns damit nun die Tangentenvektoren am Beginn ($t = 0$) und Ende ($t = 1$) der Kuve an: $B'(0) = 2 \cdot (B - A) = 2 \cdot \overrightarrow{AB}$ und $B'(1) = 2 \cdot (C - B) = 2 \cdot \overrightarrow{BC}$. Dies zeigt, dass die Kontrollstrecken $\overline{AB}$ und $\overline{BC}$ wirklich Tangtenten an die quadratische Bézierkurve sind. Für Bézierkurven höherer Ordnung kann man sich dies in ganz ähnlicher Art und Weise überlegen (vgl. [8]).

Links und Literatur

[1] M. Atiyah, *Mathematics in the 20th Century: geometry versus algebra*, Mathematics Today, 37(2), 46 – 53, 2001.

[2] Condon Kay Collection (2008). Geometric String Art, siehe www.trocadero.com/condonkay.

[3] GeoGebra (2008). Dynamische Mathematiksoftware, siehe http://www.geogebra.org.

[4] GeoGebraWiki (2008). Freier Pool von interaktiven Materialien zu GeoGebra, siehe www.geogebra.org/de/wiki/.

[5] Hohenwarter (2008). Bézierkurven mit GeoGebra, siehe www.geogebra.org/de/wiki/index.php/Bezier-CA2008.

[6] Machine Design (2007). An Alternative to Nurbs, siehe machinedesign.com/.

[7] D. Salomon, *Curves and Surfaces for Computer Graphics*, Springer Verlag, Berlin, Heidelberg, New York, 2005.

[8] Wikipedia (2007). De Casteljau Algorithmus, siehe de.wikipedia.org.

Cinderella.2 — Geometrie und Physik im Dialog

Prof. Dr. Ulrich Kortenkamp
Pädagogische Hochschule Schwäbisch Gmünd
Oberbettringer Straße 200
73525 Schwäbisch Gmünd

Prof. Dr. Dr. Jürgen Richter-Gebert
Lehrstuhl für Geometrie und Visualisierung
Technische Universität München, Garching
Boltzmannstraße 3
85748 Garching bei München

kortenkamp@cinderella.de
richter@cinderella.de

Zusammenfassung

Dynamische Geometriesysteme sind nicht nur für den Mathematikunterricht geeignet. Das DGS Cinderella bietet in seiner neuesten Version 2.0 auch einen Simulationsmodus, mit dem einfache physikalische Experimente durchgeführt werden können. Dabei kommt die Geometrie aber auch nicht zu kurz, wie wir in diesem Artikel darstellen.

Geometrie und Realität

Geometrie ist nicht nur ein spannender Teil der Mathematik, sondern war schon seit der Antike die Grundlage der Architektur. Vor über 200 Jahren erfand Gaspard Monge die darstellende Geometrie, um dadurch eine mathematische Grundlage für den Bau von Festungsanlagen zu haben. Viele Rechen- und Darstellungsmethoden, wie zum Beispiel homogene Koordinaten, die aus dem *Barycentrischen Calcul* von August Ferdinand Möbius [1] hervorgingen, sind heute unentbehrlich für CAD-Programme, ohne die wiederum die moderne Architektur nicht denkbar wäre. Daher dürfen diese Grundlagen im Architekturstudium nicht fehlen [2].

Das Geometrieprogramm Cinderella [3] ist seit Anbeginn darauf ausgelegt, eine saubere und stabile mathematische Grundlage für dynamische Geometrie zu liefern, die auch mit Ausnahmesituationen wie „unendlich fernen Punkten“ oder (für die Schule) ungewöhnlichen mathematischen Modellen, zum Beispiel hyperbolischer Geometrie, umgehen kann. Dies alles dient aber nicht nur Mathematikern, sondern ist auch für den Schuleinsatz geeignet, wie zum Beispiel die mit Cinderella gestalteten Webseiten www.geomouse.ch zeigen.

Die neueste Version, Cinderella.2 [4], geht in vielen Aspekten über die ursprünglichen Ansätze heraus. Zum Einen sind diese Erweiterungen im klassischen Geometrieteil zu finden: So werden zum Beispiel Abbildungen nicht nur — wie gewohnt — dynamisch aktualisiert, sondern stehen als eigene Objekte zur Verfügung, die wiederum miteinander verknüpft werden können.

Wesentlich spannender sind allerdings die gänzlich neuen Teile *Simulationen (CindyLab)* und *Algorithmen (CindyScript)*. In diesem Artikel stellen wir nur eine kleine Anwendung des Simulationsteils vor, weitere Informationen zu den anderen Teilen finden sich auf der Cinderella-Homepage cinderella.de, und auch im (derzeit leider nur auf Englisch verfügbaren) Handbuch doc.cinderella.de.

Die Einführung eines Simulationsmoduls, mit dem physikalische Experimente durchgeführt werden können, ist Teil der Bestrebungen der Autoren, die in der Schule oft eher abstrakt oder (über eingekleidete Aufgaben) nur bedingt realitätsnah behandelte Geometrie wieder an echte Probleme heranzuführen. Das im folgende vorgeführte Experiment ist ein Beispiel dafür, dass man mit Geometrie-Software auch fächerübergreifend mehr als bisher erreichen kann.

Die Golden Gate Bridge

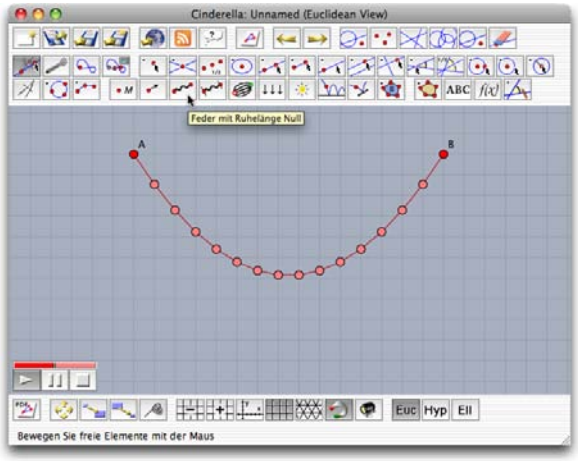
Ausgangspunkt unseres Geometrie-Experiments ist die Golden Gate Bridge in San Francisco. Als eine der längsten Hängebrücken der Welt gehört sie zu den bedeutendsten architektonischen Wahrzeichen der Vereinigten Staaten von Amerika.

Die Golden Gate Bridge. Fotografie: Aaron Logan, www.lightmatter.net/gallery/albums.php

Die Golden Gate Bridge überspannt den *Golden Gate*, die Verbindung zwischen der Bucht von San Francisco und dem Pazifik seit 1937. Sie ist eine Hängebrücke, das heißt, dass die eigentliche Brücke an Stahlseilen aufgehängt ist, die selbst an zwei monströsen Pfeilern (227m hoch!) befestigt sind. Die Stahlseile selbst haben einen Durchmesser von 92 cm — nicht gerade das, was man sich unter einem Seil vorstellt, aber notwendig, um das immense Gewicht von insgesamt 887 000 Tonnen zu halten.

Wir möchten nun diese Brücke untersuchen und herausfinden, welchen Bogen die oberen Halteseile beschreiben, und warum dieser genau so aussieht, wie er aussieht.

Die Modellierung der Seile werden wir zunächst mit (virtuellen) Gummibändern durchführen; diese werden wie Federn proportional zu der auf sie wirkenden Kraft gedehnt, haben aber eine Ruhelänge von 0. Unser Ziel ist, die Gestaltung der Brücke so durchzuführen, dass sie nicht durch im Innern wirkende Kräfte zerrissen wird. Die Gummibänder zeigen uns mit ihrer Ausdehnung genau, welche Länge die Stahlseile später haben müssen, damit diese inneren Kräfte nicht auftreten. Man kann sich also vorstellen, dass die mit Gummibändern gefundene Form dann aus Metall nachgebaut wird. Dieses Gestaltungsprinzip findet sich in der Architektur immer wieder (siehe auch [5]). So werden zum Beispiel Dachkonstruktionen wie die von Frei Otto (Olympiastadion München!) im Modell meist durch Damenstrumpfhosen realisiert — eine Art zweidimensionales Gummiband!



Hängende Gewichte in Cinderella. Ein Video, das zeigt wie diese Konstruktion erstellt wurde, findet sich unter [7]

Physik-Simulationen in Cinderella

Wir starten nun Cinderella und schalten die Werkzeugleiste für Physiksimulationen ein (über „Datei/Toolbars auswählen“, und dann „CindyLab“). Zusätzlich zu den normalen geometrischen Werkzeugen werden nun auch Werkzeuge für Massepunkte, Federn, Wände und andere Simulationsobjekte freigeschaltet [6].

Als Vorübung zu der Untersuchung der Brücke erstellen wir zunächst eine einfache Simulation, die das Durchhängen einer Reihe von Gewichten demonstriert. Dazu fixieren wir zwei „normale“ Punkte A und B, und verbinden diese dann über eine Reihe von Gummibändern.

Wir können in der Simulation einfach solche Parameter wie Reibung und Schwerkraft einstellen. Die Reibung „beruhigt“ die Simulation und sorgt dafür, dass sie sich — im wahrsten Sinne des Wortes — einpendelt. Mit dem Schwerkraftregler können wir sehen, wie stark die Gewichte durchhängen. Spielen wir ein wenig mit dem Regler, so können wir in der obigen Simulation eine durchhängende Kurve herstellen, die stark der Kurve ähnelt, die das obere Halteseile der Golden Gate Bridge beschreibt. Es handelt sich hierbei um eine *Parabel*!

Warum eine Parabel?

Die meisten Mathematiker erwarten, dass die Gummibänder hier als Kurve eine Kettenlinie (Katenoide) beschreiben, also eine Funktion der Form

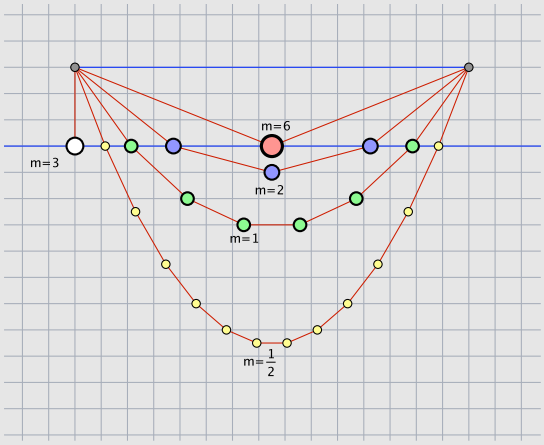
$$f(x) = a \cdot \cosh \frac{x - x_0}{a} + y_0 \,.$$

Wir können uns aber auch ohne viel Rechnen recht schnell klar machen, dass es sich um eine Parabel handeln muss. Betrachten wir eine Kette aus n Massepunkten gleicher Masse, die durch ideale masselose Federn mit Ruhelänge 0 verbunden sind und deren Enden an zwei gleich hohen Fixpunkten mit Nägeln befestigt sind. Die Schwerkraft wirkt senkrecht nach unten. Das Superpositionsprinzip gestattet uns die Situation in x- und y-Richtung getrennt zu betrachten.

In x-Richtung passiert nicht viel Spannendes: Da in diese Richtung keine Schwerkraft wirkt, pendeln sich einfach gleiche Abstände in x-Richtung zwischen den Punkten ein. Die x-Koordinaten unserer Punkte werden einfach um den immer gleichen Betrag größer (sagen wir, um 1, damit wir nicht so viel rechnen müssen).

Was passiert nun in y-Richtung? Wir nehmen der Einfachheit halber an, dass wir es mit einer ungeraden Anzahl von Massen zu tun haben (andernfalls wird die Überlegung ein klein wenig komplizierter, ist aber im Prinzip die gleiche). Das Koordinatensystem verschieben wir so, dass der unterste Punkt der hängenden Kette im Koordinatenursprung liegt. Die Masse des untersten Massepunktes zieht an den beiden benachbarten Gummibändern und streckt diese in y-Richtung auf eine Länge l. An den links und rechts daran angrenzenden Gummibändern hängt das Gewicht der untersten drei Massen, also werden diese in y-Richtung auf $3l$ gedehnt. Es folgen Dehnungen von $5l$, $7l, \ldots$ Damit ergeben sich aber für die Höhen der Massepunkte ausgehend vom untersten Punkt der Reihe nach die Werte $1l, 1l+3l = 4l, 1l+3l+5l = 9l, 1l+3l+5l+7l = 16l, \ldots$, — also liegen sie auf einer Parabel!

Das erkennt man auch gut in dem folgenden Bild, welches unter [8] auch animiert zur Verfügung steht:

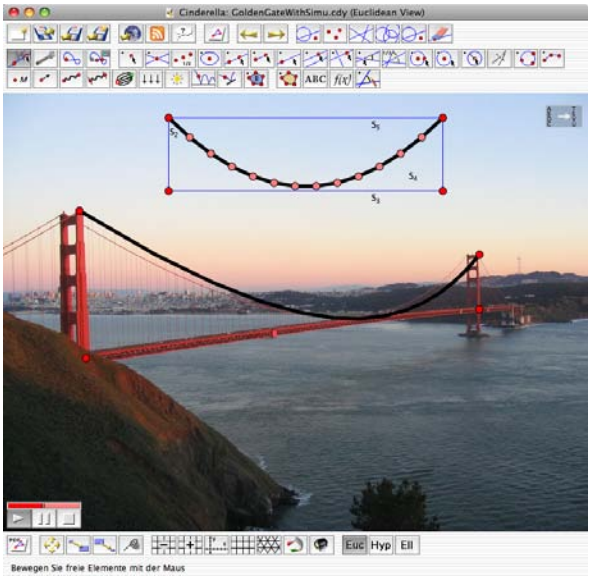


Die Mathematik, die hinter den Physik-Simulationen steckt, ist die Numerik partieller Differentialgleichungen. Mit dem *Runge-Kutta-Verfahren* löst Cinderella die Differentialgleichungen, die die Kräfteverhältnisse für den jeweiligen Versuchsaufbau beschreiben, und zeigt die Ergebnisse direkt animiert an.

Die Brücke ist zu lang!

Jetzt könnten wir schon fast überprüfen, ob die Golden Gate Bridge nach den Prinzipien gestaltet wurde, die wir zuvor in der Theorie erarbeitet haben. Eine rasche Bildersuche über Flickr[3], Google oder Wikipedia Commons[4] liefert schnell viele Ansichten der Brücke, doch auf keiner ist die Brücke verzerrungsfrei von vorn dargestellt. Wir können also die Parabel aus dem Versuch nicht direkt auf die Brücke übertragen.

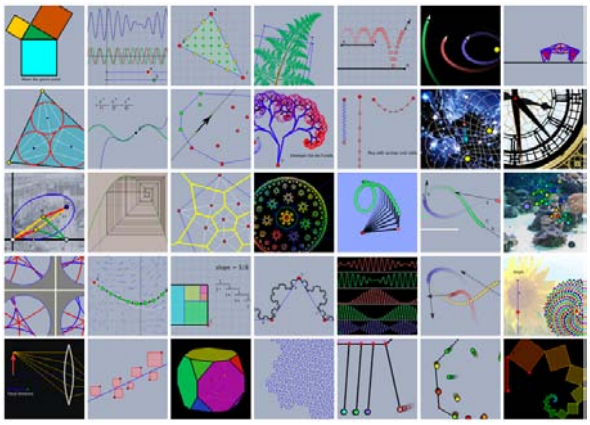
Jetzt kommt *wieder* die Geometrie in das Spiel: Wir definieren in Cinderella eine projektive Transformation — das sind die Abbildungen, die man erhält, wenn man ein Foto auf eine Wand projiziert — von den vier Ecken der Simulation auf die Spitzen und Basen der Brückenpfeiler. Diese Projektion (im Bildschirmfoto rechts oben zu sehen) wenden wir auf die Gummibänder an und erhalten als Resultat den Verlauf der „Hängeparabel“ im Foto, als hätten wir die Gewichte tatsächlich an die echte Brücke gehängt.



Unter [9] kann man diese Simulation selbst ausprobieren, das „making of“ ist im Video unter [10] zu sehen.

Lässt man die Fantasie ein wenig spielen, so fallen einem noch viele weitere spannende und fächerübergreifende Beispiele ein, die die Physiksimulation ausnutzen. Kombiniert man das ganze noch mit der eingebauten Sprache CindyScript, so ergeben sich noch mehr Möglichkeiten.

Eine Beispielsammlung, die von Gestängemechanismen und Optik über Planetensimulation bis hin zur Simulation von Fischschwärmen geht, findet man auf der Homepage von Cinderella.



Die Version 1.4 von Cinderella, unter anderem ausgezeichnet mit dem EASA 2002 und dem Deutschen Bildungssoftwarepreis digita2001 gibt es als kostenlosen Download unter cinderella.de. Die neue Version 2.0 mit Physiksimulationen und vielen weiteren Funktionen gibt es ebenfalls unter dieser Adresse.

Links und Literatur

[1] A. F. Möbius, *Der barycentrische Calcul*, Originalausgabe: Verlag von Johann Ambrosius Barth, Leipzig 1827, Nachdruck: Georg Olms, 1976.

[2] H. Pottmann, A. Asperl, M. Hofer und A. Kilian, *Architectural Geometry*, Bentley Institute Press, 2007.

[3] J. Richter-Gebert und U. H. Kortenkamp, *The Interactive Geometry Software Cinderella*, Springer, Heidelberg, 1999.

[4] J. Richter-Gebert und U. H. Kortenkamp, *Math in Motion — The Interactive Geometry Software Cinderella*, Version 2.0, 2006, siehe cinderella.de.

[5] M. Schuster, *Seminararbeit Designgeschichte: Frei Otto*, (Technische Grundlagen), 1997, siehe www.aspekt1.net/ms/fo_ref/tgrundl.html.

[6] U. Kortenkamp, *Erste Schritte mit Physiksimulationen in Cinderella.2*, 2008. YouTube Video: youtube.com/watch?v=2nrpP1PeKyI.

[7] U. Kortenkamp, *Hängende Gewichte in Cinderella.2*, 2008. YouTube Video: youtube.com/watch?v=2nrpP1PeKyI.

[8] U. Kortenkamp, *Cinderella Blog: Hängende Parabel*, 2008. Cinderella-Konstruktion unter blog.cinderella.de/archives/206-Haengende-Parabel.html.

[9] U. Kortenkamp, *Cinderella Blog: Golden Gate Simulation*, 2008. Cinderella-Konstruktion unter blog.cinderella.de/archives/210-Golden-Gate-Simulation.html.

[10] U. Kortenkamp, *Golden Gate Bridge Simulation mit Cinderella.2*, 2008. YouTube Video: youtube.com/watch?v=xHY6G2fTLPg.

[3] flickr.com

[4] Das hier verwendete Bild ist aus der Wikipedia Commons, aufgenommen von Dirk Beyer und unter der Creative Commons Share-alike-Lizenz (cc-by-sa-2.5) lizensiert.

FeliX — mit Algebra Geometrie machen

Prof. Dr. Reinhard Oldenburg
Institut für Didaktik der Mathematik und Informatik
Johann Wolfgang Goethe Universität Frankfurt a. M.
Senckenberganlage 9 und 11
60325 Frankfurt am Main

oldenbur@math.uni-frankfurt.de

Zusammenfassung

Zeichnen oder rechnen? Viele Probleme kann man geometrisch oder algebraisch lösen. Die Übersetzung zwischen geometrischer und algebraischer Welt kann man mit dem Programm FeliX live erleben: Beide Sichtweisen sind gleichberechtigt integriert.

Zum Einstieg: Eine Kurbel mit Gummiband

Mathematik liefert die Sprache, mit der man viele Situationen beschreiben kann. In Abbildung 1 ist eine drehbare Kurbel K gezeigt, an der ein Gummiband montiert ist, dessen zweites Ende an einem Punkt F fixiert ist. Der Mittelpunkt M des Bandes ist markiert. Wenn man jetzt kurbelt, auf welcher Bahn bewegt sich dann M? Vielleicht reicht die Vorstellungskraft aus, das Problem zu lösen. Im Normalfall kann man sich eine Zeichnung machen, eine Reihe von Mittelpunkten für verschiedene Kurbelstellungen einzeichnen und so die Bahn gewinnen. Alternativ rechnet man algebraisch: Wenn der Drehpunkt der Kurbel im Ursprung liegt, und der Kurbelarm die Radiuslänge r hat, erfüllen die Koordinaten (x_K, y_K) von K die Kreisgleichung $x_K^2 + y_K^2 = r^2$. Der Mittelpunkt $M(x_M, y_M)$ von K und $F(x_F, y_F)$ erfüllt die Gleichungen $2x_M = x_F + x_K$ und $2y_M = y_F + y_K$. Wenn man diese Gleichungen nach (x_K, y_K) auflöst und in die Kreisgleichung einsetzt, erhält man: $(2x_M - x_F)^2 + (2y_M - y_F)^2 = r^2$. Dies ist eine Kreisgleichung mit Mittelpunkt $\left(\frac{x_F}{2}, \frac{y_F}{2}\right)$, wie man besser erkennt, wenn man die ganze Gleichung durch $4 = 2^2$ dividiert: $\left(x_M - \frac{x_F}{2}\right)^2 + \left(y_M - \frac{y_F}{2}\right)^2 = \left(\frac{r}{2}\right)^2$.

Dass man eine Kreisbahn erhält, kann man auch geometrisch verstehen: Das Gummiband realisiert eine zentrische Streckung. Der Vorteil der (mühevollen) algebraischen Lösung ist, dass man mit dem Ergebnis gut weitere Fragen untersuchen kann, z. B. Schnittpunkte berechnen oder Tangenten anlegen. Vor allem aber kann man die algebraische Lösungsmethode systematisch auch auf kompliziertere Probleme anwenden, und dabei kann man sich von Computeralgebrasystemen helfen lassen.

Besonders elegant gestaltet sich die Lösung, wenn man FeliX verwendet. Dieses Geometrieprogramm basiert auf Computeralgebra und kann solche algebraischen Berechnungen durchführen. Abbildung 2 zeigt die entsprechende Lösung. Allerdings geht es bei FeliX nicht nur darum, Lösungen zu erhalten, sonderen vor allem darum, den Zusammenhang von Algebra und Geometrie besser zu verstehen.

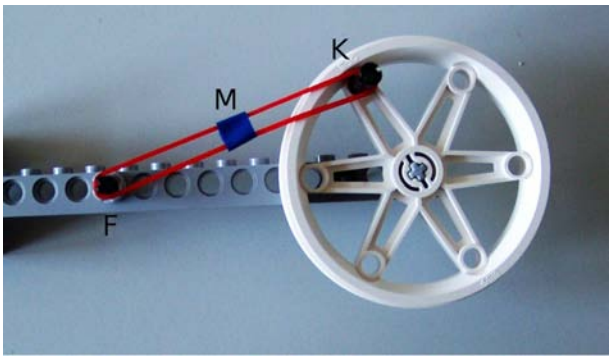


Abbildung 1: Eine Kurbel dehnt ein Gummiband

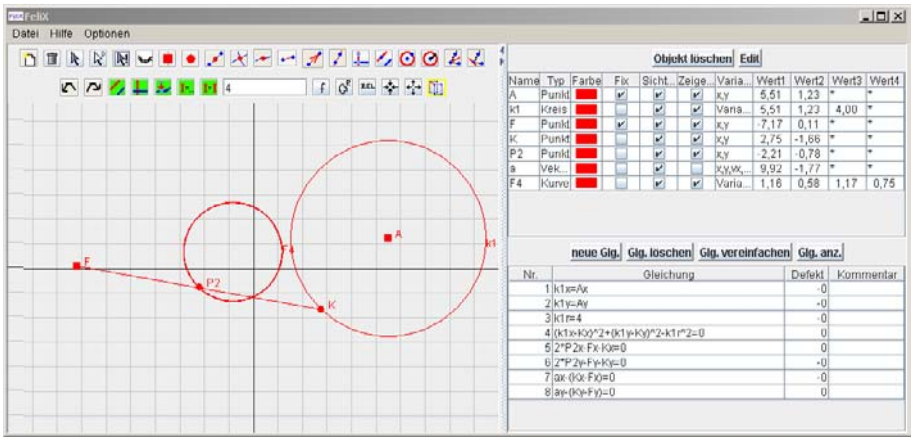


Abbildung 2: Die Situation aus Abbildung 1 mathematisch realisiert mit FeliX

FeliX — ein Algebra-Geometrie-Programm

Die Grundidee von FeliX ist recht einfach: Man erzeugt im Geometriefenster geometrische Objekte wie Punkte, Geraden, Strecken und Kreise, die mit der Maus verschoben werden können. In der Objektliste werden die Objekte mit ihren numerischen Koordinaten angezeigt. Dort kann man die Koordinaten auch ändern, was sofort eine Neupositionierung im Geometriefenster nach sich zieht.

Objektart	**Variablen**	**Bedeutung**
Punkt / point P	Px, Py	Kartesische Koordinaten des Punktes
Kreis / circle K	Kx, Ky, Kr	Mittelpunktkoordinaten und Radius
Grade / line L	La, Lb, Lc	Koeffizienten in der Gleichung ax+by=c
Strecke V oder Vektor V	Vx, Vy	Vektorkomponenten Vx, Vy

Tabelle 1: Die Objektarten in FeliX und ihre Koordinaten

Objekte alleine sind aber noch nicht so spannend. Interessant wird die virtuelle geometrische Welt erst durch die Beziehungen zwischen den Objekten. Solche Beziehungen sind beispielsweise, dass ein Punkt Mittelpunkt zweier anderer ist, dass zwei Geraden parallel oder orthogonal sind, dass ein Punkt auf einer Geraden oder auf einem Kreis liegt oder der Schnittpunkt zweier Objekte ist. Solche Beziehungen lassen sich alle mit Gleichungen ausdrücken und die Idee von FeliX ist die, dass das System beim Bewegen von Objekten mit der Maus (oder durch direkte Eingabe von neuen Koordinaten) immer die Gültigkeit einer Reihe von Gleichungen einhält. Dabei ist es egal, ob die Gleichungen vom Benutzer von Hand in die Gleichungstabelle eingetragen worden sind, oder ob sie über die Konstruktionsbuttons wie „Mittelpunkt erzeugen“ erzeugt wurden.

Ein Beispiel: Der Benutzer erzeugt zwei Punkte A und B, wählt danach das Mittelpunktswerkzeug aus der Buttonleiste und klickt die beiden Punkte an. Es erscheint im Geometriefenster ein neuer Punkt M und in der Gleichungstabelle tauchen zwei neue Gleichungen auf: `2*Mx=Ax+Bx` und `2*My=Ay+By`. Egal, an welchem Punkt man dann wie zieht, immer bleibt M der Mittelpunkt von A und B. Das ändert sich erst, wenn man die Gleichungen ändert, z.B. in `3*Mx=2*Ax+Bx` und `3*My=2*Ay+By`. Haben Sie eine Vorstellung, was das bewirkt?

Weitere interessante Möglichkeiten ergeben sich daraus, dass man auch Ungleichungen eingeben kann. Damit kann man z.B. verhindern, dass Kreise sich überlappen. Daraus resultiert ein Verhalten, wie man es von physikalischen Scheiben erwartet: Die Kreise schieben sich gegenseitig weg.

Für das weitere ist noch die Unterscheidung von fixen und nicht fixen Objekten nötig: Fixe Punkte werden als kleine Quadrate, nicht fixe als kleine Kreisscheiben dargestellt. Das Verhalten der Konstruktion beim Ziehen wird durch die Zugregel bestimmt:

Fixe Objekte ändern ihre Koordinaten nur, wenn sie direkt bewegt werden, aber nie mittelbar in Reaktion auf die Bewegung anderer Punkte.

Ob ein Objekt fix ist oder nicht, kann jederzeit umgeschaltet werden.

Kurven

Konstruktionen wie die in Abschnitt 1 führen oft zu Punkten, die sich nur auf einer bestimmten Kurve bewegen lassen. Schöne Beispiele sind die Kegelschnitte. Eine Ellipse ist etwa die Menge aller Punkte P, für die die Summe der Abstände zu zwei gegebenen Punkten A und B eine bestimmte Konstante ist. Um eine Ellipse in FeliX zu erzeugen, kreiert man also zwei fixe Punkte A und B und einen nicht-fixen Punkt P. Die Abstände muss man mit dem Satz des Pythagoras berechnen (`sqrt` ist die Wurzelfunktion). Eine geeignete Gleichung ist: `sqrt((Px-Ax)^2+(Py-Ay)^2)+sqrt((Px-Bx)^2+(Py-By)^2)=10`. Diese Gleichung bewirkt, dass sich P mit der Maus nur noch auf der Ellipse ziehen lässt, für die alle Punkte die Abstandssumme 10 zu den beiden gegebenen Punkten haben. Um die Ellipse auch als Kurve zu sehen, benutzt man den Kurven-Button und klickt dann auf P. FeliX berechnet dann die Gleichung der Kurve mit den Koordinaten von A und B als variablen Parametern und

zeichnet sie dann. Mit dieser Technik lassen sich viele interessante Kurven erzeugen, u. a. Herzkurven (Kardioiden) oder die „Hundelinie“, die eigentlich „Konchoide des Nikomedes“ (siehe [1]) heißt. Hier soll aber lieber nochmal das Lego ausgepackt werden.

Legoaffe

Das Kurbelbeispiel aus Abschnitt 1 ist etwas künstlich: Wo sind schon Gummibänder gespannt und wer interessiert sich für deren Mittelpunkt? Etwas anders liegen die Dinge bei dem Lego-Seilbahn-Roboter in Abbildung 3. Es ist schon spannend ihm zuzusehen, wie er sich an einem gespannten Seil vorwärts hangelt: Die Hand wird schnell von hinten nach vorne geführt und dann zieht er sich ein Stück weiter. Aber ob sich die Hand dabei auf einem Kreis oder einer Ellipse oder einer anderen Kurve bewegt, kann man kaum beurteilen. Mit FeliX lassen sich die Kurbel und die Schubstange aber schnell nachbilden und die Bahnkurve berechnen.

Abbildung 3: Der Motor-Kletteraffe von Lego

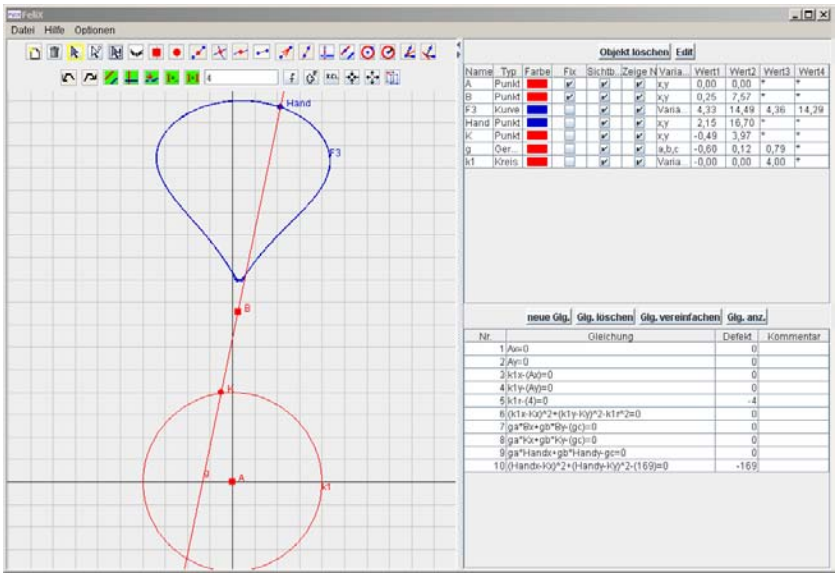


Abbildung 4: Der Kletteraffe in FeliX: Der Kreismittelpunkt ist die Motorachse, der Punkt auf dem Kreis der „Ellbogen“. Man sieht, dass sich die Hand auf einer komplexen Bahn bewegt.

Schluss

FeliX existiert in verschiedenen Varianten: Neben der kleinen eindimensionalen Fassung FeliX1D, bei der man Variablen auf einem Zahlenstrahl verschiebt, gibt es zwei Ausgaben, mit denen man die hier beschriebenen zweidimensionalen Probleme lösen kann. Eine setzt auf dem kommerziellen Computeralgebrasystem MuPAD auf, die andere benötigt MuPAD nicht und ist dadurch leichter zu installieren und vollständig kostenlos, wenn auch gegenwärtig nicht ganz so leistungsfähig. Das Programm kann von der Homepage des Autors[5] kostenlos herunter geladen werden. Es setzt eine Java-Laufzeitumgebung der aktuellen Version 1.6 voraus, die ebenfalls frei erhältlich ist.

Links und Literatur

[1] D. Haftendorn, *Konchoide des Nikomedes* (Hundekurve), siehe www.fh-lueneburg.de/mathe-lehramt/mathe-lehramt.htm.

[2] R. Oldenburg, *Bidirektionale Verknüpfung von Computeralgebra und dynamischer Geometrie*, Journal für Mathematikdidaktik, 26 (2005), 249 – 273.

[5] www.math.uni-frankfurt.de/∼oldenbur/

Enrichment, Computermathematik & Maple

Prof. Dr. Thomas Schramm
Department Geomatik
HafenCity Universität Hamburg
Hebebrandstraße 1
22297 Hamburg

Tim Buhrke
Gymnasium Wentort
Hohler Weg 16
21465 Wentorf bei Hamburg

thomas.schramm@hcu-hamburg.de
tim_buhrke@web.de

Zusammenfassung

Enrichment ist ein Förderkonzept für besonders begabte Schülerinnen und Schüler an Schleswig-Holsteiner Schulen. Diese Schulen haben sich in Verbünden organisiert und bieten schulübergreifend Kurse an. Wir berichten über einen **Workshop für Computermathematik**, der am Gymnasium Wentorf in Zusammenarbeit mit der HafenCity Universität Hamburg statt findet.

Vorgeschichte

Seit einigen Jahren verstärkt sich an den Hochschulen der Eindruck, dass sich die Leistungen der Studienanfänger in den Naturwissenschaften und der Mathematik verschlechtern. Dass dieser Eindruck nicht nur *gefühlt*, sondern regional unterschiedlich nachweisbar ist (vergl. [1]), verstärkt die Motivation etwas dagegen zu unternehmen. Härtere Aufnahmekriterien sind bei schwindenden Bewerberzahlen sicher keine zukunftsorientierte Lösung. Das Gespräch mit Schulen zu suchen eher. Beide Seiten können davon profitieren. An Schulen wird das Anforderungsprofil der Hochschulen deutlich, aber die Hochschulen können auch lernen, dass Schulabsolventen vermehrt neue Kompetenzen (z.B. Organisations-, Kritik-, Problemlösungs- und Teamfähigkeit) mitbringen, die genutzt werden müssen.

Vor diesem Hintergrund wurden vom Department Geomatik der HafenCity Universität Hamburg (ehemals an der Hochschule für Angewandte Wissenschaften Hamburg) ein- und mehrtägige Messpraktika (Turmhöhen- und astronomische Breitengradbestimmung) an Schulen in Hamburg und im Umland mit dem Motto „Mathematik macht Spaß und ist anwendbar“ durchgeführt. Ein Kontakt, geboren aus der Elternmitarbeit, hat sich dauerhaft etabliert. Mit einer Lehrerfortbildung wurde das Computeralgebrasystem (CAS) Maple am Gymnasium Wentorf eingeführt. Seither gibt es unter der Leitung der Autoren eine Schüler-Arbeitsgruppe mit dem Namen CoMa-AG zur **Co**mputer-**Ma**thematik (vergl. [2, 3]). Die Idee war (und ist) es, den Schülerinnen und Schülern aller Altersstufen gemeinsam Freude an der Arbeit mit dem Komplizierten zu vermitteln.

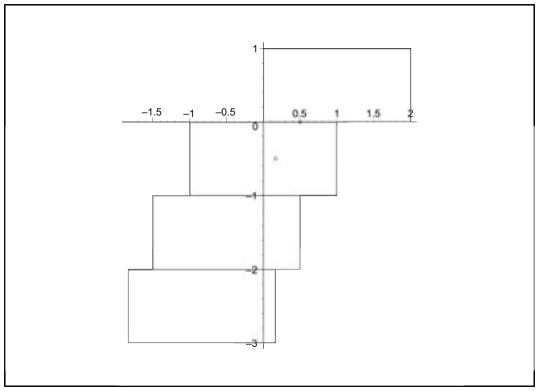
CoMa-AG

Diese Ziele wurden teilweise erreicht, insbesondere, nachdem wir unsere Konzeption etwas angepasst hatten und neben Bearbeitung mathematischer Fragestellungen auch mit sog. Mindstorm-Robotern experimentierten (de.wikipedia.org/wiki/Mindstorm), oder zur Entspannung mit thematisch naheliegenden Computerprogrammen (z.B. Crazy Machines (de.wikipedia.org/wiki/Crazy_Machines)) spielten.

Zu Beginn gaben wir eine kleine Einführung in das Computeralgebrasystem Maple und regten die Mitglieder an, neu hinzukommende einzuweisen. Wir analysierten verschiedene Fragestellungen z.B. das ...

Telefonbuchproblem

Wie weit kann man die Bücher auf einem Telefonbuchstapel verrücken, um bei einer gegebenen Anzahl eine möglichst große Strecke in der Horizontalen zu überbrücken, ohne, dass der Stapel umkippt? Da keine Telefonbücher zur Hand waren, wurde das Problem mit Bauklötzen abstrahiert und experimentiert. So nebenbei wurde dabei das Konzept des Schwerpunktes gefunden und entsprechende Zeichnungen mit Maple erstellt.



Telefonbuchstapel mit maximaler Verrückung nach rechts. Der Schwerpunkt aller über einem Buch befindlichen Bücher befindet sich jeweils genau auf der rechten Kante.

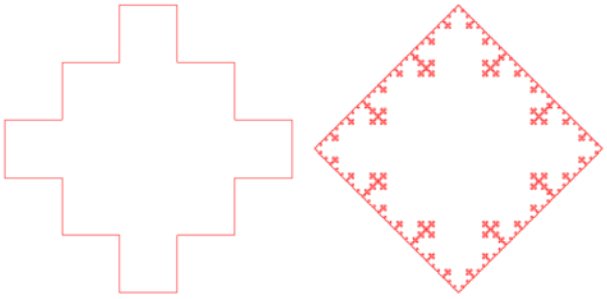
Dieses Beispiel zeigte, dass hier jede Altersstufe angesprochen werden konnte. Die jüngeren experimentierten, zeichneten auf Papier und übertrugen Koordinaten in den Plotbefehl Maples. Gemeinsam wurde eine Formel für die jeweilige „Verrückung“ aus Zahlenbeispielen entworfen, mit Maple ausprobiert und dann die Frage, die wir hier nicht beantworten wollen, diskutiert, wie weit das denn gehen kann — theoretisch und praktisch.

Ein weiteres Highlight aus der gemeinsamen Arbeit war die Konstruktion der …

Schneeflocke

Hier wurde die Frage bearbeitet, was denn mit der Gesamtfläche einer „Schneeflocke“ genannten Figur passiert, wenn man bei einem Quadrat jeweils auf die Mitte der Seiten ein neues Quadrat platziert, das ein Drittel der Seitenlänge des Ausgangsquadrats hat und dieses Verfahren auf jeder der entstehenden neuen Seiten mit jeweils genauso verkleinerten Quadraten ad infinitum Stufe um Stufe fortsetzt.

Dieser etwas kompliziert zu erklärende Vorgang ist an der Tafel schnell skizziert und koordinatenweise in Maple übertragen. Dabei zeigten gerade die jüngeren Schülerinnen und Schüler eine ungeheure Akribie und konstruierten und rechneten Stufe um Stufe der Iterationen. Ein Team brachte es bis zur 15. Stufe und stellte fest, dass die Gesamtfläche nie das Zweifache des Ausgangsquadrates übersteigt.



Links ein Bild der ersten und
rechts eines der vierten Stufe der Schneeflocke.

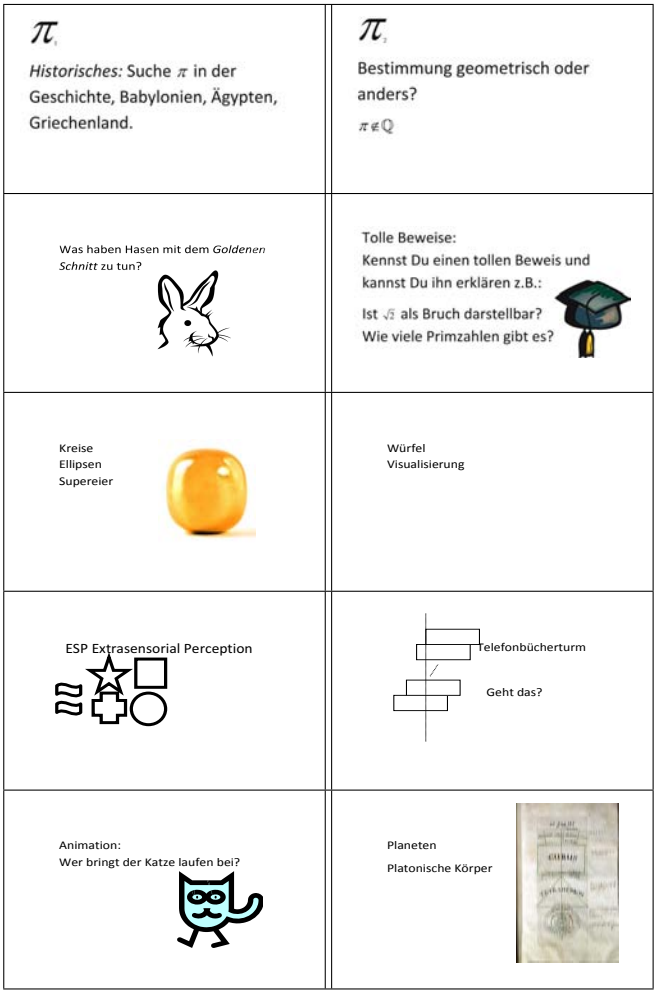
Hanna, aus der achten Klasse, hat Ihre Gedanken dazu aufgeschrieben: *Nun habe ich überlegt, ob, wenn man diesen Vorgang immer wieder wiederholt, es irgendwann vom Flächeninhalt her 2 Quadrate von der ursprünglichen Größe gibt. Obwohl ich viele Schritte ausgerechnet habe, gab es keine zwei Quadrate. Zuerst haben wir die Vermutung angestellt, dass es sogar beliebig groß wird, weil ja immer etwas dazu kommt. Aber das, was dazu kommt, ist ja immer kleiner und so kann es nie ein Ganzes werden.* Die älteren Schüler entwickeln hier mit etwas Hilfe eine Formel für die Gesamtfläche und bestimmen mit Hilfe Maples den Grenzwert, der die Vermutung bestätigt.

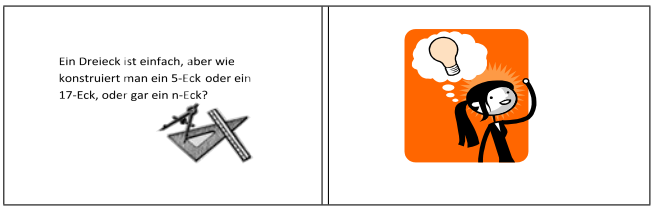
Diese Highlights dürfen jedoch nicht darüber hinwegtäuschen, dass solche Arbeitsgruppen sehr betreuungsintensiv sind. Der Workshopcharakter stellt sich nur zeitweise ein und es ist schwierig den Spannungsbogen für komplexere Probleme altersübergreifend aufrecht zu erhalten. Insofern waren wir gespannt, wie sich die Situation mit der Einführung des Enrichmentkonzeptes ändern würde.

Enrichment

Im Schuljahr 2007/2008 wurde die CoMa-AG zum Enrichment Workshop für Computermathematik erweitert. Innerhalb des Enrichment-Konzeptes bekommen besonders geeignete Schülerinnen und Schüler der umliegenden Schulen meist durch die Zeugniskonferenzen eine Empfehlung, sich für Enrichment-Kurse zu bewerben. Schlicht gesagt werden dort Schülerinnen und Schüler ausgewählt, die „mehr Futter“ benötigen und das müssen nicht unbedingt immer die Klassenbesten sein (`enrichment.lernnetz.de`). Zu Beginn des Schuljahres konnten wir 14 Teilnehmerinnen und Teilnehmer von vier Gymnasien begrüßen, die z. T. erhebliche Wege in Kauf nahmen. In der Zwischenzeit ist die Gruppe etwas geschrumpft, aber recht aktiv.

