

The *Mathematica*[®] Journal

Volume 8, Issue 1

■ Contents

□ Departments

Letters	2
Staff and Contributors	3
Calendar of Events	4
<i>Mathematica</i> News	5
New Products	10
New on <i>MathSource</i>	13
New Publications	16
Editor's Pick	20

□ Tutorials

Tricks of the Trade <i>edited by Paul Abbott</i>	27
In and Out <i>edited by Paul Abbott</i>	37

□ Columns

The <i>Mathematica</i> Programmer <i>by Roman Maeder</i>	41
Trott's Corner <i>by Michael Trott</i>	50

□ Feature Articles

Summary of Key Improvements in <i>Mathematica</i> 4.1 <i>by Wolfram Research Technical Staff</i>	63
BioEqCalc: A Package for Performing Equilibrium Calculations on Biochemical Reactions <i>by David L. Akers and Robert N. Goldberg</i>	86
Employing Complex Kinetic Diagrams to Understand the Belousov–Zhabotinskii Reaction <i>by Scot T. Martin</i>	114
A Language to Solve Finite Element Problems <i>by Naci M. Dai</i>	126
Linear and Morphological Image Processing <i>by Mariusz Jankowski</i> . .	147
Classifieds	160
Article Submissions to <i>The Mathematica Journal</i>	161

The electronic version of this issue is available at www.mathematica-journal.com/issue/v8i1.

The online edition of *The Mathematica Journal* includes downloadable versions of all articles and packages in this issue, plus a special animated graphics article, “Luge Run”, by Robert Rudd.

On the Cover

Scot Martin's chemical kinetics solver is used to visualize the rates of reactions leading to the production of chemical species.

Letters

Downloading *The Mathematica Journal*

Is it possible to download the whole issue of *The Mathematica Journal*? It is easier and cheaper to download the whole issue and read it offline instead of searching for each article and downloading it separately.

Martin Stockhammer
Visual Analysis GmbH

The Editors Reply

Thank you for a good suggestion. You now can download the notebooks and packages in the online issues of The Mathematica Journal as a .ZIP file. Text sections of the Journal (letters, news, etc.) are not included in the download.

Mileage

I've been getting *The Mathematica Journal* since 1988 or 1989, and it has been an invaluable resource to me. I can't begin to tell you how much mileage I've gotten out of it. It has really made a big difference in helping me to continually upgrade my *Mathematica* skills, and in helping me to get a broader view of the many seemingly incredible things that you can do with *Mathematica*. Keep up the good work!

C. Christopher Reed

Bookstore Sales?

I was wondering if I can purchase a copy of the latest issue (printed version) at a university bookstore or something like that. Thank you.

Mathematica User

The Editors Reply

The Journal is not currently distributed through bookstores. You can, however, order through the Wolfram Research bookstore, store.wolfram.com/catalog/books/tmj.html.

A Note to Subscribers

Please send address changes to service@mathematica-journal.com. Your current email address allows us to notify you when the new online issues are available.

Staff and Contributors

Editor: Flip Phillips, Ph.D. (Skidmore College)

Managing Editor: Joe Grohens

Associate Editor: Greg Springer

Design Team: Jeremy Davis, Malgorzata Zawislak, Al Navata

Compositor: Glenn Scholebo

Technical Specialist: Hormozd Gahvari

Archives and Web Production: Jean Buck

Web Development: Max Campos, Ankur Naik

Editorial Staff: George Beck, Maryka Baraka, Susan Kittivanichkulkrai, Marian Peden

Customer Service: Kathy Tattersall

Columnists: Paul Abbott, Roman Maeder, Michael Trott

Contributors: for this issue: David L. Akers, Naci M. Dai, Robert N. Goldberg, Mariusz Jankowski, Scot T. Martin, Robert Rudd

Editorial Advisory Board: Steve Christensen (MathTensor, Inc.; Christensen and Associates, Inc.); Donald Glaser (University of California, Berkeley); Donald Knuth (Stanford University); Wolfram Koepf (University Kassel, Germany); Silvio Levy (Mathematical Sciences Research Institute); David Lovelock (University of Arizona); Jerry Marsden (Caltech); Robert Moore (University of Washington, Seattle); Alfred van der Poorten (Macquarie University); Bill Press (Harvard University); Sam Savage (University of Chicago); Dana Scott (Carnegie Mellon University); Lawrence M. Seiford (University of Michigan); Lynn Steen (St. Olaf College); Francis Sullivan (Institute for Defense Analyses); Bill Thurston (University of California, Davis); Hal Varian (University of California, Berkeley); Rob Wolff (Apple Computer, Inc.); and Stephen Wolfram (Wolfram Research, Inc.).

Customer Services

www.mathematica-journal.com

+1-877-294-8184 (U.S. and Canada)

+1-217-398-6000

fax: +1-217-398-1621

subscriptions: subscribe@mathematica-journal.com

Advertising

The Mathematica Journal accepts advertising for all *Mathematica*-related products and services. Classified advertising is provided for *Mathematica* users. Rates are available at www.mathematica-journal.com/advertising. Contact the advertising department through the telephone numbers above, or send email to ads@mathematica-journal.com.

The Mathematica Journal is typeset in *Mathematica 4* using Janson, Futura, Math CMTT and the standard *Mathematica* fonts.

Publishing

The Mathematica Journal (ISSN 1047-5974) is published quarterly by Wolfram Media, Inc. Annual subscription rates in U.S. dollars: Electronic version: individuals, \$35; institutions, \$70; students, \$20. Electronic and print versions: individuals, \$75; institutions, \$200; students, \$50. Canadian/Mexican print subscriptions, add \$10. Print subscriptions outside North America, add \$32. Single print copy and back issue sales price: \$25. To subscribe or renew, call +1-217-398-6000 or +1-877-294-8184 (U.S. only), or visit our web site at www.mathematica-journal.com to enter your subscription information. Postmaster: Send address changes to: *The Mathematica Journal*, P.O. Box 6359, Champaign, IL 61826-6359.

Mathematica is a registered trademark of Wolfram Research, Inc. *Mathematica* is not associated with Mathematica Policy Research, Inc., or MathTech, Inc. Material in this publication and in its electronic form may not be reproduced in any form without permission of the publisher. The opinions expressed in the *Journal* are the authors' own and do not necessarily reflect those of the authors' employers, the editors, or the publishers of the *Journal*.

Calendar of Events

Professional meetings, conferences, trade shows, exhibits, and *Mathematica* training courses.

Send your event notices to calendar@mathematica-journal.com.

March

March 12–16, 2001

American Physical Society

Washington State Convention Center

Seattle, Washington

Info: www.aps.org/meet/MAR01/

March 27, 2001

FBC Technology Expo

Lawrence Berkeley National Lab

Berkeley, California

April

April 1–5, 2001

American Chemical Society

McCormick Place East

San Diego, California

Info: www.acs.org/meetings/sandiego2001.html

April 4–7, 2001

National Council of Teachers of Mathematics

Orange County Convention Center

Orlando, Florida

Info: www.nctm.org/meetings/annuals/orlando/

July

July 9–14, 2001

Rocky Mountain Mathematica

Holiday Inn, Frisco

Frisco, Colorado

Info: rmm.lfc.edu/

Mathematica News

More *Mathematica* news can be found at www.wolfram.com/news.

Mathematica 4.1 Now Shipping!

Mathematica 4.1 extends the technologies and advances pioneered by Wolfram Research, Inc. in *Mathematica* 4. Enhanced solvers, functions and an improved pattern matcher and compiler allow scientists and researchers to solve more complex problems more quickly. MathML and HTML integration helps academics and technical professionals share their work over the internet, while *J/Link* allows the design of innovative applications, combining the strengths of *Mathematica* and Java. The Japanese edition of *Mathematica* 4.1 is available for Macintosh and Windows.

For details, see the article beginning on page 62 in this issue. Additional information is available at www.wolfram.com/products/mathematica/newin41.

First MathML Conference Signifies Coming of Age: Finally the Web Does Math

Nearly two hundred leading mathematicians, scientists, and web technology experts converged on Champaign, Illinois, for the first “MathML and Math on the Web” conference, hosted by Wolfram Research, Inc.

MathML is the standard endorsed by the World Wide Web Consortium (W3C) for representing mathematical notation on the web. It was a proposal from Wolfram Research to the W3C that first began the MathML initiative. Many of the key concepts used in MathML are based on concepts originally developed in *Mathematica*, which fully supports MathML.

MathML is an XML application that fills the need for an efficient means of presenting mathematical or technical expressions on the web. Previously, such expressions had to be “frozen” in an image format such as a GIF—a static and cumbersome method—and inserted into an HTML document. The MathML standard makes math on the web a living, easily adaptable, and reusable entity, and it is now seen as being integral to the creation of new e-business and online educational opportunities. IBM, a conference cosponsor, chose this event to announce the release of techexplorer 3.0, a web browser plug-in for Netscape Navigator and Internet Explorer for rendering MathML. A technology exchange between IBM and Wolfram Research also provides techexplorer users a unique level of interoperability with *Mathematica*, uniting web-based typesetting and technical computation. Several other major browser developers, such as Microsoft, Netscape, and the W3C, also presented conference talks and product demonstrations that showed their commitment to supporting MathML.

More information about the MathML conference, including presentation abstracts and an update on the availability of conference proceedings, is at www.mathmlconference.org.

Mathematica Fashion Comes to Japan

Eri Matsui, a newcomer to fashion design in Japan, used a unique approach in creating her Fall 2000 designs. The one-time Chicago resident was inspired by *Mathematica* images created by Wolfram Research's Michael Trott. Trott was not aware of Ms. Matsui's use of *Mathematica* in her designs until contacted by Japanese Information Processing Service, or JIPS, a *Mathematica* reseller in Japan. He then gladly supplied new images especially for Matsui's use—images of knot theory and Escher patterns.

Wolfram Research's Alan Skillman and staff attended a premiere showing of the *Mathematica*-inspired designs. After the show, a JIPS representative gave a demonstration of how Matsui used *Mathematica* in fashion design.

Matsui's clients include Hikaru Nishida, a well-known Japanese singer. The singer gave a live performance shown on major Japanese television stations wearing one of Matsui's *Mathematica* creations.

The new designer's collection has attracted much attention and requests for information. Interviews with Matsui were published in many Japanese publications. Matsui's use of *Mathematica* has established her as an exciting new talent in her field.

More info: www.wolfram.com/news/archive/fashionshow.html

See also: www.cfd.or.jp/collect/97ss/97ssdigest/d115c.html



Math Olympics and Mathcamp

Every year Wolfram Research participates in educational and technological events for high school students. As cosponsor of this year's International Mathematical Olympiad (IMO) and Mathcamp 2000, Wolfram Research helped to

send students from the international, national, and local level to these events. Eight finalists from all over the U.S. and participants from 70 countries attended IMO.

To be selected for the U.S. Math Olympics team, students must complete a sequence of three exams. The eight top-scorers out of a total of 350,000 are invited to the Olympiad Awards Ceremony in Washington, D.C. Activities associated with the awards ceremony allow the U.S. team to meet some of the nation's best scientists and mathematicians.

After the awards, the U.S. team and 16 other finalists participate in an intensive month-long Mathematical Olympiad Summer Program (MOSP). The IMO competition follows the MOSP. Next year, Wolfram Research will be a major sponsor of the IMO, to be held in Seoul, Korea.

To be selected for the five-week Mathcamp 2000 program, students completed a challenging 10-problem test. The program's goal is to give students an opportunity to be creative in their thinking about advanced mathematical concepts. Participants also had a chance to make new friends and study with the great mathematicians of today.

More info: imo.math.ca

Mathematica and the Science of Secrecy

"I have no doubt that I am not a genius," says Sarah Flannery, who considers herself to be a fairly regular girl with a wide range of hobbies and interests. Yet, in January of 1999, Sarah was named the Irish Young Scientist of the Year for her work in devising a highly innovative, fast, and secure new algorithm for data encryption of internet and other electronic communications. Sarah was asked to write about her experiences. The resulting book, *In Code: A Mathematical Journey*, was published in April 2000.

Sarah's project, entitled "Cryptography—A New Algorithm versus the RSA," was widely praised for its brilliant applications of number theory and its demonstration of a strong grasp of the fundamentals of cryptography. Along the way, however, she also discovered how "wonderfully powerful" *Mathematica* is. It was "great fun learning the *Mathematica* commands to [encipher and decipher]...just seeing it work was a real kick," she wrote.

Sarah set herself to the task of learning basic matrix theory in order to construct her new algorithm, writing programs to generate examples of unfamiliar concepts, and consulting her father and numerous journals for additional information. As her mathematics knowledge and programming skills increased, she was determined to write cleaner, more sophisticated code after admiring the work of others that demonstrated the "wonderful flexibility of the *Mathematica* language."

Using *Mathematica*, Sarah implemented both the RSA and the CP algorithms and performed comparative run-time analyses on them. Her efforts were rewarded when she was able to demonstrate successfully that the CP algorithm

was 20 to 30 times faster than the RSA and to knowledgeably defend an attack on it as well.

An in-depth mathematical description and comparison of the RSA and CP algorithms, including the *Mathematica* code, is available at www.cayley-purser.ie. Sarah's story, *In Code: A Mathematical Journey*, is currently for sale throughout the UK, and can be ordered online through the Amazon.co.uk bookstore.