Unsere Ziele waren hoch gesteckt. Eigentlich wollten wir nur Themen anbieten und die Ausgestaltung den Schülerinnen und Schülern überlassen. Es sollten sich kleine Gruppen bilden, die selbstständig das Thema erarbeiten und dokumentieren sollten. Das Ziel war ein Verständnis des Problemkreises, die Dokumentation im Internet und ein Vortrag in der Gruppe. Wir wollten ein „lernerzentriertes“ Arbeitsklima schaffen und selbst nur als „Coach“ agieren. Unsere Themenangebote waren z. B.:





Die Themen wurden auf Kärtchen von den Schülerinnen und Schülern herumgereicht und es konnte frei gewählt werden. Auf die Frage, was denn zu tun sei, haben wir erst einmal bewusst nur mit der Schulter gezuckt. Wir bemerkten lediglich, dass zur Bearbeitung Maple benutzt werden könnte und wir für Fragen zur Verfügung stehen würden, man könne aber auch die anderen fragen oder im Internet suchen. Das letzte Thema sei ein Joker, für die, die schon wüßten, was sie machen wollten. Hier waren die jüngeren zuerst etwas überfordert. Zwei Mädchen aus der Oberstufe eines externen Gymnasiums offenbar nicht. Sie wählten das Fibonacci-Thema (Hasen, Goldener Schnitt) probierten etwas in Maple herum und meldeten sich nach einer Stunde als „fertig“. Sie hatten das Thema bereits im Unterricht behandelt und nicht wirklich verstanden, dass wir lediglich einen Startpunkt gegeben hatten. Auch hatten wir den Eindruck, dass sie sich im Kreise der „Kleinen“ nicht wirklich wohl fühlten. Wir hatten auch keine Chance unser Konzept zu erläutern, da sie zum nächsten Termin nicht mehr erschienen.

Der Rest der Gruppe war etwas geduldiger (mit uns). Gewählt wurden die Themen:

- π — Historisch
- Tolle Beweise
- Würfel visualisieren
- ESP
- Platonische Körper
- Fünfeck

Wir wollen im Folgenden kurz über die Einzelprojekte berichten.

π — Historisch

Hanna und Jasmin (8./7. Klasse) näherten sich dem Thema anders als (von uns) gedacht. Sie beschäftigten sich mit den Rechenkünsten und insbesondere den Darstellungen der Zahlen im alten Ägypten und Rom. Sie fassten Ihre Erkenntnisse in einer Powerpoint-Show zusammen und erzeugten auch eine HTML-Version, die bald im Internet zu bewundern sein wird [6].

Tolle Beweise

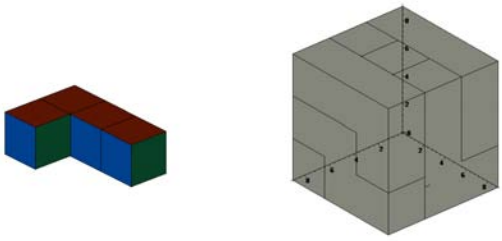
Marvin aus einer sechsten Klasse interessierte sich für Primzahlen und arbeitete allein. Er experimentierte mit Maple, um die Primfaktorzerlegung von Zahlen zu erhalten. Hierzu eigneten sich die Befehle isprime, ithprime, nextprime, prevprime und ifactor besonders gut. Er fand eine Version des *Satzes von Euklid* bei Wikipedia (de.wikipedia.org/wiki/Satz_von_Euklid) und probierte die Beispiele mit Maple aus. Es ging darum, die Argumentationskette, dass es unendlich viele Primzahlen geben muss, *wirklich* zu verstehen, so dass sie den anderen erklärt werden konnte. Hierzu fertigte er ein Word-Dokument an, das auch mit Maple und für das Internet aufbereitet wird. Während seines Vortrages tauchten weitere Fagen auf z. B.: wieso es nur eine Primfaktorzerlegung gäbe, die er noch beantworten will.

Würfel visualisieren

Kim und Nicolas aus der sechsten Klasse nahmen sich vor, das Zusammensetzen eines Puzzlewürfels mit Maple zu visualisieren. Nicolas entdeckte in Maple eine Plotstruktur PLOT3D(POLYGONS(...)), die er an das Problem der Teilwürfel anpasste. Der Plan ist für jeden Teilwürfel eine Struktur zu entwerfen und diese dann animiert zusammenzufügen. Kim benutzte ebenfalls die Maplefunktionalität, um den fertigen Würfel darzustellen. Nach harter Arbeit hatte er seine Punktfolge zusammen. Hier ein Ausschnitt

```
with(plots):
polygonplot3d([[9,0,0],[0,0,0],
[0,9,0],[0,0,0],[0,0,9],[0,9,9],
[0,0,9],[9,0,9],[9,0,0],[9,9,0],
[9,9,9], ...
[6,3,0],[6,6,0],[6,9,0],[9,9,0]],
axes=normal);
```

Es hat uns erstaunt, mit welcher Energie die beiden ihr Ziel verfolgten, allerdings auch die Hartnäckigkeit, mit der Ratschläge ignoriert wurden. Wir konnten allerdings schon häufiger beobachten, dass gerade jüngere Schüler und Schülerinnen durch große geordnete Zahlenmengen und deren Darstellungen fasziniert wurden. Hier noch zwei Figuren, die die beiden aus ihren Daten erzeugten.



Links Nicolas Teilwürfel des zu animierenden Gesamtsystems, das rechts als Kims Außenansicht zu sehen ist.

ESP

In diesem, etwas scherzhaften gemeinten Projekt, haben Jan, Tim und Tobias (10. Klasse) den Standardtest zur Bestimmung „außersinnlicher Wahrnehmung“ (engl. **e**xtrasen**s**orial **p**erception, esp) mit den Mitgliedern der Gesamtgruppe und den Leitern durchgeführt.

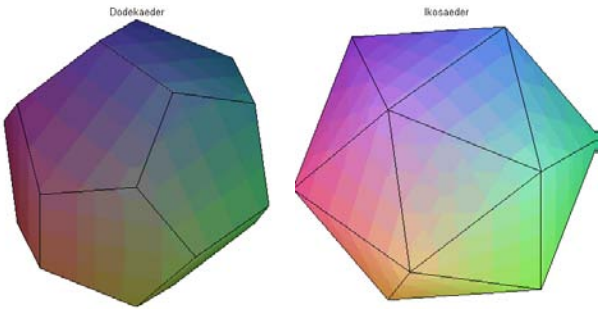
(Wir haben zu diesem Zeitpunkt vom Comeback Uri Gellers noch nichts geahnt). Schnell wurde das Konzept des Mittelwertes und der Varianz angewendet, aber auch die Frage gestellt, was denn *normal* sei. Hierzu gab es eine Extrastunde in Testtheorie. Offensichtlich ist niemand von uns paranormal begabt. Jetzt sollen die Ergebnisse mit dem Statistikpaket Maples aufbereitet und dokumentiert werden.

Platonische Körper

Diana aus einer achten Klasse hat sich dem Problem der platonischen Körper und der Verwendung für ein Modell des Planetensystems durch Kepler interessiert. Sie trug dazu Informationen aus dem Internet (de.wikipedia.org/wiki/Johannes_Kepler) zusammen und diskutierte mit uns die Modellbildung, die Kepler selbst zugunsten der heute so genannten keplerschen Gesetze verwarf. In einem ersten Kurzvortrag stellte Diana eine Powerpoint-Show zusammen. Für die endgültige Dokumentation versucht sie platonische Körper mit Maple darzustellen und diese für das keplersche Modell ineinander zu schachteln. Hierzu eignet sich das geom3d-Paket in Maple mit dem Befehl RegularPolyhedron besonders gut. Allerdings ist die Dokumentation auf Englisch und wir helfen nur ausnahmsweise. Z. B. erzeugt die Befehlsfolge

```
with(geom3d);
RegularPolyhedron(a, [5,3],
    point(o,0,0,0),1);
RegularPolyhedron(b, [3,5],
    point(o,0,0,0),1);
draw(a, title = "Dodekaeder");
draw(b, title = "Ikosaeder");
```

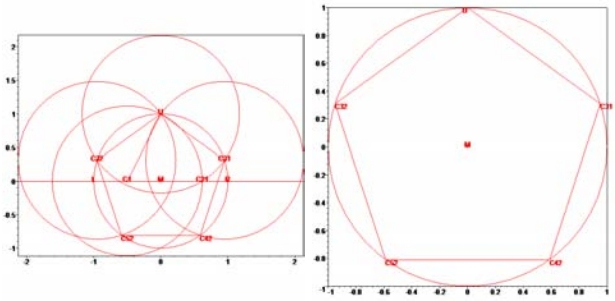
einen Dodekaeder und einen Ikosaeder.



Zwei platonische Körper für Mars- Erd- und Venusbahn

Fünfeck

Alena und Carolin aus der neunten Klasse erforschen auf Gauß' Spuren die Konstruktion von regelmäßigen Vielecken. Bei 17 sind sie noch nicht, aber das Fünfeck ist verstanden. Hier bietet sich das geometry-Paket Maples an, eine Zirkel- und Linealkonstruktion algorithmisch nachzuempfinden, auch, wenn die Syntax etwas gewöhnungsbedürftig ist.



Die algorithmische Konstruktion und das bereinigte Ergebnis eines Fünfecks mit dem geometry-Pakets Maples

Fazit

Wir glauben, dass wir mit unserem Konzept auf dem richtigen Weg sind, den Schülerinnen und Schülern nachhaltig Freude am Umgang mit mathematischen oder allgemein schwierigen Problemen zu vermitteln. Maple ist sicher ein geeignetes Werkzeug, wenn es mit Augenmaß eingesetzt wird. Wir nutzen es als allgemeine „Problemlösungsumgebung" bzw. als Erweiterung des Schreibpapiers und ermöglichen so die Behandlung des Nichttrivialen. Mittelfristig können solche Projekte dazu dienen, den Dialog zwischen Schule und Hochschule zu intensivieren, die Schülerinnen und Schüler besser auf ein mögliches Studium vorzubereiten und letztendlich den Studienerfolg zu verbessern.

Links und Literatur

[1] C. Polaczek, *Studienerfolg in den Ingenieurwisssenschaften. Eingangsvoraussetzungen — Prognose — Validität*, Proc. Minisymp. DMV Berlin 2007, Heft 01/2007 Teil 1, Wismarer Frege-Reihe, Wismar, 2007.

[2] T. Schramm und T. Buhrke, *Mathematisches Assessment in der Schul- und Ingenieurausbildung*, Proc. Minisymp. DMV Berlin 2007, Heft 01/2007 Teil 2, Wismarer Frege-Reihe, Wismar, 2007.

[3] T. Schramm, *Back to School: Mathematikförderung zwischen Universität und Schule*, Global J. of Engng. Educ., Special Edition, 10(3):315, 2006.

[4] Superei: Bild von Malene Thyssen, siehe commons.wikimedia.org/wiki/Image:Superaeg.jpg.

[5] Zeichnung: Johannes Kepler, siehe de.wikipedia.org/wiki/Bild:Kepler-1619-pl-3.jpg.

[6] Weitere Informationen unter coma.gymnasium-wentorf.de/.

Wellen als Vektoren

Roland Mechling
Oken-Gymnasium
Vogesenstraße 10
77652 Offenburg

roland@mechling.de

Dynamische Geometriesysteme (kurz DGS) können in vielen Situationen zur Klärung und Veranschaulichung komplizierter Zusammenhänge eingesetzt werden. Im folgenden Aufsatz wird das von Richard P. Feynman eingeführte Verfahren der „Zeigeraddition“ zur Behandlung von Interferenzphänomenen (vgl. [1]) dargestellt und in einigen einfachen Situationen angewendet. Die Abbildungen sind statische Bilder von dynamischen Zeichnungen, die mit einem DGS (hier: DynaGeo) erstellt wurden. Natürlich erschließen sich manche Aspekte der in den Bildern steckenden Informationen erst beim interaktiven Umgang mit den dynamischen Zeichnungen, weshalb letztere im WWW frei zur Verfügung gestellt werden (siehe unten).

Richard P. Feynman (Quelle: [2])

Jeder Oberstufenschüler hat im Unterricht bei der Einführung der trigonometrischen Funktionen wohl schon einmal ein Bild wie das folgende gesehen:

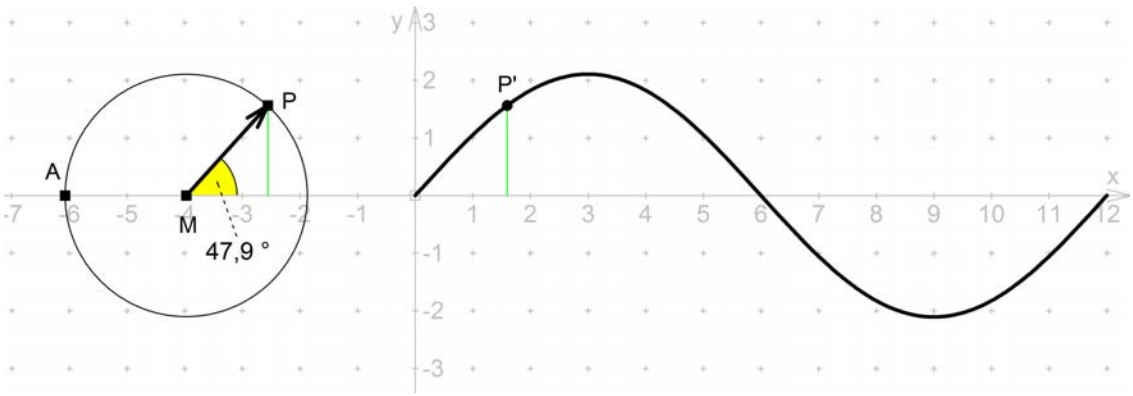


Abb. 1

Läuft P auf dem dargestellten Kreis einmal im Gegenuhrzeigersinn herum, dann beschreibt P’ die dargestellte sinusförmige Kurve. Die y-Koordinate von P’ ist dabei stets gleich der y-Koordinate von P, während die x-Koordinate von P’ proportional zu dem Winkel ist, den der Pfeil $\overrightarrow{MP}$ mit der positiven x-Richtung einschließt. Wichtig ist nun, dass das linke Bild mit dem Vektor $\overrightarrow{MP}$ genau die gleichen Informationen enthält wie das rechts dargestellte Funktionsschaubild: zu jeder möglichen Lage von P gehört eine eindeutig festgelegte Lage von P’, und umgekehrt. Also ist der rotierende Vektor $\overrightarrow{MP}$ eine vollständige Repräsentation der rechts dargestellten sinusförmigen Kurve.

Betrachten wir einen zweiten Punkt Q, der ebenfalls auf einem Kreis um M läuft, aber möglicherweise in einem anderen Abstand als P. Dabei soll der Winkel $\Delta\varphi$ zwischen den Vektoren $\overrightarrow{MQ}$ und $\overrightarrow{MP}$ stets konstant bleiben. Die beiden Vektoren sollen also synchron um M rotieren, d.h. mit derselben Winkelgeschwindigkeit. Das zu Q gehörende Schaubild ist dann ebenfalls eine sinusförmige Kurve, die aber gegenüber der zu P gehörigen Kurve phasenverschoben ist. Man erhält dann z.B. das folgende Bild.

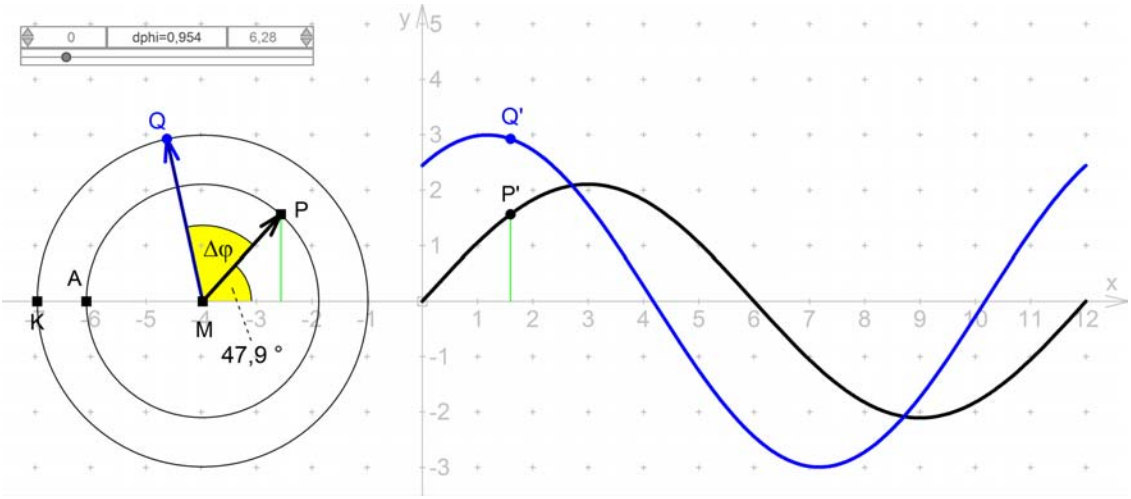


Abb. 2

Interpretieren wir die beiden Schaubilder als Momentaufnahmen von Wellen, die sich auf einem linearen Träger (z. B. einem Seil) bewegen, dann interessiert den Physiker, wie sich der Träger verhält, wenn sich beide Wellen gleichzeitig auf ihm ausbreiten. Wir wissen, dass sich die Einzelwellen ungestört überlagern, indem sich die einzelnen Auslenkungen addieren. Welche Kurve erhält man nun, wenn man die beiden sinusförmigen Kurven addiert? Wenn wir die Erkenntnis benutzen, dass die Vektoren alle Informationen über die zugehörigen Schaubilder enthalten, dann können wir statt der üblichen y-Werte-Addition der Schaubilder auf der rechten Seite viel einfacher die Vektoren auf der linken Seite addieren: dies führt zu einem Summenvektor $\overrightarrow{MR} = \overrightarrow{MP} + \overrightarrow{MQ}$, und die zum rotierenden Punkt R gehörende Kurve ist das gesuchte „Summenschaubild“ (siehe Abb. 3).

Dieses Verfahren ist verallgemeinerbar auf die Addition vieler Vektoren, und es leistet besonders dort gute Dienste, wo viele sinusförmige Größen addiert werden müssen. Als einfaches Beispiel soll hier die Beugung von Licht am Vierfachspalt betrachtet werden.

An der Stelle VSp befinde sich ein Vierfachspalt, der von einer (hier nicht eingezeichneten) von links kommenden ebenen Lichtwelle beleuchtet wird. Die Wellenlänge des Lichts (in nm) kann am Zahlobjekt „Lambda“ eingestellt werden, der Abstand g (in μm) benachbarter Spalte am Zahlobjekt „g“. Wir betrachten nur den Fall sehr enger Spalte, so dass jeder der vier Spalte als Ausgangspunkt einer einzigen Elementarwelle angesehen werden kann, so wie es das Huygens'sche Prinzip vorschreibt. Wenn wir nun nach der Lichtintensität in einem Punkt B auf dem (weit entfernten) Schirm fragen, müssen wir dort die von den vier einzelnen Spalten kommenden Wellen überlagern. Dabei ist die Phasenverschiebung $\Delta\varphi$ benachbarter Wellenstrahlen aufgrund des Gangunterschiedes um so größer, je weiter B von der optischen Achse entfernt liegt. Diese Phasenverschiebung wird im Termobjekt „dphi“ passend zu den vorgegebenen Daten (Wellenlänge, Spaltabstand, Position von B) berechnet, so wie man das im Physikkurs der Oberstufe lernt.

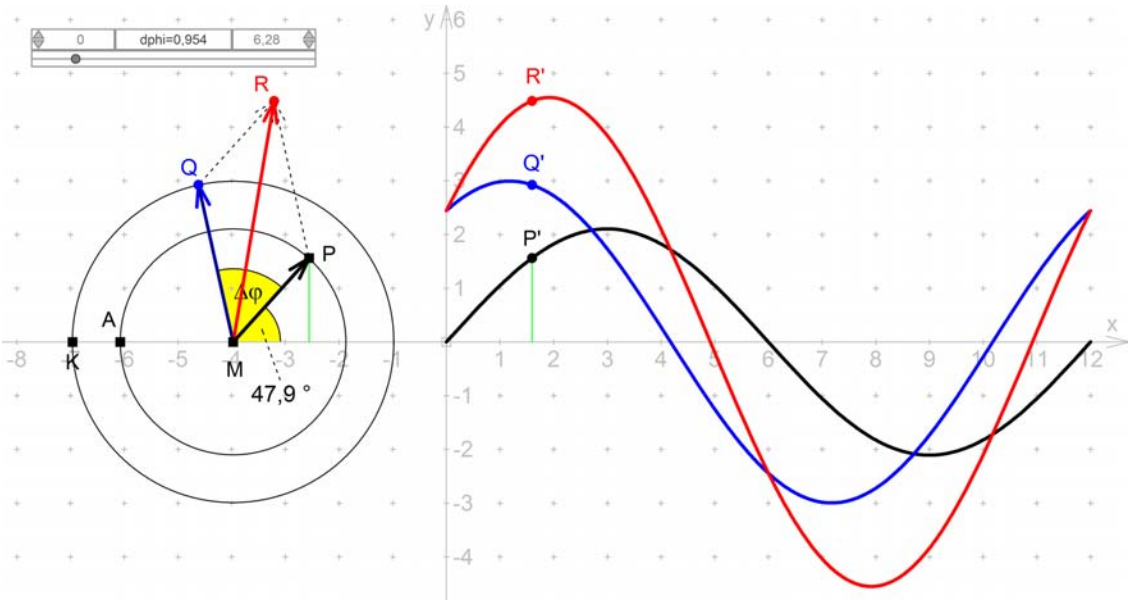


Abb. 3

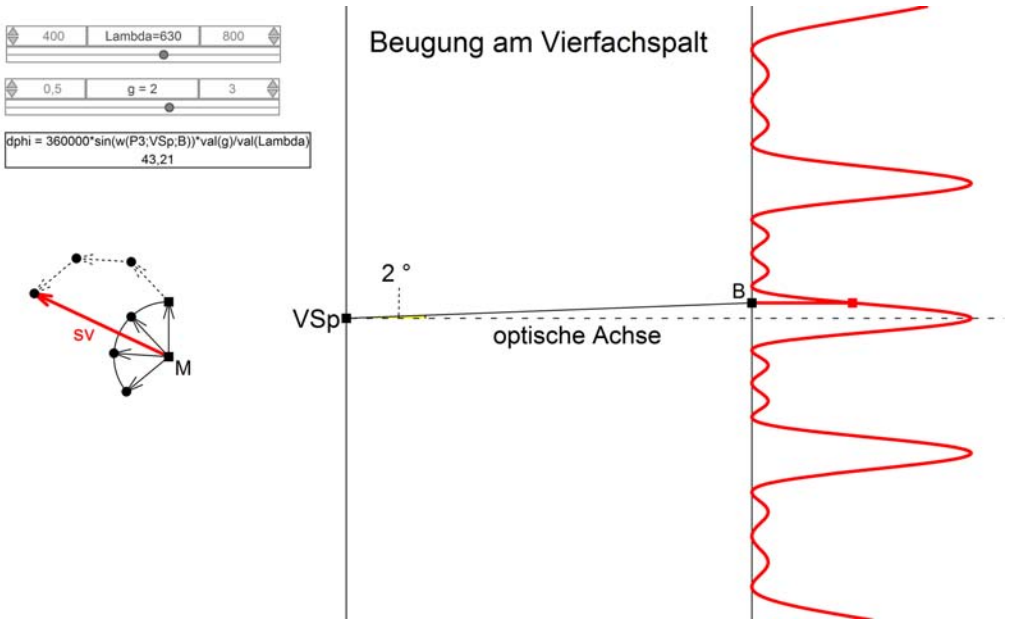


Abb. 4

Die resultierende Amplitude der in B ankommenden Lichtwelle erhalten wir nun durch Vektoraddition: die vier von M ausgehenden Vektoren repräsentieren die von den einzelnen Spalten ausgesandten Elementarwellen. Dabei kann die Richtung des ersten (!) Vektors beliebig gewählt werden; wichtig ist nur, dass der Winkel zwischen den einzelnen Vektoren die jeweils passende Phasenverschiebung „dphi“ ist. Mit Hilfe der gestrichelt gezeichneten Kopien von drei der Vektoren wird nun die Vektoraddition durchgeführt, was den Summenvektor $\overrightarrow{sv}$ ergibt. Schließlich ist die in B zu beobachtende Lichtintensität proportional zum Amplitudenquadrat, also zu $\left|\overrightarrow{sv}\right|^2$. Zu jeder möglichen Lage von B auf dem Schirm ist die so errechnete (relative) Intensität nach rechts aufgetragen. Man sieht den typischen Intensitätsverlauf des Beugungsbilds eines Vierfachspalts, mit drei Dunkelstellen und zwei schwachen Nebenmaxima zwischen benachbarten Hauptmaxima.

Besonders instruktiv ist es nun, den Punkt B auf dem Schirm entlang zu ziehen und sich dabei die Entwicklung der Vektorsumme anzuschauen. Dabei kann man verstehen, wie es zu den Dunkelstellen zwischen den Hauptmaxima kommt, und warum es eigentlich Nebenmaxima gibt.

Damit Sie nun auch in den Genuss des interaktiven Umgangs mit einer dynamischen Zeichnung kommen, sind alle hier verwendeten Konstruktionen unter `www.geometrie-online.de` frei verfügbar. Sie sind mit DynaGeo erstellt, können aber auch ohne dieses Programm studiert werden, wenn Sie Ihrem Browser die Verwendung des Viewers DynaGeoX erlauben bzw. ermöglichen. Die Website enthält noch einige weitere Beispiele zur Wellenoptik und Interferenz, so z. B. das Beugungsbild eines Zehnfachspaltes, welches durchaus schon dem eines optischen Gitters nahe kommt.

Zum Schluss noch eine Anwendung, die auf den Einsatz der „Zeigeraddition“ in der Quantenmechanik hinweist: Wenn Licht von einem Sender S über einen Spiegel zu einem Empfänger E gelangen soll, woher „weiß“ es dann, welchen Punkt des Spiegels es „anpeilen“ muss? Zum genaueren Studium dieser Situation besetzen wir den Spiegel mit vielen Testpunkten, die hier der Einfachheit halber äquidistant gewählt werden. Jeder Testpunkt soll Ausgangspunkt einer reflektierten Elementarwelle sein. Weil wir voraussetzen, dass das Licht sich im Raum vor dem Spiegel jeweils geradlinig ausbreitet, erhalten wir für jeden Testpunkt P_i einen zugehörigen möglichen Lichtweg SP_iE:

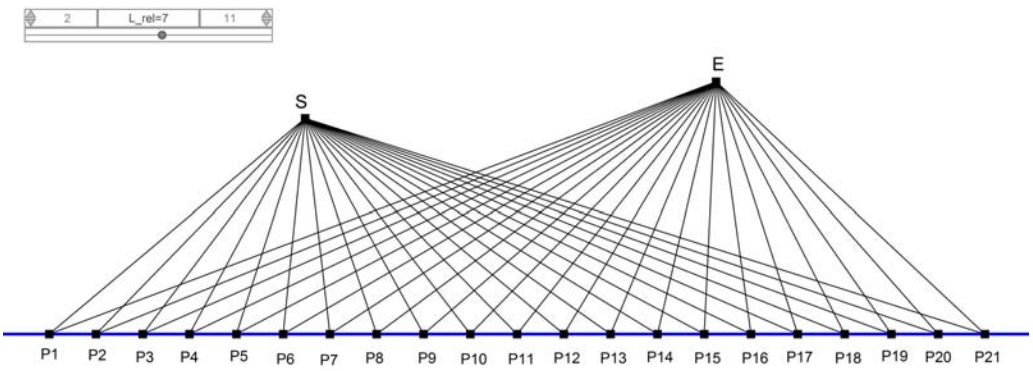


Abb. 5

Analog zum vorigen Beispiel ergibt sich die Amplitude der in E ankommenden Gesamtwelle durch Addition der Vektoren, die zu den auf den einzelnen Lichtwegen verlaufenden Wellenstrahlen gehören. Insgesamt erhält man für diese Vektorsumme das folgende Bild:

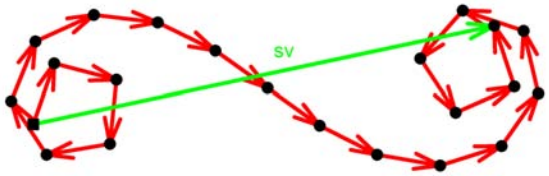


Abb. 6

Man erkennt: Der wesentliche Beitrag zum Summenvektor $\overrightarrow{SV}$ wird von den „mittleren“ Testpunkten geliefert, während die „exotischeren Wege“ über die äußeren Testpunkte nahezu wirkungslos bleiben. Die Amplitude der in E ankommenden reflektierten Welle wird also im Wesentlichen durch wenige Testpunkte auf dem Spiegel bestimmt, und zwar gerade durch diejenigen, die sich dicht bei der durch das klassische Reflexionsgesetz vorausgesagten Stelle befinden. Verwendet man mehr Testpunkte (oder geht man gar mit den Mitteln der höheren Mathematik zu unendlich vielen über), dann erhält man eine schön gerundete „Cornu-Spirale“.

Literaturverzeichnis

[1] R. P. Feynman, *QED — Die seltsame Theorie des Lichts und der Materie*, Piper Verlag, München, Sonderausgabe, 2006.

[2] Quelle des Bildes von R. P. Feynman: www.britannica.com/eb/article-9034161/Richard-P-Feynman.

Enthüllt: Schüler schummelten in Klausuren

Dr. Andreas Pallack
Wissenschaftlicher Referent für Mathematik
Soest, NRW

andreas@pallack.de

Tipps zur Gestaltung von Umfragen in Abiturzeitungen

April — für die Abiturienten 2009 kommen die letzten Sommerferien in bedrohliche Nähe. Nun schaut man sich nach Ausbildungs- oder Studienmöglichkeiten um und realisiert langsam, dass die Schulzeit tatsächlich endlich ist (abire[lat] = abgehen). April ist auch die Zeit, in der nicht ganz ernst gemeinte Fragen gesammelt werden, um z. B. die Miss und den Mister Oberstufe 2009 zu küren. Diese und andere wichtige Fakten münden schließlich in einem Printprodukt, das als Abiturzeitung betitelt wird.

Zum Ende der Schulzeit wächst auch das Verlangen nach Wahrheit. Doch die Wahrheit ist nicht immer bequem: Fragen, die aus Sicht der Befragten unangenehm sind oder diffamierend wirken, werden deswegen in Abiturzeitungen eher vermieden. Der Grund: Die Befragten antworten nicht unbedingt ehrlich. Es heißt nicht umsonst „in vino veritas“. Frei übersetzt: Manchmal bedarf es eines Kniffs, um an die Wahrheit zu kommen.

„Haben Sie schon einmal geklaut?“ Können Sie auf diese Frage mit einem lauten und ehrlichen NEIN antworten? Was glauben Sie: Wie viele Jugendliche in Ihrem Alter haben schon einmal geklaut? Wie viele würden die gestellte Frage wahrheitsgemäß beantworten? In diesem Beitrag stelle ich Ihnen eine Methode vor, mit der man auch auf solch unangenehme Fragen tragfähige Antworten erhält: Die Random-Response-Technik.

Die Abiturzeitung enthüllt: ...

Mein Abitur ist bereits einige Jahre alt (Abiturjahrgang 1992) und entsprechend sind meine Erinnerungen an diese Zeit ein wenig verblasst. Ich könnte mir aber durchaus vorstellen, dass Fragen wie:

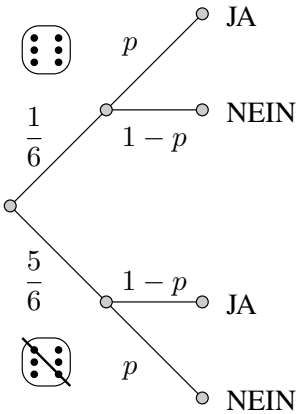
- Hast du schon einmal in einer Klassenarbeit oder Klausur geschummelt (abgeschrieben, Spickzettel benutzt ...)?
- Warst du schon mal in eine Lehrerin / einen Lehrer verknallt? oder
- Hast du schon einmal die Unterschrift deiner Eltern gefälscht (z. B. bei einer Entschuldigung)?

auch heute noch für Abiturzeitungen spannend sind.

Fragt man nur eine Person, so ist es (auch für Statistiker) nahezu unmöglich herauszufinden, ob eine Antwort wahr oder falsch ist. Werden allerdings viele Personen befragt, so kann man sich einer Technik bedienen, um der Wahrheit ein Stück näher zu kommen.

Grundidee ist, dass jeder Einzelne wahrheitsgemäß oder nicht wahrheitsgemäß antwortet, ohne dass der Fragende im Einzelfall entscheiden kann, ob die Antwort der Wahrheit entspricht oder nicht. Eine solche Befragung kann wie folgt ablaufen: Der Interviewer stellt eine Frage. Der Befragte überlegt, ob er die Frage wahrheitsgemäß mit JA oder mit NEIN beantworten müsste, ohne jedoch die Antwort zu sagen. Dann würfelt der Befragte mit einem Würfel und schaut sich das Ergebnis des Wurfs an, ohne dass der Interviewer den Würfel sieht. Wurde eine 6 gewürfelt, soll die Frage wahrheitsgemäß beantwortet werden. In allen anderen Fällen soll der Befragte lügen.

Diese Konstellation ist günstig, wenn JA die unangenehme Antwort ist. In den meisten Fällen werden die Befragten lügen (1, 2, 3, 4 oder 5 fallen eben häufiger als die 6). Das negativ besetzte *gedachte JA* kann deswegen häufig durch ein positiv besetztes *gesprochenes NEIN* ersetzt werden. Die Befragten mit reinem Gewissen werden wohl kein Problem haben, mit JA zu antworten, da das ja gerade das gelogene NEIN ist. Diejenigen, die die Wahrheit sagen sollen, sind durch die vergleichsweise eher geringe Wahrscheinlichkeit des Auftretens der 6 geschützt, da der Interviewer nicht entscheiden kann, ob die jeweilige Antwort der Wahrheit entspricht. Das folgende Baumdiagramm fasst das Gesagte zusammen. Dabei ist p z. B. der relative Anteil der Schummler:



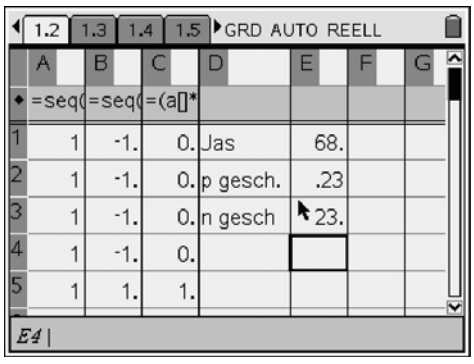
Das war's schon. Nun benötigt man nur noch ein wenig Mathematik, um die Daten einer solchen Erhebung auszuwerten.

Ein erläuterndes Beispiel

Man kann die Daten von Random-Response-Befragungen mit Formeln auswerten. Ich gehe hier jedoch bewusst einen anderen, anschaulicheren Weg: die Analyse mit Hilfe einer Simulation. Für die Simulation werden diverse digitale Werkzeuge, wie z. B. Tabellenkalkulation, Funktionenplotter und Computer-Algebra-System, verwendet. Dafür benutze ich das System TI-Nspire™ CAS, das diese Werkzeuge in sich vereint. Die Software kann unter education.ti.com/deutschland als 30 Tage Testversion kostenlos heruntergeladen werden.

Stellen Sie sich vor: Die Redaktion einer Abiturzeitung möchte wissen, ob und wie viele Schülerinnen und Schüler der 100-köpfigen Jahrgangsstufe schon einmal in einer Klassenarbeit oder Klausur (z. B. durch Abschreiben oder die Verwendung von Spickzetteln) geschummelt haben. Wir gehen nun davon aus, dass wir wissen, dass die wahre Anzahl derer, die bereits einmal geschummelt haben, 20 ist.

Die Befragung dieser Jahrgangsstufe möchte ich nun simulieren. In der folgenden Tabelle steht in Spalte A, ob die Person schon einmal geschummelt hat ($= 1$) oder nicht ($= -1$). Ob die 20 Personen zufällig verteilt sind oder — wie hier — in den ersten 20 Zeilen stehen, spielt dabei keine Rolle (beides ist ohne Beschränkung der Allgemeinheit möglich). In Spalte B wird der Wurf des Würfels simuliert. -1 bedeutet, dass eine 1, 2, 3, 4 oder 5 gefallen ist, 1, dass eine 6 gewürfelt wurde. In Spalte C wird das Produkt von Spalte A und B gebildet, um 1 vermehrt und durch 2 geteilt. Ist der so berechnete Zahlenwert 1, so antwortet die Person mit JA, ansonsten mit NEIN. Um diese Formel zu verstehen, führe man sich vor Augen, dass diejenigen, die schon einmal geschummelt haben ($= 1$), genau dann JA sagen, wenn eine 6 fällt ($= 1$). Diejenigen, die noch nie geschummelt haben ($= -1$), sagen genau dann JA, wenn keine 6 fällt ($= -1$). Die in Spalte C berechnete Größe wird also genau dann 1, wenn die Werte in Spalte A und B identisch sind.



In dieser Simulation gab es 68 JA-Antworten. Diese Zahl ist für unsere Fragestellung aber ohne Aussage: Durch das Befragungsdesign entspricht die Anzahl der JA-Antworten gerade nicht der Anzahl der Personen, die schon einmal geschummelt haben.

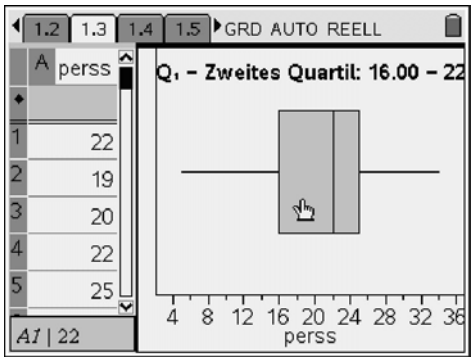
Um die Situation zu analysieren, betrachten wir das Baumdiagramm (s. o.) der möglichen Antworten. Die gesuchte Größe ist p (auch wenn wir sie hier kennen). Der relative Anteil der JA-Antworten ergibt sich zu $p_j = \frac{1}{6}p + \frac{5}{6}(1-p)$. Umformen nach p ergibt: $p = \frac{5}{4} - \frac{3}{2}p_j$.

Als Schätzer für p_j kann das Ergebnis der Simulation verwendet werden. Damit kann die relative Häufigkeit der tatsächlichen JA-Antworten geschätzt werden: $\frac{5}{4} - \frac{3}{2} \cdot \frac{68}{100} = 0,23$. Umgerechnet auf die 100 Schülerinnen und Schüler der Stufe bedeutet das, dass schätzungsweise 23 Schüler schon einmal geschummelt haben. Die wahre Zahl (20) wird durch die Schätzung leicht übertroffen. Das Ergebnis liegt aber in der richtigen Größenordnung.

Knapp vorbei ist auch daneben

Je nachdem wie intensiv Sie sich mit statistischen Fragestellungen auseinandergesetzt haben, wird Sie das Ergebnis 23 überraschen (das passt ja gut!) oder zu weiteren Fragen anregen (kann man diesem Wert vertrauen?). Die wichtigste Frage für die Redaktion einer Abiturzeitung ist sicher: Wie belastbar ist dieses Ergebnis? Könnte es sein, dass die Statistik uns an der Nase herumführt und in Wahrheit niemand geschummelt hat? Vorab: Ja, das ist möglich. Es kann sein, dass die Würfel so gefallen sind, dass die 100 schummelfreien Schülerinnen und Schüler der Jahrgangsstufe genau 68-mal JA sagten und das Ergebnis *ungefähr 23 haben es getan* eine Unterstellung ist. Es ist eben nur sehr unwahrscheinlich.

Um zu untersuchen, wie stabil die Schätzung ist, habe ich die Simulation 50-mal wiederholt. Das Ergebnis wird hier mit Hilfe eines Boxplots dargestellt.



Man erkennt, dass die Ergebnisse der Simulationen um den wahren Mittelwert schwanken. Jedoch zeigt der Boxplot auch, dass es Simulationsdurchgänge gab, die sehr kleine (z. B. 5) bzw. recht große (z. B. 33) Ergebnisse lieferten. Aussagen wie: „23 haben geschummelt" sind deswegen zu vermeiden. Sie sind im Allgemeinen nicht tragfähig. Der Schätzwert muss — im gleichen Atemzug wie er genannt wird — relativiert werden. Man kann noch nicht einmal mit 100 % Sicherheit sagen, ob überhaupt geschummelt wurde. Das ist jedoch

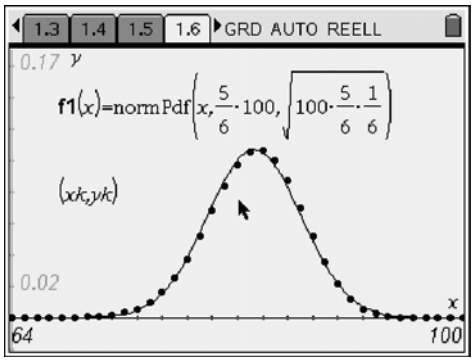
kein Beinbruch: In der Statistik kann man häufig nur Aussagen treffen, die im optimalen Fall *mit an Sicherheit grenzender Wahrscheinlichkeit* zuverlässig sind.

Die Abiturzeitung enthüllt immer noch …

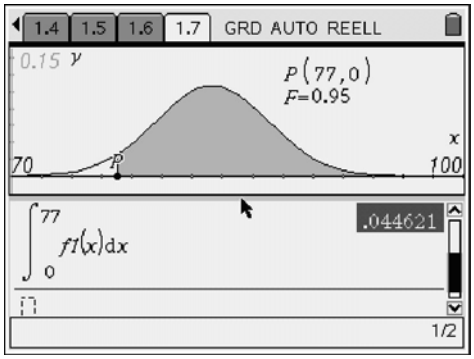
Für eine Schlagzeile ist es — aus meiner Sicht — häufig uninteressant, ob 10%, 20% oder nur einer geschummelt hat. Der Fakt, dass es passiert ist, ist entscheidend. Also untersuchen wir nun einmal, wie wahrscheinlich es bei einem bestimmten Ausfall des Experiments ist, dass niemand geschummelt hat.

Wenn die gesamte Jahrgangsstufe noch niemals geschummelt hat, sagen die Schülerinnen und Schüler genau dann JA, wenn keine 6 gewürfelt wurde. Die Wahrscheinlichkeit beim 100-fachen Wurf eines Würfels genau k mal nicht die 6 zu erhalten, berechnet sich aus $p(k) = \binom{100}{k} \cdot \left(\frac{5}{6}\right)^k \cdot \left(\frac{1}{6}\right)^{100-k}$.

Diese Verteilung kann man durch die Normalverteilung annähern:



Mein Ziel ist es nun, eine Grenze für die Anzahl der JA's zu bestimmen, bei der man davon ausgehen kann, dass mit 95 % Sicherheit mindestens einer der Befragten schon mal geschummelt hat. Dazu müssen die Einzelwahrscheinlichkeiten aufsummiert werden. Das erledigt hier ein Integral:



Die 0,95 = 95 % wird bei 77 Ja-Antworten überschritten. Die oben diskutierten 68 JA-Antworten können als mit an Sicherheit grenzender Wahrscheinlichkeit so interpretiert werden, dass Schülerinnen und Schüler der Jahrgangsstufe schon einmal geschummelt haben: Die Schlagzeile ist gerettet!

Tipps für Ihre Abiturzeitung

Natürlich sind die hier vorgestellten Fragen als (z.T. nicht ganz ernst gemeinte) Vorschläge zu verstehen. Sie sollten selbst entscheiden, welche Fragen für Ihre Stufe von Relevanz sind, und erwägen, ob das vorgestellte Verfahren für Sie nützlich sein könnte.

Natürlich wird Ihre Stufe (zumindest mit großer Wahrscheinlichkeit) nicht aus genau 100 Schülerinnen und Schülern bestehen. Damit Sie die Simulationen trotzdem ohne großen Aufwand selbst durchführen können, habe ich unter `www.pallack.de/random` eine (TI-Nspire™) Datei hinterlegt, bei der Parameter, wie zum Beispiel die Anzahl der Personen, variiert werden können. Mit dieser Datei können Sie eigene Fragestellungen analysieren und Simulationen durchführen.

Das Verfahren selbst kann auch vielfältig variiert werden: Statt eines Würfels kann man Karten verwenden, auch die Verwendung von zwei oder mehr Würfeln ist denkbar. Jedoch: Mit sinkender Wahrscheinlichkeit des für die Untersuchung relevanten Ereignisses steigt die Hemmschwelle, wahrheitsgemäß zu antworten. Auch sollte man beachten, dass die Zuverlässigkeit der Schätzung unmittelbar mit dem verwendeten Zufallsgerät und der Antwortanweisung zusammenhängt. Wählt man z.B. als Antwortanweisung WAHRHEIT (wenn die 6 nicht fällt) und JA (wenn die 6 fällt), so lässt sich die Unsicherheit des Ergebnisses gegenüber dem vorgestellten Verfahren verringern. Vertauscht man die Anweisung (WAHRHEIT (wenn die 6 fällt) und JA (wenn die 6 nicht fällt)), so steigt die Unsicherheit.

Bevor Sie starten, sollten Sie (z.B. mit Hilfe der Simulation) prüfen, ob die Verwendung der Random-Response-Technik bei Ihrer Jahrgangsstufe überhaupt Sinn macht, oder ob die relativen Schwankungen zu groß bzw. die gewonnenen Aussagen mit zu großer Unsicherheit belegt sind. Dazu müssen Sie die Anzahl der Personen mit der gesuchten Eigenschaft schätzen sowie das Zufallsgerät und die Antwortanweisung festlegen. Hilfreich ist dabei auch ein Artikel von Jörg Meyer, der sich detailliert mit der Analyse der Streuungen beim Einsatz von Methoden der Random-Response-Technik auseinandersetzt. Sie finden diesen ebenfalls auf der oben genannten Internetseite.

Abschließend möchte ich darauf hinweisen, dass das hier vorgestellte Modell davon ausgeht, dass alle Befragten bereit sind ehrlich zu antworten. Das ist nicht immer der Fall. Bei professionellen Befragungen werden deswegen häufig Modelle verwendet, die Saboteure berücksichtigen. Diese Modelle werden jedoch sehr schnell recht komplex. Aber auch durch geschickte Settings von Fragen lässt sich die Zuverlässigkeit der Messung erhöhen. Sie sollten deswegen abwägen, ob und inwiefern das vorgestellte Verfahren für Ihre Anwendung verfeinert werden muss.

Ich wünsche Ihnen viel Erfolg für die verbleibende Schulzeit, die Abiturprüfungen und natürlich viel Freude beim Gestalten und Lesen Ihrer persönlichen Abiturzeitung.

Primzahltests und Primzahlrekorde

Prof. Dr. Günter M. Ziegler
Institut für Mathematik
Technische Universität Berlin
Straße des 17. Juni Nr. 136
10623 Berlin

ziegler@math.tu-berlin.de

Die letzten Jahre haben uns eine Serie von neuen Primzahl-Rekorden beschert. So wurde im November 2005 von F. Bahr, M. Boehm, J. Franke, T. Kleinjung das RSA-640 Entschlüsselungsproblem gelöst: die Faktorisierung einer 193-stelligen Dezimalzahl. Im September 2006 wurde die bisher größte bekannte Primzahl gefunden, die Mersenne-Zahl

$$M = 2^{32.582.657} - 1,$$

mit insgesamt 9.808.358 Stellen. Möglicherweise stehen wir damit ganz kurz vor der Entdeckung einer Primzahl mit mehr als zehn Millionen Stellen, worauf ein Preisgeld von 100.000 US$ ausgesetzt ist.

Mersennesche Zahlen

Seit Januar 1996 läuft im Internet eine Suche nach immer größeren Mersenneschen Primzahlen. In dem verteilten Rechenprojekt unter dem Titel GIMPS („Great Internet Mersenne Prime Search“, www.mersenne.org), können Freiwillige übers Internet die GIMPS-Computerprogramme abrufen und „ihre“ Zahlen zum Testen zugeteilt bekommen, ihre PCs damit Sklavenarbeit leisten lassen, und die Rückmeldung übers Internet abliefern.

Marin Mersenne, 1588 – 1648 (Quelle: www-groups.dcs.st-and.ac.uk/~history/PictDisplay/Mersenne.html)

Zur Erinnerung: zu Ehren des französischen Mönches Marin Mersenne (1588 – 1648) heißen die Zahlen der Form $M_n = 2^n - 1$ *Mersennesche Primzahlen* — wenn sie prim sind. Dafür ist notwendig (schöne Übungsaufgabe aus der elementaren Zahlentheorie), dass n selbst prim ist. Aber hinreichend ist das nicht: $n = 11$ liefert das erste Gegenbeispiel. Im Jahr 1644 behauptete Mersenne, dass M_n für $n = 2, 3, 5, 7, 13, 17, 19, 31, 67, 127$ und 257 prim sei, aber keine andere Primzahl unter 257 (womit er exakt fünfmal danebengelegen hat).

Mersennesche Primzahlen sind ziemlich selten: Man weiß nicht, ob es unendlich viele Mersennesche Primzahlen gibt, man kennt inzwischen die ersten 39 und nur fünf weitere, darunter die neu gefundene $M_{32.582.657}$, die derzeit auch die größte bekannte Primzahl ist.

Dass man Zahlen mit fast zehn Millionen Stellen effektiv auf Primalität testen kann, ist die eigentliche wissenschaftliche (und programmiererische) Höchstleistung hinter dem neuen Rekord — dass $n = 32.582.657$ prim sein muss, ist ja nur eine klitzekleine Aufwärmübung für den neuen Rekord.

Primalitätstests

Nun weiß man seit Kurzem, dass es exakte Primzahltests gibt, die in Polynomzeit laufen — siehe [1]. Diese stellen einen theoretischen Durchbruch dar, sind aber für den Einsatz in der Praxis (noch) nicht geeignet. Im GIMPS-Projekt wird für jedes prime n eine Kaskade von klassischeren Tests durchlaufen, die unter www.mersenne.org/math.htm sehr schön und kapierbar beschrieben

werden. Zur algorithmischen Primzahltheorie empfehlen mir Experten das Buch [2]. Aus Computeralgebra-Perspektive finden sich Primzahltests (und sehr viel mehr Spannendes) in [3]. In *Phase I* sucht man nach kleinen Primteilern q von $2^n - 1$. Diese müssen (wieder eine hübsche Übungsaufgabe) $q \equiv 1 \bmod 2n$ und $q \equiv \pm 1 \bmod 8$ erfüllen. Mithilfe eines auf solche Faktoren zugeschnittenen „Sieb des Eratosthenes“ werden dann Primteiler von M_n bis ca. 40.000 erkannt. Dabei kann ausgenutzt werden, dass Teilbarkeitstests für Zahlen vom Typ $2^n - 1$ in Binärarithmetik sehr effektiv durchgeführt werden können.

In *Phase II* wird dann ein Spezialfall der sogenannten $(p-1)$-Methode von Pollard (1974) verwendet, mit der man Faktoren $q = 2kn + 1$ finden kann, für die $q - 1 = 2kn$ aus vielen kleinen Primfaktoren besteht, oder aber (in einer verbesserten Version) bis auf einen etwas größeren Primfaktor stark zusammengesetzt ist: Wenn man q sucht, so dass alle Primfaktoren kleiner als B sind, so bildet man dafür das Produkt $E := \prod_{p<B} p$ aller Primzahlen, die kleiner als B sind, und berechnet dann $x := 3^{E2n}$. Im ggT von $x - 1$ und $2^n - 1$ fängt man dann den gesuchten Teiler von $2^n - 1$.

www.mersenne.org

Erst in *Phase III* verwendet man dann ein Verfahren, mit dem man sicher entscheiden kann, ob $2^n - 1$ prim ist, den sogenannten Lucas-Lehmer-Test (1878, 1930/1935) für Mersenne-Zahlen: M_n ist genau dann prim, wenn $\ell_{n-1} \equiv 0 \bmod M_n$ gilt, wobei die ℓ_k durch $\ell_1 = 4$ und $\ell_n = \ell_{n-1}^2 - 2$ rekursiv definiert werden. Um das effektiv zu berechnen, muss man riesige Zahlen schnell modulo $2^n - 1$ quadrieren. Dazu werden die Zahlen in große Blöcke unterteilt, und dann arbeitet man mit Spezialversionen einer schnellen Fouriertransformation („Fast Fourier Transform“, FFT), in diesem Fall mit einer FFT bezüglich einer irrationalen Basis, die von Richard Crandell und Barry Fagin (*Mathematics of Computation* 1994) eingeführt wurde. Auf einer der WWW-Seiten des *Mathematica*-Projekts mathworld.wolfram.com, die die aktuelle Rekordmeldung verbreiten, wird suggeriert, GIMPS würde mit einer *Mathematica*-Implementierung arbeiten, aber das ist eine arge Dehnung der Tatsachen. (Es hat nur Crandall die Methode auch für die Primzahltests von *Mathematica* implementiert.) In der Tat arbeitet GIMPS mit hochoptimiertem Assembler-Code, aus Prozessorarchitekturgründen in Gleitkommaarithmetik, deren Fehler getrennt erkannt und aufgefangen werden müssen.

Primalität und Faktorisierung

Phasen I und II des GIMPS-Verfahrens spucken also im Fall von zusammengesetztem M_n wirklich Teiler aus — *wenn* sie welche finden —, die dritte und entscheidende Phase aber nicht mehr. Die Antwort heißt da dann nur noch „zusammengesetzt!“, ohne einen expliziten (Prim-)Teiler als Beweis. Es wird also ein Primalitätstest durchgeführt, aber kein vollständiges Faktorisierungsverfahren.

Und das ist auch gut so: Nicht einmal für den Spezialfall von Mersenne-Zahlen kennt man effektive Verfahren zum Faktorisieren. Ein Verfahren, mit dem man *beliebige* Zahlen mit ein paar Hundert Stellen faktorisieren könnte, wäre interessant und bedrohlich, weil die kryptographischen Verfahren, die die Sicherheit von Online-Banking und Internet garantieren sollen, darauf beruhen, dass das Faktorisieren und verwandte Probleme (wie die Berechnung von „diskreten Logarithmen“) offenbar schwer sind.

RSA

Ein Beispiel dafür ist das von Ron Rivest, Adi Shamir und Leonard Adleman 1978 publizierte Verschlüsselungsverfahren „mit öffentlichen Schlüsseln“, das sich inzwischen in fast jedem elementaren Zahlentheorie-Lehrbuch findet, gleichzeitig aber auch in der Praxis vielfältig zum Einsatz kommt — siehe die Homepage www.rsa.com der Firma von Rivest, Shamir und Adleman.

Die Sicherheit des Verfahrens gegen unerlaubtes Entschlüsseln hängt davon ab, dass es mit heutiger Technologie sehr schwer ist, Produkte von Zahlen mit 150 – 200 Stellen in ihre Primfaktoren zu zerlegen. Die Firma „RSA Securities“ hatte sogar Preise von insgesamt über 630.000 US$ auf Beispielprobleme (de.wikipedia.org/wiki/RSA_Factoring_Challenge) ausgesetzt.

Für das Faktorisieren der Zahl „RSA-576“ hat Jens Franke von der Universität Bonn im Dezember 2003 ein Preisgeld von 10.000 US$ kassiert. Im November 2005 konnte er in Teamarbeit die Faktorisierung von RSA-640 vermelden. Dabei ging es um die Zahl

3107418240490043721350750035888567930037346022842727545720161948823206
4405180815045563468296717232867824379162728380334154710731085019195485
2900733772482278352574238645401469173660247765234609

mit 193 Dezimalziffern bzw. 640 Binärziffern (bits). Dafür gab es 10.000 US$. Diese Zahl hat die Faktoren

1634733645809253848443133883865090859841783670033093231218111085238933310010450815121211816751157 9

und

$$1900871281664822113126851573935413975471896789968515493666638539088027103802104498957191261465571$$

(mit je 97 Ziffern), und die sind prim — was wiederum mit den aktuellen Methoden ganz leicht zu zeigen ist. Franke verwendete dabei das „General Number Field Sieve (GNFS)“. Dieses wurde von Lenstra, Lenstra, Manasse & Pollard 1990 eingeführt, und hat eine Laufzeit von $\exp\left(O\left(\sqrt[3]{n \log n}\right)\right)$ für n-stellige Zahlen; es ist also nicht ganz polynomial, aber *fast*. Unter Verwendung des GNFS wurden auch schon die kleineren Testprobleme von RSA-100 bis RSA-512 geknackt.

Im Mai 2006 hat RSA Secturity den „RSA Factoring Challenge“ als beendet erklärt. Die Schwierigkeit des Problems sei inzwischen hinreichend geklärt — und dies, obwohl es ja bisher keinen *Beweis* gibt, dass das Faktorisieren von Zahlen praktisch oder theoretisch wirklich schwer ist.

Die weiteren Preise des „RSA Factoring Challenge“ werden also nicht mehr ausgezahlt: die reichten ursprünglich bis zu 20.000 US$, für die Faktorisierung des Testproblems RSA-2048.

Das hat allerdings Jens Franke et al. nicht aufgehalten: Mit Hilfe von Supercomputern in Bonn, an der EPFL Lausanne und am NTT in Japan gelang ihnen die vollständige Faktorisierung von $M_{1039} = 2^{1039} - 1$, also einer Zahl mit 1039 bits (die allerdings kein RSA-Testproblem war).

Und es gibt noch mehr aktuelle Rekorde, die sich ebenfalls aufs Faktorisieren beziehen: Unter anderem versucht man eben Mersenne-Zahlen nicht nur auf Primalität zu untersuchen, sondern auch vollständig in Primfaktoren zu zerlegen. So will man im „Cunningham project“ (homes.cerias.purdue.edu/~ssw/cun/) die Zahlen der Form $b^n \pm 1$ für $b = 2, 3, 5, 6, 7, 10, 11$ und 12 bis zu hohen Werten von n vollständig faktorisieren. Als Spezialfälle enthält dies die Mersenne-Zahlen $M_n = 2^n - 1$, die nur für primes n selbst Primzahlen sein können, und die Fermat-Zahlen $F_n = 2^{2^n} + 1$. (Übungsaufgabe: Eine Zahl $2^m + 1$ kann nur dann prim sein, wenn m eine Zweierpotenz ist.) Soweit wir wissen, ist F_n nur für $n = 0, 1, 2, 3$ und 4 prim. Euler selbst hat gezeigt, dass $F_5 = 4294967297$ durch 641 teilbar ist. Das verteilte Internet-Projekt NFSNET (www.nfsnet.org) ist dabei extrem erfolgreich, und liefert fast jeden Monat einen neuen Eintrag in die Ergebnisliste. Die Erfolge basieren dabei auf dem „Special Number Field Sieve (SNFS)“ — einer schnelleren Spezialversion des GNFS, die nur für spezielle Zahlen, eben etwa vom Typ $b^n \pm 1$, anwendbar ist.

Rekordjagd

Die Rekordjagd geht weiter. Die „Electronic Frontier Foundation“ (www.eff.org/) hat schon im Jahr 2000 einmal 50.000 US$ für die erste Primzahl mit einer Million Stellen ausgezahlt. Für die Identifikation einer Primzahl mit mehr als 10 Millionen Dezimalstellen hat sie 100.000 US$ ausgesetzt. Dies heizt die Stimmung an, und das GIMPS-Projekt sucht Mitstreiter, die ihre Computer für die Rekordjagd einsetzen wollen.

Viele arme kleine PCs werden also mit Zahlen gefüttert und mit Primzahltests und mit Zerlegungsverfahren gequält werden, nur damit Herrchen vielleicht einen Teil des Ruhms (und des Preisgeldes) einkassieren kann.

Literaturverzeichnis

[1] F. Bornemann, *Ein Durchbruch für „Jedermann“*, in Computeralgebra-Rundbrief Nr. 32, 2003, S. 8 – 14.

[2] R. Crandall und C. Pomerance, *Prime Numbers — A Computational Perspective*, Springer-Verlag, New York, 2001.

[3] J. von zur Gathen und J. Gerhard, *Modern Computer Algebra*, Cambridge University Press, 2. Auflage, Cambridge, 2003.

Fehler korrigierende Codes

Prof. Dr. Felix Ulmer
Institut de Recherche Mathématiques de Rennes
IRMAR, UMR 6625 du CNRS
Université de Rennes 1
Campus de Beaulieu
F-35042 Rennes, Frankreich

perso.univ-rennes1.fr/felix.ulmer

Fehler bzw. Fälschungen erkennen

Codes gibt es im elektronischen Zeitalter fast überall: CD's (siehe den Artikel von van Lint auf Seite 53), Festplatten, digitale Bilder, ...

Die Rückseite eines Euroscheins enthält ein Codewort der Länge 12: Links einen Buchstaben, gefolgt von 11 Ziffern (z.B. U24263273615). Der Buchstabe gibt Aufschluss über das Herkunftsland (X für Deutschland, U für Frankreich, ...). Wenn man den Buchstaben durch seine Reihenfolge im Alphabet ersetzt (also $X = 24, U = 21$), muss die so gewonnene Zahl z (z.B. 2124263273615) stets Rest 8 bei der Division durch 9 ergeben.

Die Zahlen $z+1$ und $z-8$ sind genau dann durch 9 teilbar, wenn die Summe ihrer Ziffern durch 9 teilbar ist. Dieser Prozess kann *rekursiv* angewendet werden.

Fehler korrigieren: Der Zaubertrick

Der Zauberer *Hamming* bittet einen Zuschauer, sich eine ganze Zahl Z mit $0 \leq Z \leq 15$ auszudenken. Er stellt dem Zuschauer nun 7 Fragen. Bei der Beantwortung der Fragen darf der Zuschauer höchstens einmal lügen, muss aber nicht.

Frage 1 = ist $Z \geq 8$?
Frage 2 = ist Z in der Menge $\{4,5,6,7,12,13,14,15\}$?
Frage 3 = ist Z in der Menge $\{2,3,6,7,10,11,14,15\}$?
Frage 4 = ist Z eine ungerade Zahl ?
Frage 5 = ist Z in der Menge $\{1,2,4,7,9,10,12,15\}$?
Frage 6 = ist Z in der Menge $\{1,2,5,6,8,11,12,15\}$?
Frage 7 = ist Z in der Menge $\{1,3,4,6,8,10,13,15\}$?

Die Schwierigkeit liegt natürlich darin, dass der Zuschauer bei einer beliebigen Frage gelogen haben kann, oder auch nicht. Man kann beweisen, dass es unter diesen Umständen unmöglich ist, diesen Trick mit weniger als 7 Fragen durchzuführen.

Nun soll $f_i = 0$ sein, wenn die Antwort auf die i-te Frage Nein ist, und $f_i = 1$, wenn die Antwort auf die i-te Frage Ja ist. Hamming berechnet nun folgende Zahlen modulo 2 (das heißt, er teilt die gewonnenen Zahlen durch 2 und betrachtet den Rest):

$$\begin{aligned} b_1 &= f_4 + f_5 + f_6 + f_7 \pmod 2 \\ b_2 &= f_2 + f_3 + f_6 + f_7 \pmod 2 \\ b_3 &= f_1 + f_3 + f_5 + f_7 \pmod 2 \end{aligned}$$

Er bekommt so die binäre Zahl $T = \overline{b_1 b_2 b_3}^2$, also die in Basis zwei geschriebene ganze Zahl

$$T = b_1 \cdot 2^2 + b_2 \cdot 2^1 + b_3 \cdot 2^0 \leq 7.$$

Ist T gleich Null, so wurde nie gelogen, sonst wurde genau bei der Frage Nummer T gelogen. Falls gelogen wurde, so korrigiert man nun die Antwort auf die Frage T. Die ausgedachte Zahl ist die (ggfs. korrigierte) binäre Zahl $Z = \overline{f_1 f_2 f_3 f_4}^2$, also die Zahl $Z = f_1 \cdot 2^3 + f_2 \cdot 2^2 + f_3 \cdot 2 + f_4$.

Beispiel 1 Wenn sich der Zuschauer die Zahl 6 ausdenkt und bei der dritten Frage lügt, so lauten seine 7 Antworten 0100011. Daher ergibt sich $b_3 = 0$, $b_2 = 1$ und $b_1 = 1$. Hamming berechnet $\overline{011}^2 = 3$ und weiß nun, dass bei der dritten Frage gelogen wurde. Er korrigiert die Antworten zu 0110011. Er berechnet nun $\overline{0110}^2 = 6$.

Fehler korrigierende Codes

Hammings Trick liefert ein Mittel, um eine in Basis 2 geschriebene ganze Zahl $\overline{f_1 f_2 f_3 f_4}^2$ (also eine ganze Zahl zwischen 0 und $15 = 2^4 - 1$) so zu übertragen, dass der Empfänger, bei höchstens einem Übertragungsfehler, erkennen kann, ob es einen Fehler gab, und, falls ja, ihn zu korrigieren. Man sendet dafür

statt $\overline{f_1f_2f_3f_4}^2$ die Zahl $w = \overline{f_1f_2f_3f_4f_5f_6f_7}^2$, also eine Zahl $0 \le w \le 255 = 2^8 - 1$. Man beachte, dass die vier Antworten auf die ersten Fragen des Zauberers eigentlich genau die binäre Darstellung der ausgedachten Zahl sind (wenn da nicht gelogen wurde). Die Anzahl der möglichen Übertragungen ist also 256, davon sind jedoch nur 16 Übertragungen korrekt, das heißt, es gibt nur 16 Codewörter in unserem Code der Länge 7.

Der Wiederholungscode

Man kann ein Wort auch dadurch codieren, dass man es mehrmals wiederholt. Schickt man statt 1011 das Codewort 10111011, so schließt man beim Empfang auf einen Fehler, wenn beide Hälften nicht identisch sind. Schickt man statt 1011 nun 101110111011, und wird das Wort 100110111011 empfangen, so korrigiert man diesen einen ersichtlichen Fehler. Wenn bei der Übertragung höchstens einmal *gelogen* wird, so kann man diesen einen Fehler immer korrigieren. Hätten wir im obigen Zaubertrick einen *Wiederholungscode* verwendet, so hätten wir 12 statt 7 Fragen stellen müssen.

Das Prinzip des Korrigierens

Der *Hamming-Abstand* $d(w_1, w_2)$ zwischen zwei Codewörtern w_1 und w_2 misst, in wie vielen Stellen sich zwei Codewörter eines Codes unterscheiden. Wenn der kleinste Abstand zwischen zwei beliebigen Codewörtern $d = 2e + 1$ ist, so lässt sich bei höchstens $d - 1$ Fehler ein fehlerhaftes Wort als solches erkennen und bei höchstens $e = (d-1)/2$ Fehler sogar korrigieren. Hierbei wird als korrektes Wort stets das *am nächsten gelegene* Wort angegeben. Beim Zaubertrick gibt es 16 Codewörter. Die Zahl 6 ergibt dort richtig codiert 0110011, und die Zahl 14 entspricht 1110000. Diese Zahlen haben den Abstand 3, weil sie sich in 3 Stellen unterscheiden. Empfängt man die Zahl 0100011, so stellt man fest, dass diese Zahl kein Codewort ist und ersetzt die Zahl durch das nächstgelegene Codewort 0110011. Der Zaubertrick ist deshalb so interessant, weil er, ohne alle Codewörter heran zu ziehen, direkt korrigieren kann. Zwei verschiedene Codewörter des dreimaligen Wiederholungscodes haben natürlich auch Abstand 3, was beweist, dass ein solcher Code ebenfalls einen Fehler korrigieren kann. Der Hamming-Code ist in dem Sinn optimal, dass man mindestens drei zusätzliche Symbole braucht, also die Länge 7, um eine vierstellige binäre Zahl so zu übertragen, dass man gegebenenfalls einen einzelnen Fehler korrigieren kann. Wenn mehr als ein Fehler passiert, gibt es wahrscheinlich einen Übertragungsfehler, der unentdeckt bleibt.

Mathematik statt Zauberei

Die Welt der Informatik besteht aus Nullen und Einsen. In dieser Welt wollen wir immer noch addieren und multiplizieren können, und dazu begeben wir uns nun in eine Welt $\mathbb{F}_2$, in der $1 + 1 = 0$ sein soll, in der also -1 gleich 1 ist. Konkret bedeutet dies für uns, dass wir immer modulo zwei rechnen: $1 + 1 = 2 = 0 \pmod 2$. Polynome sind die Grundbausteine der *Computer-Algebra*. Deshalb wollen wir nun ein Codewort, wie 0110011, mit einem Polynom in der Variable X identifizieren:

$$0 \cdot X^6 + 1 \cdot X^5 + 1 \cdot X^4 + 0 \cdot X^3 + 0 \cdot X^2 + 1 \cdot X^1 + 1 \cdot X^0$$

Wie üblich schreiben wir einfach nur $X^5 + X^4 + X + 1$, wobei die Koeffizienten des Polynoms hier in $\mathbb{F}_2$ sind und nach den obigen Regeln addiert und multipliziert werden.

Die **Codierung** einer binären Zahl $6 = \overline{0110}^2$, also des Polynoms $X^2 + X$, kann nun (statt vieler Fragen) folgendermaßen erfolgen: Man multipliziert das zu übertragende Polynom dazu immer mit dem selben *Erzeuger-Polynom* $g = X^3 + X + 1$. Die Codierung von 6, also $X^2 + X$, ist dann

$$(X^2 + X)(X^3 + X + 1) = X^5 + X^4 + X^3 + X$$

(Man beachte, dass $X^2 + X^2 = (1 + 1)X^2 = 2 \cdot X^2 = 0 \cdot X^2 = 0$ ist). Die zu übermittelnde Zahl ist also 0111010.

Um das empfangene Wort $w = X^5 + X^4 + X^3 + X$ zu **entschlüsseln**, muss eine Division mit Rest des Polynoms $w = X^5 + X^4 + X^3 + X$ durch $g = X^3 + X + 1$ durchgeführt werden. Dies erfolgt im Prinzip wie bei ganzen Zahlen. Man betrachtet die höchsten Koeffizienten von w und g und überlegt nun, *wie oft geht g in w*, und es geht natürlich X^2 Mal. Daher zieht man X^2 Mal g, also $X^5 + X^3 + X^2$ von w ab und bekommt $w - X^2 \cdot g = X^4 + X^2 + X$ (man beachte, dass -1 gleich 1 ist). Schließlich *geht* X mal g in $w - X^2 \cdot g$ und man bekommt $(w - X^2 \cdot g) - X \cdot g = 0$. Also ist $w = (X^2 + X) \cdot g$, und wir erhalten wieder $X^2 + X$, also $6 = \overline{0110}^2$. Das Codieren entspricht einer Multiplikation und das Erkennen der Nachricht einer Division.

Hätten wir $\widetilde{w} = X^5 + X^4 + X$ statt w empfangen, so hätten wir dank $\widetilde{w} = (X^2 + X + 1) \cdot g + (X + 1)$ erkannt, dass der Rest nicht Null ist und somit ein Fehler aufgetreten ist. Diesen gilt es dann zu korrigieren.

Eine Eigenschaft des *Polynom-Codes* ist, dass die Summe zweier Codewörter (d.h. Polynome) wieder ein Codewort ist: der Code ist ein *linearer Code*. Dies folgt aus der Tatsache, dass die Summe zweier durch g teilbarer Polynome, wieder durch g teilbar ist. Diese lineare Eigenschaft hat zum Beispiel auch der *Hamming-Code*. Die additive Eigenschaft erlaubt nun, schneller

den minimalen Abstand zwischen zwei beliebigen Codewörtern zu ermitteln. Dieser entspricht der minimalen Anzahl von Einsen in einem von Null verschiedenen Codewort. Man spricht vom *Gewicht* des Codewortes, und so ist zum Beispiel das Gewicht von 0111010 einfach 4. Es genügt also, den Abstand zum Nullwort 0000000 zu messen.

Um dies zu beweisen beachte man, dass zwei Codewörter w_1 und w_2 genau dann den Abstand d haben, wenn es d Einsen im Codewort $w_1 - w_2$ gibt (hier werden die Polynome Koeffizientenweise subtrahiert).

Beispiel 2 Da die Codewörter des Polynom-Codes in aufsteigender Reihenfolge 0000000, 0001011, 0010110, 0011101, 0101100, 0100111, 0111010, 0110001, 1011000, 1010011, 1001110, 1000101, 1110100, 1111111, 1100010, 1101001 sind, ergibt sich, dass der kleinste Abstand zwischen zwei Codewörtern 3 ist. Der *Polynom-Code* kann also ebenfalls einen Fehler korrigieren.

Es kann also mehr als nur einen optimalen Code geben.

Unser Polynom-Code ist zudem noch ein *zyklischer* Code: genau dann ist $a_6a_5a_4a_3a_2a_1a_0$ ein Codewort, wenn $a_0a_6a_5a_4a_3a_2a_1$ ein Codewort ist.

Zauberlehrlinge der Mathematik

Zur Fehlerkorrektur des *Polynom-Codes* werden wir nun Mathematik benutzen, deren Begründung in der Regel im Rahmen eines Mathematikstudiums stattfindet. Ist man bereit einige Tatsachen vorab zu akzeptieren, so ist der Zugang jedoch recht einfach.

Als erstes wollen wir akzeptieren, dass in einer Welt, in der $1+1=0$ ist, es immer noch Nullstellen von Polynomen gibt. Falls α eine solche Nullstelle des *Erzeuger-Polynoms* $g = X^3 + X + 1$ ist, so ist $\alpha^3 + \alpha + 1 = 0$ oder, wegen $1 = -1$, auch $\alpha^3 = \alpha + 1$. Die Potenzen α^2 und α können nicht vereinfacht werden. Für höhere Potenzen ergeben sich jedoch folgende Gleichungen:

$$\begin{array}{lcllcl} \alpha^3 &=& \alpha+1 & \alpha^6 &=& \alpha^2+1 \\ \alpha^4 &=& \alpha^2+\alpha & \alpha^7 &=& 1 \\ \alpha^5 &=& \alpha^2+\alpha+1 & & & \end{array}$$

Man kann nachrechnen, dass, in der $1+1=0$ Welt auch α^2 und α^4 Wurzeln von X^3+X+1 sind. In der Tat ist in dieser Welt

$$X^7 - 1 \;=\; \left(X^3+X^2+1\right)(X+1)\left(X^3+X+1\right).$$

In der *Normalen Welt*, in der $1+1=2$ ist, rechnet man nach, dass dies nicht der Fall ist. In der Welt, in der $1+1=0$ ist, passieren also merkwürdige Dinge. Wir wollen nun auch akzeptieren, dass in der neuen Welt $\alpha^i \neq 0$ ist.

Nun möchten wir unsere seltsame Welt benutzen, um einen Übertragungsfehler zu korrigieren. Wichtig hierbei ist, dass alle Codewörter w des Polynom-Codes Vielfache des Erzeuger-Polynoms g sind, und somit die Form $w = h \cdot g$ haben. Da wir annehmen, dass höchstens ein Fehler bei der Übertragung aufgetreten ist, ist das empfangene $\widetilde{w}$ entweder w oder $w + X^i$. Im letzteren Fall ist das richtige Wort eigentlich $\widetilde{w} + X^i$ (immer wegen $1+1=0$). Wir schreiben $\widetilde{w}(X) = h(X) \cdot g(X)$ oder $\widetilde{w}(X) = h(X) \cdot g(X) + X^i$ um die Variable X hervorzuheben. Nun setzen wir die Nullstelle α von g in ein $\widetilde{w}$ ein. Falls $\widetilde{w}(X) = h(X) \cdot g(X)$ ist, so ist $\widetilde{w}(\alpha) = h(\alpha) \cdot g(\alpha) = h(\alpha) \cdot 0 = 0$, und falls $\widetilde{w}(X) = h(X) \cdot g(X) + X^i$, so ist $\widetilde{w}(\alpha) = \alpha^i \neq 0$. Man kann durch Einsetzen also sofort erkennen, ob die Übertragung korrekt oder fehlerhaft ist.

Beispiel 3 (Korrektur von $\widetilde{w} = X^5 + X^4 + X$)
Da $1 \cdot \alpha + 1 \cdot \alpha = 1 \cdot \alpha + (-1) \cdot \alpha = 0$ und $\alpha^2 + \alpha^2 = 0$, gilt

$$\begin{array}{rcl} \widetilde{w}(\alpha) &=& \alpha^5 + \alpha^4 + \alpha \\ &=& \left(\alpha^2+\alpha+1\right) + \left(\alpha^2+\alpha\right) + \alpha \\ &=& \alpha + 1 \end{array}$$

Es gibt also einen Fehler der Form X^i mit $\alpha^i = \alpha + 1$. Unsere obigen Berechnungen zeigen, dass $i = 3$ sein muss, und der Fehler also X^3 ist. Das korrekte Wort ist somit $w = \widetilde{w} + X^3 = \left(X^5 + X^4 + X\right) + X^3 = X^5 + X^4 + X^3 + X$. Nun ergibt sich aus der Division, dass $w = \left(X^2 + X\right) \cdot g$ ist. Daher war das gesendete Wort $X^2 + X$, was der Zahl $6 = \overline{0110}^2$ entspricht.

Die Tricks des Mathematikers

Im Gegensatz zu Zauberern verraten Mathematiker ihre Tricks nur zu gerne, hier ausnahmsweise ohne Beweise. Wichtig beim Polynom-Code ist, dass in der Welt, in der $1+1=0$ ist, das Erzeuger-Polynom g ein Teiler von $X^7 - 1$ ist. Den Abstand 3 zwischen zwei Codewörtern erreicht man dann dadurch, dass die $2 = 3 - 1$ Potenzen α und α^2 Nullstellen von g sind.

Einen Polynom-Code der Länge 15, der bei der Übertragung einer 7-stelligen binären Zahl maximal $2 = (5-1)/2$ Fehler korrigieren kann, erreicht man nun mit dem Erzeuger-Polynom $\widetilde{g} = X^8 + X^7 + X^6 + X^4 + 1$. Wenn $1+1=0$ ist, so ist $\widetilde{g}$ ein Teiler von $X^{15} - 1$, und wenn $\widetilde{\alpha}$ eine Nullstelle des Teilers $X^4 + X + 1$ von $\widetilde{g}$ ist, sind auch die $4 = 5 - 1$ Potenzen $\alpha, \alpha^2, \alpha^3, \alpha^4$ Nullstellen von $\widetilde{g}$. Codieren und Decodieren erfolgen wieder durch Mutiplikation und Division. Das Korrigieren kann man in Beispiel 20.5 in [1] nachlesen.

Literaturverzeichnis

[1] R. Lidl und G. Pilz, *Applied Abstract Algebra*, Springer, Berlin, Heidelberg, New York, 1998.

[2] *Codes correcteurs d'erreurs*, Agrégation externe de mathématiques, 2005, Épreuve de modélisation. agreg.dnsalias.org/Textes/527.pdf

Wie schnell kann man multiplizieren?

Prof. Dr. Bernd Heinrich Matzat
Interdisziplinäres Zentrum für Wissenschaftliches Rechnen
der Universität Heidelberg
Im Neuenheimer Feld 368
69120 Heidelberg

matzat@iwr.uni-heidelberg.de

Wenn ein Mensch oder ein Computer schnell rechnen können soll, ist es vordringlich, dass er die Grundoperationen Addition, Subtraktion, Multiplikation und Division sehr schnell ausführen kann. Bevor man sich jedoch mit dazu geeigneten Rechenverfahren beziehungsweise Algorithmen beschäftigt, muss man sich über die Darstellung von Zahlen Gedanken machen. In diesem Artikel geht es nur um ganze Zahlen bzw. natürliche Zahlen (mit Vorzeichen). Diese sind uns üblicherweise in der dekadischen Schreibweise, d.h. im Zehnersystem, geläufig. Die aktuelle Jahreszahl zum Beispiel ist

$$2008 \;=\; 8\cdot 1 + 0\cdot 10 + 0\cdot 10^2 + 2\cdot 10^3.$$

Für Computerrechnungen ist zumeist die 2-adische bzw. binäre Darstellung zweckmäßiger, da diese dem 0-1-Schema eines Computers besser angepasst ist. Hier ergibt sich für die aktuelle Jahreszahl

$$11111011000 \;=\; 2^3 + 2^4 + 2^6 + 2^7 + 2^8 + 2^9 + 2^{10},$$

also bereits eine Zahl mit 11 Ziffern. In diesem Artikel werden wir uns auf solche binären Darstellungen

$$\begin{aligned} x &= a_0 1 + a_1 2 + a_2 2^2 + \ldots + a_{m-1}2^{m-1} \\ &= \sum_{i=0}^{m-1} a_i 2^i \quad \text{mit} \quad a_i \in \{0,1\} \end{aligned}$$

(eventuell mit Vorzeichen) beschränken, auch wenn im realen Computer die Koeffizienten in der Regel zu Blöcken (Worten) der Länge 32 oder 64 zusammengefasst sind. Offensichtlich kann die Addition von zwei binären Zahlen der Länge höchstens m mit höchstens $2m$ Operationen mit den Koeffizienten (Bitoperationen) durchgeführt werden, die sich aus der Addition im Binärsystem mit Übertrag ergeben. Entsprechendes gilt auch für die Subtraktion.

Die erste Herausforderung stellt also die Multiplikation dar. Zuerst wollen wir feststellen, wie viele Bitoperationen wir bei der gewöhnlichen Multiplikation von Zahlen $x = \sum_{i=0}^{m-1} a_i 2^i$ und $y = \sum_{j=0}^{n-1} b_j 2^j$ der (Bit)-Länge m bzw. n benötigen. Dazu berechnen wir die Koeffizienten des Produkts z von x und y, also die Koeffizienten c_k in

$$z \;=\; \sum_{k\geq 0} c_k 2^k \;=\; \left(\sum_{i=0}^{m-1} a_i 2^i\right)\cdot\left(\sum_{j=0}^{n-1} b_j 2^j\right)$$

zum Beispiel nach dem folgendem Schema

$$\begin{aligned} z_0 \;&:=\; b_0 x, \quad z_1 \;:=\; z_0 + 2b_1 x, \;\ldots, \\ z \;&=\; z_n \;:=\; z_{n-1} + 2^n b_n x. \end{aligned}$$

Zunächst benötigen wir m Bitoperationen zur Berechnung der Koeffizienten von z_0 (für die Multiplikation der a_i mit b_0). Die Berechnung der z_k erfolgt durch Addition von z_{k-1} zu den um k verschobenen Koeffizienten von $b_k x$, wozu man höchstens $3m$ Bitoperationen benötigt, m für die Multiplikation und $2m$ für die Additionen mit Übertrag. Dies ergibt eine Gesamtzahl von höchstens $3mn$ Bitoperationen für die gewöhnliche Schulmultiplikation.

Ist es denkbar, dass man mit weniger Rechenaufwand auskommt? Hierzu hatte Karatsuba 1962 eine einfache und doch sehr wirkungsvolle Idee. Er ging aus von zwei Zahlen x und y in Binärdarstellung der Länge höchstens $n = 2^k$. Dann lassen sich x und y schreiben in der Form

$$\begin{aligned} x \;&=\; x_0 + x_1 2^{n/2}, \quad y \;=\; y_0 + y_1 2^{n/2} \\ &\text{mit} \quad 0 \;\leq\; x_i,\, y_i \;<\; 2^{n/2}. \end{aligned}$$

Das Produkt $x \cdot y$ ergibt sich dann in der gewöhnlichen Form als

$$x\cdot y \;=\; x_0 y_0 + (x_0 y_1 + x_1 y_0)\, 2^{n/2} + x_1 y_1 2^n$$

bzw. in der modifizierten Form

$$\begin{aligned} x\cdot y \;&=\; x_0 y_0 + C\cdot 2^{n/2} + x_1 y_1 2^n \quad \text{mit} \\ C \;&=\; (x_0 + x_1)\,(y_0 + y_1) - x_0 y_0 - x_1 y_1. \end{aligned}$$

Hier scheint die modifizierte Form zunächst aufwändiger zu sein. Beim näheren Hinsehen entdeckt man aber, dass die beiden Produkte $x_0 y_0$ und $x_1 y_1$ doppelt auftreten, so dass eigentlich nur drei Multiplikationen (statt vier in der gewöhnlichen Form) und einige Additionen ausgeführt werden müssen.

Wie lässt sich die Gesamtzahl der benötigten Bitoperationen berechnen? Dazu verwenden wir einen Trick und bezeichnen mit

$A(k)$ die Anzahl der Bitoperationen für die Addition von zwei Binärzahlen der Länge $n = 2^k$,

$B(k)$ die Anzahl der Bitoperationen für die Karatsuba-Multiplikation von zwei Binärzahlen der Länge 2^k.

Wegen $\frac{n}{2} = 2^{k-1}$ benötigt man für die 3 Multiplikationen in der modifizierten Form $3B(k-1)$ Bitoperationen. Die Addition $(x_0 + x_1)$ sowie $(y_0 + y_1)$ kosten zusätzliche $2A(k-1) = A(k)$ Bitoperationen. Weitere $2A(k)$ Bitoperationen werden für die Berechnung des Ausdrucks C verbraucht. Schließlich können x_0y_0 und $x_1y_1 2^n$ ohne Addition aneinandergefügt werden, so dass zur Berechnung von $x \cdot y$ nur noch die Addition von $C \cdot 2^{n/2}$ und $x_0y_0 + x_1y_1 2^n$ mit $A(k)$ Bitoperationen hinzukommt. Dies ergibt insgesamt die Bilanz

$$B(k) = 3B(k-1) + 4A(k).$$

Nach Voraussetzung ist $\frac{n}{2} = 2^{k-1}$ ebenfalls eine 2-Potenz, so dass auch $B(k-1) = 3B(k-2) + 4A(k-1)$ gilt. Insgesamt erhält man so durch sukzessives Einsetzen die Formel

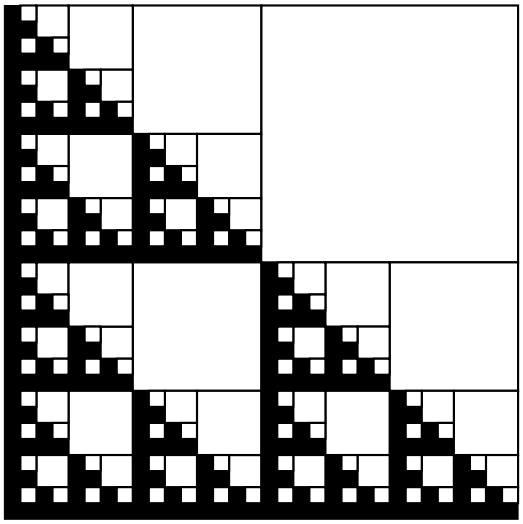
$$\begin{aligned} B(k) &= 3^2 B(k-2) + 3 \cdot 4A(k-1) + 4A(k) \\ &= \ldots \\ &= 3^k B(0) + 3^{k-1} \cdot 4A(1) + 3^{k-2} \cdot 4A(2) \\ &\quad + \ldots + 4A(k). \end{aligned}$$

Nach Obigem kennen wir $A(k) = 2 \cdot 2^k$ und $B(0) = 1$. Damit erhalten wir

$$\begin{aligned} B(k) &= 3^k + 8\left(3^{k-1} \cdot 2 + 3^{k-2} \cdot 2^2 + \ldots + 3^0 \cdot 2^k\right) \\ &= 3^k + 8 \cdot 2^k \left(1 + \tfrac{3}{2} + \left(\tfrac{3}{2}\right)^2 + \ldots + \left(\tfrac{3}{2}\right)^{k-1}\right) \\ &= 3^k + 8 \cdot 2^k \left(1 - \left(\tfrac{3}{2}\right)^k\right)\left(1 - \tfrac{3}{2}\right)^{-1} \\ &= 3^k + 16\left(3^k - 2^k\right) \\ &< 17 \cdot 3^k, \end{aligned}$$

d.h. bei der Karatsuba-Multiplikation natürlicher Zahlen der Bitlänge $n < 2^k$ benötigt man höchstens $17 \cdot 3^k = 17n^{\lg(3)}$ mit $\lg(3) = 1,59$ Bitoperationen (statt $3n^2$ bei der gewöhnlichen Multiplikation).

Was ist dadurch gewonnen? Dies kann man sich am einfachsten am folgenden Schaubild veranschaulichen. Bei jedem Karatsuba-Schritt benötigt man gegenüber der gewöhnlichen Multiplikation nur 3 statt 4 Multiplikationen mit $B(k-1)$ Bitoperationen. Da die zusätzlich benötigten Additionen nur gering ins Gewicht fallen, spart man etwa ein Viertel des Rechenaufwands. Aus obigem Viereck der bei der gewöhnlichen Multiplikation benötigten Bitoperationen kann man also ein Viertel (oben rechts) herausnehmen. Beim zweiten Karatsuba-Schritt gilt dasselbe für die verbliebenen 3 Viertel des Vierecks usw. Bei großem k kann also eine bei weitem überwiegende Anzahl von Bitoperationen eingespart werden. Insbesondere ist der Karatsuba-Algorithmus asymptotisch (für $k \to \infty$) erheblich schneller als die gewöhnliche Multiplikation.