Mathematica Site Administrator Web Site Resource

Wolfram Research, makers of *Mathematica*, has announced the new Site Administrator web site (site.wolfram.com), an online resource to help *Mathematica* Site Administrators make effective use of their *Mathematica* license program. The site includes distribution information, installation instructions for all *Mathematica* platforms and configurations, links to information about *Mathematica* products and services, the answers to Frequently Asked Questions, as well as additional *Mathematica* resources and contact information.

"6 Integers": Making Music with Mathematica

On January 25, 2000, concert goers at the University of Cincinnati (UC) College Conservatory of Music heard the world premiere of "6 Integers," a *Mathematica*-generated musical composition by Carl McTague, a freshman at UC. Computed with *Mathematica* and rendered using a synthesizer playing piano and marimbas, "6 Integers" premiered to warm applause and positive responses.

The piece "6 Integers," as created by McTague, is an expression in *Mathematica*, just as " $x^2 + 1 + p(x)$ " is an expression. Over the course of the composition, a single, coherent process explores six integers by means of 5,292 notes and four independent channels of audio. It is the first full piece generated by the Hierarchical Functional Inheritance Model (HFIM).

The HFIM is the basis of the "very promising mathematically-driven approach to composition" that McTague has been working on for several years now, says Mara Helmuth, director of the UC Center for Computer Music studio.

"Eventually, I realized that the digital computer could be of immense use in my work," he said. McTague, who has been using *Mathematica* for three years now, says he chose to do his musical work in *Mathematica* because "I think in mathematical terms and *Mathematica* lets me implement my ideas in mathematical terms."

In the case of "6 Integers," *Mathematica* evaluated the composition's "expression" to generate a tremendous hierarchical structure, which it then symbolically "flattened" to produce a list of explicit instructions with each command representing a note. A Perl script then translated these instructions into a binary format understood by synthesizers.

Says Helmuth, “One of the wonderful things about algorithmic composition is that it can allow people to experience mathematical and scientific ideas in the realm of the arts and senses, increasing modes of understanding for the non-scientist.”

“6 Integers” can be downloaded as a stereo MP3 file. To read more about Carl McTague, visit his web site at www.mctague.org/carl.

Mathematica in Economics and Finance

Increasingly, *Mathematica* is being used in the fields of economics and finance. The updated web site at www.wolfram.com/solutions/finance now includes information about software developed by Vanderbilt University Professors Philip Crooke, Luke Froeb, and Steven Tschantz and Justice Department Research Director Gregory Werden. This software aids antitrust enforcement agencies to use *Mathematica* to simulate the effects of horizontal mergers.

2001 Mathematica Developer Conference

Mark your calendar now! The 2001 *Mathematica* Developer Conference has been scheduled for October 11–13 at Hawthorn Suites Conference Center, Champaign, Illinois, USA.

The 2001 *Mathematica* Developer Conference will be an invaluable resource for users and developers interested in creating *Mathematica*-related tools, products, courseware, or literature—and for those wishing to offer services as *Mathematica* trainers, consultants, site visitors, or evangelists.

The conference will give users a chance to get the newest developer information and an excellent opportunity to meet Wolfram Research staff and other *Mathematica* experts from around the world. The conference will feature tutorials, forums, an open computer lab, special events, and many informative, practical sessions.

The main focus points of this year’s conference will be web*Mathematica*, XML applications, and other Wolfram Research web and connectivity solutions, as well as trainer certifications. Submissions on these and all other *Mathematica* topics are welcomed.

More info: www.wolfram.com/news/events/devconf01

New Products

Digital Image Processing

The new *Digital Image Processing* application package for *Mathematica* contains tools for image processing, ranging from the basic to very advanced. Since its functionality is completely integrated with that of *Mathematica*, *Digital Image Processing* can perform far more sophisticated operations than standard image processing software.

Designed for students and professionals alike, *Digital Image Processing* combines a wide array of means for processing, transforming, and manipulating images. It comes with an interactive manual and tutorial. Advanced users can take advantage of *Mathematica*'s flexibility and functionality in order to program their own image processing tools and functions.

Digital Image Processing's features include the ability to import and export more than 15 standard image formats, support for many color models, a large number of built-in filters and filter-design algorithms, the ability to process over arbitrary polygonal regions of an image, and functions to smooth, sharpen, and reduce noise in images.

The author of *Digital Image Processing* is Dr. Mariusz Jankowski of Southern Maine University. A winner of many awards in Electrical Engineering Education, Dr. Jankowski is also the author of the "Linear and Morphological Image Processing" feature article in this issue of *The Mathematica Journal*.

More info

www.wolfram.com/news/digitalimage.html

Calculus WIZ

The popular *Mathematica* application package, *Calculus WIZ*, is now available as a stand-alone application. The new *Mathematica* 4 compatible version of *Calculus WIZ* uses the power of *Mathematica* to conquer tough calculus homework. And, since it is a stand-alone application, you do not need *Mathematica* to use it.

Calculus WIZ is comprehensive, covering calculus topics ranging from limits and functions to derivatives, integrals, and differential equations. Graphics and charts bring these and many more topics to life, making them much easier to understand. Built-in solvers accept and solve actual homework problems, allowing for fast double-checking of work and avoiding a lot of extra work. Tutorials provide that needed bit of extra help. The coverage is compatible with standard calculus textbooks, so that no matter how hard the class, *Calculus WIZ* can help with it.

The stand-alone version of *Calculus WIZ* is available for Windows 95/98/NT/2000 and Macintosh, and requires 160 MB of hard-disk space for installation. A new *Mathematica* 4 compatible application package version is also available for registered users of the previous version of *Calculus WIZ* or registered users of *Mathematica*.

More info

www.wolfram.com/products/student/calcwiz/

Operations Research

Operations Research is a *Mathematica* package for solving problems in linear optimization, shortest path tasks, and combinatorial optimization. With *Operations Research* you can solve a broad range of typical problems encountered in university courses as well as in daily practical work. Together with *Mathematica* you can model and solve optimization problems in a very convenient and interactive manner.

Example notebooks containing typical problems that can be solved with *Operations Research* are also included. Each problem is explained, modeled, and solved in full detail. The notebook collection should serve as a guide to dealing with a broad class of optimization tasks.

Operations Research was developed and is supported by SoftAS GmbH.

More info

www.wolfram.com/products/applications/operationsresearch/

Mathematica French and German Language Kits

Mathematica French and German Language Kits are add-ons to the *Mathematica* system, providing the *Mathematica* user interface, documentation, and spell checking in French and German.

The entire *Mathematica* interface—including menus, palettes, dialog boxes, error and warning messages, and over a thousand pages of online help—was carefully translated by leading French and German technical professionals who have had years of experience with the *Mathematica* system.

Mathematica French and German Language Kits online documentation includes a full French or German translation of *The Mathematica Book*, the definitive reference to *Mathematica* by Stephen Wolfram, the system's creator.

Each *Mathematica* Language Kit is distributed on CD-ROM and is for use with *Mathematica* 4. Each requires approximately 85.0 MB of disk space for hard-disk installation and is available for Windows 95/98/NT/2000 and Mac OS (Power Macintosh).

More info

Mathematica French Language Kit: www.wolfram.com/products/special/flk/

Mathematica German Language Kit: www.wolfram.com/products/special/glk/

ACTEX

Wolfram Research and ACTEX Publications, Inc. have collaborated to produce a new edition of their interactive, computer-based reference on actuarial models and the modeling process. The product is designed for use by members and test takers of the Society of Actuaries (SOA) and the Casualty Actuarial Society (CAS).

Actuaries are professionals trained in math, statistics, calculus, economics, and probability who apply their skills to the assessment of risk in financial fields such as investments, insurance, and employee benefits. To become an associate member of either society, one must successfully complete a series of comprehensive examinations for which the societies publish a list of required, as well as recommended, reading and preparation materials.

ACTEX is a publisher of many of the professional and exam-preparation materials used by the society's members and test takers. The new CD-ROM, titled *An Introduction to Actuarial Models and Modeling: An Interactive Approach*, is used primarily by actuarial students preparing for the Course 3 and 4 Examinations, which are coadministered by the SOA and the CAS.

The interactive study guide, based on a full *Mathematica* kernel, was initially released last year. The revised version uses a fully customized kernel and user interface, and the accompanying examples and documentation have been updated as well.

For more information on ACTEX and *Mathematica*, see www.wolfram.com/news/actex.html. The interactive CD-ROM is currently available only through Mad River Books, www.actexmadriv.com/. A complete listing of ACTEX publications is also available.

New on MathSource[®]

The following notebooks and packages were recently published on *MathSource*. To download, select the name of any of these items or visit *MathSource* at www.mathsource.com/Content/WhatsNew/.

- *Array of Framed Plots: Shared Axes or Inverted Axes* (Michele Cappellari, item 0208-538)
- *Class Material Creation for Primary School Mathematics* (Peter Massey, Debbie Wheeler, item 0210-687)
- *A Package to View All Colors Over a Specified Range* (Ted Ersek, item 0210-700)
- *An Enhanced Version of the Package FilterOptions`Utilities`* (Ted Ersek, item 0210-711)
- *A Utility Palette* (Ted Ersek, item 0210-744)
- *Finding the Inverse of Euler Totient Function* (Maxim Rytin, item 0210-755)
- *Add Colors, Borders, etc. to Exported HTML Pages* (Reinhold Kainhofer, item 0210-788)
- *Multiple-Response Fitting* (Bruce Miller, item 0210-799)
- *Random Math Test Generation* (Tom De Vries, item 0210-801)
- *Spheroidal Wave Functions* (Le-Wei Li, item 0210-812)
- *Simulating Sunlight* (Matthew P. Szudzik, item 0210-823)
- *Formal Proofs in Axiomatic Set Theory* (Matthew P. Szudzik, item 0210-834)
- *Lie Algebras* (Rolf Sulanke, item 0210-845)
- *A Cellular Automata Simulation Program for Silicon Anisotropic Super Micro-etching Process in Aqueous KOH* (Kazume Nishidate, Mamoru Baba, Richard J. Gaylord, item 0210-867)
- *C60 Graphics* (Kazume Nishidate, Mamoru Baba, Kiyoshi Nishikawa, item 0210-878)
- *MathSaver* (Stephen Coast, item 0210-889)
- *Nine7Symbol* (Alec Wodtke, Josh Halpern, item 0210-890)
- *Formatting Mathematica Output for Email* (Ted Ersek, item 0210-902)
- *ArrowExtended.m* (David Reiss, item 0210-913)
- *Balance Chemical Reactions* (Frank Kampas, item 0210-924)
- *Exterior Differential Calculus* (Sotirios Bonanos, item 0210-935)

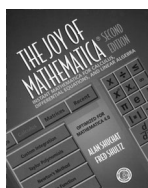
- *JMath: A Text Frontend with Command Editing (emacs/vi), History, and Symbol Completion* (Jim Radford, item 0210-946)
- *A Package to Find Options and Named Characters Matching a String Pattern* (Ted Ersek, item 0210-957)
- *Numerical Inversion of Laplace Transform* (Arnaud Mallet, item 0210-968)
- *Pattern Extractor* (Frank Kronmueller, item 0210-979)
- *Virtual Antennas, Volume 1, 1999* (N. I. Yannopoulou, P. E. Zimourtopoulos, item 0210-980)
- *Stirling Numbers Redefined* (Kwankyu Lee, item 0210-991)
- *Solution Methods for Microeconomic Dynamic Stochastic Optimization Problems* (Christopher D. Carroll, item 0211-004)
- *Ito's Lemma* (Mark Fisher, item 0211-015)
- *A Package to Eliminate Discontinuities in the Phase of an FFT* (Ted Ersek, item 0211-026)
- *Dbg—An Easy to Use Debugger* (David Bailey, item 0211-037)
- *Choleski Decomposition* (Mark C. Bourland, item 0211-048)
- *Rhombic Helicoids* (Russell Towle, item 0211-059)
- *Elect, Bead, and Star Zonohedra* (Russell Towle, item 0211-060)
- *Zonohedral Completion* (Russell Towle, item 0211-071)
- *Matrix Decomposition Using Pauli Matrices* (Heiko Feldmann, item 0211-082)
- *Normal Ordering of Quantum Operators* (A. Schnyder, item 0211-093)
- *Additional Analytical Inverse Laplace Transforms* (Arnaud Mallet, item 0211-105)
- *3D Plots of Discontinuous Functions* (Heiko Feldmann, item 0211-116)
- *Indical Tensor Package Using Notebook Interface* (Dal-Ho Park, item 0211-127)
- *Gaussian Integrals* (Frank Kuehnel, item 0211-138)
- *B-Spline Interpolation of Data with Regular Sampling* (Philippe Thévenaz, item 0211-149)
- *GPS Simulator* (Hans-Henrik Benzon, item 0211-150)
- *Solution of the Natural Convection Boundary-layer Flow Near a Heated Flat Plate* (Mark J. McCready, item 0211-161)
- *Magma Temperature Calculation from Fe-Ti Oxide Compositions* (Scott T. Dreher, item 0211-172)

- *Importance of Mass Transfer Fluctuations on Reaction Outcome in Multiphase Reactors* (Mark J. McCready, item 0211-183)
- *Colored Zonotiles* (Russell Towle, item 0211-194)
- *Introduction to Statistical Mechanics* (Michelle M. Franci, item 0211-206)
- *Introduction to the Use of Numerical Methods in Chemical Kinetics* (Michelle M. Franci, item 0211-217)
- *Inviscid Flow Past a Stationary Sphere* (Mark J. McCready, item 0211-228)
- *QuCalc: A Quantum Computation Package* (Paul Dumais, Hugo Touchette, item 0211-239)
- *Revised Installation Instructions for MathLie Packages that Accompany the Book, Symmetry Analysis of Differential Equations with Mathematica* (Eric Bynum, Gerd Baumann, item 0211-240)

New Publications

The Joy of *Mathematica*, Second Edition: Instant *Mathematica* for Calculus, Differential Equations, and Linear Algebra

Alan Shuchat and Fred Shultz, 2000, Harcourt/Academic Press, 576 pages, spiral-bound, CD-ROM.



The Joy of Mathematica is an easy manual for learning *Mathematica* by substituting menus and dialog boxes for typing commands. The user can access *Mathematica*'s power instantly, pointing and clicking in a simplified environment.

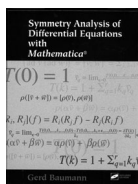
Shuchat and Shultz's book is useful as a supplemental text for calculus, differential equations, and linear algebra classes. Chapters include: Graphing in Two Dimensions; Manipulating Expressions; Solving Expressions; Solving Equations; Working with Functions of One Variable; Differentiating Functions of One Variable; Integrating Functions of One Variable; Working with Sequences and Series; Graphing in Three Dimensions; Working with Functions of Several Variables; Differentiating Functions of Several Variables; Integrating with Several Variables; Working with Vector Fields; Solving Differential Equations; Working with Vectors and Matrices; Functions; Limits and Continuity; Derivatives in One Variable; Integrals in One Variable; Sequences and Series; Parameterized Curves; Surfaces and Level Sets; Derivatives in Several Variables; Multiple Integrals; Differential Equations; Systems of Differential Equations; Matrices and Linear Equations; Vector Spaces and Linear Transformations.

Available

store.wolfram.com/view/ISBN0126407304/

Symmetry Analysis of Differential Equations with *Mathematica*

Gerd Baumann, 2000, Springer-Verlag, 521 pages, hardbound, CD-ROM.



Symmetry Analysis of Differential Equations with Mathematica provides a comprehensive introduction to the applications of symmetry analysis to differential equations. These applications, which emerged from discoveries by Sophus Lie, can be used to find exact solutions and to verify and develop numerical schemes.

The book includes both theoretical considerations and practical applications of use to physicists, chemists, mathematicians, and engineers. The CD-ROM contains *Mathematica* programs and examples. Chapters include: Elements of Symmetry Analysis; Derivatives; Symmetries of Ordinary Differential Equations; Point Symmetries of Partial Differential Equations; Non-Classical Symmetries of Partial Differential Equations; Potential Symmetries of Partial Differential Equations; Approximate Symmetries of Partial Differential Equations; Generalized Symmetries; Solution of Coupled Linear Partial Differential Equations.

Available

store.wolfram.com/view/ISBN0387985522/

Linear Algebra and Linear Operators in Engineering with Applications in *Mathematica*

H. Ted Davis and Kendall T. Thomson, 2000, Academic Press, 547 pages, hardbound.



Linear Algebra and Linear Operators in Engineering with Applications in Mathematica is an introduction to both the theory and the practical application of linear algebra and functional analysis designed for advanced engineering and physical science students.

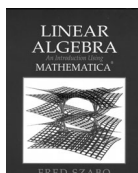
The book is self-contained, beginning with elementary principles and working up to a treatment of eigenanalysis and the spectral resolution theorem. Contents include numerous *Mathematica* examples with full code and solutions to applications throughout engineering and the physical sciences. Chapters include: Determinants; Vectors and Matrices; Solution of Linear and Nonlinear Systems; General Theory of Solvability of Linear Algebraic Equations; The Eigenproblem; Perfect Matrices; Imperfect or Defective Matrices; Infinite-Dimensional Linear Vector Spaces; Linear Integral Operators in a Hilbert Space; Linear Differential Operators in a Hilbert Space.

Available

store.wolfram.com/view/ISBN012206349X/

Linear Algebra: An Introduction Using *Mathematica*

Fred Szabo, 2000, Academic Press, 590 pages, hardbound.



Linear Algebra was written for a standard first course in linear algebra. The book emphasizes development of analytical and computational skills, and includes worked examples providing step-by-step methods for solving basic problems using *Mathematica*.

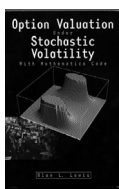
The book illustrates the application of linear algebra to problems in engineering, the natural sciences, computer animation, and statistics. Contents include: Linear Systems; Matrix Algebra; Determinants; Vector Spaces; Linear Transformations; Eigenvalues and Eigenvectors; Norms and Inner Products; Orthogonality; Singular Values and Singular Vectors.

Available

store.wolfram.com/view/ISBN0126801355/

Option Valuation under Stochastic Volatility with *Mathematica* Code

Alan L. Lewis, 2000, Finance Press, 350 pages, softbound.



Option Valuation under Stochastic Volatility with Mathematica Code is an advanced treatment of option pricing focusing on the role of stochastic volatility, which is geared for traders, money managers, and researchers. The book covers the new generation of option models where both the stock price and its volatility follow diffusion processes, explaining important features of real-world option pricing.

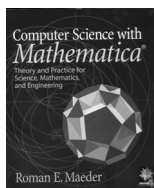
Also included are many illustrations and *Mathematica* code for the most important formulas. Chapters include: The Fundamental Transform; The Volatility of Volatility Series Expansion; Mixing Solutions and Applications; The Smile; The Term Structure of Implied Volatility; Utility-Based Equilibrium Models; Duality and Changes of Numeraire; Volatility Explosions and the Failure of the Martingale Pricing Formula; Options Prices at Large Volatility; Solutions to Models.

Available

store.wolfram.com/view/ISBN0967637201/

Computer Science with *Mathematica*: Theory and Practice for Science, Mathematics, and Engineering

Roman E. Maeder, 2000, Cambridge University Press, 389 pages, hardbound or softbound.



Computer Science with Mathematica: Theory and Practice for Science, Mathematics, and Engineering is a valuable text for introductory course work in computer science for mathematicians, scientists, and engineers. This book demonstrates that *Mathematica* is a powerful tool in the study of algorithms, allowing the behavior of each algorithm to be studied separately. Examples from mathematics, science, and engineering

are included. This book is also useful for *Mathematica* users at all levels.

Chapters include: Computers and Science; *Mathematica's* Programming Language; Iteration and Recursion; Structure of Programs; Abstract Data Types; Algorithms for Searching and Sorting; Complexity of Algorithms; Operations on Vectors and Matrices; List Processing and Recursion; Rule-Based Programming; Functions; Theory of Computation; Databases; Object-Oriented Programming.

Available

store.wolfram.com/view/ISBN0521663954/

Quantum Computation

Flip Phillips, Editor

The Mathematica Journal

■ Introduction

In late 1995, I was browsing an issue of *Science*, purloined from an unsuspecting colleague, containing an article [1] by David DiVincenzo of IBM Thomas J. Watson Research Center on the then emerging subject of quantum computation. Being a vision scientist my knowledge of quantum physics could be summed up in only a very few “easy pieces” (and in fact Feynman [2] seems to have thought of the idea pretty early on) but I was able to follow the gist of the argument enough to want to know more about it. It is a really powerful idea—being able to use quantum information to do computation. However, since the practical issues in constructing a physical realization of a quantum computer apparently required a few more years of incubation, I figured I would carefully clean my fingerprints off of the journal, return it to its rightful owner, and wait for a) someone to figure out how to build one, b) someone to port UNIX, Emacs, and T_EX to it, and finally c) someone to port it to *Mathematica*. But as with all good things that I am not able or permitted to do—such as fly an F-16 or drive a school bus on the streets and sidewalks of San Francisco, a simulator can work wonders to cure an unscratched itch.

■ Nuts and Bits

For the uninitiated, the building block of the quantum computer is the quantum-bit or *qubit*. A qubit is the superposition of the states (0 and 1) of the traditional bit with a probabilistic state instead of an absolute one. The possible states of a single qubit are written as follows.

$$\begin{array}{l} |0\rangle \\ |1\rangle \end{array} \tag{1}$$

The superposition x where a and b are complex numbers that declare the probability that x will be in the state 0 or 1, $p = |a|^2$ and $|b|^2$ respectively, is written as follows.

$$|x\rangle : |x\rangle = a|0\rangle + b|1\rangle \tag{2}$$

In vector form, we can think of the state of qubits as being represented by the following.

$$\begin{aligned} |0\rangle &= \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ |1\rangle &= \begin{pmatrix} 0 \\ 1 \end{pmatrix} \end{aligned} \quad (3)$$

A collection of two qubits can be written as follows.

$$|01\rangle = |0\rangle \otimes |1\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \quad (4)$$

n generalized qubits can be written as follows.

$$|x_1 \cdots x_n\rangle = |x_1\rangle \otimes |x_2\rangle \otimes \cdots |x_n\rangle = \begin{pmatrix} w_1 \\ \vdots \\ w_{2^n} \end{pmatrix} \quad (5)$$

As with things quantum, a set of qubits are in every possible state and when we choose to observe them they will be in some state with a given probability. Notationally, we typically see the following, where a and b are complex numbers with the probability of the device being in the state 10100101010 with $p = |a|^2$ and in the state 11101101010 with $p = |b|^2$, etc.

$$\Psi = a |10100101010\rangle + b |11101101010\rangle + \cdots \quad (6)$$

One interesting aspect of this representation is that we can take advantage of the fact that the phase information of the coefficients is actually meaningful. Using it, we can examine the interference between states, which as DiVincenzo points out can be useful from a computational standpoint, as we will see below.

■ Computation with Qubits

Consider a traditional computing machine, one can think of it as a construction of simpler, more elemental components. These basic parts are the AND, OR, and NOT gates that are conglomerated to carry out incrementally more complicated processes. This is obviously only a semantic distinction, we can easily regress further to the transistor level or even the elemental particle level should we be so inclined. I have chosen the Boolean logic level as a sort of delineation point where we consider traditional computation to be occurring. Quantum computers too are built on such a foundation. In the quantum case we can reduce the logic to a slightly simpler *controlled-NOT* element shown below.

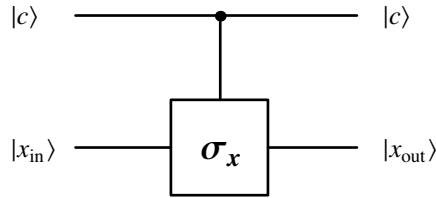


Figure 1. A graphical representation of a controlled NOT element. c represents the control qubit and x the qubit to be conditionally NOTed. In the absence of a control qubit the element acts like a traditional NOT gate by inverting the state of x .

The qubit c is used to control the NOT operation, a $|1\rangle$ results in inversion of the qubit x , and a $|0\rangle$ passes x unscathed. Mathematically, the controlled NOT can be represented by the following matrix.

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad (7)$$

Multiplying the two qubits $|cx\rangle$ by a controlled NOT matrix carries out the conditional action. For example, if we want to NOT the qubit $|0\rangle$ we set the control bit c to $|1\rangle$. This is just the multiplication of the column vector representation of $|10\rangle$ by the conditional NOT matrix.

$$\begin{aligned} |10\rangle &= |1\rangle \otimes |0\rangle = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} &= \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \end{aligned} \quad (8)$$

The resulting column vector has swapped the state of $x = |0\rangle$ to a $|1\rangle$, as expected. Had the control qubit c been $|0\rangle$ the state of x would have been left unchanged. By gathering up a mess of conditional NOTs it is a pretty straightforward task to make the more traditional gates of Boolean lore. For example, an AND gate can be created using one conditional NOT with two conditional inputs.

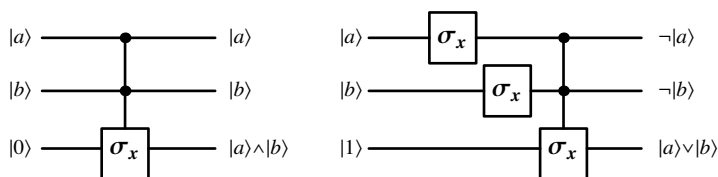


Figure 2. Left: A quantum AND gate created using a conditional NOT with two control qubits. Right: A quantum OR gate created from two standard NOTs and a conditional NOT.

Obviously from here you can expand your palette of operations even further by aggregating the gates into more sophisticated assemblies. As it turns out, quantum versions of gates such as the AND and OR above have a very important difference from their classical counterparts. A normal gate operates in an absolute fashion for a given input while its quantum sibling, by nature of the superposition of multiple states of its inputs, takes on several state pathways over the temporal duration of the computation. At the end of the computation these pathways recombine and, due to the phase information present, interfere with each other either constructively or destructively to produce a definite state. It is this feature of qubits that permitted Peter Shor [3] to devise a quantum prime factorization algorithm that operates in polynomial instead of exponential time.