Wie hilft uns diese Erkenntnis beim Rechnen? Hierbei ist zu beachten, dass beim Karatsuba-Algorithmus der Vorfaktor größer und auch die Speicherverwaltung aufwändiger ist. Der Karatsuba-Algorithmus ist also für kleine Zahlen langsamer und für große Zahlen schneller als die gewöhnliche Multiplikation. Für die Anwendung interessant ist nun, ab welcher Größe von Zahlen der Karatsuba-Algorithmus Einsparungen erbringt. Hier hat sich herausgestellt, dass der Karatsuba-Algorithmus in unserem Beispiel mit Worten der Bitlänge 1 schon oberhalb einer Bitlänge 8 (bzw. in einem Computer oberhalb der 8-fachen Wortlänge) schneller ist und in Computeralgebra-Systemen auch eingesetzt wird. Unterhalb der 8-fachen Wortlänge wird dann auf die gewöhnliche Multiplikation umgeschaltet.

Wir haben also gesehen, dass das Multiplizieren zweier Zahlen der Bitlänge n durch geschickte Anordnung der Rechenschritte deutlich schneller als mit der gewöhnlichen Multiplikation ausgeführt werden kann; der Rechenaufwand sinkt von n^2 auf $n^{\lg(3)}$ Bitoperationen (bis auf von n unabhängige Vorfaktoren). Da stellt sich nun natürlich die Frage, ob man durch weitere gute Ideen oder Einsatz von theoretischem Wissen weitere Verbesserungen erzielen kann. Das ist in der Tat möglich. Schönhage und Strassen haben gezeigt, dass mittels diskreter Fouriertransformation und modularer Arithmetik Algorithmen konstruiert werden können, die bis auf einen Vorfaktor den asymptotischen Rechenaufwand $n \lg(n) \lg\lg(n)$ verursachen. Dieser Schönhage-Strassen-Algorithmus ist zumindest asymptotisch nochmals um Klassen besser als der Karatsuba-Algorithmus. Er ist „fast linear“, da in der Formel der Exponent von n Eins ist und die Logarithmus-Faktoren erheblich langsamer wachsen als n-Potenzen. Praktisch kommt dieser Algorithmus bisher aber noch nicht zur Anwendung, da der Übergangspunkt jenseits des zur Zeit für praktische Rechnungen zugänglichen Bereichs liegt.

Leser, die mehr über schnelle Multiplikationen wissen möchten, insbesondere auch über Details des Schönhage-Strassen-Algorithmus, seien z. B. auf das Lehrbuch Modern Computer Algebra von J. von zur Gathen und J. Gerhard verwiesen.

Warum können CAS differenzieren?

Prof. Dr. Wolfram Koepf
Fachbereich Mathematik
Universität Kassel
Heinrich-Plett-Straße 40
34132 Kassel

koepf@mathematik.uni-kassel.de

Zusammenfassung

In diesem Artikel wird aufgezeigt, warum Differenzieren eigentlich gar nicht so kompliziert ist, und zwar, indem wir dem Computeralgebrasystem (CAS) *Mathematica* das Differenzieren beibringen. Nicht, dass dies nötig wäre, denn Differenzieren ist in den meisten CAS bereits eingebaut und selbst in vielen Handhelds verfügbar. Aber es ist hochinteressant zu sehen, wie dies in einem CAS realisiert werden kann. Für meine Präsentation wähle ich *Mathematica*, weil die Realisierung hier besonders einfach ist, aber auch in *Maple*, MuPAD oder REDUCE sieht ein Programm, das differenzieren kann, nicht wesentlich komplizierter aus.

Wir wollen uns zum Ziel setzen, die Differentiation in *Mathematica* zu erklären. Dabei wollen wir natürlich *nicht* die eingebaute Differentiationsprozedur `D` verwenden.

Zunächst mag man überlegen: Da die Ableitung f' einer Funktion f bzgl. der Variablen x als Grenzwert

$$f'(x) = \lim_{h\to 0} \frac{f(x+h) - f(x)}{h}$$

definiert ist, und da *Mathematica* Grenzwerte berechnen kann, programmiert man einfach die Definition und ist fertig. Aber Achtung: Wie berechnet *Mathematica* eigentlich Grenzwerte? Oder anders gefragt: Wie berechnen *Menschen* denn Grenzwerte in der Praxis?

Der obige Ableitungsgrenzwert ist immer von der Form $0/0$, und solche Grenzwerte werden in der Praxis mit Hilfe von Ableitungen berechnet! Dieses Rechenverfahren nennt man die Regel von de l'Hospital. Derartige Methoden zieht auch *Mathematica* zur Grenzwertberechnung heran, so dass die obige Definition der Ableitungsfunktion letztlich offenbar doch den eingebauten Differentiationsmechanismus verwendet, was wir ja vermeiden wollten.

Also überlegen wir von neuem: Wie berechnet man denn Ableitungen in der (schulischen) Praxis? Dies geschieht durch das Anwenden weniger *Regeln* und einiger spezieller Ableitungsresultate. Genau so — wie einem Schüler also — werden wir nun *Mathematica* sukzessiv das Differenzieren (erneut) beibringen.

Bevor wir dies tun, einige generelle *Mathematica*-Eigenheiten:

- Die üblichen mathematischen Symbole wie `+`, `-`, `*`, `/` (für die Division) können verwendet werden. Das Symbol `^` bezeichnet die Potenz. Der Malpunkt kann durch ein Leerzeichen ersetzt werden.
- Alle Funktionen in *Mathematica* werden statt wie üblich mit runden Klammern (wie z.B. `f(x)`) mit eckigen Klammern (also `f[x]`) notiert.
- Alle eingebauten Funktionen werden mit Großbuchstaben aufgerufen (wie `Sin[x]`).
- *Mathematica* erzeugt automatisch Eingabezeilen `In[`n`]` und nach Abschicken der Zeile mit `<Shift> <Enter>` Ausgabezeilen `Out[`n`]`, welche automatisch durchnummeriert werden.

Nun kann es losgehen!

Als erstes behandelt man gewöhnlich die Differentiation der Potenzfunktionen. Dies erklären wir in *Mathematica* durch

```
In[1]:=diff[c_,x_]:=0/;FreeQ[c,x]
In[2]:=diff[x_^n_.,x_]
      :=n*x^(n-1)/;FreeQ[n,x]
```

In der ersten Zeile wird erklärt, dass Konstanten die Ableitung 0 besitzen. Die Regel `diff[c,x]->0` wird nur angewandt, falls die Bedingung `FreeQ[c,x]` erfüllt ist, d.h. falls der Ausdruck `c` das Symbol `x` nicht enthält, c also konstant bzgl. x ist. Bedingungen für die Anwendung von Regeln werden hinter dem Zeichen `/;` angegeben. Dass die angegebene Regel — unter besagter Bedingung — für *beliebige* Terme `c` und `x` Anwendung finden soll, wird durch die Verwendung der Symbolik `c_` und `x_` ausgedrückt. Soll eine Regel nur für ein *bestimmtes* Symbol `y` gelten, verwendet man dagegen keinen Unterstrich `_`.

Schließlich bedeutet das `:=` Symbol eine Zuweisung, die beim *Aufruf* der Funktion ausgeführt wird und daher bei der Definition von Funktionen Verwendung findet. Dass keine augenblickliche Auswertung erfolgt, sieht man auch daran, dass keine Ausgabezeile `Out[`n`]` ausgegeben wird.

In der zweiten Zeile wird also definiert, dass die Ableitung von x^n den Wert nx^{n-1} erhalten soll, und zwar unter der Bedingung, dass n nicht von x abhängt. Mit dem Punkt beim Symbol `n_.` drücken wir aus, dass diese Regel auch noch für $n = 1$ gültig sein soll, wenn also explizit gar kein Exponent vorhanden ist.

Testen wir nun einmal, was *Mathematica* hiermit gelernt hat:

```
In[3]:=diff[y,x]
Out[3]=    0
```

Warum klappt dies scheinbar nicht? Richtig, y hängt ja nicht von x ab! Auch wenn wir das anders gemeint haben sollten. Raten kann *Mathematica* natürlich nicht. Man erhält nun aber

```
In[4]:=diff[x^(1/2),x]
Out[4]=      1/(2√x)
```

Da wir vom Exponenten nicht verlangt hatten, dass er ganzzahlig ist, können nun also bereits beliebige Potenzen abgeleitet werden. Aber beim Beispiel

```
In[5]:=diff[2*x,x]
Out[5]=      diff[2x,x]
```

liefert *Mathematica* als Ausgabe unsere Eingabe ab, da das Programm nach Durchsicht aller Regeln, die wir für `diff` bisher aufgestellt haben, keine gefunden hat, die Anwendung finden könnte. Wir hatten tatsächlich in der zweiten Zeile die Ableitungen lediglich für x^n erklärt, nicht für Vielfache davon. Dem kann aber leicht durch die nächste Regel

```
In[6]:=diff[c_*f_,x_]
      :=c*diff[f,x]/;FreeQ[c,x]
```

abgeholfen werden. Diese Regel gilt nicht nur für Potenzen, sondern für beliebige Terme `f`. Nun kann *Mathematica* das Problem von eben lösen

```
In[7]:=diff[2*x,x]
Out[7]=      2
```

und hat überhaupt gelernt, dass man Konstanten vorziehen kann. Allerdings scheitert es an der einfachen Aufgabe

```
In[8]:=diff[x+x^2,x]
Out[8]=      diff[x+x^2,x]
```

weil wir noch nicht erklärt haben, wie mit einer Summe verfahren werden soll. Da die Differentiation linear ist („Ableitung der Summe gleich Summe der Ableitungen“), erklären wir also

```
In[9]:=diff[f_+g_,x_]
      :=diff[f,x]+diff[g,x]
```

mit dem Resultat, dass nun unsere obige Anfrage

```
In[10]:=diff[x+x^2,x]
Out[10]=      1+2x
```

vereinfacht wird. Obwohl wir die Additivität nur für zwei Summanden definiert haben, wendet *Mathematica* diese Regel nun auch auf mehrfache Summen an, da *Mathematica* weiß, dass die Addition eine assoziative und kommutative Operation ist:

```
In[11]:=
       diff[x+2x^2+3x^3+4x^4+5x^5+6x^6,x]
Out[11]=      1+4x+9x^2+16x^3+25x^4+36x^5
```

Wir wenden uns nun weiteren Ableitungsregeln zu. Bislang können z. B. keine Produkte differenziert werden

```
In[12]:=diff[(2x+x^2)(5+x^2-4x^3),x]
Out[12]=      diff[(2x+x^2)(5+x^2-4x^3),x]
```

Daher teilen wir *Mathematica* nun die Produktregel mit

```
In[13]:=diff[u_*v_,x_]
       :=diff[u,x]*v+diff[v,x]*u
```

und erhalten

```
In[14]:=diff[(2x+x^2)(5+x^2-4x^3),x]
Out[14]=
      (2x-12x^2)(2x+x^2)+(2+2x)(5+x^2-4x^3)
```

Ebenso implementieren wir nun die Quotientenregel.

```
In[15]:=diff[u_./v_,x_]
       :=diff[u,x]*v-diff[v,x]*u)/v^2
```

Nun kann unsere Prozedur `diff` alle gebrochen rationalen Funktionen ableiten, z. B.

```
In[16]:=diff[(2x+x^2)/(5+x^2-4x^3),x]
Out[16]=
      -(2x-12x^2)(2x+x^2)+(2+2x)(5+x^2-4x^3)
      ---------------------------------------
                    (5+x^2-4x^3)^2
```

Bevor wir *Mathematica* nun noch die Kettenregel „beibringen“, wollen wir die Ableitungen der „elementaren“ Funktionen erklären:

```
In[17]:=diff[Log[x_],x_]:=1/x
In[18]:=diff[Sin[x_],x_]:=Cos[x]
In[19]:=diff[Cos[x_],x_]:=-Sin[x]
In[20]:=diff[Tan[x_],x_]:=1/Cos[x]^2
In[21]:=diff[Cot[x_],x_]:=-1/Sin[x]^2
In[22]:=diff[ArcSin[x_],x_]
       :=1/Sqrt[1-x^2]
In[23]:=diff[ArcCos[x_],x_]
       :=-diff[ArcSin[x],x]
In[24]:=diff[ArcTan[x_],x_]:=1/(1+x^2)
In[25]:=diff[ArcCot[x_],x_]
       :=-diff[ArcTan[x],x]
```

Ignorieren Sie diejenigen Funktionen einfach, die Sie nicht kennen. Den natürlichen Logarithmus $\ln x$ nennt *Mathematica* `Log[x]`.

Wenn man sich die obige Liste ansieht, stellt man fest, dass es uns nicht schwerfallen würde, an dieser Stelle auch für weitere Funktionen Ableitungen festzulegen. In *Mathematica* gibt es tatsächlich Hunderte weiterer Funktionen, die ebenfalls differenziert werden können.

Die Produktregel liefert nun beispielsweise

```
In[26]:=diff[Sin[x]*Cos[x],x]
Out[26]=      cos(x)^2-sin(x)^2
```

während der ähnliche Aufruf (teilweise!) scheitert:

```
In[27]:=diff[Sin[x]*Cos[2x],x]
Out[27]=      cos(x)cos(2x)-diff[sin(2x),x]sin(x)
```

Klar: Wir haben ja noch nicht die Kettenregel erklärt. Daher können bislang z. B. auch die Ableitungen

```
In[28]:=diff[Sin[ArcCos[x]],x]
```
$$\text{Out[28]=}\quad \text{diff}\left[\sqrt{1-x^2},x\right]$$
```
In[29]:=diff[(1+x)^n,x]
```
$$\text{Out[29]=}\quad \text{diff}\left[(1+x)^n,x\right]$$

noch nicht gefunden werden, obwohl *Mathematica* beim ersten Beispiel eine automatische Vereinfachung vornimmt und das zweite Beispiel sogar eine gebrochen rationale Funktion darstellt, falls n ganzzahlig ist.

Bei der Darstellung der Kettenregel müssen wir beachten, dass das Muster der Verkettung zweier Funktionen `f[g]` nicht alle verketteten Funktionen auffindet: Auch die Potenz `f^g` (intern: `Power[f,g]`) ist eine verkettete Funktion, die aber nicht als solche erkennbar ist, weil die Funktion `Power[f,g]` *zwei* Argumente hat und somit einem anderen Muster entspricht! Also wird durch die beiden Regeln

```
In[30]:=diff[f_[g_],x_]
       :=diff[f[g],g]*diff[g,x]
In[31]:=diff[f_^g_,x_]
       :=f^g*diff[g*Log[f],x]
```

die Kettenregel implementiert — insbesondere die erste erfordert Geschick und ist etwas trickreich! — und unsere obigen Beispiele liefern nun die Resultate

```
In[32]:=diff[Sin[x]*Cos[2x],x]
```
$$\text{Out[32]=}\quad \cos(x)\cos(2x)-2\sin(x)\sin(2x)$$
```
In[33]:=diff[Sin[ArcCos[x]],x]
```
$$\text{Out[33]=}\quad -\frac{x}{\sqrt{1-x^2}}$$
```
In[34]:=diff[(1+x)^n,x]
```
$$\text{Out[34]=}\quad n(1+x)^{n-1}$$

Jetzt kann *Mathematica* auch die Ableitung der Exponentialfunktion

```
In[35]:=diff[Exp[x],x]
```
$$\text{Out[35]=}\quad e^x$$

angeben, da `Exp[x]` intern durch `E^x` dargestellt wird. Der Vollständigkeit halber hätte man aber besser bei den elementaren Funktionen auch die Ableitung der Exponentialfunktion festhalten können.

Überhaupt kann *Mathematica* nun alle durch algebraische Operationen aus den elementaren Funktionen erzeugten Funktionen differenzieren. Wir betrachten einige Beispiele:

```
In[36]:=diff[x^(x^x),x]
```
$$\text{Out[36]=}\quad x^{(x^x)}\left(x^{x-1}+x^x\log(x)\,(1+\log(x))\right)$$
```
In[37]:=diff[Exp[ArcSin[x^2]],x]
```
$$\text{Out[37]=}\quad \frac{2\,e^{\arcsin\left(x^2\right)}x}{\sqrt{1-x^4}}$$

und schließlich die „Phantasiefunktion“

```
In[38]:=diff[(Exp[x^2+1]+x)
         /(Log[Exp[x]+1]),x]
```
$$\text{Out[38]=}\quad \frac{-\frac{e^x\left(e^{1+x^2}+x\right)}{1+e^x}+\left(1+2e^{1+x^2}x\right)\log\left(1+e^x\right)}{\log\left(1+e^x\right)^2}$$

Schlussbemerkungen

Man beachte, dass wir bei unserer Darstellung niemals die eingebaute Ableitungsfunktion `D` benutzt haben. Ganz im Gegenteil bilden die von uns eingeführten Definitionen im Prinzip ein vollständiges Regelwerk für die Differentiation.

Auf der anderen Seite ist zu beobachten, dass wir *sehr häufig* auf der rechten Seite unserer Funktionsdefinitionen die zu erklärende Funktion `diff` benutzt haben, dies begann bereits in der dritten Definition in Zeile `In[6]`. Die Funktion `diff` erklärt sich also Stück für Stück selbst, indem auf bereits bekannte Definitionsteile verwiesen wird.

Diese Vorgehensweise ist in sehr natürlicher Weise aufgetreten. Man nennt diese wichtige Programmiertechnik *rekursives* Programmieren, s. [2], Kapitel 2.

Ein *Mathematica*-Detail verhindert allerdings, dass die von uns programmierte Funktion `diff` bereits vollständig funktioniert. Der Befehl

```
In[39]:=diff[1/Cos[x],x]
```

liefert beispielsweise eine Fehlermeldung. Grund ist die *automatische Vereinfachung* der Funktion $1/\cos(x)$

```
In[40]:= 1/Cos[x]
```
$$\text{Out[40]=}\quad \sec(x)$$

in die — bei uns in Europa völlig ungebräuchliche — Sekansfunktion. Diese Vereinfachung kann vom Benutzer prinzipiell nicht verhindert werden.

Daher benötigen wir in *Mathematica* (mindestens) zwei weitere Definitionen

```
In[41]:=diff[Sec[x_],x_]
       :=Sin[x]/Cos[x]^2
In[42]:=diff[Csc[x_],x_]
       :=-Cos[x]/Sin[x]^2
```

um die Differentiationsprozedur zu vervollständigen.

Die vorliegende Vorgehensweise wurde in der Publikation [1] zum ersten Mal vorgestellt und in das Lehrbuch [2], Abschnitt 2.7 auf S. 40ff., aufgenommen. Von der Internetseite dieses Lehrbuchs kann man ein *Mathematica*-Notebook, aber auch *Maple*- und MuPAD-Worksheets für eine analoge Vorgehensweise in diesen CAS herunterladen.

Literaturverzeichnis

[1] W. Koepf, *Eine Vorstellung von Mathematica und Bemerkungen zur Technik des Differenzierens*, in Didaktik der Mathematik **21**, 125 – 139, 1993.

[2] W. Koepf, *Computeralgebra. Eine algorithmisch orientierte Einführung*, Springer, Berlin, Heidelberg, 2006, vergleiche www.mathematik.uni-kassel.de/~koepf/CA.

TI-*nspire*™ TECHNOLOGIE

Der neue Blick für das Ganze

Jeder Schüler lernt anders. Einer versteht schneller Formeln, ein anderer Tabellen, ein dritter eher Graphiken.

Die neue TI-Nspire™ Technologie – bestellbar als TI-Nspire™ Handheld (numerische Graphikrechner-Version) und TI-Nspire™ CAS Handheld mit jeweils entsprechender PC-Software – geht auf diese spezifischen Lerngewohnheiten ein. So können Schülerinnen und Schüler im Mathematikunterricht besser und leichter Beziehungen beobachten, Verbindungen herstellen und dokumentieren.

Das Ergebnis: Eine inspirierende Lehr- und Lernerfahrung in Mathematik. Mit einem neuen Blick für das Ganze!

Weitere Informationen zur TI-Nspire™ Technologie und zu unseren Serviceangeboten wie Fortbildungen, kostenloser Ausleihe und vieles mehr erhalten Sie im Internet: **education.ti.com/deutschland**

Oder schreiben Sie uns: ti-nspire@ti.com

TEXAS INSTRUMENTS

Ihre Erfahrung. Unsere Technologie. Mehr Lernerfolg.

Integralrechnung und Computeralgebra

Prof. Dr. Wolfram Koepf
Fachbereich Mathematik
Universität Kassel
Heinrich-Plett-Straße 40
34132 Kassel

koepf@mathematik.uni-kassel.de

Zusammenfassung

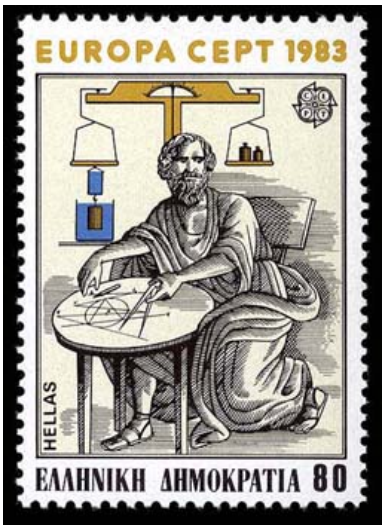
In diesem Artikel wird die historische Entwicklung der Flächen- und Volumenmessung beschrieben, die zum heutigen Integralbegriff führte. Während diese Fragestellungen bereits im Altertum untersucht wurden, ist die Differentialrechnung erst 350 Jahre alt. Aber erst mit Hilfe der Differentialrechnung lassen sich die heute unterrichteten Integrationstechniken gewinnen.

Während die Differentialrechnung vom algorithmischen Standpunkt recht einfach ist — vgl. *Warum können CAS differenzieren?* auf S. 37 — ist das Integrieren oft beschwerlich und kompliziert. Falls man kein Ergebnis findet, weiß man nicht, ob man sich nur ungeschickt angestellt hat oder ob es prinzipiell „nicht geht“ und man also vergeblich nach einer „einfachen“ Integralfunktion sucht.

Es ist weniger bekannt, dass auch diese Fragestellung bereits im 19. Jahrhundert von Liouville untersucht wurde und ebenfalls beantwortet werden kann. Auf dem Satz von Liouville aufbauend hat Risch in den 1960er Jahren einen Algorithmus angegeben, mit dem das Integrieren automatisiert wird und einem Computeralgebrasystem überlassen werden kann.

Flächen und Volumina in der Antike

In der Blüte der griechischen Mathematik wurde die Geometrie auf eine solide Grundlage gestellt. Das gesamte geometrische Wissen wurde auf wenige entscheidende Prinzipien zurückgeführt. Wir sprechen heute von *Axiomen*. Diese Erkenntnisse wurden in den Elementen des Euklid verewigt. Von dieser Basis aus war es vor allem Archimedes (282 – 212 v. Chr.), der die Flächen- und Volumenbestimmung, die die Grundlage für die Integralrechnung darstellt, voranbrachte.



Archimedes von Syrakus

Mit ihrer Beweistechnik der doppelten *reductio ad absurdum* konnten die Griechen zeigen, dass

(a) der Umfang eines Kreises proportional zum Durchmesser ist: $U = \pi\, d = 2\pi\, r$;

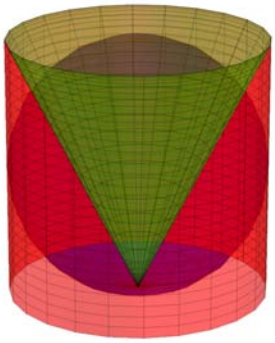
(b) der Flächeninhalt eines Kreises proportional zum Quadrat des Radius ist: $F = \sigma\, r^2$.

Es war die Erkenntnis von *Archimedes von Syrakus*, dass die beiden Proportionalitätskonstanten übereinstimmen: $\sigma = \pi$. Diese Zahl π ist eine der berühmtesten Naturkonstanten.

Archimedes approximierte die Zahl π, indem er den Flächeninhalt eines Kreises von innen und von außen durch den Flächeninhalt regelmäßiger Vielecke annäherte. Durch Betrachtung des regelmäßigen 96-Ecks (!) fand er heraus, dass

$$3{,}140845\ldots = 3\,\frac{10}{71} < \pi < 3\,\frac{1}{7} = 3{,}142857\ldots\,.$$

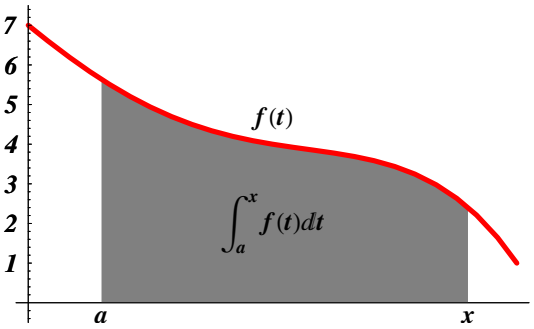
Archimedes war völlig überwältigt von seiner Erkenntnis, dass die Volumeninhalte von Kegel, Kugel und Zylinder bei gleichem Radius (s. folgende Abbildung) im Verhältnis $1 : 2 : 3$ zueinander stehen.



Das Verhältnis der Volumina von Kegel, Kugel und Zylinder

Das Integral als Flächeninhalt

Erst die Einführung des Differentiationskalküls, der im Artikel *Warum können CAS differenzieren?* beschrieben wurde, durch Isaac Newton (1642 – 1727) und Gottfried Wilhelm Freiherr von Leibniz[6] (1646 – 1716) im 17. Jahrhundert und die Erkenntnis, dass Differentiation und Integration zueinander inverse Operationen sind, hat die Integralrechnung wieder einen entscheidenden Schritt vorangebracht.



Das Integral als Flächeninhalt mit variabler oberer Grenze

Wenn für eine nichtnegative Funktion $f(t)$ durch

$$\int_a^b f(t)\,\mathrm{d}t$$

der Flächeninhalt im Intervall $[a, b]$ zwischen dem Graphen von $f(t)$ und der t-Achse bezeichnet wird, so ist die Einführung der *Integralfunktion*

$$\int f(x)\,\mathrm{d}x := \int_a^x f(t)\,\mathrm{d}t$$

mit variabler oberer Grenze x relevant, und der Hauptsatz der Differential- und Integralrechnung besagt, dass

$$\int f'(x)\,\mathrm{d}x = f(x)\ .$$

Diese Erkenntnis wurde zuerst von Newtons Lehrer Isaac Barrows ausgesprochen, der sie aus physikalischen Erwägungen herleitete (wie im übrigen auch Archimedes die Volumenformel für die Kugel!).

Mit dem Hauptsatz wird nämlich aus der *Produktregel*

$$(u(x)\,v(x))' = u'(x)\,v(x) + v'(x)\,u(x)$$

durch Integration sofort die *Regel der partiellen Integration*

$$u(x)\,v(x) = \int u'(x)\,v(x)\,\mathrm{d}x + \int v'(x)\,u(x)\,\mathrm{d}x\ ,$$

eine Integrationstechnik, die sich — bei Kenntnis eines der Integrale — zur Berechnung des zweiten Integrals auf der rechten Seite eignet. Ebenso folgt aus der Kettenregel die *Substitutionstechnik für Integrale*.

Mit diesen neuentwickelten Methoden gelang es Newton, aus seinem *Kraftgesetz* $F = m\,a$ (Kraft = Masse mal Beschleunigung) und dem Gravitationsgesetz über die Anziehungskraft zweier Massen die *Keplerschen Planetengesetze* rein mathematisch herzuleiten! Kepler hatte empirisch beobachtet, dass sich Planeten auf Ellipsen bewegen, wobei sich die Sonne in einem der beiden Brennpunkte befindet, der Radiusvektor eines umlaufenden Planeten in gleichen Zeiten gleiche Flächeninhalte durchläuft und die Quadrate der Umlaufzeiten umgekehrt proportional zu den Kuben der zugehörigen großen Halbachsen sind. Newtons Erfolg zeigt die Mächtigkeit dieser Integrationsverfahren.

Dennoch blieben Fragen offen. Leibniz war natürlich klar, wie man ein Polynom integriert. Er fragte sich, ob sich auch jede gebrochen rationale Funktion elementar integrieren lässt. Hier blieb er sich unsicher, da er die Integralfunktion

$$\int \frac{1}{x^4+1}\,\mathrm{d}x$$

nicht bestimmen konnte. Dies lag daran, dass es ihm nicht gelang, das Nennerpolynom in Faktoren zu zerlegen. Eine Faktorisierung des Nenners in reelle quadratische Faktoren liefert mittels einer Partialbruchzerlegung — einer Darstellung als Summe — sofort das Resultat. Hätte Leibniz mit komplexen Zahlen gerechnet, die damals noch nicht anerkannt waren, so hätte er das Problem lösen können. Die dritte binomische Formel liefert

$$x^4 + 1 = (x^2 + i)(x^2 - i)\ ,$$

wobei i eine Zahl mit der Eigenschaft $i^2 = -1$ ist. Man nennt i die *imaginäre Einheit*. Aus dieser Darstellung kann man mittels der pq-Formel eine Faktorisierung in vier lineare Faktoren bestimmen, die aber ebenfalls die imaginäre Einheit enthalten. Fasst man allerdings jeweils zwei Faktoren wieder geschickt zusammen, so erhält man die reelle Faktorisierung

$$x^4 + 1 = (x^2 + \sqrt{2}\,x + 1)\,(x^2 - \sqrt{2}\,x + 1)\ ,$$

mit welcher sich Leibniz' Fragestellung lösen lässt: Die Integralfunktion lässt sich nun durch Logarithmen (und inverse Tangensfunktionen), also durch „elementare Funktionen", darstellen. Man teste dies mit einem Computeralgebrasystem! Dies fand Leibniz' „wissenschaftlicher Ziehsohn" Johann Bernoulli (1667 – 1748) heraus, der auch feststellte, dass man bei *allen* gebrochen rationalen Funktionen so vorgehen kann. Allerdings wollen wir festhalten, dass das Symbol $\sqrt{2}$, welches in der Eingabe nicht vorkam, für die Darstellung der Integralfunktion unverzichtbar ist.

Der Bernoulli-Algorithmus hat aber ein generelles Problem: Er beruht auf der Erkenntnis, dass sich jedes reelle Polynom in lineare und quadratische reelle Faktoren zerlegen lässt. Leider ist hierfür aber kein *Algorithmus* bekannt, und man kann sogar beweisen, dass

[6] Jurist, Naturwissenschaftler, Politiker, Philosoph, Historiker, Theologe und Diplomat. Er wird von manchen als einer der *letzten Universalgelehrten* bezeichnet.

es einen generellen Algorithmus gar nicht geben kann, falls der Grad des Polynoms größer als 4 ist. Soll ein Computeralgebrasystem jedoch derartige Integrationsprobleme exakt symbolisch lösen können, so müssen wir dies dem System „beibringen“, und hierfür wird ein Algorithmus benötigt, der die Lösung berechnet.

Rationale Integration und der Risch-Algorithmus

Zum Glück kann man aber die Fragestellung modifizieren. In der Regel hat ja die gegebene gebrochen rationale Funktion keine beliebigen reellen Koeffizienten, sondern beispielsweise — wie in unserem obigen Fall — ganzzahlige Koeffizienten. In dieser Situation kann man aber — anders als im „allgemeinen reellen Fall“ — die Integralfunktion algorithmisch bestimmen! Es stellt sich heraus, dass man in diesem Fall die vollständige Faktorisierung des Nennerpolynoms nicht benötigt, sondern eine sogenannte quadratfreie Faktorisierung genügt, welche man leicht algorithmisch bestimmen kann. Dies hat Ostrogradsky 1845 erkannt.

Dieser Algorithmus zerlegt die Integralfunktion jeder gebrochen rationalen Funktion in einen gebrochen rationalen Anteil und einen Teil, welcher als Summe logarithmischer Funktionen dargestellt werden kann, wobei inverse Tangensfunktionen zu den Logarithmen gerechnet werden können. Dieser Algorithmus liefert dann auch beispielsweise automatisch die gesuchte Zahl $\sqrt{2}$, die zur Darstellung der Lösung unseres obigen Beispiels benötigt wird. Die algorithmische Integration gebrochener rationaler Funktionen wird in Kapitel 12 des Lehrbuchs [5] ausführlich behandelt.

Die rationale Integration ist natürlich nur ein Spezialfall. Sehen wir uns die Integration elementarer Funktionen einmal etwas genauer an. Die Integralfunktionen

$$\int e^x \,\mathrm{d}x = e^x \quad \text{und} \quad \int x\, e^{x^2} \,\mathrm{d}x = \frac{1}{2}\, e^{x^2}$$

kann man leicht ausrechnen. Sucht man nun aber nach der Integralfunktion $\int e^{x^2} \,\mathrm{d}x$ und findet hierfür keine Darstellung, so stellt sich die Frage, ob es eine *elementare* Integralfunktion von e^{x^2} überhaupt gibt, also eine Funktion, die sich mit Hilfe rationaler Operationen aus Exponential- und Logarithmusfunktionen darstellen lässt. So lange man auch sucht: Bei diesem Beispiel wird man nicht fündig werden! Dies hat Liouville im Jahr 1835 gezeigt. Eine sehr schöne Darstellung des *Satzes von Liouville* und des Beweises, dass e^{x^2} keine „einfache“ Integralfunktion hat, findet man in [1], Abschnitt 6.6.

Der Satz von Liouville ist viel allgemeiner. Er liefert eine ähnliche Aussage, wie wir am Beispiel der gebrochen rationalen Funktionen bereits kennengelernt haben: Ist die Integralfunktion einer elementaren Funktion $f(x)$ elementar, so besteht sie aus zwei Teilen, einem Anteil, welcher sich rational aus den Funktionen, die in $f(x)$ bereits vorkommen, zusammensetzt, sowie einem Teil, der als Summe von Logarithmusfunktionen dargestellt werden kann.

Risch [6] hat 1969 erkannt, dass man dies zu einem Algorithmus zusammenfügen kann. Der Satz von Liouville liefert einen *Ansatz* für die gesuchte Integralfunktion, deren Bestandteile man dann mit Hilfe einer Art Koeffizientenvergleich bestimmen kann. Die gesuchten Logarithmen erfüllen eine Differentialgleichung, deren Lösung algorithmisch bestimmt werden kann. Die Details sind natürlich wesentlich komplizierter als im rationalen Fall.

Anders als bei einer Serie von Substitutionen und partiellen Integrationen liefert dieser Algorithmus nach endlich vielen Schritten entweder einen *Beweis* dafür, dass keine elementare Integralfunktion existiert, oder die Integralfunktion wird berechnet und kann ausgegeben werden. Der *Risch-Algorithmus* zur Integration elementarer Funktionen ist beispielsweise in *Maple* eingebaut.

Abschließende Bemerkungen

Dieser Artikel gibt einen Einstieg in die Theorie der algorithmischen Integration. Weiterführende Bücher zum Thema sind [2] und [4]. Viele der historischen Bemerkungen stammen aus dem sehr lesenswerten Buch von Edwards [3].

Literaturverzeichnis

[1] E. Behrends, *Analysis: Band 2*, Vieweg, Wiesbaden, 2004.

[2] M. Bronstein, *Symbolic Integration I*, Springer, Berlin, 1997.

[3] C. H. Edwards Jr., *The Historical Development of the Calculus*, Springer, New York, 1979.

[4] K. O. Geddes, S. R. Czapor und G. Labahn, *Algorithms for Computer Algebra*, Kluwer, Boston, Dordrecht, London, 1992.

[5] W. Koepf, *Computeralgebra. Eine algorithmisch orientierte Einführung*, Springer, Berlin, Heidelberg, 2006, vergleiche www.mathematik.uni-kassel.de/~koepf/CA.

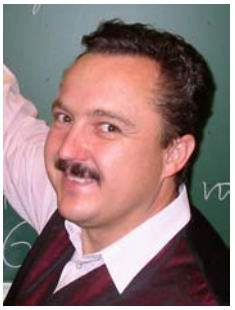
[6] R. H. Risch, *The problem of Integration in Finite Terms*, Trans. Amer. Math. Soc. **139**, 167 – 189, 1969.

Alles logisch, oder was?

Prof. Dr. Martin Kreuzer
Fakultät für Informatik und Mathematik
Universität Passau
Innstraße 33
94030 Passau

Stefan Kühling
Kremser Straße 61
94032 Passau

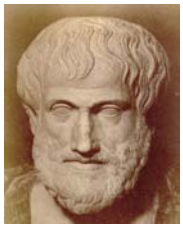
Martin.Kreuzer@Uni-Passau.de
Kuehling@FIM.Uni-Passau.de



Zusammenfassung

Die Logik ist eines der ältesten Teilgebiete der Mathematik. Logische Knobelaufgaben (sogenannte „Logeleien“) sind bei mathematischen Schülerwettbewerben sehr beliebt. Wir stellen eine Methode vor, mit der man die Lösung einer (aussagen-) logischen Aufgabe auf die Berechnung einer reduzierten Gröbner-Basis eines Polynomideals mittels eines Computeralgebrasystems zurückführen kann.

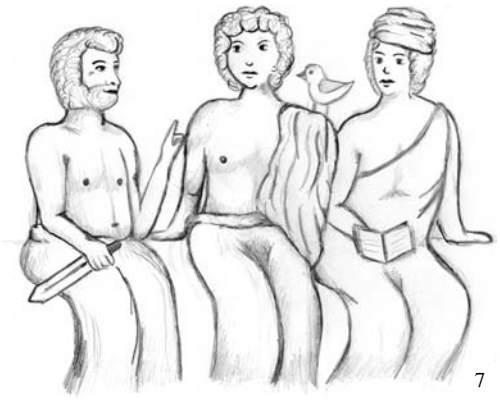
Ich weiß nicht, was soll es bedeuten? (Die Logelei)



Arsitoteles

Die Logik in der Mathematik fasziniert die Menschen schon seit dem Altertum. Das Wort $\lambda o\gamma o\sigma$ bedeutet hier „vernünftiges Denken“. Die erste Blütezeit der Logik erlebte ihren Höhepunkt mit den umfassenden Schriften des Aristoteles (384 – 322 v. Chr.).

Beispiel 1 Im Olymp sitzen drei Götter nebeneinander und unterhalten sich. Der linke Gott sagt, dass er neben dem Gott der Lüge steht. Der mittlere Gott sagt, dass er der Gott der Diplomatie sei. Der rechte Gott sagt, dass er neben dem Gott der Wahrheit steht. Wer ist hier wer?



Unter Verwendung von Aristoteles *Satz vom ausgeschlossenen Dritten*, der besagt, dass eine Aussage nur entweder wahr oder falsch sein kann, finden wir die Lösung:

Der rechte Gott kann nicht der Gott der Wahrheit sein, weil er dann die Wahrheit spräche und der mittlere Gott somit ebenfalls der Gott der Wahrheit wäre. Wäre dieser der Gott der Wahrheit, spräche er die Wahrheit und wäre zugleich der Gott der Diplomatie, was nicht geht. Also muss der linke Gott der Gott der Wahrheit sein. Da er damit die Wahrheit spricht, ist der mittlere Gott der Gott der Lüge und somit der rechte der Gott der Diplomatie.

Solche Aufgaben nennt man manchmal auch *Logeleien*. Doch wie soll man sie lösen, wenn man keine so einfachen Argumentationen wie eben findet?

$\alpha||\exists\sigma\ \ [o\gamma\wedge\sigma\subset\vdash]\ \vee\ \omega.\alpha.\sigma$? (Logische Formeln)

Eine *Aussage* ist ein sprachliches Gebilde, das entweder *wahr* oder *falsch* ist. Wir bezeichnen Aussagen mit Großbuchstaben $A, B, C, \ldots$ und übersetzen die Angaben durch *logische Verknüpfungen* dieser Aussagen, z. B.

$A \vee B$	bedeutet	„es gilt A oder B oder beides“
$A \wedge B$	bedeutet	„es gilt A und zugleich B“
$\neg A$	bedeutet	„nicht A“
$A \Rightarrow B$	bedeutet	„wenn A gilt, so gilt auch B“
$A \Leftrightarrow B$	bedeutet	„A gilt genau dann, wenn B gilt“

Nachdem wir die Voraussetzungen so in logische Formeln gefasst haben, prüfen wir deren mögliche Wahrheitswerte mit einer *Wahrheitstafel*. Betrachten wir diese Methode am folgenden

Beispiel 2 Ein Lehrer erhält vier identische Lösungen einer Mathe-Schulaufgabe. Um festzustellen, wer hier von wem abgeschrieben hat, befragt er die Schüler.

Angela: Wenn Peer gemogelt hat, dann auch Frank-Walter.

Frank-Walter: Peer oder Ulla hat abgeschrieben.

Peer: Entweder Angela oder Ulla hat abgeschrieben.

Ulla: Die Aufgabe wurde entweder von Peer oder von Angela gelöst.

[7] Besten Dank an Jennifer Heidenreich für die Gestaltung dieses Bildes.

Wir kürzen die Aussage „X hat abgeschrieben“ mit X ab und erhalten die folgenden Übersetzungen in logische Formeln:

Angela: $P \Rightarrow F$
Frank-Walter: $P \vee U$
Peer: $(A \wedge \neg U) \vee (\neg A \wedge U)$
Ulla: $(A \wedge \neg P) \vee (\neg A \wedge P)$

Bezeichnen wir (wie üblich) *wahr* mit 1 und *falsch* mit 0, so sind die Wahrheitswerte der verwendeten Verknüpfungen gegeben durch

X	Y	$\neg X$	$X \vee Y$	$X \wedge Y$	$X \Rightarrow Y$	$X \Leftrightarrow Y$
0	0	1	0	0	1	1
0	1	1	1	0	1	0
1	0	0	1	0	0	0
1	1	0	1	1	1	1

Dabei ist zu beachten, dass die Formel „wenn X gilt, so auch Y“ wahr ist, falls X nicht gilt, da dann ja die Voraussetzung nicht erfüllt ist und über den Wahrheitswert von Y nichts behauptet wird. Jetzt können wir die Wahrheitstafel für Beispiel 2 aufstellen.

A	F	P	U	$P \Rightarrow F$	$P \vee U$	$(A \wedge \neg U) \vee (\neg A \wedge U)$	$(A \wedge \neg P) \vee (\neg A \wedge P)$
0	0	0	0	1	0	0	0
0	0	0	1	1	1	1	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	1
0	1	0	0	1	0	0	0
0	1	0	1	1	1	1	0
0	1	1	0	1	1	0	1
0	1	1	1	1	1	1	1
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	1
1	0	1	0	0	1	1	0
1	0	1	1	0	1	0	0
1	1	0	0	1	0	1	1
1	1	0	1	1	1	0	1
1	1	1	0	1	1	1	0
1	1	1	1	1	1	0	0

Die Aufgabe hat also eine eindeutige Lösung: alle haben von Angela abgeschrieben. Für größere Aufgaben mit vielen Aussagen wird die Methode der Wahrheitstafeln schnell zu aufwändig. Ein klarer Fall für die Computeralgebra!

Die Wahrheit ist eine Nullstelle (Rechen mit Wahrheitswerten)

Um mit den Wahrheitswerten 0 und 1 Computerberechnungen durchführen zu können, betrachten wir sie als eine Art Zahlen und führen folgende Addition und Multiplikation ein.

$+$	0	1
0	0	1
1	1	0

$\cdot$	0	1
0	0	0
1	0	1

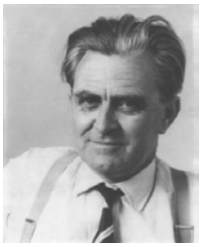
Die Addition entspricht also einem „exklusiven oder“ (d.h. es gilt A oder B, aber nicht beides) und die Multiplikation entspricht der „und“-Verknüpfung. Statt Aussagensymbolen $A, B, C, \ldots$ verwenden wir nun Variablen $a, b, c, \ldots$ Da diese Variablen nur die Werte 0 und 1 annehmen sollen, erfüllen sie die Gleichungen $a^2 = a$, $b^2 = b$, usw. Aus $a + a = 0$ folgt außerdem, dass $-a = a$ erfüllt ist. Mit anderen Worten, unsere Variablen nehmen nur Werte im Zahlenbereich $\mathbb{Z}/(2) = \{0, 1\}$ an.