Quantum algorithms are important topics in computer science at present but these algorithms are, of course, only theoretically realized since there is no actual quantum hardware to actually perform them on. Furthermore, two major stumbling blocks lie in the way of a physical realization of a quantum computer: the error correction and decoherence problems. For some problems error correction is trivial, such as in the prime factorization case. The reversibility of the process by multiplying the factors together gives a simple check to the algorithm. Other nontrivially reversible or irreversible problems require more robust error correction. Decoherence is another kettle of metaphorical fish—the quantum computation needs to take place in isolation from the outside environment, lest it be affected by its quantum dynamics. This contamination would cause damage to the phase information, which is critical to the eventual constructive/destructive reconstitution of the qubits at the end of the computation.

■ Simulating the Qubit

It turns out that a few years after the appearance of the DiVincenzo article, Gershenfeld and Chuang [4] used nuclear magnetic resonance to make a coffee-cup-sized quantum computer. The $|0\rangle$ and $|1\rangle$ states are represented by the upward and downward spin of the electrons respectively. It is a truly exciting contraption, getting us closer to a practical realization all of the time. Unfortunately, we are still going to have to wait a little bit for the real thing to make it into our personal digital assistants, but that does not mean we cannot get started using the tried and true concept of the simulator.

□ The SENKO Quantum Computer Simulator

The SENKO Corporation has produced a very novel cross-platform quantum computing simulator for Macintosh and Windows. It has a straightforward user interface that allows you to easily map out your quantum circuits and run them. It has a complete set of gates including various flavors of conditional NOTs, swap, exponentiation, discrete Fourier transform, observation, and arbitrary gates. If you have enough memory you can have 32 qubits (remember, that is $2^{32} \approx 4$ billion possible states so we are talking some serious RAM requirements). The process is reasonably simple, you set up your quantum circuitry, set the initial state of the qubits, and start the simulation. At the end of the execution the various possible states of the qubits are available along with their respective probabilities, and complex coefficients.

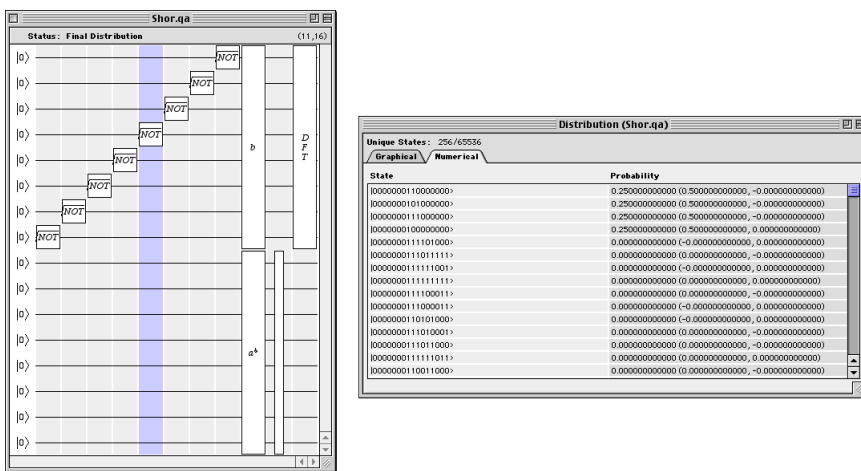


Figure 3. Two windows from the SENKO Quantum Computer Simulator. On the left, the actual configuration of quantum gates and input qubits and on the right the configuration of the qubits and their associated probabilities after the simulation has been run.

Most important to *Mathematica* users is the ability to import and export arbitrary gates *via MathLink*. Custom gates can be easily imported *via* the **ImportMatrix**·**FromMathematica**[] command (the semantics of the command names are a little confusing, since you are actually exporting it from the program you are typing the command into). For example, we can create a controlled NOT manually and import it using the following.

$$\text{ImportMatrixFromMathematica} \left[\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \right]$$

Process successful.

If you are feeling particularly saucy you can import that matrix into the SENKO simulator for further processing and graphical manipulation.

```
ImportMatrixFromMathematica[t]
```

```
Process successful.
```

■ Summary

Quantum computation offers wildly exciting possibilities from both a hardware and, perhaps more importantly, algorithmic implementation point of view. In an exciting *Nature* article [6], Seth Lloyd probes the ultimate limits of laptop computing from a physical standpoint. Hopefully, on the way to this theoretical *cul-de-sac* we will solve the implementation barriers between us and a truly usable quantum computer. Just like my fantasy of F-16 flight and reckless bus driving, and until I can get a few fluorine atoms to cooperate, we can all still take some solace in a simulator.

■ References

- [1] D. P. DiVincenzo, "Quantum computation," *Science*, **270** (5234), 1995 pp. 255–261.
- [2] R. P. Feynman, "Quantum mechanical computers," *Optics News*, **11**, 1985 pp. 11–20.
- [3] P. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," *Proceedings of the 35th Annual Symposium on Foundations of Computer Science* (Shafi Goldwasser, ed.) Los Alamitos, CA: IEEE Computer Society Press, 1994 pp. 124–134.
- [4] N. A. Gershenfeld and I. L. Chuang, "Bulk spin-resonance quantum computation," *Science*, **275** (5298), 1997 pp. 350–356.
- [5] P. Dumas, <http://crypto.cs.mcgill.ca/QuCalc/>
- [6] S. Lloyd, "Ultimate limits to computation," *Nature*, **406** (6799), 2000 pp. 1047–1054.
- [7] Center for Quantum Computation, <http://www.qubit.org/>

Tricks of the Trade

Edited by Paul Abbott

*Department of Physics
University of Western Australia
Nedlands WA 6907, Australia
tmj@physics.uwa.edu.au
physics.uwa.edu.au/~paul*

This is a column of programming tricks and techniques, most of which, we hope, will be contributed by our readers.

■ Mathematical Typesetting

□ Spanning Characters

In mathematical notation one commonly uses a large opening curly brace with all the cases listed to the right. However, by default, curly braces do not scale to the full height of the set of cases on the right, as shown.

$$f(x) = \left\{ \begin{array}{ll} 1 & \text{if } 0 < x \leq 1, \\ \frac{1}{2} & \text{if } x = 0, \\ 0 & \text{otherwise.} \end{array} \right.$$

A regular opening parenthesis instead does scale properly.

$$f(x) = \left(\begin{array}{ll} 1 & \text{if } 0 < x \leq 1, \\ \frac{1}{2} & \text{if } x = 0, \\ 0 & \text{otherwise.} \end{array} \right.$$

Neil Soiffer (soiffer@wolfram.com) writes: Very large brackets and braces for their common uses in *Mathematica* (function call, lists) can look silly. Hence, as defined in **UnicodeFontMapping.tr**, they are set to grow to at most “2.01” (factor of two, with a fudge factor) times their normal size when **SpanMaxSize** is set to **Automatic**. Spanning can be controlled by selecting Expression Input ▸ Spanning Characters ▸ Expand Indefinitely from the Edit menu. Alternatively, if you use the Option Inspector and type in **Span** you will see the Spanning Character Options. You can set **SpanMaxSize** around the character, at the cell level, notebook level, or globally. You may not be happy with what the brackets and

braces look like for function calls and lists, though. Here is the result of setting **SpanMaxSize→Infinity** at the character level.

$$f(x) = \begin{cases} 1 & \text{if } 0 < x \leq 1, \\ \frac{1}{2} & \text{if } x = 0, \\ 0 & \text{otherwise.} \end{cases}$$

You could also change the value in **UnicodeFontMapping.tr**, but then the notebook is no longer “portable” unless you also send the modified **UnicodeFontMapping.tr**, and it would make all list braces grow arbitrarily large.

There is another problem: in each case the opening bracket is displayed in color, indicating that it is unmatched. One solution is to select the unmatched bracket, use the Option Inspector and type in **Unmatched**. Here is the result of setting **"UnmatchedBracketStyle"→None** at the character or cell level.

$$f(x) = \begin{cases} 1 & \text{if } 0 < x \leq 1, \\ \frac{1}{2} & \text{if } x = 0, \\ 0 & \text{otherwise.} \end{cases}$$

In **TraditionalForm**, a better alternative is to close off the unmatched left bracket with **\[AutoRightMatch]** (or **ESC \right. ESC**) which keeps things properly bracketed. This will not work for **StandardForm** unless you set **DelimiterMatching→None** for the cell (or any larger scope). Here is the result of closing off the unmatched left bracket.

$$f(x) = \begin{cases} 1 & \text{if } 0 < x \leq 1, \\ \frac{1}{2} & \text{if } x = 0, \\ 0 & \text{otherwise.} \end{cases}$$

Similarly, to close off an unmatched right bracket, use **\[AutoLeftMatch]** (or **ESC \left. ESC**).

□ Inline Equations

Here is an easy and powerful way of producing correctly formatted two-dimensional inline equations by converting formulas in text cells into two-dimensional form. Consider the following cell.

Here is some text with an identity, $\text{Integrate}[x^2 \sin[x],x]=\text{Integrate}[x^2 \sin[x],x]$.

1. Select the formula (i.e., $\text{Integrate}[x^2 \sin[x],x]=\text{Integrate}[x^2 \sin[x],x]$) and use **Edit ▸ Expression Input ▸ Start Inline Cell** (or **CTRL[9]**) to turn this into an *inline cell*.

Here is some text with an identity,
 $\text{Integrate}[x^2 \sin[x], x] = \text{Integrate}[x^2 \sin[x], x]$.

2. Select the inline cell and use **Cell ▸ Convert To ▸ TraditionalForm**.

Here is some text with an identity, $\int x^2 \sin(x) dx = \int x^2 \sin(x) dx$.

3. Select the right-hand integral (you can make this selection by triple-clicking on this integral) and use **Kernel ▸ Evaluation ▸ Evaluate In Place**.

Here is some text with an identity, $\int x^2 \sin(x) dx = 2 x \sin(x) - (x^2 - 2) \cos(x)$.

■ Object-Oriented Quantum Mechanics

Frank J. Kampas

fkampas@msn.com

One of the fundamental principles of object-oriented programming is that the internal structures of objects are private (hidden). The response of the objects to “messages” passed to them is independent of their internal structures. This is similar to the fact that measurements upon quantum mechanical states give definite results independent of the various ways in which the states may be represented (e.g., wave functions, vectors in a Hilbert space). This similarity is illustrated here by constructing objects which represent both the states of a quantum mechanical harmonic oscillator and the operators associated with measurements on those states.

Object-oriented programming can be implemented in *Mathematica* using Roman Maeder’s package **Classes.m** (available from www.mathsource.com/Content/0205-186), which is described in [1, p. 60] and [2, p. 293]. First we load the package.

```
In[1]:= << Classes.m
```

Now we can construct the simple harmonic oscillator class, **sho**. The class members are states of definite energy, described by two instance variables: a complex amplitude **a** and a nonnegative integer quantum number **n**.

- **new** is the constructor.
- **amp** and **num** return the amplitude and quantum number of the object.
- **mult** multiplies the amplitude of the state by a constant.

- **raise** and **lower** increase and decrease the quantum number, respectively.
- The **energy** function corresponds to measuring the energy of the state and returns the state multiplied by its energy. $\hbar$ is Planck's constant divided by 2π , and ω is the radial frequency of the corresponding classical harmonic oscillator.
- **inprd** returns the inner product of two states. The inner product of states s_1 and s_2 is designated by $\langle s_1 | s_2 \rangle$ in Dirac bracket notation [3, p. 19].

Note that the raising and lowering operators do not correspond to physical variables, but will be useful for defining other functions. Also note that the states are not normalized unless the magnitude of a is 1.

```

ln[2]:= Class[sho, Object, {a, n},
  (
    new      If[MatchQ[
               #2, _Integer?NonNegative]
             , new(super);
               a = #1; n = #2,
             Print["unsuitable quantum
                   number"]] &
    amp      a &
    num      n &
    mult      new(sho, a #1, n) &
    raise     new(sho, a Sqrt[n + 1], n + 1) &
    lower     If[n > 0, new(sho, a Sqrt[n], n - 1),
               new(sho, 0, 0)] &
    energy    new(sho, (n + 1/2) hbar omega a, n) &
    inprd     If[n == num(#1), a* amp(#1), 0] &
  )
];

```

We use a simple trick for computing the complex conjugate of complex expressions involving real variables using a replacement rule.

```
ln[3]:= x_* := x /. Complex(a_, b_) -> Complex(a, -b)
```

The Dirac bracket notation can be implemented using the **Notation** package.

```
ln[4]:= << Utilities`Notation`;
```

```
ln[5]:= SetOptions[Notation, WorkingForm -> TraditionalForm]
```

```
Out[5]:= {WorkingForm -> TraditionalForm, Action -> CreateNotationRules}
```

```
ln[6]:= Notation((x_ | y_) ==> inprd[x_, y_])
```

```
ln[7]:= Notation((x_ | m_ | y_) ==> inprd[x_, m_ @ y_])
```

It is also useful to have a notation for **mult**.

```
ln[8]:= Notation((a_ \otimes x_) ==> mult[x_, a_])
```

The following steps illustrate the construction of a state s with an amplitude of 1 and a quantum number of 2, and the determination of the expectation value of its energy, which is $\langle s | \text{energy} | s \rangle / \langle s | s \rangle$ in Dirac notation.

In[9]:= $s = \text{new}(\text{sho}, 1, 2);$

In[10]:= $\frac{\langle s | \text{energy} | s \rangle}{\langle s | s \rangle}$

Out[10]:= $\frac{5 \omega \hbar}{2}$

The next step is the construction of a class whose members are linear combinations of states with definite energy. Members of class **lincom** are sequences of objects of type **sho**.

- **new** is the constructor.
- **states** returns a list of the **sho** components making up a variable of type **lincom**.
- **mult** and **inprd** are similar to the functions in class **sho** and are based upon them.
- The functions x and p are the position and momentum operators, respectively.

```
In[11]:= Class lincom, Object, {shos},
(
  new      (new(super); shos == #1) &
  states   {shos} &
  mult     new(lincom, Sequence @@
              Table[mult({shos}[[i]], #1),
                    {i, Length[{shos}]]]) &
  inprd     Plus @@ Flatten [Outer [inprd,
                                    {shos}, states(#1)]] &
  x        mult (new(lincom, Sequence @@
                      Join [raise /@ {shos}, lower /@
                            {shos}]),  $\sqrt{\frac{\hbar}{2 m \omega}}$ ) &
  p        mult (new(lincom, Sequence @@
                      Join [raise /@ {shos}, lower /@
                            (mult(#1, -1) &) /@
                            {shos}]),  $i \sqrt{\frac{m \omega \hbar}{2}}$ ) &
)
```

Using the two classes, it is possible to calculate the matrix elements of the x and p operators, which are defined as $\langle n | x | m \rangle$ and $\langle n | p | m \rangle$, where n and m are states with amplitude 1 and quantum numbers n and m , respectively.

Single energy state members of the class **lincom** are constructed first.

```
In[12]:= s_i := new(sho, 1, i); l_i := new(lincom, s_i)
```

The following are scaled matrix representations of x and p (see [4, p. 103]).

$$\text{In[13]:= Table}[\langle l_i | x | l_j \rangle, \{i, 0, 5\}, \{j, 0, 5\}] / \sqrt{\frac{\hbar}{2m\omega}}$$

$$\text{Out[13]= } \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & \sqrt{2} & 0 & 0 & 0 \\ 0 & \sqrt{2} & 0 & \sqrt{3} & 0 & 0 \\ 0 & 0 & \sqrt{3} & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 & 0 & \sqrt{5} \\ 0 & 0 & 0 & 0 & \sqrt{5} & 0 \end{pmatrix}$$

$$\text{In[14]:= Table}[\langle l_i | p | l_j \rangle, \{i, 0, 5\}, \{j, 0, 5\}] / \sqrt{\frac{\hbar m \omega}{2}}$$

$$\text{Out[14]= } \begin{pmatrix} 0 & -i & 0 & 0 & 0 & 0 \\ i & 0 & -i\sqrt{2} & 0 & 0 & 0 \\ 0 & i\sqrt{2} & 0 & -i\sqrt{3} & 0 & 0 \\ 0 & 0 & i\sqrt{3} & 0 & -2i & 0 \\ 0 & 0 & 0 & 2i & 0 & -i\sqrt{5} \\ 0 & 0 & 0 & 0 & i\sqrt{5} & 0 \end{pmatrix}$$

It is also possible to construct time-dependent states $s(i, t)$ after constructing a function e_i which gives the energy of a state with quantum number i .

```
In[15]:= e_i := <s_i | energy | s_i>
```

```
In[16]:= s(i_, t_) := e^{-\frac{i e_i t}{\hbar}} \otimes s_i
```

Constructing an equal mixture of the two states $s(0, t)$ and $s(1, t)$,

$$\text{In[17]:= It = new}\left(\text{lincom}, \sqrt{\frac{1}{2}} \otimes s(0, t), \sqrt{\frac{1}{2}} \otimes s(1, t)\right);$$

we compute the expectation value of the position operator, $\langle x \rangle$.

```
In[18]:= <It | x | It> // ExpToTrig
```

$$\text{Out[18]= } \frac{\sqrt{\frac{\hbar}{m\omega}} \cos(t\omega)}{\sqrt{2}}$$

We see that $\langle x \rangle$ oscillates back and forth at frequency ω .

■ Maxwell Construction

The equation of state of an ideal gas of N noninteracting classical particles reads [5, Section 1.8]

$$p V = N k_B T, \quad (1)$$

where k_B is the Boltzmann constant. A simple approximation, taking into account the interaction between particles, is the van der Waals equation of state,

$$\left(p + \frac{a}{V^2}\right)(V - b) = N k_B T, \quad (2)$$

with parameters a and b . It turns out that there exists a critical temperature T_c below which loops appear in the p - v diagram. These loops are forbidden for thermodynamic reasons. For $T < T_c$, a phase transition between gas and liquid takes place; at $T = T_c$ the difference between the liquid and gaseous phases vanishes. T_c and the corresponding critical volume V_c are determined by the condition that the first and second derivatives of $p(V, T)$ with respect to V vanish (i.e., $\frac{\partial p(V, T)}{\partial V} = 0$ and $\frac{\partial^2 p(V, T)}{\partial V^2} = 0$). We define $p(V, T)$ by solving for p in the van der Waals equation (2),

$$\text{In}[1]:= p(V_ , T_) = p /. \text{Solve}\left[\left(p + \frac{a}{V^2}\right)(V - b) = N k_B T, p\right] // \text{First} // \text{Simplify}$$

$$\text{Out}[1]= \frac{T N k_B V^2 + a(b - V)}{V^2 (V - b)}$$

and determine the critical values V_c and T_c .

$$\text{In}[2]:= \{V_c, T_c\} =$$

$$\{V, T\} /. \text{First}[\text{Solve}[\{\frac{\partial p(V, T)}{\partial V}, \frac{\partial^2 p(V, T)}{\partial V^2}\} = \{0, 0\}, \{V, T\}]]$$

$$\text{Out}[2]= \left\{3 b, \frac{8 a}{27 b N k_B}\right\}$$

The critical pressure p_c is defined by $p_c = p(V_c, T_c)$.

$$\text{In}[3]:= p_c = p(V_c, T_c)$$

$$\text{Out}[3]= \frac{a}{27 b^2}$$

We rescale p , V , and T by p_c , V_c , and T_c , respectively.

$$\text{In}[4]:= \tilde{p}(v_ , t_) = \text{Apart}\left[\frac{p(v V_c, t T_c)}{p_c}, v\right]$$

$$\text{Out}[4]= \frac{8 t}{3 v - 1} - \frac{3}{v^2}$$

Then we denote the dimensionless parameters using $\tilde{x} = x/x_c$ and obtain a dimensionless form of the van der Waals equation (2).

```
In[5]:=  $\tilde{T} == (\tilde{T} /. \text{Factor}[\text{First}[\text{Solve}[\tilde{p} == \tilde{p}(\tilde{V}, \tilde{T}), \tilde{T}]]])$ 
```

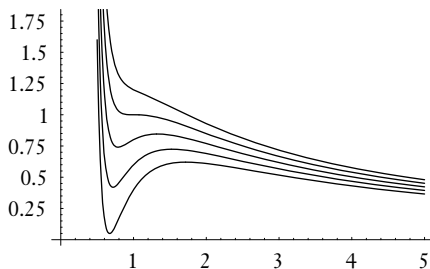
```
Out[5]:=  $\tilde{T} == -\frac{1}{8} \left( \tilde{p} + \frac{3}{\tilde{V}^2} \right) (1 - 3 \tilde{V})$ 
```

Plotting a set of $\tilde{p} - \tilde{V}$ curves for $\tilde{T} = 0.85, 0.9, \dots, 1.05$ over $0.5 \leq \tilde{V} \leq 5$ displays the unphysical van der Waals loops which arise for $\tilde{T} < 1$.

```
In[6]:= vanderWaals =
```

```
Plot[Evaluate[Table[ $\tilde{p}(v, t)$ , { $t$ , 0.85, 1.05, 0.05}]], { $v$ , 0.5, 5}];
```

From In[6]:=



Over a certain temperature range, three volumes $v_1 \leq v_2 \leq v_3$ correspond to each pressure $\tilde{p}$. For thermodynamic reasons, the transition from the gas with large volume v_3 , to the liquid with small volume v_1 , takes place at the pressure $\tilde{p}_t$ for which

$$\int_{v_1}^{v_3} \tilde{p}(v, t) dv = (v_3 - v_1) \tilde{p}_t. \quad (3)$$

At $\tilde{p}_t$, gas and liquid are simultaneously present for all volumes v with $v_1 \leq v \leq v_3$; one observes a two-phase mixture. Geometrically, equation (3) means that the area between the curve $\tilde{p}(v)$ and the straight line $\tilde{p}_t = \text{constant}$ for $v_1 \leq v \leq v_2$ is the same as the corresponding area for $v_2 \leq v \leq v_3$. This is known as the *Maxwell construction*. We want to construct the line $\tilde{p}_t = \text{constant}$, i.e., the two-phase line. For the two-phase line we need two equations in order to determine the two unknown volumes v_1 and v_3 :

$$\begin{aligned} \tilde{p}(v_1, t) &= \tilde{p}(v_3, t), \\ \int_{v_1}^{v_3} \tilde{p}(v, t) dv &= (v_3 - v_1) \tilde{p}(v_1, t). \end{aligned} \quad (4)$$

There is no simple analytical solution to equations (4) so, for each temperature $\tilde{T} = 0.85, 0.9, 0.95$, we determine v_1 and v_3 using **FindRoot**. Fairly accurate initial guesses are required, and these can be obtained from the plots of the unphysical van der Waals $\tilde{p} - \tilde{V}$ equation.

The two equations read

```
In[7]:=  $\mathcal{E}_1(t\_)= \tilde{p}(v_1, t) == \tilde{p}(v_3, t)$ 
```

```
Out[7]:=  $\frac{8t}{3v_1 - 1} - \frac{3}{v_1^2} == \frac{8t}{3v_3 - 1} - \frac{3}{v_3^2}$ 
```

and

$$\begin{aligned} \text{In}[8]:= & \mathcal{E}_2(\mathbf{t}_-) = \\ & \left(\int_a^b \tilde{p}(v, t) dv /. \{a \rightarrow v_1, b \rightarrow v_3\} // \text{Expand} // \text{Simplify} \right) = \tilde{p}(v_1, t) (v_3 - v_1) \\ \text{Out}[8]:= & -\frac{8}{3} t (\log(3 v_1 - 1) - \log(3 v_3 - 1)) - \frac{3}{v_1} + \frac{3}{v_3} == \left(\frac{8 t}{3 v_1 - 1} - \frac{3}{v_1^2} \right) (v_3 - v_1) \end{aligned}$$

An automatic way to find initial guesses follows.

$$\begin{aligned} \text{In}[9]:= & \text{initial}[\mathbf{t}_- /; t < 1] := \\ & \text{Module}[\{v, v1, v3\}, \{v1, v3\} = v /. \text{Solve}[\partial_v \tilde{p}(v, t) = 0, v] // \text{Rest}; \\ & \left(v /. \text{Solve}[\tilde{p}(v, t) = \tilde{p}\left(\frac{v1 + v3}{2}, t\right), v] \right) \{1, 3\}] \end{aligned}$$

These initial guesses allow the volumes to be determined using **FindRoot**.

$$\begin{aligned} \text{In}[10]:= & \text{volumes}(\mathbf{t}_- /; t < 1) := \text{Module}[\{v1, v3\}, \{v1, v3\} = \text{initial}(t); \\ & \text{FindRoot}[\text{Evaluate}[\{\mathcal{E}_1(t), \mathcal{E}_2(t)\}, \{v1, v1\}, \{v3, v3\}]]] \end{aligned}$$

Here are v_1 and v_3 for each temperature $\tilde{T} = 0.85, 0.9, 0.95$.

$$\begin{aligned} \text{In}[11]:= & \text{Table}[\{t, \text{volumes}[t]\}, \{t, 0.85, 0.95, 0.05\}] \\ \text{Out}[11]:= & \begin{pmatrix} 0.85 & \{v_1 \rightarrow 0.55336, v_3 \rightarrow 3.12764\} \\ 0.9 & \{v_1 \rightarrow 0.603402, v_3 \rightarrow 2.34884\} \\ 0.95 & \{v_1 \rightarrow 0.684122, v_3 \rightarrow 1.72707\} \end{pmatrix} \end{aligned}$$

Now we display the Maxwell construction for the van der Waals equation. Defining

$$\text{In}[12]:= \tilde{p}_{\text{Maxwell}}(\mathbf{v}_-, \mathbf{t}_-) := \tilde{p}(v, t)$$