Nun wollen wir die Aussage A mit dem *Polynom* $a+1$ übersetzten. Dies macht die Suche nach dem Wahrheitswert 1 für A zu einer Suche nach einer *Nullstelle* des Polynoms $a + 1$ über dem Zahlbereich $\mathbb{Z}/(2)$. Die Verknüpfungen übersetzen sich wie folgt:

logische Formel	A	$\neg A$
polynomiale Übersetzung	$a+1$	a

$A \vee B$	$A \wedge B$	$A \Rightarrow B$	$A \Leftrightarrow B$
$ab + a + b + 1$	$ab + 1$	$ab + a$	$a + b$

Sucht man nach simultanen Nullstellen mehrerer Polynome $f_1, \ldots, f_k$, so kann man stattdessen nach Nullstellen des erzeugten *Polynomideals* I suchen. Letzteres besteht aus allen Summen $f_1 g_1 + \ldots + f_k g_k$, wobei $g_1, \ldots, g_k$ beliebige Polynome sind. Beim Auffinden der Nullstellen eines solchen Polynomideals hilft uns die Berechnung einer reduzierten *Gröbner-Basis* mittels des *Buchberger-Algorithmus*.



Wolfgang Gröbner

Bruno Buchberger

Die reduzierte Gröbner-Basis besteht aus besonders einfachen Polynomen mit denselben Nullstellen, an denen wir die Lösung der Aufgabe oft sofort ablesen können. In Beispiel 2 ergeben sich etwa die folgenden Polynomübersetzungen.

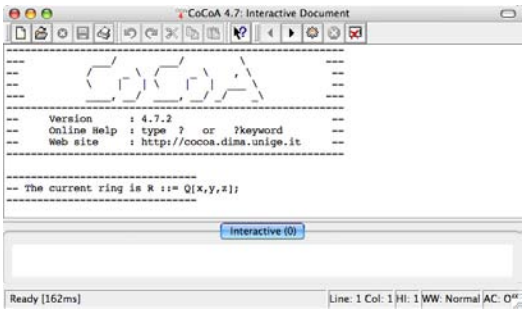
A	F	P	U	$P \Rightarrow F$	$P \vee U$
$a+1$	$f+1$	$p+1$	$u+1$	$pf+p$	$pu+p+u+1$

$(A \wedge \neg U) \vee (\neg A \wedge U)$	$(A \wedge \neg P) \vee (\neg A \wedge P)$
$a + u + 1$	$a + p + 1$

Hierbei ist $\{a, f+1, p+1, u+1\}$ eine reduzierte Gröbner-Basis, so dass wir die Nullstellen $a = 0$, $f = 1$, $p = 1$ und $u = 1$ unmittelbar ablesen können.

Alles wird durch den Kakao gezogen (Logische Berechnungen mit CoCoA)

So, jetzt wollen wir uns nicht mehr so anstrengen und die gesuchte reduzierte Gröbner-Basis automatisch berechnen lassen, z. B. von einem Computeralgebrasystem wie CoCoA (vgl. [1] oder [2]).



Nach der Installation und dem Starten des Systems lösen z. B. die folgenden Befehle das Beispiel 2:

```
Use Z/(2)[a,f,p,u];
K:=Ideal(a^2-a, f^2-f, p^2-p, u^2-u);
I:=Ideal(pf+p, pu+p+u+1, a+u+1, a+p+1)+K;
ReducedGBasis(I);
```

Die erste Zeile legt dabei den gewünschten Polynomring fest, die zweite definiert das *Körperideal*, das die Nullstellen auf 0 und 1 beschränkt, die dritte Zeile das zur Aufgabe gehörige Polynomideal und die vierte liefert die gesuchte Antwort.

Die Antwort von CoCoA ist $[a, f+1, p+1, u+1]$. Sie bedeutet, dass $a = 0$, $f = 1$, $p = 1$ und $u = 1$ gelten muss, im Einklang mit der Lösung durch die Wahrheitstafelmethode.

Zum Abschluss betrachten wir noch ein komplizierteres Beispiel, bei dem eine Lösung mittels einer Wahrheitstafel schon recht anstrengend wäre.

Beispiel 3 (Der Bulle von Transsylvanien)

Inspektor Craig hat wieder einmal schwierige Fälle zu lösen. Sein augenblickliches Revier ist Transsylvanien, ein Land, das gleichermaßen von Vampiren wie Menschen bewohnt wird. Hinzu kommt, dass ein beträchtlicher Teil der Bevölkerung von einer grausamen Geisteskrankheit heimgesucht wird, die ihre Opfer so sehr verwirrt, dass diese immer das Gegenteil dessen sagen, was sie eigentlich meinen. Erschwerend für seine Ermittlungen wirkt sich auch die Tatsache aus, dass Vampire im Gegensatz zu Menschen auch noch lügen (ein geisteskranker Vampir sagt also wieder die Wahrheit).

In seinem ersten Fall handelt es sich um zwei Schwestern namens Lucy und Minna. Inspektor Craig muss herausfinden, wer von ihnen ein Vampir ist. Er weiß, dass eine der beiden Schwestern ein Vampir ist und die andere ein Mensch. Über den Geisteszustand der Betreffenden ist nichts bekannt. Hier ist das Transkript der Untersuchung:

Insp. Craig: „Erzählen Sie mir etwas, das Sie beide betrifft."

Lucy: „Wir sind beide verrückt."

Insp. Craig: „Stimmt das?"

Minna: „Natürlich nicht."

Hieraus konnte Inspektor Craig zu jedermanns Zufriedenheit nachweisen, welche der Schwestern ein Vampir war. Sie auch?

Zur Modellierung dieser Aufgabe definieren wir sechs Aussagen:

$$\begin{aligned} A &= \text{„Lucy sagt die Wahrheit"} \\ B &= \text{„Lucy ist ein Mensch"} \\ C &= \text{„Lucy ist geistig gesund"} \\ D &= \text{„Minna sagt die Wahrheit"} \\ E &= \text{„Minna ist ein Mensch"} \\ F &= \text{„Minna ist geistig gesund"} \end{aligned}$$

Die Voraussetzung über die Wahrheitsliebe bei Menschen und Vampiren liefert die beiden Formeln $A \Leftrightarrow (B \Leftrightarrow C)$ und $D \Leftrightarrow (E \Leftrightarrow F)$. Ferner wissen wir $B \Leftrightarrow \neg E$. Der Dialog übersetzt sich zu $A \Leftrightarrow (\neg C \wedge \neg F)$ sowie $D \Leftrightarrow \neg A$.

Beachten wir, dass sich die Äquivalenz $G \Leftrightarrow F$ zweier logischer Formeln mit der Summe $g + h$ der zugehörigen Polynome übersetzen lässt, so erhalten wir nachstehende CoCoA-Berechnung:

```
Use Z/(2)[a,b,c,d,e,f];
K:=Ideal(a^2-a,b^2-b,c^2-c,d^2-d,e^2-e,f^2-f);
I:=Ideal(a+b+c+1, d+e+f+1, b+e+1,
                        cf+c+f+a+1, d+a+1)+K;
ReducedGBasis(I);
```

Das Ergebnis $[f^2+f, a+f+1, b, c+f, d+f, e+1]$ hat nun folgende Interpretation: Es gibt zwei Lösungen, eine mit $f = 0$ und eine mit $f = 1$. Beide Male gilt $b = 0$ und $e = 1$, d. h. Lucy ist ein Vampir und Minna ein Mensch.

Die hier vorgestellte Methode nennt man das *Gröbner-Basis-Beweissystem* für die Aussagenlogik. Weitere Beispiele und Aufgaben findet der Leser auf der Webseite dieses Heftes [3] und in [4].

Links und Literatur

[1] CoCoA — A System for Doing Computations in Commutative Algebra, verfügbar unter cocoa.dima.unige.it.

[2] ApCoCoA — Applied Computations in Commutative Algebra, siehe www.apcocoa.org.

[3] Die Webseite dieses Sonderhefts: www.fachgruppe-computeralgebra.de/JdM/.

[4] M. Kreuzer und S. Kühling, *Logik für Informatiker*, Pearson, München, 2006.


DU KANNST MEHR MATHE,
ALS DU DENKST.
www.jahr-der-mathematik.de
22 KM/H
DRUCKPUNKT 01
360°
DRUCKPUNKT 02
30°
F'(t)<0
1
2
3
−162 CM = ∫ F'(t)dt
Wissenschaftsjahr 2008
Mathematik
Alles, was zählt

Sichere Kommunikation über unsichere Leitungen?

StR Markus Meiringer
Goethe-Gymnasium Regensburg
Goethestraße 1
93049 Regensburg

meiringer@gmx.de

Zusammenfassung

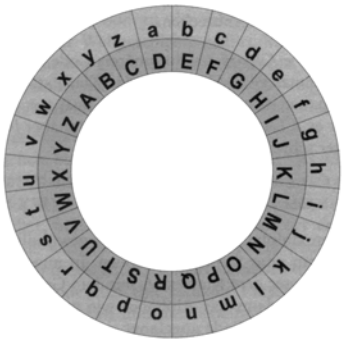
Am Beispiel der Cäsar- und Vigenère-Verschlüsselung wird das Rechnen in Restklassen motiviert und mit Hilfe von Derive umgesetzt. Die Frage nach perfekter Sicherheit liefert den Ausgangspunkt, um über den Schlüsselaustausch nach Diffie-Hellman und den RSA-Algorithmus nachzudenken.

Viele Male und auf vielerlei Weise hat die Kryptographie Eingang in Romane aus den Bestsellerlisten gefunden. Dan Brown lässt etwa in „Sakrileg“ seine Hauptdarsteller Sophie Neveu, eine Kryptologin, und Robert Langdon, einen Symbol-Forscher, immer wieder neu durch verschlüsselte Botschaften näher an ihr Ziel gelangen. In „Diabolus“ geht es sogar darum, ob eine geheime Verschlüsselungsmaschine tatsächlich alle denkbaren Kryptosysteme entschlüsseln kann. Es wird hierbei die Frage nach dem perfekten Verschlüsselungsalgorithmus aufgeworfen.

Cäsarchiffrierung

Schon auf Kinder im Grundschulalter wirken Geheimschriften äußerst anziehend. Meist ersetzen sie einen Buchstaben durch ein anderes Zeichen — in der großen Hoffnung: je komplizierter das Zeichen, desto schwerer zu knacken … Diese Idee einer sog. Substitutionschiffre geht auf G. J. Cäsar zurück, der die Buchstaben des Alphabets einfach ein Stück verschoben hat. Er verschlüsselte etwa ein A mit D, dann ein B mit E, und schließlich wird aus Z ein C. Dies kann mit einem Chiffrierrad wie dem abgebildeten leicht nachgemacht werden. Zum Verschlüsseln sucht man den Buchstaben am äußeren Rand und ersetzt ihn mit dem entsprechenden inneren. Da man mit der Einstellung des Rades, etwa „Verschiebung um 3“, sowohl ver- als auch entschlüsseln kann, nennt man derartige Kryptosysteme symmetrisch. Will man die Idee des Chiffrierrades in ein Computerprogramm umsetzen, sollten die Buchstaben in Zahlen umgewandelt werden, beispielsweise das A in die Zahl 1, das B in 2 und schließlich das Z in 26 (oder 0).

Die oben erwähnte Verschlüsselung nach Cäsar könnte man nun durch eine einfache Addition darstellen: A ($\widehat{=}$ 1) wird also zu D ($\widehat{=}$ 1 + 3). Ein Problem ergibt sich bei den Buchstaben am Ende des Alphabets, etwa muss Y ($\widehat{=}$ 25) mit B ($\widehat{=}$ 2) verschlüsselt werden, jedoch ergibt $25 + 3 = 28$. Man muss nun lediglich den Rest dieser Zahl bei der Division durch 26 betrachten. Für diesen Rest kann man auch $28 \mod 26$ (sprich: 28 modulo 26) schreiben, und es ergibt sich $2 \equiv 28 \mod 26$. Mit dieser Art „Gleichheitszeichen“ kann man Umformungen wie gewohnt durchführen — lediglich bei der Division kann es Probleme geben ($4 \equiv 22 \mod 6 \Rightarrow 4+5 \equiv 22+5 \mod 6$ und $4 \cdot 7 \equiv 22 \cdot 7 \mod 6$, aber $4 \equiv 22 \mod 6 \not\Rightarrow 4 : 2 \equiv 22 : 2 \mod 6$, was ja $2 \equiv 11 \mod 6$ bedeuten würde). Um diesem Problem zu entgehen verwendet man häufig Primzahlen: $4 \equiv 18 \mod 7 \Rightarrow 4 - 10 \equiv 18 - 10 \mod 7$ und jetzt auch $4 : 2 \equiv 18 : 2 \mod 7$.



Chiffrerad

Möchte man den Buchstaben R $\widehat{=}$ 18 mit einer Verschiebung um 10 verschlüsseln, muss in Derive 6 nur `MOD (18+10,26)` berechnet werden und es ergibt sich 2 $\widehat{=}$ B. Auch ein Text etwa „VERSCHLUESSELNISTSCHOEN“ kann in eine Zahlenfolge übersetzt werden `[22, 5, 19, 3, 8, 12, 21, 5, 19, 19, 5, 12, 14, 9, 19, 20, 19, 3, 8, 15, 5, 14]` und mit folgendem Programm verschlüsselt werden.

```
caesar(s,text):=PROG(
    i:=1,
    LOOP(
      IF(i=DIM(text)+1,RETURN text),
      text↓i:=MOD(text↓i+3,26),
      i:=i+1))
```

Fragen:

a) Schreibe ein ähnliches Programm zum Entschlüsseln.

b) Ergibt sich auch eine Möglichkeit zur Verschlüsselung, wenn man nicht die Addition mit einer Zahl, sondern die Multiplikation mit 2 oder 3 betrachtet?

Vigenère-Verschlüsselung

Mehr Sicherheit als die Cäsarchiffre bietet die sog. Vigenère-Verschlüsselung, die über lange Zeit hin als sicher galt. Hierzu benötigt man ein Schlüsselwort, etwa KRYPTO. Nun wird das Chiffrierrad so eingestellt, dass A zunächst mit dem ersten Buchstaben des Schlüsselwortes K verschlüsselt wird, damit wird der erste Buchstabe x_1 des Klartextes $x_1x_2x_3x_4x_5\cdots x_n$ verschlüsselt. Danach stellt man das Chriffrierad auf den nächsten Buchstaben des Schlüsselwortes und verschlüsselt x_2 usw. Ist man beim O, dem letzten Buchstaben, angelangt, beginnt man wieder mit dem K. Mit dieser Verschlüsselung lautet unser Text von oben nun „GWQIWWWMDIMTWFHINHNZNUH“.

Werden das Schlüsselwort $k_1k_2k_3\cdots k_l$ und der Geheimtext $y_1y_2y_3\cdots y_n$ auch als Zahlenfolgen betrachtet, so gilt:

$$y_i = (x_i + k_{i \bmod l}) \bmod 26$$

Hier bedeutet das „$i \bmod l$“ die Stelle des aktuellen Buchstaben des Schlüsselwortes, der sich ja alle l Schritte wiederholt. Die Addition dieses Schlüsselbuchstabens modulo 26 entspricht einer Cäsarverschiebung. Eine Umsetzung dieser Idee in Derive 6 könnte so aussehen:

```
vigenere(schluessel,text):= PROG(
    i:=1,
    LOOP(
       IF(i=DIM(text)+1,RETURN text),
       k:=MOD(i-1,DIM(schluessel))+1,
       text↓i:=MOD(text↓i+schluessel↓k,26),
       i:=i+1))
```

Verwendet man das Schlüsselwort KRYPTO $\widehat{=}$ `[11,18,25,16,20,15]`, kann nun verschlüsselt werden.

Fragen:

a) Mit `text↓i` kann man sich auf den `i`-ten Buchstaben beziehen und `DIM(text)` gibt die Länge des Textes an. Warum wurde bei `k:=MOD(i-1,DIM(schluessel))+1` zuerst 1 subtrahiert und dann wieder addiert?

b) Schreibe ein Programm zum Entschlüsseln.

So sicher diese Verschlüsselung auch scheint, kann man mit dem sog. Kasiski-Angriff und dem Friedman-Angriff das Schlüsselwort mit heutiger Technik sehr schnell bestimmen und den Geheimtext knacken (vgl. [2] S. 16 – 19).

Man stellt also fest, dass solche Verschlüsselungen keine Sicherheit garantieren, da sie sich einem Angriff geeigneter Entschlüsselungsmaschinen in kürzester Zeit geschlagen geben müssen. Also bieten die Cäsarchiffre und Vigenère-Chiffrierung keine perfekte Sicherheit? Doch! Es kommt nur auf die Länge an: ein einzelner Buchstabe mit dem Cäsarcode verschlüsselt, bietet perfekte Sicherheit. Niemand kann sagen, welcher Buchstabe da verschlüsselt wurde, wenn er nicht die Einstellung der Chiffrierscheibe kennt. Das kann man sich zu Nutzen machen und die Chiffrierscheibe als eine Art Glücksrad betrachten, um sie vor der Verschlüsselung eines neuen Buchstaben mit Schwung zu drehen und dann zu verschlüsseln. Dieses System liefert perfekte Sicherheit, die aber so perfekt ist, dass auch der berechtigte Empfänger den Geheimtext nicht mehr lesen kann. Die einzige Möglichkeit wäre, diese zufällige Buchstabenfolge zu übermitteln. Dieser Schlüssel besteht dann aus ebenso vielen Buchstaben wie der zu übermittelnde Text und könnte nur ein einziges Mal verwendet werden.

Schlüsselaustausch nach Diffie-Hellman

Es bleibt somit das Problem des Schlüsselaustausches: Wie können zwei Nutzer eine natürliche Zahl vereinbaren, ohne dass ein Dritter, der die Leitung abhört, diese natürliche Zahl mitbekommt. Eine erste Idee erhält man aus den Potenzgesetzen: Alice und Bob vereinbaren eine natürliche Zahl g als Basis, die veröffentlicht wird. Sowohl Alice als auch Bob denken sich jeweils eine weitere natürliche Zahl a bzw. b als Exponent aus, die sie geheim halten. Alice berechnet nun g^a und schickt diesen Wert an Bob, der das entsprechende mit g^b macht. Jeder der beiden potenziert nun den erhaltenen Wert mit seiner geheimen Zahl und beide erhalten $(g^a)^b = g^{ab} = (g^b)^a$. Leider kann aber Eve (sie ist „evil“) aus den Zahlen g^a und g^b durch einfaches Logarithmieren a und b bestimmen und kennt somit das Geheimnis g^{ab}.

Durch eine geringe Änderung aber kann diese Idee sicher gemacht werden: man vereinbart zuerst eine Primzahl p und wieder eine Zahl g. Alice und Bob berechnen nun die Potenzen modulo dieser Primzahl und kennen somit den geheimen Schlüssel $g^{ab} \bmod p$.

Zur Verdeutlichung ein kleines Zahlenbeispiel:

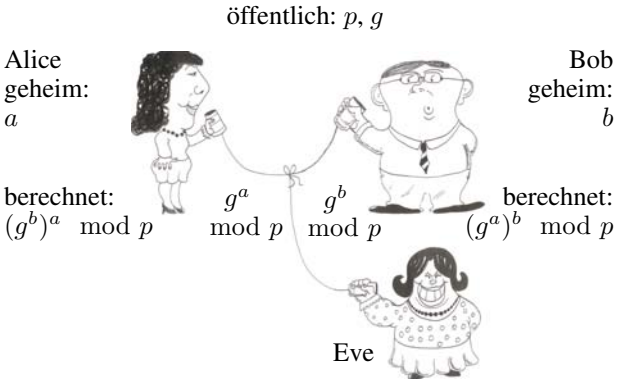
öffentlich: $p = 23$ und $g = 5$

Alice: $a = 3$ und $g^a \bmod 23 \equiv 5^3 \bmod 23 \equiv 125 \bmod 23 \equiv 10$

Bob: $b = 17$ und $g^b \bmod 23 = 5^{17} \bmod 23 \equiv 762\,939\,453\,125 \bmod 23 \equiv 15$

Alice: $g^{ab} \bmod 23 \equiv (g^b)^a \bmod 23 \equiv 15^3 \bmod 23 \equiv 3375 \bmod 23 \equiv 17$

Bob: $g^{ab} \bmod 23 \equiv 10^{17} \bmod 23 \equiv 17$



Schlüsselaustausch

Die Berechnung mit dem Taschenrechner liefert bei der Eingabe 5^{17} leider etwa $7,6293945 \cdot 10^{11}$.

Folgendes Vorgehen, das sich an den Algorithmus „square and multiply“ anlehnt (vgl. [3] S. 125 oder [2] S. 121), führt aber zum Ziel, wenn man bedenkt, dass $25 \equiv 2 \mod 23$ und $256 \equiv 5 \mod 23$. Somit gilt: $5^{17} \mod 23 \equiv 5 \cdot (5^2)^8 \mod 23 \equiv 5 \cdot 25^8 \mod 23 \equiv 5 \cdot 2^8 \mod 23 \equiv 5 \cdot 256 \mod 23 \equiv 5 \cdot 3 = 15$. Hat man es mit viel größeren Zahlen wie in `MOD(5^17, 23)` zu tun, kann man diese Idee auf folgende rekursive Art umsetzen:

```
powermod(a, n, p) :=
    IF (n=0,
    1,
    IF(MOD(n, 2)=0,
       MOD(powermod(a, n/2, p)^2, p),
       MOD(powermod(a, n-1, p)·a, p) ))
```

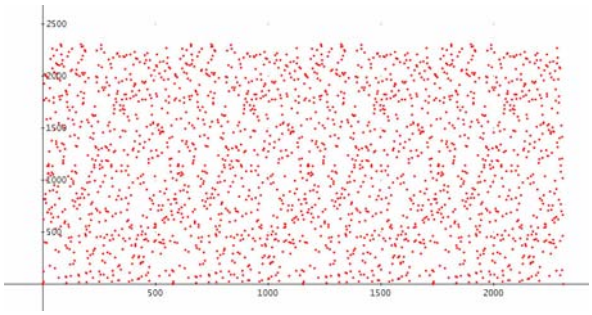
Unsere Angreiferin Eve hat es nun um einiges schwerer, da sie nicht mehr wie gewohnt logarithmieren kann. Sie sucht zwar immer noch eine Lösung der Gleichung $5^x \mod 23 \equiv 17$ nun aber mit einer endlichen Grundmenge $x \in \{0; 1; 2; \ldots; 22\}$, weshalb man derartige Fragestellungen als Problem des diskreten Logarithmus bezeichnet. Sie hat als Lösungsstrategie die Möglichkeit alle Werte für x auszuprobieren, was sich bei 23 verschiedenen Zahlen noch relativ leicht, auch mit einem Taschenrechner, machen lässt. Mit dem Befehl `SELECT(MOD(5^x, 23)=10, x, 0, 22)` sortiert man aus den ganzen Zahlen von 0 bis 22 diejenigen aus, die die Gleichung erfüllen.

Eine weitere Möglichkeit ist durch folgendes Programm in Derive gegeben, das bei einer gefundenen Lösung abbricht und somit Rechenzeit spart:

```
dlog(n, g, p) :=PROG(
    i:=0,
    LOOP(IF(MOD(g^i, p)=n, RETURN i),
       i:=i+1) )
```

Somit erhält man durch `dlog(10, 5, 23)` sofort 3 und damit die geheime Zahl von Alice. Aber immer noch ist es viel schwieriger den Wert des diskreten Logarithmus zu bestimmen, als umgekehrt die diskrete Exponentialfunktion zu berechnen. Bei Anwendungen in der Praxis sind zur Zeit für die Primzahlen Größen von 1024 bit oder gar 2550 bit üblich. Das sind Zahlen der Ordnung $2^{1024} = 10^{1024 \cdot \frac{\ln 2}{\ln 10}} \approx 10^{308}$ bzw. $2^{2550} = 10^{2550 \cdot \frac{\ln 2}{\ln 10}} \approx 10^{767}$. Man hat dazu bislang keine Verfahren gefunden, bei so großen Primzahlen mit mehreren hundert Stellen eine effiziente Berechnung durchzuführen. Natürlich findet man den diskreten Logarithmus durch obiges Verfahren, aber der dafür benötigte Zeitbedarf ist viel zu groß. Man kann das auch mit Derive durch den Aufruf von `dlog(n, g, p)` ausprobieren. Verwendet man eine vierstellige Primzahl p, dauert die Berechnung noch unter einer Sekunde; bei fünfstelligen Primzahlen dauert es ca. eine Sekunde, etwa zehnmal so lange. Bei achtstelligen Primzahlen ist man dann schon im 1000 Sekunden Bereich und setzt man das auf Primzahlen mit 300 Stellen fort, würde man $10^{300-5}\text{s} \approx 3 \cdot 10^{287}\text{a}$ erhalten. Die Welt besteht seit ca. 4,6 Milliarden Jahren, gut gerundet 10^{10} Jahre, also müsste man 10^{277} mal so lange warten. Es gibt zwar weitaus bessere Verfahren, wie den Baby-Step-Giant-Step-Algorithmus oder Silver-Pohlig-Hellman-Algorithmus (vgl. [2] S. 137 und [1] Kapitel 9), jedoch bleibt die Rechenzeit stets ineffizient hoch.

Zur weiteren Veranschaulichung kann man sich die Graphen mit Derive ansehen: 5^x zeichnet sich im Graphikfenster wunderbar, jedoch sollte man bei der diskreten Version `VECTOR([x, MOD(5^x, 23)], x, 0, 22, 1)` berechnen und die entstehende Matrix (Tabelle) zeichnen lassen und dann das „Durcheinander“ anschauen (`MOD(5^x, 23)` alleine liefert nicht das gewünschte Ergebnis).



$5^x \mod 2309$

Um dem Angreifer noch etwas Wind aus den Segeln zu nehmen, sollte man bei der Wahl von g vor der Veröffentlichung sehr sorgfältig vorgehen. Würde man etwa $g = 22$ wählen, so gäbe es nur wenige Möglichkeiten für Potenzen von g, denn $\{22^x \mod 23 \mid x \in [0; 22]\} = \{1; 22\}$. Man berechnet die Werte mittels Derive durch den Aufruf `VECTOR(MOD(22^x, 23), x, 0, 22, 1)`.

Kontrolliert die Angreiferin Eve die Datenleitung zwischen A und B vollständig, so könnte sie eine zufällige Zahl e wählen und mit A über einen Schlüssel $g^{ae} \mod p$ und mit B über einen Schlüssel $g^{eb} \mod p$ korrespondieren. Für diesen Man-in-the-middle-Angriff (vgl. [2] S. 234) gibt es zwar auch Lösungen, die diesen Rahmen aber sprengen würden.

RSA-Verschlüsselung

Mit der Idee des diskreten Logarithmus kann man auch ein Verschlüsselungsverfahren, das sog. ElGamal-Verfahren, entwickeln. Da das Wissen, wie man verschlüsselt, nach aktuellem Stand der Forschung keine Schlussfolgerungen auf die Entschlüsselung zulässt, nennt man dieses Verfahren asymmetrisch. Solche Protokolle werden auch als public-key-Verfahren bezeichnet, da man den Schlüssel zum Verschlüsseln bedenkenlos veröffentlichen kann. Das beruht auf sog. „Einwegfunktionen“, die in einer Richtung leicht zu berechnen sind und in der anderen in angemessener Zeit kein Resultat liefern. In unserem Beispiel war das der diskrete Logarithmus; beim RSA-Algorithmus, einem anderen asymmetrischen Verfahren, macht man sich zu Nutzen, dass die Multiplikation zweier Primzahlen leicht zu bewerkstelligen, jedoch eine Zahl in die Primfaktoren zu zerlegen in realistischer Zeit unmöglich ist.

Man wählt hierzu zwei verschiedene Primzahlen z.B. $p = 179$ und $q = 113$, die in der Praxis von der gleichen Größenordnung (bis zu 1024 Bits) sein sollen. Nun sucht man Zahlen e und d, sodass $\operatorname{ggT}(e;(p-1)(q-1)) = 1$, $e \cdot d = 1 \mod (p-1)(q-1)$ und $e, d < (p-1)(q-1)$. Beispielsweise erfüllen $e = 211$ und $d = 14267$ mit $n = pq = 20227$ und $(p-1)(q-1) = 19936$ diese Eigenschaften und es gilt für ein $k \in \mathbb{Z}$

$$e \cdot d = k \cdot (p-1)(q-1) + 1.$$

Die Zahl e findet man sehr leicht, da man nur eine beliebige teilerfremde Zahl suchen muss. Häufig ist $e = 3$ schon eine Möglichkeit, die leider eine Sicherheitslücke auftut, die durch die sog. Low-Exponent-Attacke ([1] S. 119) ausgenutzt werden kann. Hingegen ist es schwieriger ein geeignetes d zu finden, das sich nach dem euklidschen Algorithmus (vgl. [3] S. 116f), also durch sukzessive Division von $(p-1)(q-1)$ durch e mit Rest ergibt:

$$\begin{array}{rcrcrcr}
19936 & = & 94 & \cdot & 211 & + & 102 \\
211 & = & 2 & \cdot & 102 & + & 7 \\
102 & = & 14 & \cdot & 7 & + & 4 \\
7 & = & 1 & \cdot & 4 & + & 3 \\
4 & = & 1 & \cdot & 3 & + & 1 \\
3 & = & 3 & \cdot & 1 & + & 0
\end{array}$$

Da der letzte auftretende Rest 1 ist, kann man folgern, dass $\operatorname{ggT}(211, 19936) = 1$. Dies gelingt mit Derive durch

```
euklid(a,b):=PROG(
   IF(a=0,
      b,
      IF(b=0,
      a,
      LOOP(r:=MOD(a,b),
         a:=b,b:=r,
         IF(r=0,RETURN a)))))
```

Um d zu finden, rechnet man nun von unten nach oben zurück und es ergibt sich:

$$\begin{aligned}
1 &= 4 - 1 \cdot 3 \\
&= 4 - 1 \cdot (7 - 4) = -7 + 2 \cdot 4 \\
&= -7 + 2 \cdot (102 - 14 \cdot 7) \\
&= 2 \cdot 102 - 29 \cdot 7 \\
&= 2 \cdot 102 - 29 \cdot (211 - 2 \cdot 102) \\
&= -29 \cdot 211 + 60 \cdot 102 \\
&= -29 \cdot 211 + 60 \cdot (19936 - 94 \cdot 211) \\
&= -5669 \cdot 211 + 60 \cdot 19936
\end{aligned}$$

Um in Derive auf die gewünschten Zahlen zu schließen, kann man statt obiger Berechnung sehr elegant rekursiv vorgehen:

```
euklidplus(a,u,v,b,x,y) :=
   If(b=0
      [a,u,v]
      euklidplus(b,x,y,MOD(a,b),
         u-FLOOR(a,b)·x,v-FLOOR(a,b)·y))
```

Komfortabler wird es, da die Startwerte für u, v, x und y stets gleich sind, mit

```
supereuklid(a,b):=euklidplus(a,1,0,b,0,1).
```

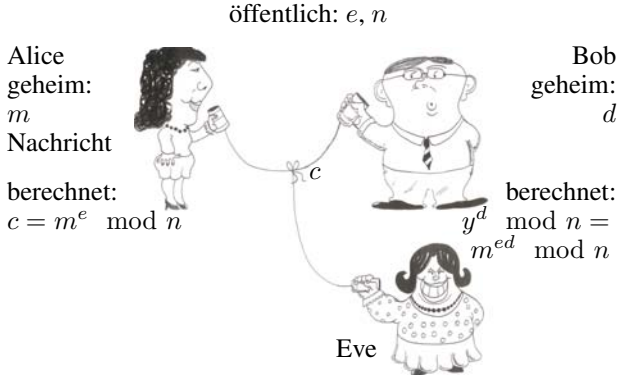
Mit `supereuklid(211,19936)` erhält man nun `[1,-5669,60]` und kann somit $\operatorname{ggT}(211, 19936) = 1$ als Kombination der Zahlen 211 und 19936 schreiben:

$$1 = -5669 \cdot 211 + 60 \cdot 19936$$

Weitere Möglichkeiten finden sich, wenn man zu -5669 das k-fache von 19936 addiert und von 60 das k-fache subtrahiert (warum?). Für $k = 1$ ergibt sich also:

$1 = 14267 \cdot 211 - 151 \cdot 19936$, womit $d = 14267$ gefunden ist.

Wesentlich ist, dass man die Zahlen e und d nur dann gut finden kann, wenn man p und q kennt. Zum Verschlüsseln kann man e und n bedenkenlos veröffentlichen. Jedoch muss der private Schlüssel d geheim gehalten werden. Insbesondere lässt sich exakt nachweisen, dass es genauso schwierig ist aus e und n den geheimen Schlüssel d zu berechnen, wie die Zahl n in ihre Primfaktoren p und q zu zerlegen.



RSA-Verschlüsselung

Eine Mitteilung besteht nun wieder aus einer Zahl m, die kleiner als n sein soll, sodass ein beliebiger Text in eine Ziffernfolge und diese dann in Zahlen geeigneter Größe aufgespalten werden muss, die schließlich der Reihe nach verschlüsselt werden. Diese Zahl wird modulo n mit e potenziert und man erhält den Geheimtext c:

$$c \equiv m^e \mod n.$$

Zum Entschlüsseln potenziert man c mit dem geheimen Schlüssel d und erhält nun m':

$$m' \equiv c^d \mod n.$$

Das sollte doch nun die ursprüngliche Nachricht von Alice sein. Ein Versuch mit Derive liefert für $m = 12457$ mit `MOD(12457^211, 20227)` die Zahl $c = 2339$. Bob entschlüsselt nun mit `MOD(2339^14267, 20227)` und erhält tatsächlich 12457.

Ist das Zufall oder hat Bob wirklich immer diese Möglichkeit? Hier hilft eine Version des sog. kleinen Satzes von Fermat oder Satzes von Euler, der besagt, dass für $n = pq$ gilt:

$$a^{(p-1)(q-1)} \equiv 1 \mod n.$$

In [1] (S. 35) und [2] (S. 124 ff) kann man den Nachweis des Satzes samt einer allgemeineren Version nachlesen. Bob kann also die ursprüngliche Nachricht m berechnen, da $m' \equiv c^d \equiv m^{ed} \equiv m^{k\cdot(p-1)(q-1)+1} \equiv \left(m^{(p-1)(q-1)}\right)^k \cdot m \equiv 1^k \cdot m \equiv m \mod n$.

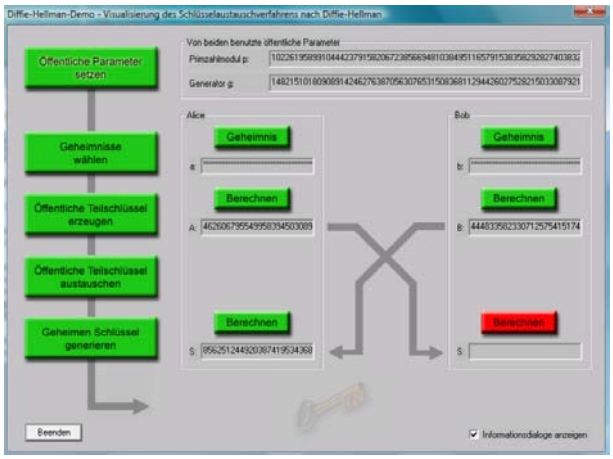
Fragen:

a) Wie viele Schlüssel braucht man mit dem Vigenère- und mit dem RSA-Verfahren, wenn man sich mit 5 Leuten austauschen möchte?

b) Suche zwei größere Primzahlen und entwirf ein RSA-Verschlüsselungssystem. Wende dabei den Euklidschen Algorithmus mit Bleistift und Papier an und experimentiere mit dem Programm, ver- und entschlüssle sodann.

Eine Möglichkeit mit den verschiedensten Verschlüsselungsverfahren zu experimentieren bietet CrypTool 1.4.10 (`www.cryptool.de`).

In der Praxis werden häufig sogenannte Hybridverfahren benutzt. Dabei verwendet man bei der Übertragung des Schlüssels eines symmetrischen Verfahrens (z.B. Vigenèreverfahren) ein asymmetrisches Verfahren (etwa Schlüsselaustausch nach Diffie-Hellman). Leider nutzt das zur Verwirklichung der perfekten Sicherheit wenig, da es keinen Sinn macht einen Schlüssel, der so lang wie der Originaltext ist, mit einem asymmetrischen Verfahren zu übertragen, um danach ein symmetrisches zum Verschlüsseln zu verwenden.



CrypTool

Vielleicht hat man Lust bekommen weiter zu experimentieren. Dazu lässt sich im Literaturverzeichnis und mit Hilfe des Programms CryptTool einiges finden. Weiteres Probieren ist natürlich mit einem CAS wie Derive oder durch Programmierung mit einer herkömmlichen Programmiersprache möglich. Die hier verwendeten Beispiele findet man unter `www.fachgruppe-computeralgebra.de/JdM/`.

Ein abschließender Dank gilt Thomas Sigl für die künstlerische Umsetzung von Alice, Bob und Eve.

Literaturverzeichnis

[1] J. Buchmann, *Einführung in die Kryptographie*, Springer, New York, 1999.

[2] A. Beutelspacher, H. B. Neumann und T. Schwarzpaul, *Kryptografie in Theorie und Praxis*, Vieweg, Wiesbaden, 2005.

[3] A. Beutelspacher, J, Schwenk und K.-D. Wolfenstetter, *Moderne Verfahren der Kryptographie*, Vieweg, Wiesbaden, 2001.

[4] W. Stich, *Kryptographie*, Vorlesungsmitschrift, 2006.

Die Mathematik der Compact Disc

Prof. Dr. Jack H. van Lint (†)

www.win.tue.nl/∼jlint

Dieser Artikel ist eine gekürzte Version des Artikels gleichen Titels, welcher in dem Buch Alles Mathematik [1] *erschien und welchen wir hier mit freundlicher Genehmigung des Vieweg Verlags abdrucken dürfen. Das Buch* Alles Mathematik*, welches eine Vielzahl weiterer interessanter populärwissenschaftlicher Artikel über mathematische Anwendungen enthält, erscheint 2008 in der dritten Auflage. Der Autor Jack van Lint ist leider am 28. September 2004 verstorben, s.* www.win.tue.nl/∼jlint.

Zusammenfassung

Jeder verwendet heute ganz selbstverständlich Compact Discs. Warum ist aber die Musikübertragung auf einer CD reiner als auf einer herkömmlichen Schallplatte? Die Antwort lautet, um einen populären Slogan abzuwandeln: There is mathematics inside! Genauer gesagt, ein Zweig der Diskreten Mathematik, nämlich die Theorie der Fehler korrigierenden Codes. In diesem Artikel soll über die Anwendung solcher Codes auf das Design des Compact-Disc-Audio-Systems berichtet werden, das von Philips Electronics und Sony entwickelt wurde.

Wörter und Codes

Wir werden das folgende leicht verständliche mathematische Konzept verwenden. Betrachten wir zwei n-Tupel $\boldsymbol{a} = (a_1, a_2, \ldots, a_n)$ und $\boldsymbol{b} = (b_1, b_2, \ldots, b_n)$, wobei die a_i und b_i aus einer Menge Q, genannt das *Alphabet*, stammen. Der *Hamming-Abstand* von $\boldsymbol{a}$ und $\boldsymbol{b}$ ist definiert durch

$$d(\boldsymbol{a}, \boldsymbol{b}) = \text{ Anzahl der } i \text{ mit } a_i \neq b_i.$$

Wir nennen $\boldsymbol{a}$ und $\boldsymbol{b}$ *Wörter der Länge* n über dem Alphabet Q.

Die Analogie dieses Begriffes „Wörter“ zur gewöhnlichen Sprache ist absichtlich. Nehmen wir an, wir lesen ein Wort (sagen wir, in einem Buch auf Deutsch) und bemerken, dass das Wort zwei Druckfehler enthält. Dann heißt das in unserer Terminologie, dass das gedruckte Wort Hamming-Abstand 2 zum korrekten Wort hat. Der Grund, warum wir die 2 Druckfehler erkennen, liegt darin, dass in der deutschen Sprache nur ein einziges Wort mit so einem kleinen Abstand zum gedruckten (fehlerhaften) Wort existiert. Das ist auch schon das grundlegende Prinzip der Fehler korrigierenden Codes: Entwirf eine Sprache (genannt *Code*) von Wörtern einer festen Länge über einem Alphabet Q, so dass je zwei Codewörter mindestens Abstand $2e + 1$ haben. Diese Größe $2e + 1$ heißt dann der Minimalabstand des Codes. Offensichtlich ist ein Wort, das höchstens e Fehler enthält, näher zum korrekten Wort als zu jedem anderen Codewort.

Ein einfaches Beispiel

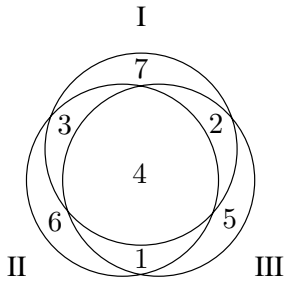
Ein Beispiel solch codierter Information ist jedem von uns geläufig: Es sind die *Strich-Codes*, mit denen heute die meisten Produkte versehen sind. Zunächst wird jedes Produkt mit einer Folge von 12 Zahlen (aus 0 bis 9) identifiziert. Jede Zahl wird ersetzt durch ein Codewort mit sieben 0'en und 1'en. Auf dem Produkt werden 0 und 1 durch einen dünnen Strich dargestellt: 0 durch einen weißen Strich und 1 durch einen schwarzen Strich. Zum Beispiel wird 5 codiert durch 0110001, so dass auf dem Produkt ein weißer Strich erscheint, eine Einheit breit, gefolgt von einem schwarzen Strich, zwei Einheiten breit, einem weißen Strich, drei Einheiten breit, und einem abschließenden schwarzen Strich. Wird nun das Produkt an der Kasse des Supermarktes gescannt, so kann es passieren, dass ein Bit (d.h. eine 0 oder 1) falsch gelesen wird. Die Codewörter für die Zahlen und für die Produkte sind so gewählt, dass der Scanner den Fehler entdeckt und der Kassiererin ein Signal gibt, den Scan-Vorgang zu wiederholen. Jeder von uns hat dies sicher schon des öfteren erlebt.

Einer der frühesten Erfolge von Fehler korrigierenden Codes war die Qualität der Bilder von Satelliten,

z.B. vom Mars. Wären diese Bilder ohne die Verwendung von Codes zur Erde übertragen worden, so wäre überhaupt nichts zu erkennen gewesen. Die Kontrollstationen hätten nur einen unverständlichen Bildsalat erhalten (in der Fachsprache: random noise).

Strich-Code und Viking-Aufnahme vom Mars (Jet Propulsion Laboratory of the California Institute of Technology)

Der erste effiziente Fehler korrigierende Code wurde von R. W. Hamming 1948 in den Bell Laboratories entworfen. Eine binäre Folge (d.h. eine Folge von 0 und 1) wurde in Blöcke zu je 4 Bits zerlegt. Hammings Idee war es, jedem 4-Block drei *Korrekturbits* anzuhängen, mit Hilfe der folgenden Regel. Betrachten wir das folgende Diagramm:



Die 4 Bits (z.B. 1101) werden in die Teile mit den Nummern 1 bis 4 geschrieben. Die Korrekturbits kommen von den Teilen 5 bis 7. Die Regel ist, dass jeder Kreis eine *gerade* Anzahl von 1'en enthalten muss. Eine Konfiguration von 7 Bits, welche *einen* Fehler enthält, ergibt einen oder mehrere Kreise mit einer ungeraden Anzahl von 1'en. Das Bit, welches in all diesen Kreisen mit ungerader Parität enthalten ist, aber nicht in jener mit gerader Parität, muss dann das fehlerhafte Bit sein.

Wir sehen in diesem Beispiel, dass jedes Wort der Länge 7 genau 4 Informationsbits enthält. Daher sagen wir, dass der Hamming-Code ein binärer $[7, 4]$-Code ist mit *Informationsrate* $\frac{4}{7}$. Wir bemerken, dass die einfache Wiederholung der 4 Informationsbits einen $[8, 4]$-Code ergibt, also einen Code mit Informationsrate $\frac{4}{8} = \frac{1}{2}$ (schlechter als im Hamming-Code). Und dieser Code kann zwar einen Fehler entdecken, aber nicht korrigieren.