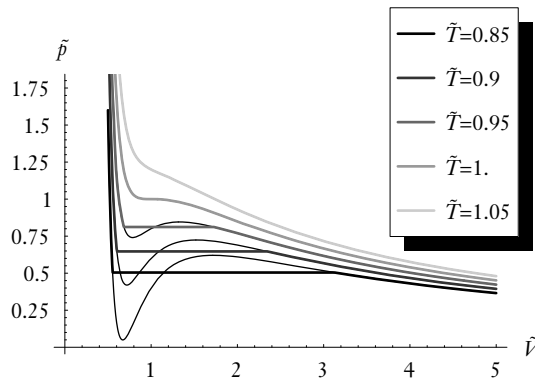
and

$$\begin{aligned} \text{In}[13]:= & \tilde{p}_{\text{Maxwell}}(\mathbf{V}_-, \mathbf{t}_- /; t < 1) := \text{With}[\{\text{vols} = \text{volumes}(t)\}, \\ & \text{If}[V < v_1 \vee V > v_3 /. \text{vols}, \tilde{p}(V, t), \tilde{p}(v_1 /. \text{vols}, t)]] \end{aligned}$$

here are plots of the Maxwell construction.

$$\begin{aligned} \text{In}[14]:= & << \text{Graphics`Legend`} \\ \text{In}[15]:= & \text{Plot}[\text{Evaluate}[\text{Table}[\tilde{p}_{\text{Maxwell}}(v, t), \{t, 0.85, 1.05, 0.05\}]], \{v, 0.5, 5\}, \\ & \text{AxesLabel} \rightarrow \{\tilde{V}, \tilde{p}\}, \text{PlotRange} \rightarrow \text{PlotRange}[\text{vanderWaals}], \\ & \text{PlotStyle} \rightarrow \text{Table}[\{\text{AbsoluteThickness}[1], \text{GrayLevel}[t]\}, \{t, 0, 1, \frac{1}{5}\}], \\ & \text{PlotLegend} \rightarrow \text{Table}["\tilde{T} = " <> \text{ToString}[t], \{t, 0.85, 1.05, 0.05\}], \\ & \text{LegendPosition} \rightarrow \{0.4, -0.1\}, \text{Prolog} \rightarrow \text{vanderWaals}[1]]; \end{aligned}$$

From In[15]:=



The three lowest curves (i.e., $\tilde{T} = 0.85, 0.9, 0.95$) show the separation into liquid (small volume) and gaseous phases. Along each two-phase line, liquid and gas coexist in thermal equilibrium.

■ References

- [1] R. E. Maeder, *The Mathematica Programmer*, Burlington: Academic Press, 1994.
- [2] J. Gray, *Mastering Mathematica*, Burlington: Academic Press, 1994.
- [3] P. A. M. Dirac, *The Principles of Quantum Mechanics*, 4th ed., Oxford: Oxford University Press, 1958.
- [4] J. Feagin, *Quantum Methods with Mathematica*, New York: TELOS (Springer-Verlag), 1993.
- [5] W. Kinzel and G. Reents, *Physics by Computer*, Berlin: Springer-Verlag, 1998.

In and Out

Edited by Paul Abbott

*Department of Physics
University of Western Australia
Nedlands WA 6907, Australia
tmj@physics.uwa.edu.au
physics.uwa.edu.au/~paul*

In and Out offers readers an opportunity to ask questions of the experts. *The Mathematica Journal* encourages readers to submit problems in care of the editor.

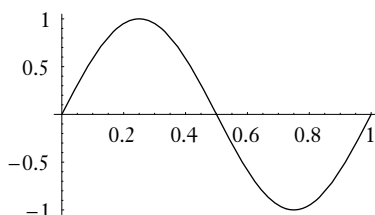
■ Automatic Grouping of Animation Cells

Q: Is there a way to automatically close the group of graphics cells in an animation after they have been generated?

A: P. J. Hinton (paulh@wolfram.com) writes: You can achieve this using the following code snippet.

```
In[1]:= Do[Plot[sin(2  $\pi$  f t), {t, 0, 1}], {f, 1, 3}];  
SelectionMove[EvaluationNotebook[], After, EvaluationCell];  
SelectionMove[EvaluationNotebook[], Next, CellGroup];  
FrontEndTokenExecute["OpenCloseGroup"]
```

From In[1]:=



For this method to work, you need to set **Cell Grouping** ▸ **Automatic Grouping** under the **Cell** menu (and not have modified the **CellGroupingRules** for **Graphics** cells in your style sheet).

■ Calculus with Anticommuting Variables

Q: How can I perform matrix calculus (products, determinants, and so on) with anticommuting variables?

A: Daniel Lichtblau (danl@wolfram.com) answers.

Say you encapsulate all anticommuting variables with the head $\mathcal{A}$. Then you can define product rules, here using $\otimes$, to handle the commutation relations and furthermore distribute over **Plus**, pull out “scalars” (everything not with head $\mathcal{A}$) using ordinary **Times**, and so on. Matrix multiplication is then done using just **Inner** with this new product specified in place of **Times**. Each rule below, followed by an example, illustrates this method.

1. Repeated anticommuting variables.

$\text{In}[1]:= a \otimes \mathcal{A}(x) \otimes b \otimes \mathcal{A}(x) \otimes c := 0$

$\text{In}[2]:= a \otimes \mathcal{A}(x) \otimes b \otimes \mathcal{A}(x) \otimes c$

$\text{Out}[2]= 0$

2. Canonical order for a product of anticommuting variables.

$\text{In}[3]:= \mathcal{A}(x) \otimes \mathcal{A}(y) \otimes \mathcal{A}(z) \dots /; !(\text{OrderedQ}[\{x, y, z\}]) :=$
 $\text{Signature}[\{x, y, z\}] (\otimes \text{Sequence} @@ \mathcal{A} / @ \text{Sort}[\{x, y, z\}])$

$\text{In}[4]:= \mathcal{A}(z) \otimes \mathcal{A}(y)$

$\text{Out}[4]= -(\mathcal{A}(y) \otimes \mathcal{A}(z))$

$\text{In}[5]:= \mathcal{A}(z) \otimes \mathcal{A}(y) \otimes \mathcal{A}(x)$

$\text{Out}[5]= -(\mathcal{A}(x) \otimes \mathcal{A}(y) \otimes \mathcal{A}(z))$

3. Reduction for a single term.

$\text{In}[6]:= \otimes a /; \text{Head}[a] \neq \text{Times} := a$

$\text{In}[7]:= \otimes \mathcal{A}(z)$

$\text{Out}[7]= \mathcal{A}(z)$

4. Distribute over **Plus**.

$\text{In}[8]:= x \otimes y \text{ Plus } \otimes z := (x \otimes \#1 \otimes z \ \&) / @ y$

$\text{In}[9]:= x \otimes (a + b + c) \otimes z$

$\text{Out}[9]= x \otimes a \otimes z + x \otimes b \otimes z + x \otimes c \otimes z$

5. Distribute over **Times**.

$\text{In}[10]:= x \otimes y \text{ Times } \otimes z := x \otimes \text{Sequence} @@ y \otimes z$

$$\text{In}[11]:= x \otimes (a \, b \, c) \otimes z$$

$$\text{Out}[11]= x \otimes a \otimes b \otimes c \otimes z$$

6. Distribute over **CircleTimes**.

$$\text{In}[12]:= x_ \otimes y_ \text{CircleTimes} \otimes z_ := x \otimes \text{Sequence} @@ y \otimes z$$

$$\text{In}[13]:= x \otimes (a \otimes b \otimes c) \otimes z$$

$$\text{Out}[13]= x \otimes a \otimes b \otimes c \otimes z$$

7. Null argument **CircleTimes**.

$$\text{In}[14]:= \text{CircleTimes}[] = 1;$$

8. Pull out “scalars.”

$$\text{In}[15]:= a_ \otimes b_ /; \text{Head}[a] \neq \mathcal{A} \vee \text{Head}[b] \neq \mathcal{A} := a \, b$$

$$\text{In}[16]:= a \otimes \mathcal{A}(b)$$

$$\text{Out}[16]= a \, \mathcal{A}(b)$$

$$\begin{aligned} \text{In}[17]:= & \bigotimes \text{vars_} /; \text{Length}[\{\text{vars}\}] > 2 := \text{Module}[\{\text{vl} = \{\text{vars}\}, \text{antis}, \text{coms}\}, \\ & \text{antis} = \text{Cases}[\text{vl}, \mathcal{A}(_)] ; \text{coms} = \text{Complement}[\text{vl}, \text{antis}] ; \\ & \text{Times} @@ \text{coms} \text{CircleTimes} @@ \text{antis} /; \text{Length}[\text{coms}] > 0] \end{aligned}$$

$$\text{In}[18]:= \mathcal{A}(c) \otimes \mathcal{A}(b) \otimes d \otimes \mathcal{A}(a)$$

$$\text{Out}[18]= -d \, \mathcal{A}(a) \otimes \mathcal{A}(b) \otimes \mathcal{A}(c)$$

Now consider a 3×3 matrix with mixed commuting (x_i) and anticommuting ($\mathcal{A}(j)$) entries.

$$\text{In}[19]:= m_1 = \begin{pmatrix} \mathcal{A}(1) & x_1 & \mathcal{A}(2) \\ x_2 & \mathcal{A}(1) + 2 \, \mathcal{A}(2) & 3 \, \mathcal{A}(1) - \mathcal{A}(2) \\ \mathcal{A}(1) - \mathcal{A}(2) & x_3 & \mathcal{A}(2) - 3 \, \mathcal{A}(1) \end{pmatrix};$$

We find its inner product with itself using the above rules.

$$\text{In}[20]:= \text{Inner}[\text{CircleTimes}, m_1, m_1]$$

$$\text{Out}[20]= \begin{pmatrix} x_1 \, x_2 - \mathcal{A}(1) \otimes \mathcal{A}(2) & 2 \, x_1 \, \mathcal{A}(1) + 2 \, x_1 \, \mathcal{A}(2) + x_3 \, \mathcal{A}(2) & 4 \, \mathcal{A}(1) \otimes \mathcal{A}(2) + 3 \, x_1 \, \mathcal{A}(1) - x_1 \, \mathcal{A}(2) \\ -2 \, \mathcal{A}(1) \otimes \mathcal{A}(2) + 2 \, x_2 \, \mathcal{A}(1) + 2 \, x_2 \, \mathcal{A}(2) & x_1 \, x_2 + 3 \, x_3 \, \mathcal{A}(1) - x_3 \, \mathcal{A}(2) & x_2 \, \mathcal{A}(2) - 7 \, \mathcal{A}(1) \otimes \mathcal{A}(2) \\ 3 \, \mathcal{A}(1) \otimes \mathcal{A}(2) + x_2 \, x_3 & x_1 \, \mathcal{A}(1) - 2 \, x_3 \, \mathcal{A}(1) - x_1 \, \mathcal{A}(2) + 3 \, x_3 \, \mathcal{A}(2) & \mathcal{A}(1) \otimes \mathcal{A}(2) + 3 \, x_3 \, \mathcal{A}(1) - x_3 \, \mathcal{A}(2) \end{pmatrix}$$

To do determinants one can use Laplace expansion, again substituting $\otimes$ for **Times**.

$$\text{In}[21]:= \|(a_)\| := a$$

For efficiency we memo-ize the results so that we do not recompute minor expansions but instead look them up once they are computed.

$\text{In}[22]:= \text{||m_? MatrixQ|| /; Length[m] === Length[m[[1]]] :=}$
 $\text{||m|| = With[{l = Length[m], r = Drop[m, \{1\}]},}$
 $\sum_{j=1}^l (-1)^{j+1} m[[1, j]] \otimes \text{||Drop[r, None, \{j\}||}]$

$\text{In}[23]:= \text{||m}_1\text{||}$

$\text{Out}[23]= -2 \mathcal{A}(1) \otimes \mathcal{A}(2) x_1 + 3 x_2 \mathcal{A}(1) x_1 - x_2 \mathcal{A}(2) x_1 + \mathcal{A}(1) \otimes \mathcal{A}(2) x_3 + x_2 x_3 \mathcal{A}(2)$

We conclude by clearing the rules we have assigned.

$\text{In}[24]:= \text{Clear[CircleTimes, DoubleBracketingBar]}$

Fast Computation of Equivalence Classes

Roman E. Maeder

*MathConsult Dr. R. Mäder
Samstagernstrasse 58a
8832 Wollerau, Switzerland
maeder@mathconsult.ch
www.mathconsult.ch*

This installment in our column can be summed up as “the more you know about your equivalence relations, the faster your programs will run.” Partitioning data into classes or picking representative elements is a problem that appears in many forms. All interesting equivalence relations are generated by a preorder or a classifying function. This fact can be used to develop efficient programs.

■ Equivalences

One type of question keeps reappearing in Usenet discussions and *Mathematica* user groups. It comes in many forms and recognizing that it is a question about equivalences is already half the answer. Here is one formulation:

Given a set (or list), partition it into its equivalence classes. Each element belongs to exactly one class and two elements in a class are equivalent. Instead of the classes themselves we may be interested simply in the *indices* of the elements. From the indices we can easily find the elements, but not vice versa. The equivalence relation is given as a function of two arguments that returns **True** if the two arguments should be considered equivalent, and **False** otherwise.