Von Musik zu Audiobits

Bevor wir uns den Codes zuwenden, wollen wir besprechen, wie Musik in eine Folge sogenannter Audiobits umgewandelt wird. Im CD-System wird das Analogsignal 44.100 Mal pro Sekunde bemustert (in der Fachsprache „sampled"). Diese Rate von 44,1 kHz ist ausreichend, um Frequenzen bis zu 20.000 Hz hören zu können. Die Muster werden uniform in 16 Bits zerlegt, und da wir Stereo-Musik empfangen wollen, so entspricht jedes Muster einer Folge von 32 Bits. So eine Folge von 32 Bits wird aufgefasst als 4 aufeinanderfolgende Bytes (ein Byte ist eine Folge von 8 Bits). Für den Codierungsvorgang werden diese Bytes als Elemente das Körpers $\mathbb{F}_{2^8}$ aufgefasst. Wer nicht weiß, was ein (mathematischer) Körper ist: In einem Körper gelten die üblichen Rechenregeln, d.h. wir können Bytes addieren, multiplizieren und dividieren, wie wir es gewohnt sind.

Genauso wie bei den Hamming-Codes wird die Folge der Bytes in Gruppen fester Länge zerlegt, an die dann Korrekturbytes angehängt werden. Im CD-System wird eine Folge von 24 Bytes in zwei Schritten zu einem Codewort der Länge 32 Bytes verwandelt. Die dabei verwendeten Codes sind sogenannte *Reed-Solomon-Codes* vom Typ $[28, 24]$ bzw. $[32, 28]$, die im nächsten Abschnitt erklärt werden. Schließlich wird noch ein Extrabyte hinzugefügt, welches die Kontroll- und Anzeigeinformation enthält. So kann der CD-Player (und der Hörer) erkennen, auf welcher Spur sich die CD gerade befindet. Also: Sechs Muster führen zu 33 Datenbytes, von denen jedes aus 8 Bits besteht.

Eine Sekunde Musik führt zu einer Folge von 4.321.800 Bits auf der Tonspur. Falls der CD-Player ein Bit mit der sehr kleinen Wahrscheinlichkeit von $\frac{1}{10.000}$ falsch interpretieren würde, so würden immer noch hunderte Fehler pro Sekunde passieren!

Es gibt mehrere Gründe für Fehler auf einer CD. Es könnte Schmutz auf der CD sein, Luftblasen im Plastikmaterial, Ungenauigkeiten beim Druck, Fingerabdrücke, Kratzer, Oberflächenfehler. Es sei angemerkt, dass Fehler hauptsächlich hintereinander auftreten (sogenannte „burst errors"). Da wir Codes verwenden, die *zufällige* Fehler korrigieren, so erfordert dies ein systematisches Austauschen von Bytes, damit Bytes, die in den Codewörtern aufeinanderfolgen, nicht mehr benachbart sind auf der CD.

Reed-Solomon-Codes

Das Codierungssystem, das auf einer Compact Disc verwendet wird, hat den Fachnamen CIRC (Cross-Interleaved Reed-Solomon-Code). Der Inhalt dieses Abschnittes sind die mathematischen Prinzipien, die in der Fehlerkorrektur auf CDs eine Rolle spielen. Wie wir gesehen haben, besteht das Code-Alphabet auf der CD aus $2^8 = 256$ Buchstaben, welche den Körper $\mathbb{F}_{2^8}$ bilden. Um die etwas komplizierten Rechnungen in $\mathbb{F}_{2^8}$ zu vermeiden, illustrieren wir die Arbeitsweise der Reed-Solomon-Codes an einem Beispiel über einem Alphabet mit 31 Elementen, d.h. über dem Körper $\mathbb{F}_{31}$.

Die Elemente von $\mathbb{F}_{31}$ sind $0, 1, 2, \ldots, 30$ und die Rechenoperationen werden wie üblich ausgeführt, wobei das Ergebnis jeweils modulo 31 reduziert wird. Das heißt: Wollen wir z.B. 6 und 9 multiplizieren, so erhalten wir $6 \times 9 = 54$ und dividieren anschließend das Resultat durch 31. Das „Produkt“ 6×9 in $\mathbb{F}_{31}$ ist dann der Rest 23

$$6 \times 9 = 54 = 31 + 23.$$

Wir schreiben kurz: $6 \times 9 = 23$ (modulo 31).

Als ein Beispiel betrachten wir ein Buch mit 200 Seiten, das beschrieben werden soll mit einer Druckgröße, die 3000 Symbole pro Seite erlaubt. Wir identifizieren die Buchstaben a bis z mit den Zahlen 1 bis 26, den Zwischenraum mit 0, und benutzen 27 bis 30 für Punkt, Komma, Strichpunkt und ein weiteres Symbol.

Wir werden vom Drucker informiert, dass die Technik, die er benutzt, nicht allzu gut ist. Tatsächlich ist sie so schlecht, dass jedes Symbol auf einer Seite mit Wahrscheinlichkeit $0{,}1\,\%$ inkorrekt ist. Das bedeutet, dass im Durchschnitt 3 Druckfehler pro Seite passieren, was eindeutig zu viel ist. Man bedenke nur, wie lästig schon ein gelegentliches Klopfen auf der Schallplatte ist. Um die Situation zu verbessern, verwenden wir Ideen aus der Codierungstheorie.

In einem ersten Ansatz zerlegen wir die Folge der Symbole in Gruppen zu je 4. Vor jeder Gruppe von 4 Symbolen werden zwei 2 Korrektursymbole eingefügt. Wir bezeichnen das resultierende Codewort mit $\boldsymbol{a} = (a_0, a_1, \ldots, a_5)$, wobei a_0 und a_1 die Korrektursymbole und a_2, a_3, a_4, a_5 die 4 Informationssymbole sind (alles in $\mathbb{F}_{31}$).

Betrachten wir das Word CODE. Dieses Wort korrespondiert zu $(a_2, a_3, a_4, a_5) = (3, 15, 4, 5)$. Die Codierungsregeln, um a_0 und a_1 zu berechnen, sind nun:

(i) $a_0 + a_1 + a_2 + a_3 + a_4 + a_5 = 0$ (modulo 31)

(ii) $a_1 + 2a_2 + 3a_3 + 4a_4 + 5a_5 = 0$ (modulo 31).

Daraus erhalten wir $a_0 = 3$ und $a_1 = 1$, so dass CODE als CACODE codiert wird.

Gehen wir zum anderen Ende und nehmen wir an, wir erhalten EPGOOF. Wenn wir die Korrektursymbole auslassen, so würde GOOF als gesendetes Word resultieren. Aber wir sehen sofort, dass ein Fehler passiert sein muß, denn EPGOOF korrespondiert zu $(a_0, a_1, \ldots, a_5) = (5, 16, 7, 15, 15, 6)$ und die Summe der a_i ist

$$5 + 16 + \ldots + 6 = 64 = 2 \quad \text{(modulo 31)},$$

was unserer Codierungsregel widerspricht. Wir vermuten, dass wahrscheinlich eines der Symbole a_i durch $a_i + 2$ ersetzt wurde. Der Wert des Fehler in a_i ist 2, das heißt wir werden $2i$ als Fehlerwert erhalten, wenn wir $a_0, \ldots, a_5$ in Gleichung (ii) einsetzen. Einsetzen ergibt (modulo 31)

$$\begin{aligned} a_1 + 2a_2 + \ldots + 5a_5 &= 16 + 14 + 45 + 60 + 30 \\ &= 165 = 10, \end{aligned}$$

also ist $i = \frac{10}{2} = 5$, das heißt der Fehler passierte in der fünften Position. Anstelle von 6 sollte 4 stehen — und wir decodieren EPGOOF in GOOD! Wie schon gesagt, die arithmetischen Operationen in $\mathbb{F}_{2^8}$ sind ein wenig komplizierter, aber alle Reed-Solomon-Codes verwenden Codierungs- and Decodierungsregeln wie die Gleichungen (i) und (ii).

Wenn wir nun auch wissen, dass dieser Code einen Fehler in einer 6-Gruppe korrigieren kann, so müssen wir ehrlicherweise auf ein Problem eingehen, das auch auf der CD auftritt. Der Code hat eine Informationsrate von $\frac{4}{6} = \frac{2}{3}$. Da wir dieselbe Anzahl von Seiten derselben Größe benutzen wollen, so müssen wir also 1½-mal so viele Symbole pro Seite unterbringen. Zu unserer Ernüchterung teilt uns aber der Drucker mit, dass die Druckfehlerwahrscheinlichkeit der 1½-mal kleineren Druckgröße *doppelt so hoch* ist! Mit anderen Worten: Mit dem kleineren Font müssen wir im Durchschnitt 9 Fehler pro Seite *vor der Fehlerkorrektur* in Kauf nehmen. Und nach der Korrektur? Ein Wort mit 6 Symbolen wird korrekt decodiert, falls es höchstens einen Fehler enthält. Die Wahrscheinlichkeit, dass mehr als ein Fehler passiert ist

$$\sum_{i=2}^{6} \binom{6}{i} (0{,}002)^i (0{,}998)^{6-i} \approx 0{,}00006\,.$$

Im Buch wird es einige Wörter geben, die zwei oder mehr Druckfehler enthalten, aber es werden im Schnitt nicht mehr als 10 Wörter im gesamten Buch sein — eine bemerkenswerte Verbesserung!

Mit einer geringfügigen Veränderung wird die Situation dramatisch besser. Dazu verwenden wir einen etwas komplizierteren Reed-Solomon-Code. Wir zerlegen die Symbole in Gruppen von 8, die wir mit $(a_4, a_5, \ldots, a_{11})$ bezeichnen. Davor fügen wir 4 Korrekturzeichen hinzu nach den folgenden Regeln:

(i) $a_0 + a_1 + a_2 + a_3 + \ldots + a_{11} = 0$ (modulo 31)

(ii) $a_1 + 2^k a_2 + 3^k a_3 + \ldots + 11^k a_{11} = 0$ (modulo 31) $(k = 1, 2, 3)$.

Die Codierung besteht also aus der Lösung von 4 Gleichungen mit den 4 Unbekannten a_0, a_1, a_2, a_3. Der Leser kann sich sofort wie oben überlegen, wie ein Fehler korrigiert wird. Nehmen wir an, es passierten zwei Fehler mit den Abweichungen e_1 und e_2 in Positionen i und j. Einsetzen in die Gleichungen (i) und (ii) ergibt sofort

$e_1 + e_2$ (modulo 31) und

$$i^k e_1 + j^k e_2 \text{ (modulo 31)} \quad (k = 1, 2, 3).$$

Daraus können wir e_1 und e_2 eliminieren und erhalten eine quadratische Gleichung mit i und j als Lösungen. Danach können wir e_1 und e_2 ermitteln und die beiden Fehler korrigieren. Der neue Code ist also ein 2-Fehler korrigierender Code.

Glücklicherweise ergibt das kein neues Problem beim Drucken. Dieser neue Reed-Solomon-Code hat wiederum Informationsrate $\frac{2}{3}$ $(= \frac{8}{12})$ wie der vorige, so dass die Druckfehlerwahrscheinlichkeit wieder $= 0{,}2\,\%$ pro Symbol ist. Die Decodierung wird nur versagen, wenn ein gedrucktes Wort von 12 Symbolen mehr als zwei Fehler enthält. Die Wahrscheinlichkeit, dass dies passiert, ist

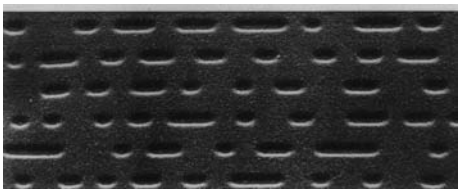
$$\sum_{i=3}^{12} \binom{12}{i} (0{,}002)^i (0{,}998)^{12-i} < 0{,}000002.$$

Das gesamte Buch hat 600.000 Symbole, also 75.000 Wörter der Länge 8. Da $75.000 \times 0{,}000002 = 0{,}15$ ist, so ist es unwahrscheinlich, dass überhaupt ein Druckfehler auftritt! Das ist nun tatsächlich eine eindrucksvolle Verbesserung!

Auf einer CD ist die Informationsrate, wie im letzten Abschnitt erwähnt, $\frac{24}{32} = \frac{3}{4}$. Wie eben erläutert (am Beipiel des kleineren Fonts) wird die Fehlerwahrscheinlichkeit für ein Bit dadurch erhöht im Vergleich zur Situation, wenn keine Korrektur durchgeführt wird. Wie wir schon erwähnt haben, kann eine CD ohne weiteres 500.000 Bitfehler enthalten. Allerdings passieren diese Fehler nicht zufällig (wie in unserem Beispiel des Buches), sondern meistens in „bursts“ (hintereinander). Es kann vorkommen, dass einige 1000 aufeinanderfolgende Symbole (z.B. durch einen Kratzer) fehlerhaft sind. Dem wird entgegengewirkt, indem benachbarte Informationssymbole in *verschiedenen* Codewörtern erscheinen (genannt „Interleaving“). Eines muss man sich außerdem beim Design eines CD-Systems vor Augen halten: Der Speicher, der für die Bits zur Verfügung steht, kann nicht zu groß sein. Das impliziert natürlich Beschränkungen bei der Länge der Codewörter und beim Interleaving. Alle Berechnungen, die zur Fehlerkorrektur benötigt werden, passieren im Bruchteil einer Sekunde, und das ist der Grund, warum wir Musik hören, praktisch unmittelbar, nachdem der CD-Player eingeschaltet wird. In dem Buch-Beispiel sahen wir, dass ein längerer Code zwar bessere Korrekturleistung bringt, aber auch mehr Berechnungen benötigt.

Die Compact Disc

Die Musik wird auf einer Compact Disc in Digitalform als eine 5 km lange spiralförmige Spur aufgezeichnet, welche aus einer Folge von sogenannten *Pits* besteht. Die Teile zwischen aufeinanderfolgenden Pits heißen *Lands*. Die Steigung der Spur ist $1{,}6\,\mu$m, die Breite ist $0{,}6\,\mu$m und die Tiefe der Pits ist $0{,}12\,\mu$m. Die Abbildung zeigt eine Vergrößerung mehrerer paralleler Spuren.



Vergrößerung mehrerer paralleler Spuren einer CD

Die Spur wird optisch von einem Laserstrahl gescannt. Jeder Land/Pit oder Pit/Land Übergang wird als 1 interpretiert, und alle anderen Bits als 0. So wird z.B. ein Pit der Länge $1{,}8\,\mu$m gefolgt von einem Land der Länge $0{,}9\,\mu$m übersetzt in die Folge 100000100. Der Durchmesser des Lichtstrahls ist ungefähr $1\,\mu$m. Wenn er auf ein Land fällt, so wird er fast völlig reflektiert. Da die Tiefe eines Pits ungefähr ¼ der Wellenlänge des Lichtes im Material der Disc ist, so bewirkt Interferenz, dass weniger Licht von den Pits in die Öffnung des Objektives reflektiert wird.

Es gibt mehrere Anforderungen an die Folge der Bits, die in der CD beachtet werden müssen. Jedes Pit oder Land muss mindestens 3 Bits lang sein. Das wird gefordert, um zu vermeiden, dass der Lichtstrahl zwei aufeinanderfolgende Übergänge zwischen Pits und Lands gleichzeitig registriert, was eine sogenannte Intersymbol-Interferenz bewirken würde. Jedesmal wenn ein Übergang auftritt, wird die „Bit-Uhr“ im CD-Player synchronisiert. Da dies so rechtzeitig erfolgen muss, um einen Verlust an Synchronisation zu vermeiden, darf kein Pit oder Land länger als $3{,}3\,\mu$m sein, d.h. zwei 1'en in der Sequenz sind durch höchstens 10 0'en getrennt. Die Einschränkung an die maximale Landlänge ist außerdem notwendig für den Mechanismus, der den Laser auf der Spur hält. Eine letzte Anforderung ist, dass der Niedrig-Frequenz Anteil des Signals auf ein Minimum beschränkt ist. Dies bedingt, dass die Gesamtlänge der Pits und Lands von Beginn der Spur an ungefähr gleich ist. Diese Bedingung wird für den Mechanismus benötigt, der entscheidet, ob etwas „hell“ oder „dunkel“ ist. (Das vermindert auch den negativen Effekt eines Fingerabdruckes auf der CD.)

Jedes Byte in einem Codewort wird in 17 sogenannte Kanalbits auf der CD umgewandelt. Wenn die CD gelesen wird, so werden diese Bits zurück verwandelt in Bytes. Dann folgt De-Interleaving, Fehlerkorrektur, Digital-zu-Analog Konversion und schließlich hören wir Musik, mit Dank an Mozart und andere — und an die *Mathematik*.

Literaturverzeichnis

[1] M. Aigner und E. Behrends, *Alles Mathematik*, Vieweg, Braunschweig, Wiesbaden, 3. Auflage, 2008.

Wie rechnen Quanten?

Prof. Dr. Ehrhard Behrends
Fachbereich Mathematik und Informatik
Freie Universität Berlin
Arnimallee 2 – 6
14195 Berlin

behrends@math.fu-berlin.de

Dieser Artikel ist eine gekürzte Version des Artikels gleichen Titels, der in dem Buch Alles Mathematik [1] *erschien und den wir hier mit freundlicher Genehmigung des Vieweg Verlags abdrucken dürfen. Das Buch* Alles Mathematik*, das eine Vielzahl weiterer interessanter populärwissenschaftlicher Artikel über mathematische Anwendungen enthält, erscheint 2008 in der dritten Auflage.*

Die Idee, Gesetze der Quantenmechanik für den Bau von Computern mit neuen Eigenschaften zu bauen, geht auf den Physiker *R. P. Feynman* zurück: 1982 schlug er vor, für die komplizierten Rechnungen im Zusammenhang mit Elementarteilchen-Modellen eigens dafür konzipierte Rechner zu verwenden. Diese noch sehr vage Idee wurde von *D. Deutsch* aufgegriffen, der Ende der achziger Jahre ein theoretisches Modell — also so etwas wie einen möglichen Bauplan — für einen Quantencomputer entwarf.

All das war allerdings nur wenigen Spezialisten bekannt. Gewaltiges Aufsehen erregten erst Arbeiten von *P. Shor*. Der konnte zeigen: Wenn es gelingt, einen funktionierenden Quantencomputer zu bauen, dann sind gewisse Verfahren der Kryptographie, die heute als absolut sicher gelten, relativ schnell zu überlisten.

Peter Shor bekam für diese Leistung 1998 auf dem Weltkongress der Mathematiker in Berlin den Nevanlinna-Preis, das ist die mit Abstand höchste Auszeichnung, die man für Arbeiten aus dem Bereich Informatik/Mathematik bekommen kann. Nach allgemeiner Überzeugung hat er diesen Preis auch verdient, denn seit seinen grundlegenden Arbeiten ist wirklich ein neues Kapitel physikalischer Forschung entstanden. Probleme der Quantencomputer, Quantenkryptographie und der Quanteninformation werden heiß diskutiert, im Umfeld gibt es Detailprobleme, die mit viel Geld gefördert werden (Teleportation, Selbstkorrektur von Quantenkanälen, ...).

Trotz aller Anstrengungen gibt es heute noch keinen Quantencomputer, der diesen Namen verdient, und viele meinen sogar, dass das zu unseren Lebzeiten auch nicht passieren wird (wenn überhaupt jemals). Die physikalischen Schwierigkeiten sind immens; sie sollen aber hier nicht diskutiert werden, denn sie können von Fachleuten aus der Physik viel besser dargestellt werden. Ziel des Artikels ist vielmehr ein Teilaspekt des Themas „Quantencomputer“: Welche neue *Mathematik* wird benötigt, um ihre neuen Möglichkeiten voll auszuschöpfen? Es soll versucht werden zu erklären, wie denn ein Angriff à la Shor auf ein sicheres Kryptosystem aussehen könnte; jedenfalls wenn es die Physiker irgendwann einmal schaffen würden, die Schwierigkeiten zu überwinden.

Warum sind Primzahlen in der Kryptographie wichtig?

Im Artikel von M. Meiringer über Kryptographie auf S. 48 wurde gezeigt, dass die Sicherheit des berühmten *RSA-Algorithmus* auf der Schwierigkeit der Faktorisierung großer Zahlen $n = p \cdot q$ beruht. Es gibt zur Zeit kein Verfahren, das wesentlich besser wäre als systematisches Probieren, um aus n auf p und q zu schließen.

> Als ein für die Kryptographie realistisches Beispiel gehen wir davon aus, dass p und q jeweils 200 Stellen haben. Dann hat $n = p \cdot q$ vierhundert Stellen, und Ausprobieren erfordert 10^{200} (eine 1 mit zweihundert Nullen!) Rechenschritte. Für derartige Zahlen gibt es schon keine eigenen Namen mehr. Man kann sich überlegen, dass alle Computer dieser Welt an diesem Faktorisierungsproblem scheitern müssen. Selbst wenn sie hundertmal so schnell rechnen wie heute theoretisch möglich ist. Und selbst wenn sie sich seit Beginn der Welt nur diesem Problem gewidmet hätten.

Allerdings ist das RSA-Verfahren nur so lange sicher, wie niemand in der Lage ist, p und q aus n herauszulesen. Denn wer das könnte, wäre auch in der Lage, die verschlüsselte Nachricht zu decodieren. Oben wurde ausgeführt, warum man heute meint, dass RSA sicher ist. Aber:

Quantencomputer könnten schnell p und q ermitteln!

Dafür hat Peter Shor 1994 ein Verfahren vorgestellt. Trotzdem können alle Kryptographen noch relativ ruhig schlafen, weil funktionierende Quantencomputer in weiter Ferne sind. Die zugrunde liegenden mathematischen Ideen sind aber interessant, und um die soll es im Folgenden gehen.

Eine mathematische Vorbereitung: Periodenlängen

Das Problem besteht darin, die zwei Faktoren einer aus zwei Primzahlen zusammengesetzten Zahl zu finden: Rekonstruiere p, q aus $n := p \cdot q$. Viele Verfahren sind erdacht worden, um das zu erleichtern oder umzuformen. Für unsere Zwecke ist eine Idee wichtig, die das Faktorisierungsproblem in eine andere Fragestellung transformiert, die für klassische Computer haargenau den gleichen Schwierigkeitsgrad hat.

Um sie vorzustellen, muss eine Vokabel eingeführt werden: Was heißt „a *modulo* b“? Es sollen zwei Zahlen a und b gegeben sein, in der Regel ist a viel größer als b. Zunächst teilt man a durch b und schaut sich den Rest an, der beim Teilen übrig bleibt. Diese Zahl, sie liegt zwischen 0 und $b-1$, wird „a modulo b“ genannt. Modulares Rechnen wurde auch im Kryptographie-Artikel (vgl. S. 48) betrachtet.

Dieses „modulo“ spielt nun eine wichtige Rolle. Wir beginnen mit einer Zahl n, die die Form $p \cdot q$ mit Primzahlen p, q hat, als illustrierendes Beispiel denken wir an $n = 15 = 3 \cdot 5$. Jemand gibt nun eine Zahl x vor, die irgendwo zwischen 1 und n liegt. Kann man x dazu verwenden, einen Faktor von n zu finden?

Ideal wäre es, wenn n und x einen von 1 verschiedenen Teiler gemeinsam hätten (wenn also in unserem Beispiel etwa $x = 10$ wäre). Jeder Computer findet den in Bruchteilen von Sekunden, das Verfahren heißt „Euklidischer Algorithmus“ und war schon vor über 2000 Jahren bekannt.

Wir nehmen also an, dass x und n *keinen* Teiler gemeinsam haben, man sagt, dass x und n *teilerfremd* sind. Dann rechnen wir nach und nach die Zahlen

x modulo n,

x^2 modulo n,

x^3 modulo n,

...

aus. Die Zahlentheorie kann beweisen, dass mit Garantie irgendwann einmal die Zahl 1 herauskommt. Wir nennen denjenigen Exponenten r, für den zum ersten Mal x^r modulo n gleich 1 ist, die *Periode* von x.

> In unserem Beispiel $n = 15$ starten wir zur Illustration mit $x = 7$. Die Zahlen n und x sind teilerfremd, wir können also die Periode von 7 ausrechnen.
>
> Dazu müssen wir so lange die Reste von $7, 7^2, 7^3, \ldots$ modulo 15 bestimmen, bis wir erstmals 1 erhalten:
>
> 7 modulo 15 ist gleich 7;
>
> 7^2 modulo 15 ist gleich 4;
>
> 7^3 modulo 15 ist gleich 13;
>
> 7^4 modulo 15 ist gleich 1.
>
> Folglich ist die Periode von $x = 7$ gleich 4.

Was nutzt das? Angenommen, unser x ist so, dass die Periode r eine gerade Zahl ist: $r = 2 \cdot s$. Dann können wir doch die Gleichung[8] $x^r = 1$ unter Verwendung der Abkürzung $y := x^s$ als $y^2 = 1$ bzw. als $(y+1) \cdot (y-1) = 0$ umschreiben. Und „gleich 0 modulo n“ bedeutet Teilbarkeit durch n und damit durch p und q. Wenn man nun noch die Tatsache verwendet, dass eine Primzahl ein Produkt nur dann teilt, wenn es einen der Faktoren teilt, so liefert uns die Kenntnis von y die Kenntnis von p und q. (Ich habe ein bisschen geschummelt: Für die Argumentation ist wichtig, dass $y+1$ nicht Null modulo n ist. Das erklärt den Zusatz in der nachstehenden Definition).

Zusammengefasst können wir also sagen, dass wir aus der Periode von x „mit etwas Glück“ einen Teiler von n bekommen. Wir präzisieren das in der

Definition: Eine Zahl x zwischen 1 und n soll *gut* heißen, wenn x zu n teilerfremd ist, die Periode von x eine gerade Zahl $r = 2 \cdot s$ ist und x^s modulo n *nicht* die Zahl $n-1$ ist.

Bemerkenswerterweise ist es nun so, dass es gute Zahlen im Überfluss gibt[9]. Greift man zufällig eine heraus, so ist sie mit mehr als fünfzig Prozent Wahrscheinlichkeit gut. Das führt zur folgenden **Strategie zur Lösung des Faktorisierungsproblems:**

- Suche mit einem Zufallsgenerator eine Zahl x zwischen 1 und n.
- Mit sehr viel Glück hat sie einen gemeinsamen Teiler mit n, dann ist man fertig. Mit mindestens fünfzig Prozent Wahrscheinlichkeit ist x gut, und dann kann man mit Hilfe der Kenntnis der Periode ebenfalls faktorisieren.
- Sollte man kein Glück gehabt haben, wiederhole man die ersten beiden Schritte. Irgendwann wird es schon klappen, denn laut Wahrscheinlichkeitsrechnung ist die Wahrscheinlichkeit für „Pech“ in k aufeinanderfolgenden Schritten höchstens $0{,}5^k$. Sie müssen schon ein echter Pechvogel sein, wenn es zehnmal schiefgeht, diese Wahrscheinlichkeit ist kleiner als ein Promille.

Für sich genommen ist das eine interessante, aber recht nutzlose Umschreibung des Problems, denn:

> **Für einen klassischen Computer ist das Berechnen der Periode genauso kompliziert wie das Faktorisieren selber!**

(Das Finden von Zufallszahlen x dagegen ist leicht, das gehört heutzutage zu den Standardaufgaben.)

Hier sollen auf spektakuläre Weise Quantencomputer zum Einsatz kommen. Ihre einzige Aufgabe (beim Faktorisierungsproblem) besteht darin, für ein vorgelegtes x die Periode von x zu bestimmen. Alles andere, also das zufällige Erzeugen von x und die weiteren Rechnungen wie etwa die mehrfache Ausführung des euklidischen Algorithmus, kann den klassischen Rechnern überlassen bleiben.

[8] Wir lassen das „modulo“ der Einfachheit halber weg und rechnen so wie mit gewöhnlichen Zahlen; das ist wirklich legitim!

[9] Der Beweis ist elementar, aber etwas länglich.

Etwas Quantenmechanik

Hier wollen wir uns wirklich auf das Notwendigste beschränken. Auch nach 100 Jahren ist die Quantenmechanik immer noch eine Wissenschaft, die den meisten ein Buch mit sieben Siegeln ist. Das ist auch ganz verständlich, denn es handelt sich um ein (hervorragend funktionierendes) Modell der Welt im atomaren Bereich, das nach Gesetzen funktioniert, die der menschlichen Lebenserfahrung total zuwiderlaufen.

Was uns interessiert, kann am folgenden Beispiel demonstriert werden. Wir denken an eine Situation, bei der Teilchen von atomarer Größenordnung betrachtet werden und bei der durch die Versuchsanordnung klar ist, dass genau eine von zwei Möglichkeiten verwirklicht werden kann:

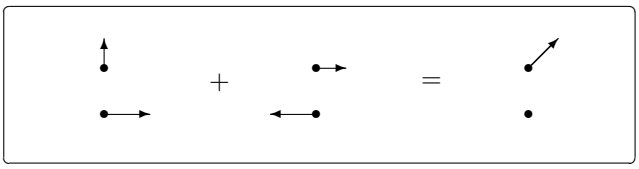
- Ein Photon, das auf eine Glasplatte schräg auftrifft, kann hindurchgehen oder gebrochen werden;
- ein Elektron kann bei einer Messung einen Spin „up“ oder „down“ haben;
- ein Teilchen (etwa ein zu einem Atomkern gehörendes Elektron) kann genau eines von zwei Energieniveaus einnehmen.

Wir wollen für den Augenblick die möglichen Ergebnisse A und B nennen. Fundamental ist dann die Feststellung, dass das Weitestgehende, das sich theoretisch aussagen lässt, *wahrscheinlichkeitstheoretische Aussagen* sind: Physiker können eine Zahl a zwischen 0 und 1 berechnen, so dass die Wahrscheinlicheit für eine A-Messung gleich a (und folglich die für eine B-Messung gleich $1-a$) ist. Es ist also — etwas unwissenschaftlich ausgedrückt — so, als ob ein Photon kurz vor Erreichen der Glasplatte würfelt, ob es nun hindurchgehen will oder lieber reflektiert werden möchte. Auf die philosophischen Probleme in diesem Zusammenhang können wir hier natürlich nicht eingehen.

Die Wahrheit ist etwas komplizierter, leider benötigen wir gleich diese Verfeinerung. Sie besagt:

> Man stelle sich A und B als Punkte der Ebene vor. An A und B denke man sich „Pfeile“ angebracht, die eine beliebige Richtung haben können. Einzige Bedingung: Misst man die Längen l_A und l_B dieser Pfeile, so muss die Zahl $l_A^2 + l_B^2$ den Wert 1 ergeben.
>
> Die Wahrscheinlichkeit, A zu messen, ist dann gerade die Zahl l_A^2.

Zwei Ergänzungen sind nun noch wichtig. *Erstens* ist es so, dass dieses Pfeil-Bild den Zustand *vor* der Messung beschreibt. Wird zum Beispiel gemessen, dass A eingetreten ist, so verändert sich der Zustand schlagartig: Es gibt nun einen Pfeil der Länge 1 bei A, und der bei B ist verschwunden. Aus dem Blickwinkel der Quantencomputer ist es bedauerlicherweise so, dass „Messung“ sehr weit interpretiert werden muss, jede Wechselwirkung mit anderen Systemen hat die gleiche Auswirkung wie eine Messung. Und *zweitens* kann es so etwas wie eine *Überlagerung* geben. Das muss man sich so vorstellen, dass man manchmal nicht weiß, welches von zwei A-B-Pfeildiagrammen für die Beschreibung einer Situation das richtige ist, zum Beispiel, weil ein Photon sich für einen von zwei Spalten zum Durchgehen entschieden hat, wir aber nicht wissen, für welchen. Dann kommt es zur Überlagerung, das richtige Modell entsteht dann so, dass man die Pfeile der einzelnen Modelle per Vektoraddition zusammensetzt (und hinterher den Maßstab noch so abändert, dass die Summe der Längenquadrate wieder Eins ist). Da sich Vektoren je nach Richtung verstärken, abschwächen oder sogar ganz auslöschen können, kommt es zu merkwürdigen Phänomenen, die klassisch nicht erklärbar sind.



Qbits: Die Bausteine eines Quantencomputers

Der Ausgangspunkt ist ganz einfach, wir knüpfen an den vorigen Abschnitt an: Grundbaustein eines Quantencomputers ist eine physikalische Situation, die bei Messung genau eines von zwei Ergebnissen produziert. Eben noch haben wir sie A und B genannt, ab jetzt sollen sie 0 und 1 heißen. Man spricht dann von einem **Qbit**. (Der Name soll natürlich daran erinnern, dass der Grundbaustein eines klassischen Computers ein Bit ist, also eine Einheit, die die Werte 0 und 1 annehmen kann.)

Der wesentliche Unterschied ist der folgende: Ein **Bit** ist in einem der Zustände 0 oder 1. Definitiv. Ein **Qbit** dagegen ist mit einer gewissen Wahrscheinlichkeit in 0 bzw. 1, die einzelnen Wahrscheinlichkeiten werden durch die Länge der Pfeile bei 0 bzw. 1 bestimmt. Nur durch eine Messung können wir den Zustand erfahren, der dann aber unwiederbringlich verändert wurde. Rein formal gesehen ist ein Bit ein spezielles Qbit, ein Bit im Zustand 0 etwa entspräche einem Qbit, bei dem 0 einen Pfeil der Länge Eins trägt (und der Pfeil bei 1 verschwindet).

Nun kann man mit einem Qbit recht wenig anfangen, wir brauchen viele. Die Lösung kann nicht darin bestehen, einfach einzelne Qbits nebeneinanderzupacken, man möchte auch noch die Wechselwirkungen ausnutzen.

Nehmen wir etwa zwei Qbits, Q1 und Q2. Sind beide im Zustand 0, so wollen wir den Gesamtzustand mit 00 bezeichnen, analog sind die Zustände 01, 10

und 11 zu verstehen. Überlassen wir beide sich selber, so wird irgendeiner dieser gemeinsamen Zustände vorliegen, wir wissen aber nicht, welcher. Wieder ist es so, dass nur Wahrscheinlichkeiten vorausgesagt werden können. Diesmal sind vier Pfeile — je einer für $00, 01, 10, 11$ — vorzuschreiben. Die Längen, zum Quadrat genommen, müssen sich zu Eins summieren, diese Quadrate stehen für Wahrscheinlichkeiten. Hat etwa der Pfeil bei 01 die Länge $0{,}7$, so werden wir mit Wahrscheinlichkeit $0{,}7 \cdot 0{,}7 = 0{,}49$ (das sind 49 Prozent) Q1 im Zustand 0 und Q2 im Zustand 1 messen.

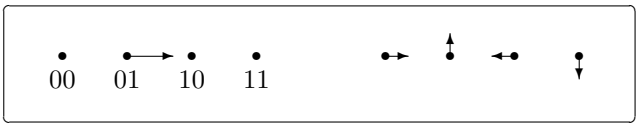
Mit wachsender Anzahl der Qbits sind nun immer mehr Pfeile zu beherrschen. Genauer: Bei L Qbits spielen 2^L Pfeile eine Rolle. Es ist dieser Punkt, der zu den gewaltigen Möglichkeiten führt, man könnte es als massive Parallelität des Rechners interpretieren[10].

Nach diesen Vorbereitungen können wir das **Anforderungsprofil** für einen Quantencomputer grob spezifizieren.

Forderung 1: Der Quantencomputer muss über L Qbits verfügen. Für die Anwendung, die wir planen, ist $L = 2.000$ eine realistische Größenordnung. Es muss möglich sein, diese Qbits beliebig zu initialisieren: Denken wir uns beliebige 2^L Pfeile aus, für die die quadrierten Längen sich zu Eins summieren, so muss es möglich sein, den Computer in einen Zustand zu bringen, dass die 2^L möglichen Zustände durch genau diese Pfeile charakterisiert sind.

Ein Computer, in denen man nur etwas einlesen kann, ist noch ziemlich uninteressant, daher folgt nun die

Forderung 2: Es muss möglich sein, an diesen Qbits im Wesentlichen die gleichen Operationen vorzunehmen wie an gewöhnlichen Bits: Negation, Konjunktion, Disjunktion, … (Das ist ein heikler Punkt, denn man kann aus theorie-inhärenten Gründen einen Quantenzustand nicht kopieren, ohne vorher eine Messung vorzunehmen, durch die die vorher mühsam produzierte Überlagerung der einzelnen Zustände schlagartig aufgehoben würde.) Es kommt noch eine spezielle Forderung hinzu, die für unsere Zwecke unerlässlich ist. Der Quantencomputer muss die diskrete *Fouriertransformation* beherrschen. Vereinfacht ausgedrückt geht es darum, mittels einer rechner-internen Operation — die die Überlagerungen nicht zerstört — einen bei einem Zustand stehenden Pfeil in eine Pfeilfamilie zu verwandeln. Dabei wird der Ausgangspfeil hergenommen, stark verkürzt und unter gewissen Drehwinkeln an die einzelnen Zustände angehängt. Nachstehend sehen wir einen Ausgangszstand (er ist deterministisch beim Zustand 01) und daneben seine diskrete Fouriertransformation; oberflächlich gesehen wird zunächst garantiert Zustand 01 bei einer möglichen Messung produziert, und nach Transformation sind alle Zustände gleichwahrscheinlich.



Das Wichtigste aber ist, dass wir auch noch die Richtungen der Pfeile kontrollieren können, das wird gleich entscheidend sein.

Als letztes bestehen wir noch auf

Forderung 3: Es muss möglich sein, die einzelnen Operationen so durchzuführen, dass Überlagerung gewährleistet ist, dass sich also die Wahrscheinlichkeitspfeile entsprechend der Vektoraddition zusammensetzen. Sind also zum Beispiel zu Beginn nur zwei Zustände Z1 und Z2 — beschrieben durch die Pfeile P1 (bei Z1) und P2 (bei Z2) — möglich und werden mit Z1 und Z2 Rechnungen durchgeführt, die zu Ergebnissen E1 und E2 führen, so soll der Rechner nach der Rechnung in einem Gesamtzustand sein, für den die Pfeil-Längen und -Richtungen an den einzelnen möglichen Zuständen aus P1 und P2 sowie aus den zu E1, E2 gehörigen Pfeilen mittels Vektoraddition entstanden sind.

Wie faktorisiert man mit einem Quantencomputer große Zahlen?

Nun können wir die Idee von Shor in den Grundzügen nachvollziehen. Wir erinnern daran, dass wir für eine Zahl $n = p \cdot q$ allein aus dem n die Zahlen p und q finden wollen und dass es reicht, zu einem x zwischen 1 und n die Periode zu berechnen. Das geht nach Shor so:

1. **Schritt:** Die Zahl n ist gegeben, sie soll mit L Ziffern im Zweiersystem darstellbar sein (hat n im Zehnersystem zum Beispiel 90 Stellen, so führt das — da 10^3 ungefähr 2^{10} ist — auf etwa 300 Stellen im Dualsystem). Verschaffe Dir einen Quantencomputer mit $3L$ Qbits. Er soll die im vorigen Abschnitt beschriebenen Eigenschaften haben.
2. **Schritt:** Organisiere den Computer so: Die ersten $2L$ Qbits sollen „das erste Register“ heißen, die letzten L Qbits taufe man als „das zweite Register“.
3. **Schritt:** Suche ein zufälliges x zwischen 1 und n. Das können wir einem klassischen Computer übertragen.
4. **Schritt:** Präpariere den Computer: Zu allen Zuständen der Form $x \cdots x0 \cdots 0$ (irgendwelche Nullen und Einsen in den ersten $2L$ Qbits, nur Nullen im zweiten Register) gehört ein nach rechts zeigender Pfeil, und alle haben die gleiche Länge.

[10] In gewisser Weise führen L Qbits zu einem Computer, der 2^L Zahlen gleichzeitig verarbeiten kann, allerdings wird jede einzelne nur mit einer gewissen Wahrscheinlichkeit eine Rolle spielen. Dieses exponentielle Ansteigen von 2^L mit wachsendem L ist kaum vorstellbar, man kommt sonst damit selten in Berührung. Höchstens einmal bei Kettenbriefen oder beim Wundern über die Fabel vom Schachbrett und den Reiskörnern.

5. Schritt: Verändere den Zustand des Computers auf folgende Weise. Zur Zeit sind doch alle Zustände der Form $a0\cdots0$ gleichwahrscheinlich, wo a den Zustand des ersten Registers bezeichnet. Das soll in einen Gesamtzustand übergehen, bei dem die Wahrscheinlichkeit für $a0\cdots0$ auf ay übertragen wird, wobei y — ein Wert im zweiten Register — für „x^a modulo n, geschrieben im Zweiersystem“ steht. Das ist sinnvoll, da mit n auch y höchstens L Dualziffern hat.

***Zusammen*:** Der Computer ist jetzt in einem Zustand, wo man bei einer Messung im ersten Register ein a und im zweiten das zugehörige x^a modulo n finden würde. Dabei kommen alle möglichen a mit gleicher Wahrscheinlichkeit vor, und immer noch zeigen alle Wahrscheinlichkeitspfeile nach rechts.

6. Schritt: Das ist wohl der entscheidende Schritt. Jetzt müssen Fouriertransformation und Überlagerung gleichzeitig ablaufen, ohne sich zu stören. Nur so kann der gewünschte Effekt erzielt werden, der auf der folgenden Idee beruht.

Erstens entstehen doch bei der Fouriertransformation Wahrscheinlichkeitspfeile, die — je nach Zustand — in alle möglichen Richtungen zeigen. Zweitens werden sie nach dem Gesetz des Kräfteparallelogramms überlagert. Und drittens gilt doch: Zeigen „sehr viele“ Pfeile in verschiedene Richtungen, so ist der resultierende Pfeil sehr klein[11]. Nur dann, wenn die Pfeile alle in die gleiche Richtung zeigen, gibt es eine bemerkenswerte Resultierende.

Im vorliegenden Fall werden nun die Zustände im ersten Register einer Fouriertransformation unterworfen. Es ist dann so, dass nur solche Zustände by — mit einem b der Länge $2L$ aus dem ersten Register und einem y der Länge L aus dem zweiten — einen von Null verschiedenen Wahrscheinlichkeitspfeil haben, wenn $r\cdot b$ ein Vielfaches von 2^{2L} ist. Dabei steht r für die gesuchte Periode von x.

7. Schritt: Nun soll das erste Register gemessen werden (die Werte des zweiten sind nicht so wichtig). Aufgrund der zum vorigen Schritt gemachten Bemerkungen erhalten wir ein b, so dass rb ziemlich genau ein Vielfaches von 2^{2L} sein muss. Mit elementaren Methoden ist es dann leicht, daraus das r zu ermitteln (Kettenbruchentwicklung!).

8. Schritt: Teste, ob x gut ist. Das kann wieder ein klassischer Computer übernehmen, da die Periode bekannt ist. Falls ja, ist damit ein Teiler von n gefunden, die Begründung findet sich oben in Abschnitt 2. Falls nein, fange noch einmal beim dritten Schritt an. Es ist dann ziemlich sicher, dass vergleichsweise schnell die Faktorisierung gefunden wird.

Zusammenfassung

Alles war doch ziemlich verwickelt, daher soll hier noch einmal auf die wichtigsten Punkte hingewiesen werden:

- Gewisse heute als sicher geltende kryptographische Verfahren, insbesondere der public-key-Algorithmus RSA, sind dann nicht mehr sicher, wenn man eine Technik kennt, aus einer zusammengesetzten Zahl $n = p\cdot q$ die Faktoren herauszulesen.
- Wenn es jemand schafft, auf schnelle Weise die Periode eines beliebigen x auszurechenen, so ist das Problem gelöst. Als neuer Aspekt kommt hinzu: Man muss es eventuell mehrfach versuchen, denn das Verfahren ist nur mit einer gewissen positiven Wahrscheinlichkeit erfolgreich. Diese Wahrscheinlichkeit ist in unserem Fall beruhigend hoch (höher als 50 Prozent).
- Quantencomputer könnten genau das leisten. Sie müssen allerdings genügend kompliziert sein (einige tausend Qbits) und gewisse Forderungen erfüllen. Diese sind beim heutigen Stand der Technik nicht einmal ansatzweise zu verwirklichen. Das Hauptproblem ist, ein kompliziertes quantenmechanisches System so abzuschirmen, dass es keine Dekohärenz (= durch Messung oder Wechselwirkung zustande gekommener Verlust des Überlagerungszustands) gibt.

  Falls so ein Quantencomputer wirklich zur Verfügung steht, ist nichts weiter zu tun, als ihn wie oben beschrieben zu präparieren, die erforderlichen Zustandsänderungen durch geeignete Gatter vorzunehmen und dann das erste Register zu messen. Mit einer hohen Wahrscheinlichkeit führt das zur Periode des vorgelegten x.