This program by Xah Lee appeared on the newsgroup comp.soft-sys.math.mathematica in March 1998 [1]. (In another discussion, the same function was named **MatchingPositions**.)

```

In[1]:= EquivalenceIndices[li_, sameQ_: SameQ] :=
  Module[{indices = Range[Length[li]], eq, res = h[]},
    While[indices != {},
      With[{el = li[[First[indices]]]},
        eq = Select[indices, sameQ[el, li[[#]]] &];
        res = h[res, eq];
        indices = Complement[indices, eq]
      ]
    ];
    List @@ Flatten[res, Infinity, h]
  ]

```

The idea is to pick the first element, find all remaining elements that are equivalent, then remove them and continue with the remaining elements until no more remain. Instead of working with the elements we use their indices and access the element with index i with `li[[i]]`. The result is first built up in a recursive structure (a stack), then converted to a list in one step, using **Flatten**. This method has complexity $O(n^2)$, where n is the length of the input list.

The default predicate generates the trivial equivalence class where each element is only equivalent to itself. The resulting index set allows us to quickly locate where the identical elements appear in the original list.

Once we have the indices, we can easily get the elements themselves.

```

In[2]:= EquivalenceClasses[li_, sameQ_: SameQ] :=
  Part[li, #] & /@ EquivalenceIndices[li, sameQ]

```

For the trivial equivalence **SameQ** the result of **EquivalenceClasses[1]** is a partitioning of `l` into lists of identical elements.

For an example, let us group all built-in *Mathematica* symbols by their length. First we obtain a list of all symbol names in the **System`** context.

```

In[3]:= allSymbols = Names["System`*"]; Length[allSymbols]

```

```

Out[3]= 1846

```

Here we measure the time it takes to split the list into lists of symbols with the same length. Using pure functions (of two arguments) is a convenient way to express arbitrary equivalences.

```

In[4]:= Timing[sameLength = EquivalenceClasses[
  allSymbols, StringLength[#1] == StringLength[#2] &];]

```

```

Out[4]= {0.716667 Second, Null}

```

There are this many different lengths of built-in symbols.

```

In[5]:= Length[sameLength]

```

```

Out[5]= 33

```

One of the equivalence classes is shown here. Note that it is not necessarily the one with the shortest symbols. The ordering of the classes is not defined.

```
In[6]:= sameLength[[1]]
```

```
Out[6]= {Abort, Above, Adams, AddTo, After, Alias, Apart, Apply, Array, AtomQ,
        Begin, Below, BitOr, Blank, Block, Boxed, Break, Cases, Catch,
        CForm, Check, Clear, Close, Count, Cross, Debug, Depth, Empty,
        Enter, Equal, EvenQ, False, First, Floor, Frame, FreeQ, Gamma,
        Heads, Infix, Inner, Input, Label, Level, Limit, Links, ListQ,
        MapAt, Minus, NameQ, Names, Needs, Order, Outer, Pause, Plain,
        Point, Power, Prime, Print, Range, Reals, Right, Roots, Round,
        SameQ, Share, Short, Solve, Sound, Space, Split, Stack, Table,
        Throw, Ticks, Times, Trace, TrueQ, Union, Unset, UpSet, Using,
        V2Get, Value, Which, While, Write, $Echo, $Line, $Path, $Post}
```

We can speed up this computation significantly by first sorting the symbols by their lengths and then splitting the result at those places where the lengths of adjacent elements differ. The predicate for **Split** is actually the reverse: the list is split where the predicate is **False**, so we use `==` instead of `≠`.

```
In[7]:= Timing[sameLength2 =
        Split[Sort[allSymbols, StringLength[#1] ≤ StringLength[#2] &],
        StringLength[#1] == StringLength[#2] &];]
```

```
Out[7]= {0.433333 Second, Null}
```

This method has a complexity of $O(n \log n)$, the complexity of sorting. The longer the input the larger the difference in run time between the two methods. Intuitively, the reason that this trick works is that the equivalence relation alone gives no information beyond a simple yes/no, and a negative answer does not help us at all. Essentially we have to compare every element with every other one. An ordering relation gives more information; consequently, sorting can be done much faster.

Because of the sorting, the first equivalence class is the one with the shortest symbols.

```
In[8]:= sameLength2[[1]]
```

```
Out[8]= {C, D, E, I, K, N, O}
```

Note how we found the ordering predicate `StringLength[#1] ≤ StringLength[#2] &` from the equivalence relation `StringLength[#1] == StringLength[#2] &` by replacing the equality by `≤`. The key is to find an ordering that is *compatible* with the equivalence. *Mathematica*'s default ordering predicate `OrderedQ[{#1, #2}] &` would not have worked. It is not compatible with the desired equivalence, because it does not guarantee that strings of the same length end up adjacent to each other; rather, it sorts strings lexicographically.

This column was motivated also by an analysis of a deliberate slowdown introduced in the **Union** function of *Mathematica* 4. The union of a list of elements can be found by first finding the equivalence classes, then taking just one of the equivalent elements (called a transversal).

```
In[9]:= EquivalenceTransversal[li_, sameQ_ :> SameQ] :=
        Part[li, First /@ EquivalenceIndices[li, sameQ]]
```

To demonstrate the effect, let us generate a long list of random integers with many repetitions and find a transversal, or simply the union of the list elements.

```
In[10]:= sampleIntegers = Table[Random[Integer, {1, 104}], {105}];
```

The built-in **Union** sorts the data, then takes one of each run of adjacent identical elements, which is quite fast. If every integer in the given range appears at least once, the length of the result will be 10,000.

```
In[11]:= Timing[Length[Union[sampleIntegers]]]
```

```
Out[11]= {0.183333 Second, 10000}
```

If we provide our own equivalence relation as an option, *Mathematica* switches to a slower $O(n^2)$ algorithm. Version 3 used sorting in all cases, and it was then recognized that the command failed with user-supplied equivalences that were not compatible with canonical ordering. The equivalence relation **SameQ** gives the same result as the built-in default, but the computation is much slower.

```
In[12]:= Timing[Length[Union[sampleIntegers, SameTest -> SameQ]]]
```

```
Out[12]= {255.533 Second, 10000}
```

The mathematical discipline that makes these notions precise is *universal algebra* [2], so let us have a look at the theory behind equivalences and the orders that generate them.

■ Preorders

An **equivalence** is a binary relation $\sim$ that is reflexive, symmetric, and transitive. An **equivalence class** is the set of all elements of the domain of an equivalence relation that are pairwise equivalent.

A **preorder** is a reflexive and transitive relation $\leq$ (the third property of an *order*, $x \leq y \wedge y \leq x \implies x = y$, may be missing). Call a preorder **total**, if $x \leq y \vee y \leq x$ for all x and y . Preorders are almost like orders; however, it may happen that x is less than y and y is less than x , without x being identical to y . Such a preorder generates an equivalence and it is exactly those problem elements that will be equivalent. The generated equivalence $x \sim y$ is therefore defined this way: $x \sim y \iff x \leq y \wedge y \leq x$. Note that a preorder induces a total order on the equivalence classes.

Mathematica's sorting algorithm works also if we specify a total *preorder* in place of a total *order* as an ordering predicate. Sorting a list using a total preorder brings equivalent elements into adjacent positions, and the partitioning can then be done in linear time. After sorting, the list $l = \{e_1, e_2, \dots, e_n\}$ has the property $e_i \leq e_{i+1}$ for all $1 \leq i < n$. If $e_{i+1} \leq e_i$ as well, the two elements are equivalent, otherwise they are not; therefore, we can split at the places where this condition is violated.

This procedure takes a preorder as an argument and finds the index partition of the equivalence generated by it. Note how the indices of the elements are tacked

onto the elements before sorting, so we know after sorting where the elements were originally. The default ordering predicate is `OrderedQ[{#1,#2}]&`, which we can write more elegantly as `Composition[OrderedQ,List]`.

```
In[13]:= PreorderIndices[li_, preQ_: Composition[OrderedQ, List]] :=
  Map[Last,
    Split[Sort[Transpose[{li, Range[Length[li]]}],
      preQ[First[#1], First[#2]] &],
      preQ[First[#2], First[#1]] &],
    {2}]
```

This method has complexity $O(n \log(n))$, because of sorting. The default predicate is canonical ordering, which leads to the trivial `SameQ` equivalence. We can use it as the basis of another, faster classification function. This time we implement it directly, without the detour through `PreorderIndices`.

```
In[14]:= PreorderClasses[li_, preQ_: Composition[OrderedQ, List]] :=
  Split[Sort[li, preQ], preQ[#2, #1] &]
```

```
In[15]:= Timing[sameLength3 = PreorderClasses[
  allSymbols, StringLength[#1] ≤ StringLength[#2] &];]
```

```
Out[15]= {0.45 Second, Null}
```

The computation of a transversal of our sample random integers using the default preorder is already much faster than is the built-in `Union` function.

```
In[16]:= Timing[First /@ PreorderClasses[sampleIntegers];]
```

```
Out[16]= {23.5 Second, Null}
```

One important source of preorders are canonical simplifiers.

■ Classifiers and Canonical Simplifiers

A **classifying function** f for an equivalence $\sim$ has the property $x \sim y \iff f(x) = f(y)$; that is, two elements are equivalent if the result of applying the classifier to both elements is the same. Note that the range of f need not be the same as the domain. For example, the length function on symbol names is a classifier for the relation introduced in the section on Equivalences. The length function appeared in all our ordering predicates and equivalence relations and we may wonder whether we could not speed up the code by computing the length once for each element and then working with the lengths only.

A **canonical simplifier** S is a classifier with the property $S(S(x)) = S(x)$ (here, the range is the same as the domain). This condition implies $S(x) \sim x$. A canonical simplifier thus allows to identify a unique (simplest) element in every equivalence class. For every x , $S(x)$ is this simplest element in the equivalence class of x .

If the range of a classifier is totally ordered, this order $\leq$ induces a total preorder on the domain: $x \preceq y \iff f(x) \leq f(y)$. The equivalence class generated by this preorder is of course the one classified by f .

Because *Mathematica* expressions are totally ordered (by the so-called *canonical ordering*), every classifier introduces a preorder: $x \approx y \iff \text{OrderedQ}[\{f(x), f(y)\}]$. It follows that whenever we can find a classifier or canonical simplifier, we can use the faster method involving sorting to find equivalence classes.

The code for partitioning into equivalence classes using a classifier is similar to the code for **PreorderIndices** above. The main advantage is that we can use the built-in ordering for sorting the list and **SameQ** for splitting, because we can first apply the classifier to all elements just once, instead of repeatedly in the preorder predicate (shown in red).

```
In[17]:= ClassifyIndices[li_, cf_: Identity] :=
  Map[Last,
    Split[Sort[Transpose[{cf /@ li, Range[Length[li]]}],
      SameQ[First[#1], First[#2]] &],
    {2}]
```

The default canonical simplifier **Identity** induces canonical ordering and generates the trivial equivalence.

```
In[18]:= ClassifyClasses[li_, cf_: Identity] :=
  Part[li, #] & /@ ClassifyIndices[li, cf]
```

This function was called **Classify** in a related Usenet discussion in 1995 [3], where it was implemented directly, without **ClassifyIndex**. However, the detour through the indices does not slow down the computation (as it did in the examples with **PreorderClasses**). The original solution was rather more complicated, due to the lack of the **Split** function in earlier versions of *Mathematica*.

```
In[19]:= Classify[li_, cf_: Identity] :=
  Map[Last,
    Split[Sort[{cf[#], #} & /@ li],
      SameQ[First[#1], First[#2]] &],
    {2}]
```

This method is much faster than the earlier ones.

```
In[20]:= Timing[sameLength4 = ClassifyClasses[allSymbols, StringLength];]
Out[20]:= {0.1 Second, Null}
```

The code for computing a transversal (or union) based on classifiers is much faster than the built-in **Union** with a user-defined equivalence relation for large inputs.

```
In[21]:= ClassifyTransversal[li_, cf_: Identity] :=
  Part[li, First /@ ClassifyIndices[li, cf]]

In[22]:= Timing[Length[ClassifyTransversal[sampleIntegers]]]
Out[22]:= {4.71667 Second, 10000}
```

If the classifier is a canonical simplifier there is one preferred transversal, where each class is represented by its canonical element. It is particularly easy to code.

```
In[23]:= CanonicalTransversal[li_, s_: Identity] := Union[s/@li]
```

Incidentally, we could also express **EquivalenceIndices** in terms of **Classify`Classes**, using the mapping from indices to list elements as the classifying function: two indices are equivalent if the corresponding list elements are the same.

```
In[24]:= EquivalenceIndices0[list_] :=  
ClassifyClasses[Range[Length[list]], list[[#]] &]
```

■ Examples

□ Lists under Reversal

This example appeared in the Usenet discussion that motivated this article [1]. The example was formulated in terms of permutations, but it works for arbitrary lists.