Ich persönlich glaube nicht, dass man jemals auf diese Weise Zahlen einer interessanten Größenordnung faktorisieren kann. Das Thema „Quantenkryptographie“ hat aber noch andere Aspekte, die sicher schneller zu greifbaren Ergebnissen führen. Dazu gehört zum Beispiel die abhörsichere Übertragung von Schlüsseln: man benötigt im wesentlichen ein einziges Qbit, es sind schon Testläufe über einige Dutzend Kilometer erfolgreich durchgeführt worden.

Da ist die enthaltene Mathematik aber eher uninteressant. Wichtiger sind die beteiligten physikalischen Phänomene, deswegen wurde in diesem Artikel auch nicht darauf eingegangen.

Literaturverzeichnis

[1] M. Aigner und E. Behrends, *Alles Mathematik*, Vieweg, Braunschweig, Wiesbaden, 3. Auflage, 2008.

[11] Davon kann sich jeder am Beispiel resultierender Kräfte überzeugen: Wenn mehrere Hunde an einer Decke in verschiedene Richtungen zerren, wird sich die kaum von der Stelle bewegen.

Computeralgebra in der Systembiologie

Prof. Dr. Reinhard Laubenbacher
Virginia Bioinformatics Institute
and Mathematics Department
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061-0123, USA

Prof. Dr. Bernd Sturmfels
Department of Mathematics
University of California, Berkeley
Berkeley, CA 94720-3840, USA

reinhard@vbi.vt.edu
bernd@math.berkeley.edu

Zusammenfassung

Die Systembiologie beschäftigt sich mit dem Studium ganzer biologischer Systeme statt mit deren einzelnen Bestandteilen. Da heutzutage Technologien verfügbar sind, mit denen man hohe Aufkommen von Messdaten erzeugen kann, und da fortschrittliche mathematische Modellierungstechniken entwickelt wurden, verspricht dieses Gebiet wichtige neue Einsichten zu liefern. Zugleich sind immer leistungsfähigere Computer verfügbar und die Computeralgebra wurde zu einem nützlichen Werkzeug für viele Anwendungsbereiche ausgebaut. In diesem Artikel erklären wir die Anwendung der Computeralgebra in der Systembiologie an Hand eines bekannten Netzwerks zur Genregulation, nämlich dem *lac-Operon* des Bakteriums *E. coli.*

Systembiologie

In der zweiten Hälfte des 20. Jahrhunderts erlebte die Molekularbiologie eine dramatische Revolution, die mit der Entdeckung der DNA Struktur begann. Seit damals hat eine Reihe von technologischen Fortschritten den Forschern die Möglichkeit eröffnet, immer detailliertere Messungen von immer mehr molekularen Zellbestandteilen vorzunehmen. DNA Mikroarrays zum Beispiel sind kleine Silikonchips, die mit kurzen DNA Stücken übersät sind und dazu verwendet werden können, die Aktivitätsniveaus tausender verschiedener Gene in einer Gewebeprobe gleichzeitig zu messen. Schon bald könnte es möglich sein, in großem Maßstab quantitative Messungen innerhalb einer Zelle vorzunehmen. Solche globalen Schnappschüsse eines molekularen Prozesses eröffnen die Aussicht, die Veränderungen zu studieren, die unentwegt als zusammenhängender dynamischer Prozess mit komplizierten Interaktionen in einer Zelle vorgehen, anstatt wie bisher die Einzelteile isoliert zu betrachten. Und genau dies ist das Thema des aufstrebenden Forschungszweigs der *Systembiologie* [1].

Biologische Netzwerke sind in der Regel sehr komplex. Viele Größen beeinflussen sich gegenseitig auf nichtlineare Art. Das macht es schwierig solche Systeme ohne die Hilfe raffinierter mathematischer Konzepte und Werkzeuge zu untersuchen. Es ist nicht einmal klar, was die richtige formale Sprache ist, um molekulare Systeme zu beschreiben. Eine kennzeichnende Eigenschaft systembiologischer Forschung ist die intensive Anwendung mathematischer Methoden. Seit kurzem verwendet man dabei die *Computeralgebra* für biologische Problemstellungen. Dieses Gebiet der Mathematik kombiniert symbolische Berechnungen auf Computern mit den Konzepten der abstrakten Algebra.

Computeralgebra

Die Computeralgebra entwickelt Progamme, die mit Symbolen statt mit Gleitkommazahlen rechnen. Softwaresysteme für die Computeralgebra reichen von weithin bekannten kommerziellen Produkten wie `Maple`, `Mathematica` oder `Magma` bis hin zu einer Palette spezialisierterer Systeme, von denen viele kostenlos sind und ihre speziellen Berechnungen schneller durchführen. Ein wichtiges Thema in der Computeralgebra ist die Lösung nichtlinearer algebraischer Gleichungen. In der Systembiologie taucht dieses Problem auf, wenn man versucht, die Gleichgewichtszustände eines dynamischen Systems zu bestimmen. Betrachten wir zum Beispiel das folgende System von zwei Gleichungen, wobei x und y die Unbestimmten sowie k_1 und k_2 Parameter seien:

$$x^2 + k_1xy - 1 = y^2 + k_2xy - 1 = 0$$

Mittels der Computeralgebratechnik der sogenannten *Gröbner-Basen* kann man diese beiden Gleichungen in die folgende äquivalente Form umschreiben:

$$\begin{aligned}(k_1k_2 - 1)y^4 + (k_2^2 - k_1k_2 + 2)y^2 - 1 &= 0\\ k_2x + (1 - k_1k_2)y^3 + (k_1k_2 - k_2^2 - 1)y &= 0\end{aligned}$$

In der ersten Gleichung kommt x nicht vor. Mit Hilfe der Lösungsformel für quadratische Gleichungen können wir y in Abhängigkeit von den Paramtern k_1, k_2

ausdrücken. Die zweite Gleichung liefert dann x in Abhängigkeit von y und k_1, k_2. Eine feinere Analyse zeigt, dass es für $k_1 k_2 < 1$ immer genau vier reelle Lösungen (x, y) gibt, aber für $k_1 k_2 > 1$ nur zwei.

Das lac-Operon

Zur Illustration der Anwendung der Computeralgebra in der Systembiologie betrachten wir ein Netzwerk zur Genregulation, das von Jacob und Monod [2] entdeckt wurde, die dafür 1965 den Nobelpreis für Medizin erhielten. Das *E(scheria) coli lac(tose) Operon* ist eines der ersten und am besten verstandenen Beispiele der Regulierung von Genexpressionen. Die Genregulation in Bakterien dient dazu, dass sich die Zellen an Veränderungen in der Nahrungsversorgung anpassen, so dass ihr Wachstum und die Zellteilung optimiert werden. E. coli kann Glucose oder Lactose als Energie- und Kohlenstoffquellen nutzen. Wenn die Zellen in einer glucosereichen Substanz wachsen, ist die Aktivität der im Lactose-Metabolismus involvierten Enzyme sehr niedrig, selbst wenn Lactose verfügbar wäre. Sobald jedoch die Glucose verbraucht und noch Lactose vorhanden ist, nimmt die Aktivität dieser Enzyme zu. Diesen Prozess nennt man *Induktion* [3].

Eine Gruppe von Genen, die von einem gemeinsamen Promotor und Operator reguliert werden, heißt ein *Operon*. Solche Gene sind typischerweise in Zweiergruppen angeordnet. Ein Operon enthält auch Kontrollelemente (genannt Transskriptionsfaktoren), die sich an die regulierenden Elemente in der DNA binden können und die die Transskription der Strukturgene aktivieren oder verhindern. Transskriptionsfaktoren, die die Transskription stimulieren, heißen Induktoren. Sie binden sich an regulierende Elemente in der DNA, die man Promotoren nennt. Die Repressoren andererseits binden sich an Elemente der DNA, die man Operatoren nennt. Sie spielen bei der Unterdrückung der Transskription eine Rolle.

Die Abbildungen 1 und 2 zeigen das lac-Operon: die Strukturgene für drei Enzyme, die im Lactose-Metabolismus involviert sind (LacZ, LacY, LacA), ein Strukturgen, das ein Repressor-Protein beschreibt (LacR) und drei Kontrollelemente, die zur Regulierung der Transskription beitragen. Das LacY Gen codiert Lactose-Permease, die beim Transport der Lactose in die Zelle hilft, LacZ codiert β-Galactosidase, ein Enzym, das Lactose in Glucose und Galactose umwandelt, zwei Zuckerarten, die von der Zelle weiter abgebaut werden können, und LacA beschreibt Thiogalactosid-Transacetylase, ein Enzym, dessen Funktion noch nicht bekannt ist. Die Strukturgene LacZ, LacY und LacA werden nur dann exprimiert (d.h. hergestellt), wenn Lactose in der Zelle vorhanden ist. Fehlt die Lactose, so verbindet sich der lac-Repressor R mit der Operator-Region O und die RNA-Polymerase, die an den Promotor P gebunden ist, kann sich nicht über diese Region hinwegbewegen. Also findet keine Transskription von LacZ, LacZ und LacA statt.

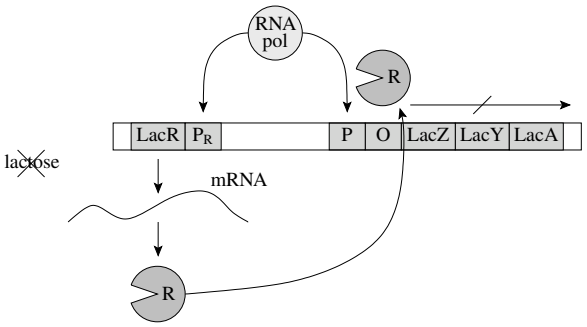


Abb. 1: Das lac-Operon ohne Lactose

Sobald Lactose in die Zelle eintritt, wird es in ein ähnliches Molekül (also ein Isomer) umgewandelt, das man Allolactose nennt. Dies geschieht ebenfalls durch die Wirkung der β-Galactosidase. Allolactose ist der Induktor des lac-Operons. Sie bindet an den lac-Repressor R an und bewirkt eine Änderung der Konformation, die die Anbindung von R an die Operator-Region verhindert. Die RNA-Polymerase kann sich damit an der DNA entlang bewegen, die Transskription der drei Gene findet statt, und Lactose wird metabolisiert.

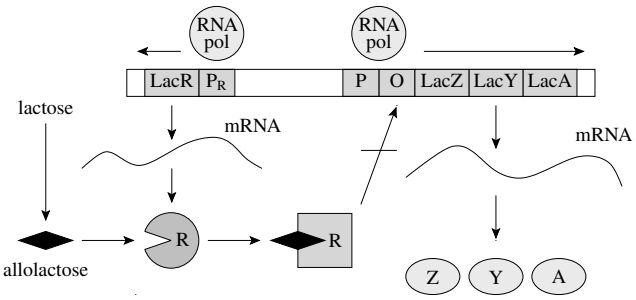


Abb. 2: Das lac-Operon mit Lactose

Ein diskretes Modell

Zuerst stellen wir ein diskretes Modell für dieses Netzwerk zur Genregulation vor. Es hat die Form eines *booleschen Netzwerks*. Wie alle Modelle ist es eine stark vereinfachte Darstellung der biologischen Details und Mechanismen. Was wir hier beschreiben möchten, ist die grundlegende dynamische Eigenschaft des lac-Operons, nämlich seine *Bistabilität*. Einfach ausgedrückt heißt dies, dass das Operon entweder EIN oder AUS ist, und jeder dieser beiden Werte entspricht genau einem Gleichgewichtszustand des Systems.

Das Modell besitzt fünf Variablen: M repräsentiert die Konzentration der mRNA für die Gene LacZ, LacY und LacA, A die Konzentration der intrazellularen Allolactose, B die Konzentration der β-Galactozidase (codiert durch LacZ), P die Konzentration der Lactose-Permease und L die Konzentration der intrazellulären Lactose. Das Modell ist quantitativ in dem Sinne, dass es extrem grob gerundete Messwerte für diese Konzentrationen verwendet: es beschreibt nur das Vorhandensein (1) oder Fehlen (0) der entsprechenden Substanz. Die Beziehungen zwischen den Variablen werden dann durch logische Formeln beschrieben, und zwar für jede Variable eine.

Beispielsweise hängen die biologischen Mechanismen, die zur Transskription der Gene LacZ, LacY und LacA führen, vom Vorhandensein der Allolactose ab, die ja die Wirkung des Repressor-Gens abblocken muss. Anders ausgedrückt, die boolesche Funktion, die den Zustand der booleschen Variablen M bestimmt, ist $f_M = A$. Entsprechend kann man die Struktur der anderen Formeln ableiten:

$$f_B = M, \quad f_A = A \vee (L \wedge B),$$
$$f_P = M, \quad f_L = P \vee (L \wedge \neg B).$$

Nun müssen wir prüfen, ob dieses einfache Modell, das auf sehr wenigen Annahmen beruht, das dynamische Verhalten besitzt, das für das lac-Operon wesentlich ist: seine Bistabilität. Diese Analyse läuft darauf hinaus, dass wir die langfristige Dynamik, also die Gleichgewichtszustände und periodischen Zustände des Modells bestimmen. Ein Zustand des Systems ist ein binäres 5-Tupel (M, B, A, L, P). Zum Beispiel steht $(0, 0, 1, 0, 1)$ für den Zustand, in dem Allolactose und Lactose-Permease vorhanden sind, aber die anderen Molekülarten fehlen.

Die booleschen Funktionen in diesem Modell übersetzen wir wie im Artikel *Alles logisch, oder was?* auf Seite 44 in Polynome. Diese verwenden den binären Zahlbereich $\mathbb{Z}/(2) = \{0, 1\}$, also die Arithmetik modulo 2. Um eine boolesche Funktion in ein Polynom zu übersetzen, benutzen wir die Tatsache, dass $a \wedge b$ und $a \cdot b$ dieselben booleschen Werte annehmen. Ebenso sehen wir, dass $a \vee b = a + b + a \cdot b$ und $\neg a = a + 1$ gilt. Somit erhalten wir:

$$f_M = A, \quad f_B = M, \quad f_A = A + LB + ALB,$$
$$f_P = M, \quad f_L = P + L + LB + LP + LPB.$$

In einem Gleichgewichtszustand des Systems ändern die Funktionen die Werte der Variablen nicht. Das heißt, wenn (M, B, A, L, P) ein Gleichgewichtszustand ist, so gilt $f_M(M, B, A, L, P) = M$ und Analoges für die anderen vier Funktionen. Ein Gleichgewichtszustand ist folglich eine Lösung des nachstehenden algebraischen Gleichungssystems:

$$M = A, \quad B = M, \quad A = A + LB + ALB,$$
$$P = M, \quad L = P + L + LB + LP + LBP.$$

Mit Hilfe eines der erwähnten Computeralgebrasysteme lösen wir dieses Gleichungssystem und erhalten drei Gleichgewichtszustände:

$$(1, 1, 1, 1, 1), \ (0, 0, 0, 0, 0) \ \text{ und } \ (0, 0, 0, 1, 0).$$

Die ersten beiden Lösungen sind biologisch vernünftig, aber die dritte ist nicht sinnvoll, denn sie würde bedeuten, dass das Bakterium die vorhandene intrazelluläre Lactose nicht metabolisiert. Dies ist ein Hinweis darauf, dass unser Modell nicht sehr genau ist und modifiziert werden muss. Es ist so klein, dass wir sogar alle Zustandsübergänge graphisch veranschaulichen können (siehe Abb. 3), was für größere Modelle nicht mehr geht.

Wenn wir das lac-Operon genauer studieren, sehen wir, dass ein Problem bei diesem Modell darin besteht, dass nicht alle Molekülarten, die die Dynamik des Systems beeinflussen, auch repräsentiert werden. Wir laden die Leser ein, das Modell so zu modifizieren, dass es die biologische Realität genauer widerspiegelt, z. B. an Hand der Materialien, die auf der Webseite [4] veröffentlicht sind.

Ein kontinuierliches Modell

Als das älteste bekannte Netzwerk zur Genregulation ist das lac-Operon intensiv studiert worden. Viele verschiedene mathematische Modelle wurden dafür konstruiert. Der üblichste Modelltyp basiert auf gewöhnlichen Differentialgleichungen. Als Beispiel betrachten wir hier das sehr einfache Modell eines dynamischen Systems aus [5]. Es besteht aus drei Gleichungen, die die Konzentration von R und die Änderungsraten von M und A modellieren. (Die Bedeutungen der Variablen sind die gleichen wie im letzten Abschnitt.) Die drei Gleichungen sind:

$$\begin{aligned} R &= \frac{1}{1 + A^n} \\ \frac{dM}{dt} &= c_0 + c\,(1 - R) - \gamma\, M \\ \frac{dA}{dt} &= ML - \delta\, A - \frac{vMA}{h + A} \end{aligned}$$

Dabei sind $c_0, c, \gamma, v, \delta, h$ und L gewisse Modellparameter, n ist eine feste positive ganze Zahl, und die Konzentrationen R, M und A sind Funktionen der Zeit t.

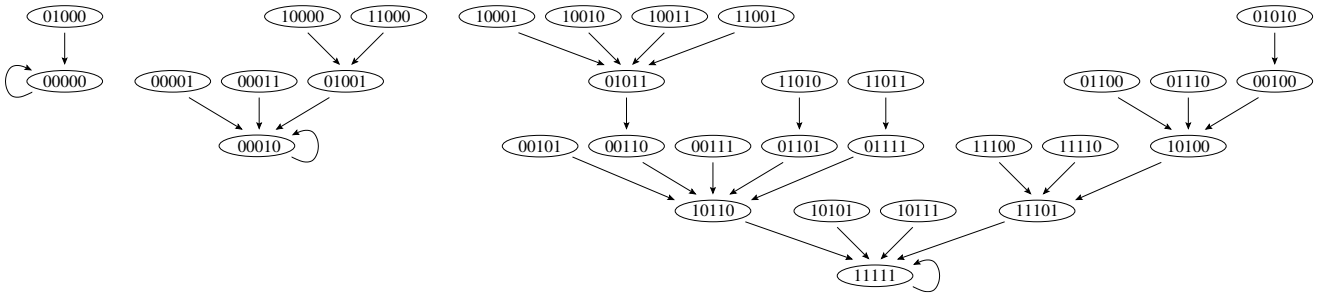


Abb. 3: Die Dynamik des diskreten Modells

Dieses Modell basiert ebenfalls auf bestimmten Annahmen, die in [5] genau erklärt werden. Es ist sowohl vom biologischen als auch vom mathematischen Standpunkt stark simplifiziert, aber selbst ein einfaches Modell kann nützlich sein. Das Ziel einer Modellierung ist, die wesentlichen Eigenschaften eines Systems herauszufinden, d.h. diejenigen Bestandteile und dynamischen Zusammenhänge, die für die Ausführung der biologischen Funktionen entscheidend sind.

Im Folgenden wollen wir die Dynamik des kontinuierlichen Modells analysieren, indem wir seine Gleichgewichtszustände bestimmen, genau wie wir es beim diskreten Modell getan haben. Dazu formulieren wir die Aufgabe in einer Art und Weise, die sie für die Methoden der Algebra zugänglich macht. Zuerst setzen wir die rechten Seiten der Differentialgleichungen gleich null:

$$\begin{aligned} c_0 + c \cdot \left(1 - \frac{1}{1+A^n}\right) - \gamma \cdot M &= 0 \\ M \cdot L - \delta \cdot A - \frac{v\, M\, A}{h + A} &= 0 \end{aligned}$$

Dies ist ein System von zwei algebraischen Gleichungen in den zwei Unbekannten A und M, das von den verschiedenen Parametern abhängt. Die Werte der dritten Unbestimmten R sind dabei durch die Gleichung $R = 1/(1 + A^n)$ festgelegt.

Wie in [5], Abschnitt 5.2 ausführlich erklärt wird, lassen wir die Konzentration L der Lactose unbestimmt und legen die anderen Parameter wie folgt fest:

$$\begin{gathered} c = \gamma = v = 1, \quad c_0 = 0.05 \\ h = 2, \quad n = 5, \quad \delta = 0.2 \end{gathered}$$

Nach einigen Umformungen und der Elimination von M nimmt das System die folgende Gestalt an:

$$4A^7 + (29 - 21L)A^6 - 42LA^5 + 4A^2 + (9 - L)A - 2L = 0$$

Dies ist ein Polynom vom Grad 7 in A. Mit Hilfe der Methoden der Computeralgebra können wir zwei Werte

$$\begin{gathered} L_1 = 0.684538965813\ldots \\ \text{und} \\ L_2 = 1.510539839844\ldots \end{gathered}$$

finden, so dass es für alle Werte von L zwischen L_1 und L_2 genau drei positive Gleichgewichtszustände gibt. Für $L = 1$ sind die Gleichgewichtszustände (R, M, A) unseres Systems z.B. gerade

$$\begin{gathered} (0.2272,\ 0.0506,\ 0.9994), \\ (0.6907,\ 0.1859,\ 0.8642), \\ (2.3717,\ 1.0368,\ 0.0132). \end{gathered}$$

Die obige Gleichung $4A^7 + (29 - 21L)A^6 + \ldots$ ist die definierende Gleichung des folgenden Verzweigungsdiagramms in der (A, L)-Ebene.

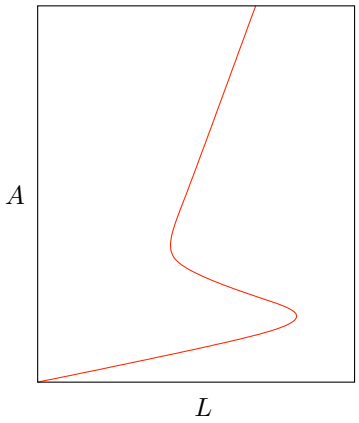


Abb. 4: Allolactose-Konzentration im Gleichgewichtszustand

Dieses Bild zeigt die Konzentration der Allolactose im Gleichgewichtszustand als Funktion der Lactose-Konzentration. Wie in [5] ausgeführt wird, zeigt unser Ergebnis mit drei Gleichgewichtszuständen, dass das Modell die wesentlichen Eigenschaften des lac-Operons korrekt wiedergibt.

Zusammenfassung

Es wird allgemein anerkannt, dass die moderne Molekularbiologie von neuen mathematischen Techniken stark profitieren kann. Sie erlauben die Konstruktion detaillierter Modelle biologischer Netzwerke auf Systemebene. Umgekehrt liefern die Probleme, die in der aktuellen Forschung in der Biologie betrachtet werden, wichtige Anregungen für die weitere Forschung in der Mathematik. Ein kürzlich erschienener Artikel [6] drückte dies treffend wie folgt aus:

Mathematics is biology's next microscope, only better; biology is mathematics' next physics, only better.

Wir haben hier versucht, an Hand mathematischer Modelle des lac-Operons zu zeigen, wie Algebra zu einer formalen Beschreibung und einem analytischen Verständnis biologischer Phänomene beitragen kann. Ein Ziel war zu erklären, dass unterschiedliche Typen mathematischer Modelle (diskrete und kontinuierliche) Einsichten in biologische Mechanismen liefern können. Ferner haben wir vorgeführt, wie die Computeralgebra, die traditionell in der Biologie nicht verwendet wurde, ein mächtiges Werkzeug sein kann, das dabei hilft, biologische Modelle zu konstruieren und zu analysieren.

Literaturverzeichnis

[1] U. Alon, *An Introduction to Systems Biology: Design Principles of Biological Circuits*, Chapman and Hall, 2006.

[2] F. Jacob und J. Monod, *Genetic regulatory mechanisms in the synthesis of proteins*, J. Molecular Biology **3** (1961), 318 – 356.

[3] H. Lodish, A. Berk, L. Zipursky, P. Matsudaira, D. Baltimore und J. Darnell, *Molecular Cell Biology*, W. H. Freeman and Company, New York, 2000.

[4] A. Martins, P. Vera-Licona und R. Laubenbacher, *Model your genes the mathematical way — a mathematical biology workshop for secondary school teachers*, verfügbar unter polymath.vbi.vt.edu/mathbio2006.

[5] R. D. Boer, *Theoretical Biology*, Undergraduate Course at Utrecht University, verfügbar unter theory.bio.uu.nl/rdb/books.

[6] J. Cohen, *Mathematics is biology's next microscope, only better; biology is mathematics' next physics, only better*, PloS Biology **2** (2004), 2017 – 2023.

Übersetzung aus dem Englischen: Martin Kreuzer

Weitere Literaturhinweise und eine ausführlichere Version des Artikels in englischer Sprache findet der Leser auf der Webseite dieses Sonderhefts.

www.fachgruppe-computeralgebra.de/JdM/

Simulation der Erwärmung von Zugbremsscheiben

Prof. Dr. Dieter Hackenbracht
Fachbereich Informatik und Ingenieurwissenschaften
Fachhochschule Frankfurt — University of Applied Sciences
Nibelungenplatz 1
60318 Frankfurt am Main

hackenbr@fb2.fh-frankfurt.de

Zusammenfassung

Untersucht wird die thermische Belastung der Bremsscheiben eines Zuges. Das Lösen der Modellgleichungen sowie die Darstellung und Prüfung der Ergebnisse erfolgen mit Hilfe des Computeralgebra-Systems Mathematica.

Modellbildung und Simulation

Ohne sichere Bremsen läuft im Eisenbahnwesen nichts. Die Zugbremsen sind eine zentrale Komponente für die Gewährleistung der Betriebssicherheit. Genaue Kenntnisse über die Beanspruchung des Bremssystems im Betrieb ermöglichen eine robustere Auslegung dieses Systems und eine Modifikation der beanspruchenden Prozesse. Eine solche Beanspruchung resultiert aus der Erwärmung der Bremsscheiben während des Bremsvorgangs. Eine typische konstruktive Auslegung einer Radbremsscheibe ist in Abbildung 1 (Quelle: Wikipedia) dargestellt.

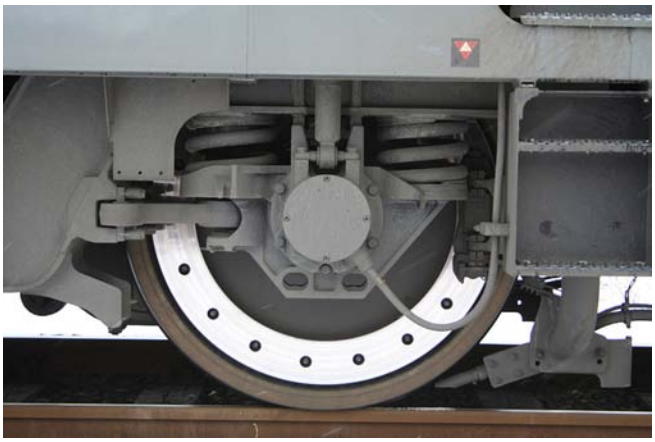


Abb. 1: Radbremsscheibe

Dabei interessieren sowohl der zeitliche Verlauf der Temperatur an ausgewählten Stellen (z.B. an Punkten auf der Oberfläche der Bremsscheibe) als auch die räumliche Verteilung zu einem gegebenen Zeitpunkt.

Ziel des Projektes war es, für einen typischen Fahrzyklus (Bremsen, Halten, Wiederanfahren, etc.) und eine typische Bremsscheibengeometrie den Temperaturverlauf (Erwärmung, Abkühlung, etc.) zu berechnen. Dazu müssen zunächst die Modellgleichungen aufgestellt werden. Als *geometrisches* Modell wurde eine einfache zylindrische Scheibe mit symmetrischer Belastung (Bremsbeläge auf beiden Seiten) zugrunde gelegt. Das *physikalische* Modell beschreibt die auftretenden physikalischen Prozesse: während des Bremsvorgangs wird über die Bremsbeläge Energie in die Bremsscheibe eingetragen und dann über Wärmeleitung, Konvektion und Strahlung in der Scheibe verteilt bzw. an die Umgebung wieder abgegeben. Daraus ergibt sich das *mathematische* Modell, nämlich eine Gleichung für die gesuchte Temperaturfunktion. Diese Funktion hängt von der Zeit t und von den Raumkoordinaten ab. Die relevanten räumlichen Koordinaten sind hier die axiale Koordinate (die z-Richtung parallel zur Radachse) und die radiale Koordinate (die r-Richtung von der Radachse zum Radkranz). Da die Gleichung die Veränderung der Temperatur modelliert, treten die Ableitungen der gesuchten Funktion nach den Variablen auf: bei der Gleichung handelt es sich damit um eine Differentialgleichung, nämlich die Wärmeleitungsgleichung, die den Temperaturausgleich innerhalb der Scheibe beschreibt. Der Temperaturausgleich zwischen Scheibe und Umgebung findet an der Oberfläche der Scheibe (den „Rändern“) statt und wird in den Randbedingungen für die Wärmeleitungsgleichung formuliert, ebenso die Energiezufuhr beim Bremsen. Parameter in der Gleichung und den Randbedingungen sind die Daten zur Beschreibung des Fahrzyklus (Geschwindigkeit und Bremskraft als zeitabhängige Größen), die geometrischen Daten der Bremsscheibe, die Materialeigenschaften sowie Angaben zur Beschreibung des Wärmeübergangs durch Konvektion und Strahlung.

Die sich daran anschließende Simulation umfasst zunächst die numerische Lösung der Modellgleichung und geeignete Darstellungen des Ergebnisses. Der Zusatz *numerisch* weist darauf hin, dass die Gleichung nicht exakt, sondern nur approximativ gelöst werden kann. Umfangreiche Kontrollrechnungen (z.B. Energiebilanzen durch geeignete numerische Integration der Lösung) sowie die Berechnung mit unterschiedlicher räumlicher Auflösung dienen der Absicherung dieser

Lösung und sind unerlässlich (s.u.). Zur Simulation wurde das Computeralgebra-System Mathematica eingesetzt. Es bietet die Möglichkeit, die gerade genannten Aufgaben mit einem einzigen Programm zu bewältigen. Es stellt hier insbesondere die Methoden zur numerischen Lösung der (partiellen) Differentialgleichung zur Verfügung, so dass man sich auf die eigentliche Problemstellung konzentrieren kann. Das verwendete mathematische Modell ist ein kontinuierliches Modell: die gesuchte Funktion soll eine stetige Funktion der Raumkoordinaten sein. Bei der Lösung der partiellen Differentialgleichung durch Mathematica wird das Problem allerdings automatisch räumlich diskretisiert und anschließend wird die Funktion durch Interpolation für alle Werte der Raumkoordinaten zur Verfügung gestellt.

Die Ergebnisse können wie folgt zusammen gefasst werden:

- die Rechnung liefert zuverlässige Ergebnisse über die thermische Belastung der Bremsscheibe;
- die numerischen Ungenauigkeiten sind über Optionen für das Lösungsverfahren, die Mathematica zur Verfügung stellt, gut kontrollierbar;
- die Rechnung wurde mit einem anderen Algorithmus wiederholt; dabei wurde die räumliche Diskretisierung „von Hand“ vorgenommen, um die partielle Differentialgleichung in ein System gewöhnlicher Differentialgleichungen umzuwandeln. Die Ergebnisse sind vergleichbar, aber die direkte („automatische“) Berechnung durch Mathematica war besser;
- ein Computeralgebra-System wie Mathematica ist ein wertvolles Werkzeug bei der Modellierung und Simulation komplexer anwendungsbezogener Fragestellungen.

Die Ergebnisse stehen der Deutsche Bahn AG zur Verfügung und werden dort zur Simulation der Bremsbelastung bei realen Streckenprofilen eingesetzt.

Fallbeispiel: einmaliger Bremsvorgang

Der zeitliche Verlauf der Bremskraft für einen einmaligen Bremsvorgang wurde wie in Abbildung 2 modelliert.

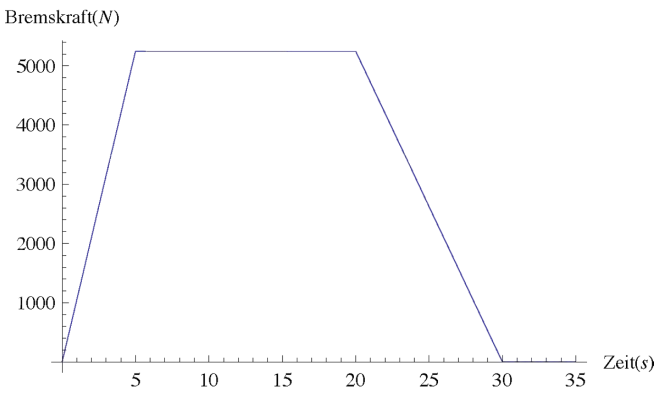


Abb. 2: Bremskraft

Abbildung 3 zeigt den Temperaturverlauf während des Bremsvorgangs im räumlich eindimensionalen Modell (es wird räumlich nur eine z-Abhängigkeit der Temperatur unterstellt). Zu Beginn des Bremsvorgangs hat die Bremsscheibe die konstante Temperatur 20° C (Umgebungstemperatur). Die Temperaturfunktion wird im betrachteten Gebiet ($0 < t < 50$, $0 < z < 0.1$) dargestellt.

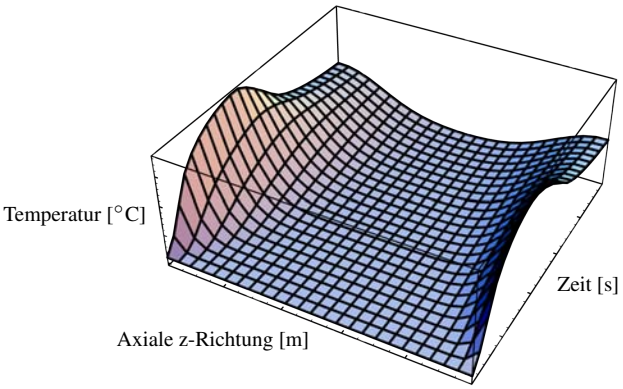


Abb. 3: Eindimensionales Modell

Abbildung 4 wurde mit dem räumlich zweidimensionalen Modell (z- und r-Abhängigkeit der Temperatur) generiert. Sie ist eine Momentaufnahme und zeigt den radialen Verlauf der Temperatur auf einer seitlichen Scheibenoberfläche ($z = 0$ bzw. $z = 0.1$) zur Zeit $t = 20\,\mathrm{s}$. Die Temperatur erreicht zu diesem Zeitpunkt lokal ein Maximum, und zwar mit einem deutlich höheren Wert als im eindimensionalen Modell (das eine Art Mittelung darstellt). Zur Einschätzung der tatsächlichen Belastung der Bremsscheibe ist also das volle zweidimensionale Modell erforderlich. Die Berechnung, die zu Abb. 4 führt, ist aber nicht genau genug: deutlich erkennbar ist der unphysikalische Verlauf der Temperatur am „rechten“ Rand ($r = 0.4$).

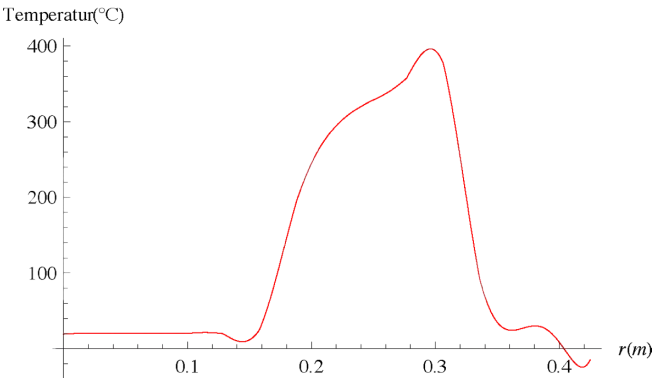


Abb. 4: Zweidimensionales Modell

Erst eine genauere Rechnung mit den entsprechenden Optionen für den Mathematica-Befehl zur Lösung der partiellen Differentialgleichung liefert einen vernünftigen Verlauf, wie Abbildung 5 zeigt. Eine sorgfältige Überprüfung der numerischen Ergebnisse ist also unerlässlich. Dazu dienen neben der Kontrolle des berechneten Temperaturverlaufes anhand der zugehörigen Grafiken die bereits erwähnten physikalischen Überlegungen (so muss z.B. die Energiebilanz

stimmen: die Differenz zwischen eingespeister und wieder an die Umgebung abgegebener Energie muss in der Bremsscheibe verbleiben) sowie der Einsatz eines alternativen mathematischen Algorithmus.

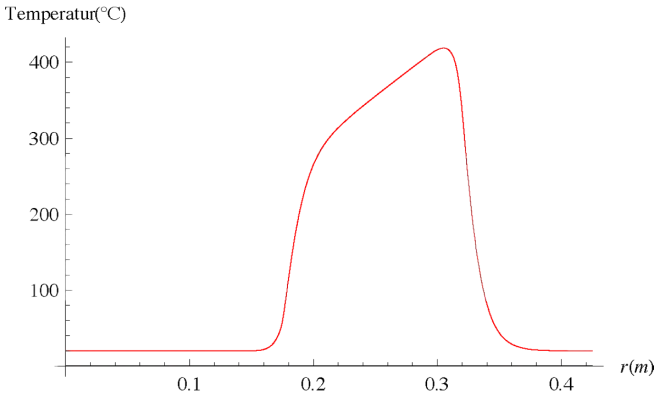


Abb. 5: Genauere Rechnung

Danksagung

Das vorgestellte Projekt wurde zusammen mit der Deutschen Bahn AG (DB Systemtechnik, Minden) durchgeführt. Für die angenehme Zusammenarbeit danke ich insbesondere Herrn Karbstein, Herrn Dr. Spieß und Herrn Söde. Der Firma ADDITIVE (Friedrichsdorf), vor allem Herrn Heilemann und Herrn Schönau, danke ich für die hilfreiche Unterstützung bei Problemen mit Mathematica. Herrn Horn vom Verlag Harri Deutsch (Frankfurt) danke ich für die Umsetzung der Vorlage nach LaTeX sowie die hilfreichen Kommentare.

Algebraisches Erdöl

Prof. Dr. Martin Kreuzer
Fakultät für Informatik und Mathematik
Universität Passau
Innstraße 33
94030 Passau

Dr. Hennie Poulisse
Shell International Exploration & Production
Kessler Park 1
2288 GS Rijswijk, Niederlande

Martin.Kreuzer@Uni-Passau.de
Hennie.Poulisse@Shell.com

Zusammenfassung

Mit Hilfe der Computeralgebra werden neuartige Modellgleichungen berechnet, die es gestatten, das Verhalten eines Ölfelds unter Produktionsbedingungen über längere Zeiträume korrekt vorherzusagen. Dabei werden symbolische Berechnungen mit numerischen Verfahren kombiniert, um ein sogenanntes *approximatives Verschwindungsideal* einer Menge von Datenpunkten zu bestimmen. Approximatives Verschwinden bedeutet hier, dass der Wert einer Modellgleichung an den Input-Datenpunkten nur ungefähr null sein muss, da die Daten Messfehler enthalten können. Das Verfahren wird an einem konkreten Beispiel unter Verwendung des Computeralgebrasystems ApCoCoA ausprobiert.

Die Erdölförderung und ihre Probleme

Bei der Förderung von Erdölvorkommen treten eine Reihe von Problemen auf, die bisher mit traditionellen Techniken der Geologie, der Physik und der angewandten Mathematik nicht zufriedenstellend gelöst werden konnten. Eine der Hauptaufgaben dabei ist, Modellgleichungen für die Produktion von Öl oder Gas aus Reservoiren zu finden, die es erlauben, das Verhalten des Systems über längere Zeiträume vorherzusagen. Der klassische Ansatz dafür ist, physikalische Gleichungen aus der „Flüssigkeitsdynamik in porösen Medien“ zu verwenden und sie an die jeweilige geologische Situation anzupassen. Auf Grund seismischer und anderer Messungen hat man eine grobe Vorstellung von der Gestalt des Reservoirs.

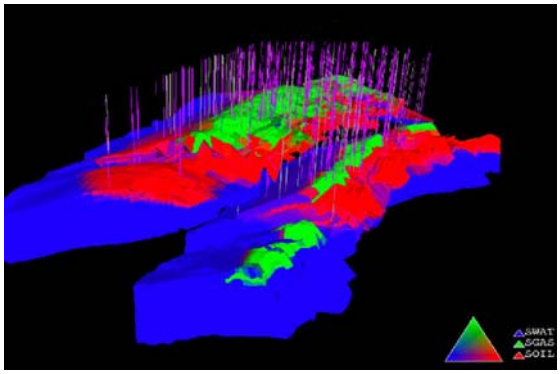


Abb. 1: Simulation eines Ölfelds und seiner Bohrungen

Allerdings sind die klassischen Modelle aus mehreren Gründen für Vorhersagen nicht geeignet. Zum einen kennt man die meisten der in den partiellen Differentialgleichungen der physikalischen Modelle auftretenden Strukturkonstanten nicht bzw. nicht genau genug (z.B. die exakte Durchlässigkeit der Erdschichten, unterirdische Verwerfungen etc.), und andererseits besitzen die Differentialgleichungen meist viele Lösungen, so dass sie so ziemlich an jede denkbare zukünftige Entwicklung angepasst werden können. Derartige Modelle spiegeln also mehr unsere Ansichten über das physikalische System als dessen wahre Struktur wider.

Die Folgen dieser Unkenntnis sind dramatisch. In fast allen Fällen wird weniger als 30% des in einem Resevoir vorhandenen Erdöls oder Erdgases gefördert, bevor eine weitere Förderung unmöglich wird. Eine typische Ursache dafür ist die Bildung sogenannter *Dry Spots*, d.h. das Reservoir zerfällt in viele kleinere Reservoire, aus denen nicht mehr wirtschaftlich gefördert werden kann.

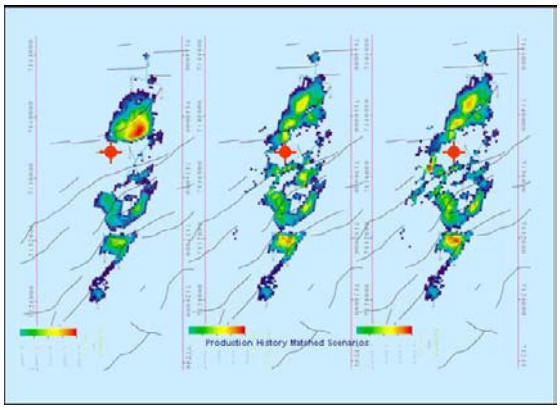


Abb. 2: Zeitliche Entwicklung eines Ölfelds

Ferner kann es zum sogenannten *Water Breakthrough* kommen, d.h. ein falsches Reservoirmanagement führt zu einem frühzeitigen Wassereinbruch, und ab diesem Zeitpunkt wird nur noch Wasser gefördert. Oder aber zu Beginn wird zu viel Gas gefördert, so dass der Reservoir-Druck zu schnell abfällt und eine weitere Förderung wegen des Druckverlusts unmöglich wird. All diese Prozesse sind nach heutigem Kenntnisstand irreversibel, so dass in diesen Fällen eine weitere Bewirtung des Felds ausgeschlossen ist und große Mengen Öl im Boden verbleiben müssen. Die Folgen für die Umwelt sind beträchtlich: da seit Jahren kein großes Ölvorkommen mehr gefunden wurde, wird inzwischen in ökologisch sehr sensitiven Bereichen wie Alaska gebohrt und gefördert.

Modellgleichungen aus der Computeralgebra

Der Ansatz des *Algebraic Oil* Projekts, das die Firma Shell Int. Exploration & Production in Zusammenarbeit mit dem Lehrstuhl für Symbolic Computation der Universität Passau betreibt, ist radikal anders. Als Grundlage für die gesuchten Modellgleichungen sollen nur Messdaten herangezogen werden. Aus diesen Daten werden die Gleichungen dann mit Hilfe der Computeralgebra berechnet. Um dieses Verfahren zu erklären, beginnen wir zunächst mit der klassischen Situation bei approximativen Interpolationsaufgaben.

Gegeben sei eine Menge von (exakten) Punkten $X = \{p_1, \ldots, p_s\}$ in $\mathbb{R}^n$. Die Gleichungen, die wir suchen, sind *Polynome*. Ein Polynom ist ein Ausdruck der Form $f = c_1t_1 + c_2t_2 + \ldots + c_mt_m$, wobei die c_i reelle Zahlen und die t_i *Terme* $t_i = x_1^{a_1} \cdots x_n^{a_n}$ (mit $a_j \geq 0$) in den Unbestimmten $x_1, \ldots, x_n$ sind. Wir suchen nun nach Polynomen, die an den Punkten von X *Nullstellen* besitzen, d.h. wenn $p_j = (p_{j1}, \ldots, p_{jn})$ die Koordinatendarstellung des Punkts p_j ist, so liefert die Einsetzung $x_k \mapsto p_{jk}$ in f das Ergebnis $f(p_{j1}, \ldots, p_{jn}) = 0$.

Die Menge aller Polynome ist der *Polynomring* $\mathbb{R}[x_1, \ldots, x_n]$. Die Teilmenge $I(X)$ aller Polynome, die an allen Punkten von X Nullstellen besitzen, bildet das sogenannte *Verschwindungsideal* von X. Die Elemente des Verschwindungsideals können als polynomiale Modelle aufgefasst werden, d.h. als Gleichungen, die polynomiale Relationen unter den Input-Daten darstellen. Modelliert man eine Output-Datenreihe ebenfalls durch ein polynomiales Modell, so kann man die Äquivalenz zweier solcher Modelle bzgl. der Input-Daten überprüfen. Man braucht dazu nur zu testen, ob das Differenzpolynom der beiden Modelle im Verschwindungsideal liegt, was in der Computeralgebra mit Hilfe einer Gröbner-Basis möglich ist.

Liegt die Punktmenge X exakt vor, so kann man ihr Verschwindungsideal mittels des Buchberger-Möller-Algorithmus [1] berechnen. Für Berechnungen mit echten Messdaten, die ja meist Messfehler beinhalten, ist dieser Algorithmus weniger geeignet. In diesem Fall sucht man ja auch nur Polynome, die auf den Datenpunkten *approximativ* verschwinden, d.h. deren Funktionswert betragsmäßig kleiner als eine vorgegebene Schranke $\varepsilon > 0$ ist. Da alle Polynome $f = c_1t_1 + \ldots + c_mt_m$ mit extrem kleinen Koeffizienten c_j and den Punkten von X approximativ verschwinden, sind wir in Wirklichkeit sogar nur an solchen Polynomen interessiert, deren Koeffizientenvektor normiert ist, d.h. für die $c_1^2 + \ldots + c_m^2 = 1$ gilt, und die an den Punkten von X approximativ verschwinden. Genau diese Polynome berechnet der approximative Buchberger-Möller-Algorithmus, der in [2] eingeführt wurde.

Ein Beispiel mit ApCoCoA

Zur Illustration betrachten wir das folgende Beispiel, wobei wir die Implementation des approximativen Buchberger-Möller-Algorithmus im Computeralgebrasystem ApCoCoA (vgl. [3]) verwenden. Mit Hilfe der Befehle

```
M:=Mat([[0.9817,-0.191],[0.191,0.9817]]);
U:=[];
P:=Mat([[1],[0]]);
For I:=1 To 500 Do
  P:=M*P;
  P[1,1]:=FloatApprox(P[1,1],10^(-6));
  P[2,1]:=FloatApprox(P[2,1],10^(-6));
  Append(U,[P[1,1],P[2,1]]);
EndFor;
```

erzeugen wir zunächst 500 Punkte, die in der Nähe des Einheitskreises liegen. (Aufgabe: Analysiere dieses Programm! Tipp: Der Punkt $(1, 0)$ wird jeweils um ca. $11°$ weitergedreht.) Dann betrachten wir die Kreise vom Radius $\sqrt{2}$ um den Punkt $(2, 2, 2)$, die in den Ebenenen $E_1 : x - z = 0$ und $E_2 : x + y - z = 6$ liegen. Mit

```
A:=[[2+P[2],2+P[1]*1.414,2+P[2]]|P In U];
B:=[[2+P[1]+P[2]*0.577, 2-2*P[2]*0.577,
     2-P[1]+P[2]*0.577]|P In U];
C:=Mat(Concat(A,B));
```

erzeugen wir dann 1000 Punkte, die wie zwei Gürtel stark gestört um die beiden Kreise liegen. (Aufgabe: Erkläre dieses Programm!)

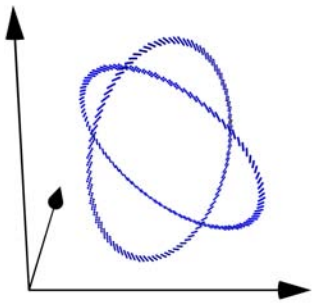


Abb. 3: 1000 Punkte in der Nähe zweier Kreise

Nun suchen wir nach polynomialen Relationen, die an diesen Punkten approximativ verschwinden. Wir machen über die Gestalt der zu berechnenden Gleichungen nur geringe Annahmen: die in den Relationen vorkommenden Variablen seien bekannt, und die gesuchten Relationen seien polynomialer Natur. Der ApCoCoA-Befehl

```
L:=Numerical.GBasisOfPoints(C,0.08,False);
```

berechnet eine sogenannte Gröbner-Basis des approximativen Verschwindungsideals der 1000 Punkte, wobei $\varepsilon = 0.08$ verwendet wird. Die resultierende Liste $L = [f_1, f_2, \dots]$ enthält Polynome f_1, f_2 mit

$$f_1 \approx -0.07x^2 - 0.07y^2 - 0.07z^2 + 0.29x + 0.29y + 0.29z - 0.71 \quad \text{und}$$
$$f_1 - 2.3 \cdot f_2 \approx -0.07x^2 - 0.07xy + 0.07yz + 0.07z^2 + 0.43x - 0.43z$$

Das erste ist ziemlich genau das -0.07-fache der definierende Gleichung der Sphäre

$$(x-2)^2 + (y-2)^2 + (z-2)^2 = 2$$

und das zweite entspricht der Gleichung

$$(x-z)(x+y+z-6) = 0$$

der beiden Ebenen, die wir ja zur Konstruktion der Punkte verwendet hatten.

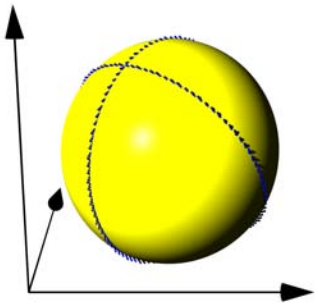


Abb. 4: Die Sphäre enthält die Punkte approximativ

Der klassische Buchberger-Möller-Algorithmus hätte hier nicht funktioniert, da er jeden Punkt als exakte Nullstelle der Polynome realisiert hätte. Die approximative Variante hingegen findet in den (z. B. durch Messfehler) gestörten Daten die eigentliche Information und liefert auch bei 1000 Datenpunkten einfache und verständliche Ergebnisse.

An dieser Stelle können wir zwei weitere Vorteile des neuen Algorithmus vermerken. Zum einen ist es möglich, auch nicht-polynomiale Modelle zu finden, zum Beispiel Wurzeln oder exponentielle Funktionen der Variablen. Dazu genügt es, diese als zusätzliche Datenreihen der Berechnung hinzuzufügen. Bei der Anwendung in der Ölförderung etwa wurden Wurzeln aus Differenzen von Druckmessungen verwendet, da diese typische Bestandteile von physikalischen Gesetzen der Flüssigkeitsdynamik sind. Dass dieses Vorgehen funktioniert, liegt an der Stabilität des Verfahrens gegenüber Erweiterungen des Modellhorizonts: fügt man dem Input unnötige neue Datenreihen hinzu, so werden diese automatisch ignoriert und die ursprüngliche Lösung wird gefunden. Wurden die neuen Datenreihen aus den vorhandenen berechnet, sind aber von diesen genügend unabhängig und für die Modellbildung nützlich, so werden sie mit einbezogen.

Der zweite und entscheidende Vorteil dieser Art der Modellbildung ist der, dass nicht nur eine optimale Approximation der Input-Daten bestimmt wird (wie es bei klassischen Methoden der approximativen Interpolation der Fall wäre), sondern polynomiale Gleichungen möglichst einfacher Form. Dies wird auch durch die Anwendung in der Ölförderung bestätigt, in der nicht nur bereits bekannte, sondern auch vollständig neue, durch weitere Messungen verifizierte Gesetzmäßigkeiten entdeckt wurden. Die so gewonnenen Ergebnisse erlauben Vorhersagen hoher Güte, wie die folgende Abbildung zeigt.

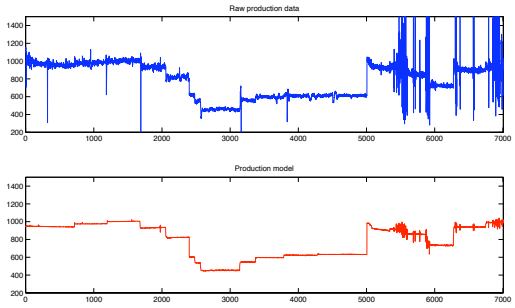


Abb. 5: Gemessene vs. vorhergesagte Gesamtproduktion

Interaktionen und ihre Interpretationen

Um das Entstehen von „Dry Spots“ und anderen unerwünschten Effekten zu verhindern, ist es wichtig, die Ölförderung geeignet zu steuern. Dazu muss man die physikalischen Interaktionen innerhalb eines Reservoirs verstehen.

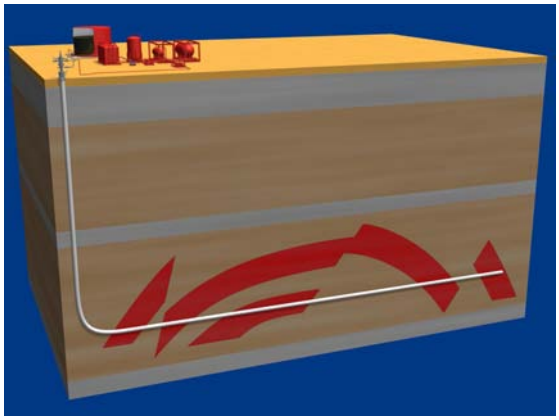


Abb. 6: Schematischer Aufbau einer Multi-Zonen-Quelle

Eine Multi-Zonen-Ölquelle besteht typischerweise aus verschiedenen *Taschen*, also Bereichen, die Öl oder Gas beinhalten. Zu den Taschen korrespondieren verschiedene Förderpunkte, die mit Sensoren für Drücke und Temperaturen ausgestattet sind. Man kann die Produktion einzelner Förderpunkte in einem Testbetrieb separat messen. Lässt man jedoch alle Förderpunkte gleichzeitig produzieren, so beeinflussen sie sich gegenseitig, weil das Öl in verbundenen Taschen zu verschiedenen Förderpunkten sickern kann. Die Kenntnis über

die Verbundenheit bzw. Unabhängigkeit von Taschen spielt eine entscheidende Rolle bei der optimalen Bewirtung eines Felds.

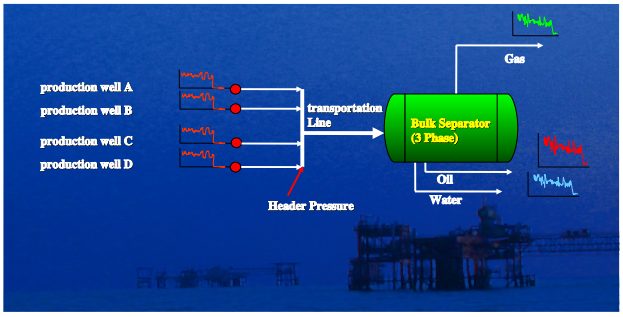


Abb. 7: Schematischer Aufbau der Messumgebung

Um die Interaktionen verschiedener Taschen zu berechnen, stellen wir die Modellgleichung f der Gesamtproduktion als Kombination der Modellgleichungen f_i der einzelnen Förderpunkte dar: $f = g_1 f_1 + \ldots + g_m f_m$. Dabei sind die Koeffizienten g_i Polynome in *allen* Unbestimmten, also auch in den nicht zum i-ten Förderpunkt gehörigen. Die Berechnung einer derartigen Darstellung nennt man das explizite Idealzugehörigkeitsproblem. Sie ist mit den Ergebnissen des approximativen Buchberger-Möller-Algorithmus problemlos möglich. Aus den Polynomen g_i kann man Rückschlüsse über die tatsächliche Struktur des Felds ziehen. Dadurch kann man das Verhalten des Ölfelds vorhersagen und somit das Produktionssystem wesentlich gezielter steuern. Zum Beispiel kann eine hohe Gasproduktion eines Förderpunkts genutzt werden, um das Öl eines anderen Förderpunkts leichter zu machen und somit effizienter zu gewinnen, allerdings nur wenn das Mischverhältnis korrekt ist.

Mit der skizzierten Berechnung des approximativen Verschwindungsideals konnten feldspezifische Gesetzmäßigkeiten nachgewiesen werden, die durch physikalische und statistische Untersuchungen bestätigt wurden, aber durch ihre unerwartete Form höchstwahrscheinlich unentdeckt geblieben wären. Dabei ist zu bedenken, dass insbesondere die hohe Komplexität des Systems und die vielen Einzelgrößen eine klassische Deduktion dieser Gesetzmäßigkeiten verhindern. Die Geschwindigkeit der Berechnung ist hoch genug um auch komplexe Datensätze von mehreren 10000 Punkten bearbeiten zu können. Die vielversprechenden Ergebnisse machen Hoffnung, dass man die beschriebenen Methoden auch auf andere Problemstellungen übertragen kann.

Literaturverzeichnis

[1] B. Buchberger und H. M. Möller, *The construction of multivariate polynomials with preassigned zeros*, Lect. Notes Comp. Sci. **144**, Springer, Heidelberg 1982.

[2] D. Heldt, M. Kreuzer, S. Pokutta und H. Poulisse, *Approximate computation of zero-dimensional polynomial ideals*, verfügbar unter `www.fim.uni-passau.de/~kreuzer`.

[3] ApCoCoA — Applied Computations in Commutative Algebra, siehe `www.apcocoa.org`.

Springer

springer.de

Wegweisende Titel der Algebra bei Springer

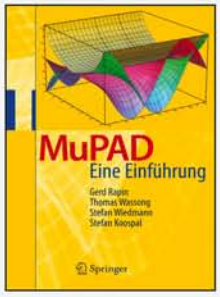
Computeralgebra

Eine algorithmisch orientierte Einführung

W. Koepf, Universität Kassel

Einführung in die moderne Computeralgebra: Während die ersten neun Kapitel den Standardkanon abdecken, behandeln die restlichen drei Kapitel Themen, welche in dieser Form noch nicht in Lehrbuchform erschienen sind. Ein Angebot für weiterführende Vorlesungen. Die betrachteten Algorithmen werden in Sitzungen mit dem Computeralgebrasystem Mathematica programmiert und getestet. Alle Sitzungen stehen auch als Worksheets für Maple oder MuPAD im Internet bereit. Daher kann Mathematica durch Maple oder MuPAD ersetzt werden. Reale Implementierungen ersetzen Pseudocode, so dass Algorithmen sofort anwendbar und überprüfbar sind. Kenntnisse der höheren Algebra werden nicht vorausgesetzt, dennoch werden alle Beweise geführt. Ein elementares Buch mit ausführlichem Index, das sich gut als Nachschlagewerk für Algorithmen der Computeralgebra eignet.

2006. XIV, 515 S. Brosch.
ISBN 978-3-540-29894-6 ► **€ (D) 39,95 | € (A) 41,07 | *sFr 65,50**



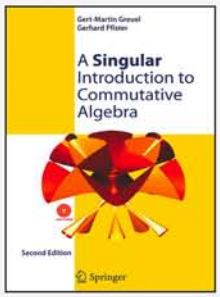
MuPAD

Eine Einführung

G. Rapin, T. Wassong, S. Wiedmann, S. Koospal, Georg-August Universität Göttingen

Das Computeralgebrasystem MuPAD ist eines der unentbehrlichen Werkzeuge für Lehre und Forschung in den Natur- und Ingenieurwissenschaften. Es überzeugt durch klare Strukturen, vielfältige Visualisierungsmöglichkeiten und effiziente Berechnungen. Diese kompakte, übersichtliche und elementare Einführung demonstriert anhand vieler Beispiele den Aufbau der Sprache und ihre wichtigsten Sprachelemente – basierend auf der MuPAD-Version Pro 4.0.

2007. XII, 195 S. 43 Abb. Brosch.
ISBN 978-3-540-73475-8 ► **€ (D) 19,95 | € (A) 20,50 | *sFr 32,50**

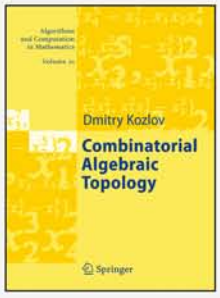


A Singular Introduction to Commutative Algebra

G. Greuel, G. Pfister, University of Kaiserslautern, Germany

The second edition is substantially enlarged by a chapter on Groebner bases in non-commtative rings, a chapter on characteristic and triangular sets with applications to primary decomposition and polynomial solving and an appendix on polynomial factorization including factorization over algebraic field extensions and absolute factorization, in the uni- and multivariate case.

2nd, extended ed. 2008. XX, 690 p. 49 illus. With CD-ROM. Hardcover
ISBN 978-3-540-73541-0 ► ***€ (D) 53,45 | € (A) 54,95 | sFr 87,00**



Combinatorial Algebraic Topology

D. Kozlov, University of Bremen, Germany

This is the first comprehensive treatment of combinatorial algebraic topology in book form. It constitutes a swift walk through the main tools of algebraic topology. The book contains an in-depth discussion of the major research techniques of combinatorial algebraic topology. In addition, several new themes are included. This book will rapidly bring the reader to the forefront of modern research.

2008. XX, 390 p. 115 illus. (Algorithms and Computation in Mathematics, Volume 21)
Hardcover
ISBN 978-3-540-71961-8 ► ***€ (D) 85,55 | € (A) 87,95 | sFr 139,50**
Softcover
ISBN 978-3-540-73051-4 ► ***€ (D) 37,40 | € (A) 38,45 | sFr 61,00**

Bei Fragen oder Bestellung wenden Sie sich bitte an ► Springer Distribution Center GmbH, Haberstr. 7, 69126 Heidelberg ► **Telefon:** +49 (0) 6221-345-4301 ► **Fax:** +49 (0) 6221-345-4229 ► **Email:** SDC-bookorder@springer.com
► € (D) sind gebundene Ladenpreise in Deutschland und enthalten 7% MwSt; € (A) sind gebundene Ladenpreise in Österreich und enthalten 10% MwSt. Die mit * gekennzeichneten Preise für Bücher und die mit ** gekennzeichneten Preise für elektronische Produkte sind unverbindliche Preisempfehlungen und enthalten die landesübliche MwSt. ► Preisänderungen und Irrtümer vorbehalten.
► Springer-Verlag GmbH, Handelsregistersitz: Berlin-Charlottenburg, HR B 91022. Geschäftsführer: Haank, Mos, Hendriks

Literatur zur Computeralgebra

Prof. Dr. Johannes Grabmeier
Fakultät Betriebswirtschaft und Wirtschaftsinformatik
Hochschule für angewandte Wissenschaften — FH Deggendorf
Edlmairstraße 6 + 8
94469 Deggendorf

johannes.grabmeier@fh-deggendorf.de

Eine Bibliographie zum Thema Computeralgebra ist naturgemäß so breit wie das Gebiet selbst. Ich gehe bei meinen Empfehlungen von den entsprechenden Nennungen in den verschiedenen Kapiteln des „Computer Algebra Handbooks“ [17] aus und ergänze diese durch die seitdem erschienenen wichtigen Bücher. Auch eine Liste für „Computeralgebra im Unterricht“ bzw. besonders zu empfehlende Bücher für Schüler und Lehrer wird angegeben. Darüber hinaus empfehle ich die entsprechende Rubrik des Computeralgebra-Rundbriefs mit Neuerscheinungen und Buchbesprechungen zur Computeralgebra — sowohl in der Vergangenheit, siehe www.fachgruppe-computeralgebra.de, als auch in der Zukunft.

Bücher über Computeralgebra insgesamt

Zunächst seien Bücher genannt, die den Anspruch haben das Gebiet als ganzes zu behandeln. Das erste Buch dieser Art war von J. H. Davenport et al. [11] 1988, dann 1989 und in englischer Übersetzung 1992 von M. Mignotte [35]. Es folgten A. G. Akritas 1989 [1], K. O. Geddes et al. [15] 1992. Wegen seiner Breite und Tiefe besonders herauszuheben ist das 1999 erschienene Werk „Modern Computer Algebra“ von J. von zur Gathen und J. Gerhard [14]. 2003 erschien das Handbuch der Computeralgebra [17], in dem Themen, Anwendungen und Systeme der Computeralgebra von mehr als 100 Autoren behandelt werden. Dazu kommen zwei deutschsprachige Bücher von M. Kaplan [23] und W. Koepf [32].

Computeralgebra und Unterricht

Speziell auf die Bedürfnisse des Einsatzes von Computeralgebra in der Schule wird in den Büchern, deren Großteil ich aus einer Auswahl von Heiko Knechtel[12] entnommen habe, eingegangen. Grundsätzliches zu Computeralgebra im Schulunterricht und zur mathematischen Modellbildung findet sich in [3, 20, 41] bzw. in [44].

Bücher mit Anwendung des im Schulunterricht gerne eingesetzten Systems Derive sind [19, 40, 4, 5]. Wie die letzten beiden behandeln auch die Bücher [45, 16] das Thema Analysis. Weiter verweise ich auch auf [2, 46]. Ein graphischer Taschenrechner mit symbolischen Funktionen ist der Ausgangspunkt für [12, 24, 25, 26, 27, 28].

Bücher zu Spezialthemen der Computeralgebra

Zu den Spezialthemen der Computeralgebra: Für grundlegende und schnelle Algorithmen der Langzahlarithmetik ist in erster Linie der Klassiker von D. Knuth zu nennen [29, 15]. Zur Irreduzibiltät und Faktorisierung von Polynomen siehe [30, 9]. Zum Thema Lösen von polynomialen Gleichungen mit Gröbnerbasen und Anwendungen [15, 10, 6, 33, 34, 36, 37]. Algorithmen zur algebraischen Zahlentheorie [39, 9]. Algorithmische Methoden in der Gruppentheorie werden in [22, 8, 43, 42] und in einer Gesamtschau in [21] behandelt. Die Theorie der symbolischen Summation findet sich in den Büchern [18, 38, 31], die der symbolischen Integration in [15] und vor allem in der Monographie von M. Bronstein. [7]. Symbolische Behandlung von Differentialgleichungen wird in [13] abgehandelt.

Literaturverzeichnis

[1] A. G. Akritas, *Elements of Computer Algebra with Applications,* Wiley, New York, 1989.

[2] A. Bartholome, H. Kern und J. Rung, *Zahlentheorie für Einsteiger — eine Einführung für Schüler, Lehrer, Studierende und andere Interessierte*, Vieweg, Wiesbaden, 5. Auflage, 2006.

[3] B. Barzel, J. Böhm, P. Drijvers, D. Janssens, D. Sjöstrand und A. J. P. Watkins, *Neue Wege im Mathematikunterricht*, Pädagogisches Institut Niederösterreich, Hollabrunn (Österreich), 1999.

[4] R. Baumann, *Analysis 1 — Ein Arbeitsbuch mit Derive*, Klett-Verlag, Stuttgart, 2003.

[5] R. Baumann, *Analysis 2 — Ein Arbeitsbuch mit Derive*, Klett-Verlag, Stuttgart, 2002.

[6] Th. Becker und V. Weispfenning in Zusammenarbeit mit H. Kredel, *Gröbner Bases, A Computational Approach to Commutative Algebra*, Springer, New York, Berlin, Heidelberg, 1993.

[7] M. Bronstein, *Symbolic Integration I — Transcendental Functions, 2nd Ed.*, Springer, Heidelberg, 2005.

[8] G. Butler, *Fundamental Algorithms for Permutation Groups*, Springer, New York, Berlin, Heidelberg, 1991.

[12] Fachreferent Schule der Fachgruppenleitung Computeralgebra (2005 – 2008)

[9] H. Cohen, *A Course in Computational Algebraic Number Theory*, Springer, Berlin, Heidelberg, New York, 1996.

[10] D. Cox, J. Little und D. O'Shea, *Ideals, Varieties and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*, Springer, New York, 1992.

[11] J. H. Davenport, Y. Siret und E. Tournier, *Computer Algebra: Systems and Algorithms for Algebraic Computation*, AcademicPress, London, 1989.

[12] M. Ebenhöh und G. Steinberg, *Ausgewählte Aufgaben zur Analysis*, Schroedel Verlag GmbH, Hannover, 1999.

[13] W. Fakler, *Algebraische Algorithmen zur Lösung von linearen Differentialgleichungen*, Teubner, Stuttgart, 1997.

[14] J. von zur Gathen und J. Gerhard, *Modern Computer Algebra*, Cambridge University Press, Cambridge, 1999.

[15] K. O. Geddes, S. R. Czapor und G. Labahn, *Algorithms for Computer Algebra*, Kluwer, Boston, 1992.

[16] A. M. Gleason, D. Hughes-Hallett, D. Lovelock, D. O. Lomen, W. G. McCallum, *Calculus Multivariable*, John Wiley and Sons, Inc., Chichester (UK), 4th Edition, 2004.

[17] J. Grabmeier, E. Kaltofen und V. Weispfenning, *Computer Algebra Handbook: Foundations, Applications, Systems*, Springer, Berlin, Heidelberg, New York, 2003.

[18] R. L. Graham, D. E. Knuth und O. Patashnik, *Concrete Mathematics*, Addison-Wesley, Reading, Massachusetts, 1989, 1994.

[19] H.-W. Henn, *Realitätsnaher Mathematikunterricht mit Derive*, Frd. Dümmlers Verlag, Bonn, 1997.

[20] H. Heugl, W. Klinger und J. Lechner, *Mathematikunterricht mit Computeralgebra-Systemen*, Addison-Wesley, Bonn, Reading, 1996.

[21] D. F. Holt, B. Eick und E. A. O'Brien, *Handbook of Computational Group Theory*, Chapman and Hall/CRC, London, 2005.

[22] D. L. Johnson, *Presentations of Groups*, Cambridge University Press, Cambridge, 1990.

[23] M. Kaplan, *Computeralgebra*, Springer, Berlin, Heidelberg, New York, 2006.

[24] H. Knechtel, H. Kramer und U.-H. Krüger, *Mathe Open End — Band 1, Differentialrechnung*, Westermann Verlag, 2001.

[25] H. Knechtel, U.-H. Krüger und R. Kühl, *Mathe Open End — Band 2, Integralrechnung*, Westermann Verlag, 2005.

[26] H. Knechtel, W. Weiskirch, *Abituraufgaben mit Graphikrechnern und Taschencomputern — Band 1*, Schroedel-Verlag, 2001.

[27] H. Knechtel, W. Weiskirch, *Abituraufgaben mit Graphikrechnern und Taschencomputern — Band 2*, Schroedel-Verlag, 2005.

[28] H. Knechtel, W. Weiskirch, *Abituraufgaben mit Graphikrechnern und Taschencomputern — Band 3*, Schroedel-Verlag, 2008.

[29] D. E. Knuth, *The Art of Computer Programming 2 — Seminumerical Algorithms*, Addison-Wesley, Reading, Massachusetts, Second Edition, 1981.

[30] N. Koblitz, *A Course in Number Theory and Cryptography*, Springer, Berlin, Heidelberg, New York, 1987.

[31] W. Koepf, *Hypergeometric Summation — An Algorithmic Approach to Summation and Special Function Identities*, Vieweg, Braunschweig, Wiesbaden, 1998.

[32] W. Koepf, *Computeralgebra: Eine algorithmisch orientierte Einführung*, Springer, Berlin, Heidelberg, New York, 2006.

[33] M. Kreuzer und L. Robbiano, *Computational Commutative Algebra 1*, Springer, Berlin, Heidelberg, New York, 2000.

[34] M. Kreuzer und L. Robbiano, *Computational Commutative Algebra 2*, Springer, Berlin, Heidelberg, New York, 2005.

[35] M. Mignotte, *Mathematics for Computer Algebra*, Springer, Berlin, Heidelberg, New York, 1992.

[36] F. Mora, *Solving Polynomial Equation Systems I: The Kronecker-Duval Philosophy*, Cambridge University Press, Cambridge, 2002.

[37] F. Mora, *Solving Polynomial Equation Systems II: Macaulay's Paradigm and Gröbner Technology*, Cambridge University Press, Cambridge, 2005.

[38] M. Petkovšek, H. S. Wilf und D. Zeilberger, $A = B$, A K Peters, Wellesley, Massachusetts, 1996.

[39] M. E. Pohst und H. Zassenhaus, *Algorithmic Algebraic Number Theory*, Cambridge University Press, Cambridge, 1989.

[40] G. Scheu, *Arbeitsbuch Computeralgebra mit DERIVE*, Bildungsverlag Eins, 1993.

[41] H. Scheuermann, *Computereinsatz im anwendungsorientierten Analysisunterricht*, Verlag Franzbecker, Hildesheim, 1999.

[42] Á. Seress, *Permutation Group Algorithms*, Cambridge University Press, Cambridge, 2003.

[43] C. C. Sims, *Computation with Finitely Presented Groups*, Cambridge University Press, Cambridge, 1994.

[44] T. Sonar, *Angewandte Mathematik, Modellbildung und Informatik — Eine Einführung für Lehramtsstudenten, Lehrer und Schüler*, Vieweg, Braunschweig, Wiesbaden, 2001.

[45] G. B. Thomas und R. L. Finney, *Calculus and Analytic Geometry*, Addison-Wesley, Reading, Massachusetts, 9th Edition, 1999.

[46] T. Westermann, *Mathematische Probleme lösen mit Maple — Ein Kurzeinstieg*, Springer, Berlin, Heidelberg, New York, 3. aktualisierte Auflage, 2008.

Fachgruppenleitung Computeralgebra 2008 – 2011

Sprecher,
Vertreter der DMV:
Prof. Dr. Wolfram Koepf
Fachbereich Mathematik
Universität Kassel
Heinrich-Plett-Str. 40
34132 Kassel
0561-804-4207, -4646 (Fax)
koepf@mathematik.uni-kassel.de
www.mathematik.uni-kassel.de/∼koepf

Fachreferent Internet:
Dr. Hans-Gert Gräbe, apl. Prof.
Institut für Informatik
Universität Leipzig
Postfach 10 09 20
04009 Leipzig
0341-97-32248
graebe@informatik.uni-leipzig.de
www.informatik.uni-leipzig.de/∼graebe

Fachexperte Physik:
Dr. Thomas Hahn
Max-Planck-Institut für Physik
Föhringer Ring 6
80805 München
089-32354-300, -304 (Fax)
hahn@feynarts.de
wwwth.mppmu.mpg.de/members/hahn

Fachreferent Themen und Anwendungen:
Prof. Dr. Florian Heß
Institut für Mathematik
Technische Universität Berlin
Straße des 17. Juni Nr. 136
10623 Berlin
030-314-25062, -29953 (Fax)
hess@math.tu-berlin.de
www.math.tu-berlin.de/∼hess

Fachreferent CA-Systeme und -Bibliotheken:
Prof. Dr. Gregor Kemper
Zentrum Mathematik – M11
Technische Universität München
Boltzmannstr. 3
85748 Garching
089-289-17454, -17457 (Fax)
kemper@ma.tum.de
www-m11.ma.tum.de/∼kemper

Fachreferent CA an der Hochschule:
Prof. Dr. Gunter Malle
Fachbereich Mathematik
Technische Universität Kaiserslautern
Gottlieb-Daimler-Straße
67663 Kaiserslautern
0631-205-2264, -3989 (Fax)
malle@mathematik.uni-kl.de
www.mathematik.uni-kl.de/∼malle

Fachreferent Schule:
StD Dr. Jörg Meyer
Schäfertrift 16
31789 Hameln
05151-54236
J.M.Meyer@t-online.de

Redakteur Rundbrief:
Dr. Markus Wessler
Fakultät für Betriebswirtschaft
Fachhochschule München
Am Stadtpark 20
81243 München
089-1265-2711, -2714 (Fax)
markus.wessler@hm.edu

Stellvertretende Sprecherin,
Fachreferentin Fachhochschulen:
Prof. Dr. Elkedagmar Heinrich
Fachbereich Informatik
Hochschule für Technik,
Wirtschaft und Gestaltung Konstanz
78462 Konstanz
07531-206-343, -559 (Fax)
heinrich@htwg-konstanz.de
www.in.fh-konstanz.de/inhalte/de/KONTAKT/persseiten_nbc/heinrich.html

Fachreferent Computational Engineering,
Vertreter der GAMM:
Prof. Dr. Klaus Hackl
Lehrstuhl für Allgemeine Mechanik
Ruhr-Universität Bochum
Universitätsstr. 150
44780 Bochum
0234-32-26025, -14154 (Fax)
klaus.hackl@rub.de

Fachreferent Lehre und Didaktik:
Prof. Dr. Hans-Wolfgang Henn
Fachbereich Mathematik
Technische Universität Dortmund
44221 Dortmund
0231-755-2939, -2948 (Fax)
wolfgang.henn@mathematik.tu-dortmund.de
www.wolfgang-henn.de

Fachexperte Industrie:
PD Dr. Michael Hofmeister
Siemens AG
Corporate Technology
Discrete Optimization
Otto-Hahn-Ring 6
81739 München
089-636-49476, -42284 (Fax)
michael.hofmeister@siemens.com
www.ct.siemens.com

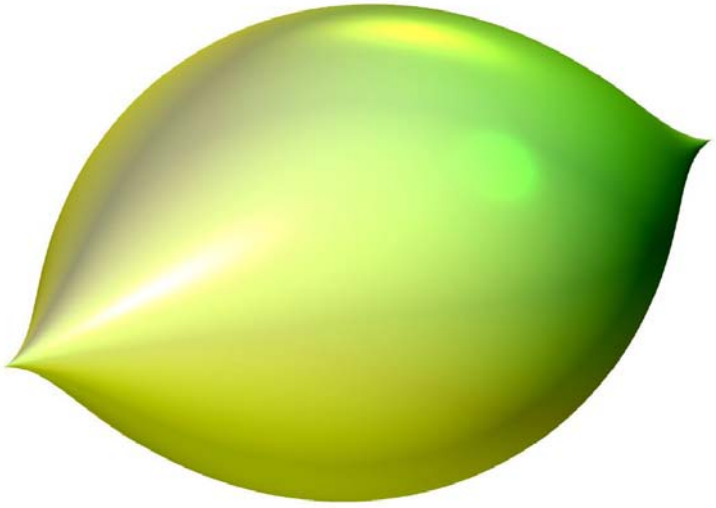
Fachreferent Jahr der Mathematik:
Prof. Dr. Martin Kreuzer
Fakultät für Informatik und Mathematik
Universität Passau
Innstr. 33
94030 Passau
0851-509-3120, -3122 (Fax)
martin.kreuzer@uni-passau.de
www.fim.uni-passau.de/∼kreuzer

Fachreferent ISSAC 2010,
Vertreter der GI:
Prof. Dr. Ernst W. Mayr
Lehrstuhl für Effiziente Algorithmen
Fakultät für Informatik
Technische Universität München
Boltzmannstraße 3
85748 Garching
089-289-17706, -17707 (Fax)
mayr@in.tum.de
www.in.tum.de/∼mayr/

Fachreferentin Publikationen und Besprechungen:
Prof. Dr. Eva Zerz
Lehrstuhl D für Mathematik
RWTH Aachen
Templergraben 64
52062 Aachen
0241-80-94544, -92108 (Fax)
eva.zerz@math.rwth-aachen.de
www.math.rwth-aachen.de/∼Eva.Zerz/

IMAGINARY — mit den Augen der Mathematik

Eine interaktive Ausstellung des Mathematischen Forschungsinstituts Oberwolfach für das Jahr der Mathematik 2008. Präsentiert werden Visualisierungen, interaktive Installationen, virtuelle Welten, 3D-Objekte und ihre theoretischen Hintergründe aus der algebraischen Geometrie und Singularitätentheorie. Die abstrakte Mathematik wird zu Bildern, imaginär wird zu *image*. Virtuelle Welten machen Mathematik zu beeinflussbarer Kunst und zu verstehbarer Wissenschaft. Ein einzigartiges Erlebnis für alle! Siehe www.imaginary2008.de.



Zitrus $x^2+z^2 = y^3(1-y)^3$

Die Idee von IMAGINARY ist — wie der Name schon vermuten lässt — die visuelle und ästhetische Komponente der Mathematik als Blickfang zu verwenden, um den BesucherInnen mathematische Hintergründe auf interaktive Weise zu erklären. Das Unvorstellbare der Mathematik wird zu Bildern, die auch selbst erzeugt werden können. Dabei werden die durch Polynome beschriebenen Objekte mittels Computeralgebra-Programmen visualisiert. Ein Kernstück ist hier das Programm Surfer zur Visualisierung algebraischer Geometrie in Echtzeit, mit dem während der Ausstellungen auf großen Touch Screens intuitiv und einfach Bilder algebraischer Flächen kreiert werden können.

Der Surfer eignet sich hervorragend, um Einblicke in die Algebra und Geometrie im Schulunterricht zu ermöglichen. Je nach Vorkenntnissen der SchülerInnen können einfache Objekte gemeinsam erzeugt und verändert oder kompliziertere Zusammenhänge erläutert werden. Das Programm gibt es kostenlos zum Download (siehe www.imaginary2008.de/surfer.php), z. B. für Projekte, die die Schüler zu Hause machen. Didaktisches Material zur Anleitung sowie Informationen zur Mathematik gibt es ebenfalls unter www.imaginary2008.de.

Spektrum der Wissenschaft und das Mathematische Forschungsinstitut Oberwolfach veranstalten gemeinsam einen Kreativ-Wettbewerb zum Mitmachen unter www.spektrum.de/mathekunst (siehe auch www.zeit.de/matheskulptur).

Die Termine und Orte der Ausstellung finden Sie unter
www.imaginary2008.de/wannwo.php.

Mathematisches
Forschungsinstitut
Oberwolfach

www.mfo.de

Wissenschaftsjahr 2008
Mathematik
Alles, was zählt

Aufnahmeantrag für Mitgliedschaft in der Fachgruppe Computeralgebra

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

Titel/Name: _____ Vorname: _____

Privatadresse
Straße/Postfach: _____

PLZ/Ort: _____ Telefon: _____

E-mail: _____ Telefax: _____

Dienstanschrift
Firma/Institution: _____

Straße/Postfach: _____

PLZ/Ort: _____ Telefon: _____

E-mail: _____ Telefax: _____

Gewünschte Postanschrift: [ ] Privatadresse [ ] Dienstanschrift

1. Hiermit beantrage ich zum 1. Januar 200___ die Aufnahme als Mitglied in die Fachgruppe

   Computeralgebra (CA) (bei der GI: 0.2.1).

2. Der Jahresbeitrag beträgt €7,50 bzw. €9,00. Ich ordne mich folgender Beitragsklasse zu:

   [ ] **€7,50** für Mitglieder einer der drei Trägergesellschaften

   [ ] GI Mitgliedsnummer: _____

   [ ] DMV Mitgliedsnummer: _____

   [ ] GAMM Mitgliedsnummer: _____

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

   [ ] **€7,50**. Ich bin aber noch nicht Mitglied einer der drei Trägergesellschaften. Deshalb beantrage ich gleichzeitig die Mitgliedschaft in der

   [ ] GI [ ] DMV [ ] GAMM.

   und bitte um Übersendung der entsprechenden Unterlagen.

   [ ] **€9,00** für Nichtmitglieder der drei Trägergesellschaften. [ ] Gleichzeitig bitte ich um Zusendung von Informationen über die Mitgliedschaft in folgenden Gesellschaften:

   [ ] GI [ ] DMV [ ] GAMM.

3. Die in dieses Formular eingetragenen Angaben werden elektronisch gespeichert. Ich bin damit einverstanden, dass meine Postanschrift durch die Trägergesellschaften oder durch Dritte nach Weitergabe durch eine Trägergesellschaft wie folgt genutzt werden kann (ist nichts angekreuzt, so wird c. angenommen).

   [ ] a. Zusendungen aller Art mit Bezug zur Informatik, Mathematik bzw. Mechanik.

   [ ] b. Zusendungen durch wiss. Institutionen mit Bezug zur Informatik, Mathematik bzw. Mechanik.

   [ ] c. Nur Zusendungen interner Art von GI, DMV bzw. GAMM.

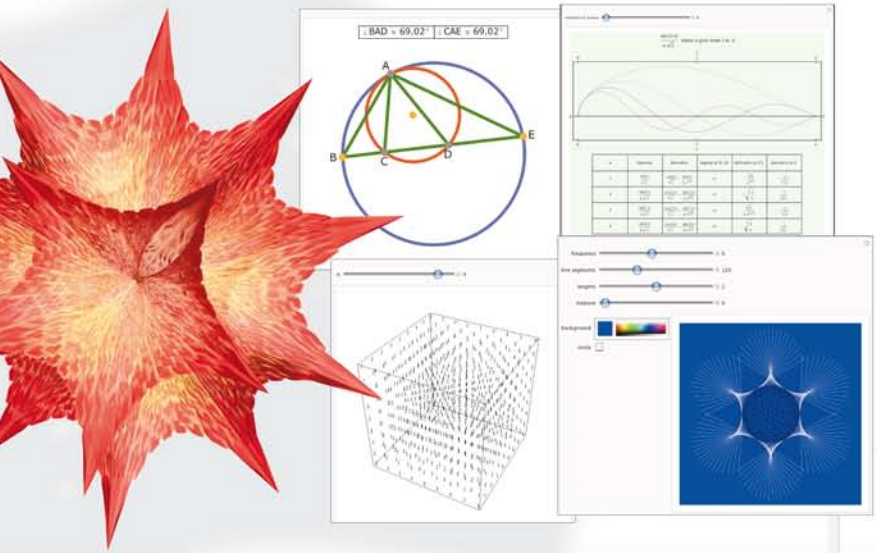
Ort, Datum: _____ Unterschrift: _____

Bitte senden Sie dieses Formular an:

Sprecher der Fachgruppe Computeralgebra
Prof. Dr. Wolfram Koepf
Fachbereich Mathematik
Universität Kassel
Heinrich-Plett-Str. 40
34132 Kassel
0561-804-4207, -4646 (Fax)
koepf@mathematik.uni-kassel.de

http://www.additive-net.de/software

Attraktive Schul- und Campuslizenzen von ADDITIVE



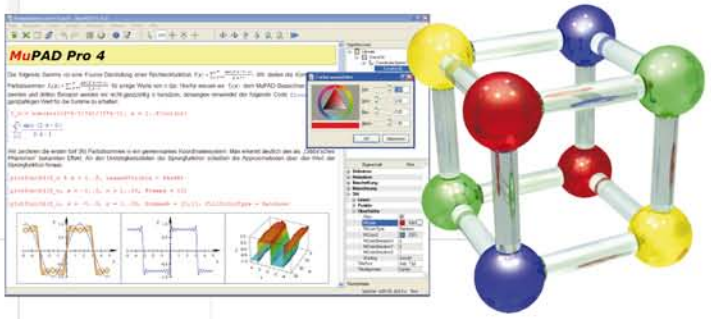
Wolfram Mathematica®6

http://www.additive-mathematica.de

MuPAD Pro 4

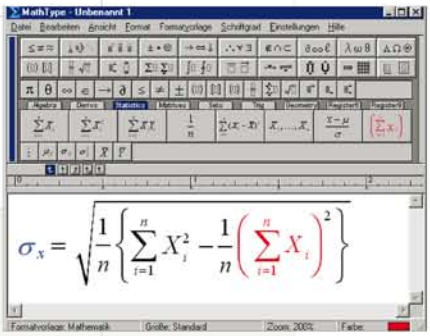
Computer-Algebra und mehr...

http://www.additive-net.de/mupad



Das Werkzeug zum Schreiben von Formeln seit der Erfindung der Kreide

http://www.additive-net.de/mathtype



MathType™

Möchten Sie gerne Ihre komplette Schule oder Ihren Campus mit Lizenzen bedienen?
Wir bieten zu unseren Produkten attraktive und günstige Schul- und Campuslizenzmodelle, direkt auf Ihre Bedürfnisse zugeschnitten.
Sprechen Sie uns an, wir beraten Sie gerne:
Telefon: 06172-5905-30 oder E-Mail: schule@additive-net.de

ADDITIVE GmbH
Max-Planck-Straße 22b • D-61381 Friedrichsdorf/Ts.
Telefon 06172-5905-0 · Fax 06172-776 13
E-Mail: info@additive-net.de
http://www.additive-net.de

ADDITIVE
Soft- und Hardware für Technik und Wissenschaft

Die richtigen Adressen

für Ihren anspruchsvollen Mathematikunterricht mit **CAS**.

www.schroedel.de

www.westermann.de

westermann®

Schroedel®

Wir beraten Sie gern:
Bildungsmedien Service GmbH
Postfach 4941 · 38023 Braunschweig
Telefon: (0 18 05) 21 31 00
0,14 Euro/Min. aus dem dt. Festnetz, abweichende Preise aus dem Mobilfunk
Telefax: (05 31) 70 86 64
www.schroedel.de
www.westermann.de

AZ_MA_ijdM_2008