Two lists are equivalent if they are the same or one is the reverse of the other.

```
In[25]:= eq1 = (#1 === #2 || #1 === Reverse[#2]) &;
```

The preorder that generates this equivalence compares the smaller of the lists and its reversal (under canonical ordering).

```
In[26]:= pre1 = OrderedQ[{min[#1, Reverse[#1]], min[#2, Reverse[#2]]}] &;
```

Here **min** returns the smaller of its two elements (the built-in **Min** works only for numbers).

```
In[27]:= min[a_, b_] := If[OrderedQ[{a, b}], a, b]
```

The canonical simplifier chooses the smaller one of the lists and its reverse.

```
In[28]:= can1 = min[#, Reverse[#]] &;
```

We generate all permutations of the integers from 1 to 4.

```
In[29]:= perm4 = Permutations[Range[4]]
```

```
Out[29]:= {{1, 2, 3, 4}, {1, 2, 4, 3}, {1, 3, 2, 4}, {1, 3, 4, 2},  
           {1, 4, 2, 3}, {1, 4, 3, 2}, {2, 1, 3, 4}, {2, 1, 4, 3}, {2, 3, 1, 4},  
           {2, 3, 4, 1}, {2, 4, 1, 3}, {2, 4, 3, 1}, {3, 1, 2, 4}, {3, 1, 4, 2},  
           {3, 2, 1, 4}, {3, 2, 4, 1}, {3, 4, 1, 2}, {3, 4, 2, 1}, {4, 1, 2, 3},  
           {4, 1, 3, 2}, {4, 2, 1, 3}, {4, 2, 3, 1}, {4, 3, 1, 2}, {4, 3, 2, 1}}
```

Every equivalence class has two elements.

```
In[30]:= TableForm[ClassifyClasses[perm4, can1], TableDepth -> 1]
```

```
Out[30]//TableForm=
```

```
{ {1, 2, 3, 4}, {4, 3, 2, 1} }
{ {1, 2, 4, 3}, {3, 4, 2, 1} }
{ {1, 3, 2, 4}, {4, 2, 3, 1} }
{ {1, 3, 4, 2}, {2, 4, 3, 1} }
{ {1, 4, 2, 3}, {3, 2, 4, 1} }
{ {1, 4, 3, 2}, {2, 3, 4, 1} }
{ {2, 1, 3, 4}, {4, 3, 1, 2} }
{ {2, 1, 4, 3}, {3, 4, 1, 2} }
{ {2, 3, 1, 4}, {4, 1, 3, 2} }
{ {2, 4, 1, 3}, {3, 1, 4, 2} }
{ {3, 1, 2, 4}, {4, 2, 1, 3} }
{ {3, 2, 1, 4}, {4, 1, 2, 3} }
```

The canonical transversal consists of all lists with the property that they are smaller than their reversal.

```
In[31]:= CanonicalTransversal[perm4, can1]
```

```
Out[31]= { {1, 2, 3, 4}, {1, 2, 4, 3}, {1, 3, 2, 4}, {1, 3, 4, 2},
          {1, 4, 2, 3}, {1, 4, 3, 2}, {2, 1, 3, 4}, {2, 1, 4, 3},
          {2, 3, 1, 4}, {2, 4, 1, 3}, {3, 1, 2, 4}, {3, 2, 1, 4} }
```

Here, we generate all permutations of the integers from 1 to 6 and measure the timings for the three algorithms.

```
In[32]:= perm6 = Permutations[Range[6]];
```

```
In[33]:= Through[Apply[Function[data, Timing[#1[data, #2]]][[1]]] &,
                Transpose[
                  {{EquivalenceClasses, PreorderClasses, ClassifyClasses},
                   {eq1, pre1, can1}}], {1}]@perm6]
```

```
Out[33]= {6.83333 Second, 0.65 Second, 0.2 Second}
```

□ Lists under Permutations

This example shows very nicely the power of canonical simplifiers.

Two lists are equivalent if they are permutations of each other. It is rather cumbersome to find out whether one list is a permutation of another one. It is much easier to sort both lists and then compare the results. The lists are permutations of each other, if and only if the sorted lists are identical.

Here is the equivalence predicate, testing the sorted lists for equality.

```
In[34]:= eq2 = Sort[#1] === Sort[#2] &;
```

The corresponding preorder compares the sorted lists under canonical ordering.

```
In[35]:= pre2 = OrderedQ[{Sort[#1], Sort[#2]}] &;
```

Finally, here is the corresponding canonical simplifier, which is quite straightforward.

```
In[36]:= can2 = Sort;
```

Our test example is a list of 2000 lists of four random integers each.

```
In[37]:= data2 = Table[Random[Integer, {1, 5}], {2000}, {4}];
```

Finally, we find the timings for the three methods.

```
In[38]:= Through[Apply[Function[data, Timing[#1[data, #2];][[1]]] &,
    Transpose[
        {{EquivalenceIndices, PreorderIndices, ClassifyIndices},
        {eq2, pre2, can2}}], {1}]@data2]
```

```
Out[38]:= {3.53333 Second, 1.4 Second, 0.133333 Second}
```

■ References

- [1] <http://library.wolfram.com/mathgroup/archive/1998/Mar/msg00042.html>
- [2] W. Wechler, "Universal algebra for computer scientists," *EATCS Monographs on Theoretical Computer Science*, Vol. 25, Springer-Verlag, 1992.
- [3] <http://library.wolfram.com/mathgroup/archive/1995/Apr/msg00247.html>

■ Additional Material

The package **Equivalences.m** contains the code from this article and works with *Mathematica* Version 3.0 and later on any machine.

Visualization of Riemann Surfaces IIb

Compositions of Elementary Transcendental Functions

Michael Trott

Wolfram Research, Inc.
mtrott@wolfram.com

This is a continuation of the article from TMJ 7:4, focusing on the intricate surfaces that arise from the mathematics of inverse trigonometric and hyperbolic functions.

■ The Weierstrass Continuation Method for $\text{Log}[1+z]$

Here we will construct a part of the Riemann surface of $\ln(1+z)$ by Weierstrass's method. It is not applicable for more complicated examples. But it is instructive to see this method carried out in detail for a real example, since it is taught in every complex analysis course. In addition, this method fits well with the subject of this article. On the other hand, all elementary transcendental functions are built only from log functions, so knowledge of the analytic continuation of log is sufficient to deal with all functions under consideration here.

We start with an analytic function $f(z)$ having a Taylor expansion around z_0 . The superscript 0 on the $a_i^{(0)}$ refers to the expansion point z_0 .

$$f(z_0 + \delta) = \sum_{i=0}^{\infty} a_i^{(0)} \delta^i$$

Inside the disk of convergence of the Taylor expansion we can re-expand $f(z)$ around a regular point z_1 . There $f(z)$ has again a Taylor expansion (with coefficients $a_i^{(1)}$).

$$f(z_1 + \delta) = \sum_{i=0}^{\infty} a_i^{(1)} \delta^i$$

Using elementary algebra, it can be shown [1] that the coefficients $a_i^{(1)}$ can be expressed through the coefficients $a_i^{(0)}$ in the following way.

$$a_i^{(1)} = \sum_{j=i}^{\infty} \binom{j}{i} a_j^{(0)} (z_1 - z_0)^{j-i}$$

We use this formula repeatedly by encircling the branch point $z = -1$ of $\ln(1 + z)$. The i th coefficient ($i \neq 0$) of the Taylor series of $\ln(1 + z)$ around $z = 0$ is given by $-\frac{(-1)^i}{i}$.

`ln[1]:= Series[Log[1 + z], {z, 0, 6}]`

`Out[1]= z -  $\frac{z^2}{2}$  +  $\frac{z^3}{3}$  -  $\frac{z^4}{4}$  +  $\frac{z^5}{5}$  -  $\frac{z^6}{6}$  + 0[z]7`

`ln[2]:=  $\sum_{i=1}^{\infty} -\frac{(-1)^i}{i} z^i$`

`Out[2]= Log[1 + z]`

The function **analyticContinuation** implements this formula for calculating new coefficients of the Taylor series from the old ones. (This function makes heavy use of *Mathematica*'s symbolic summation capabilities.) Starting with the series term of the form **SeriesTerms**[*term0*, *termi*, *i*] (in the index variable *i*) it calculates the series terms around the next expansion point. Because of its different form, we keep the zeroth order term as an extra element in **SeriesTerms**.

`ln[3]:= analyticContinuation[SeriesTerms[term0_, termi_, i],  $\delta$ _]:=`
`Module[{j}, SeriesTerms[term0 +  $\sum_{j=1}^{\infty} (\text{termi} /. i \rightarrow j) \delta^j$ ,`
 `$\sum_{j=1}^{\infty} \text{Binomial}[j, i] (\text{termi} /. i \rightarrow j) \delta^{j-i}$ , i] // Simplify]`

Here is the first step of the analytic continuation, $\delta = z_1 - z_0$.

`ln[4]:= analyticContinuation[SeriesTerms[0, - $\frac{(-1)^i}{i}$ , i],  $\delta$ ]`

`Out[4]= SeriesTerms[Log[1 +  $\delta$ ], - $\frac{(-\delta)^i \delta^{-i} (1 + \delta)^{-i}}{i}$ , i]`

Be aware that such terms as $(-\delta)^i \delta^{-i}$ in the previous expression do not “simplify” to -1 . They contain the branch cut information relevant for us. We now carry out the analytic continuation repeatedly along a path formed by, say, 10 points $z_0, z_1, \dots, z_{10}$. We do not specify explicit numerical values for the z_i yet so that the formulas will be easier to read.

`ln[5]:= logSeriesTerms = FoldList[analyticContinuation,`
`SeriesTerms[0, - $\frac{(-1)^i}{i}$ , i], {z1 - z0, z2 - z1, z3 - z2,`
`z4 - z3, z5 - z4, z6 - z5, z7 - z6, z8 - z7, z9 - z8, z10 - z9}]];`

Here are the Taylor coefficients of the first few steps. The zeroth term contains explicit logarithms.

```
In[6]:= Take[logSeriesTerms, 4] // FullSimplify
```

$$\begin{aligned} \text{Out[6]} = & \left\{ \text{SeriesTerms}\left[0, -\frac{(-1)^i}{i}, i\right], \right. \\ & \text{SeriesTerms}\left[\text{Log}[1 - z_0 + z_1], -\frac{(z_0 - z_1)^i (-z_0 + z_1)^{-i} (1 - z_0 + z_1)^{-i}}{i}, i\right], \\ & \text{SeriesTerms}\left[\text{Log}[1 - z_0 + z_1] + \text{Log}\left[\frac{1 - z_0 + z_2}{1 - z_0 + z_1}\right], \right. \\ & \quad \left. -\frac{\left(\frac{z_1 - z_2}{1 - z_0 + z_1}\right)^i \left(\frac{1 - z_0 + z_2}{1 - z_0 + z_1}\right)^{-i} (-z_1 + z_2)^{-i}}{i}, i\right], \\ & \text{SeriesTerms}\left[\text{Log}[1 - z_0 + z_1] + \text{Log}\left[\frac{1 - z_0 + z_2}{1 - z_0 + z_1}\right] + \text{Log}\left[\frac{1 - z_0 + z_3}{1 - z_0 + z_2}\right], \right. \\ & \quad \left. -\frac{\left(\frac{z_2 - z_3}{1 - z_0 + z_2}\right)^i \left(\frac{1 - z_0 + z_3}{1 - z_0 + z_2}\right)^{-i} (-z_2 + z_3)^{-i}}{i}, i\right] \left. \right\} \end{aligned}$$

We can now use **Sum** to get closed-form expressions for all the series.

```
In[7]:= summedSeries[SeriesTerms[s0_, si_, i_], z_] := s0 + Sum[si z^i, {i, 1, Infinity}]
```

```
In[8]:= summedLogSeries = Simplify[summedSeries[#, z]] & /@ logSeriesTerms;
```

```
In[9]:= Take[summedLogSeries, 5]
```

$$\begin{aligned} \text{Out[9]} = & \left\{ \text{Log}[1 + z], \text{Log}[1 - z_0 + z_1] + \text{Log}\left[\frac{1 + z - z_0 + z_1}{1 - z_0 + z_1}\right], \right. \\ & \text{Log}[1 - z_0 + z_1] + \text{Log}\left[\frac{1 - z_0 + z_2}{1 - z_0 + z_1}\right] + \text{Log}\left[\frac{1 + z - z_0 + z_2}{1 - z_0 + z_2}\right], \\ & \text{Log}[1 - z_0 + z_1] + \text{Log}\left[\frac{1 - z_0 + z_2}{1 - z_0 + z_1}\right] + \text{Log}\left[\frac{1 - z_0 + z_3}{1 - z_0 + z_2}\right] + \text{Log}\left[\frac{1 + z - z_0 + z_3}{1 - z_0 + z_3}\right], \\ & \text{Log}[1 - z_0 + z_1] + \text{Log}\left[\frac{1 - z_0 + z_2}{1 - z_0 + z_1}\right] + \\ & \quad \left. \text{Log}\left[\frac{1 - z_0 + z_3}{1 - z_0 + z_2}\right] + \text{Log}\left[\frac{1 - z_0 + z_4}{1 - z_0 + z_3}\right] + \text{Log}\left[\frac{1 + z - z_0 + z_4}{1 - z_0 + z_4}\right] \right\} \end{aligned}$$

We see mainly the **Log** terms. If we would neglect branch cuts (the whole point here is, of course, not to do this), then all terms just reduce to **Log[1 + z - z₀ + z_i]**.

```
In[10]:= PowerExpand /@ Rest[summedLogSeries]
```

$$\begin{aligned} \text{Out[10]} = & \{ \text{Log}[1 + z - z_0 + z_1], \text{Log}[1 + z - z_0 + z_2], \\ & \text{Log}[1 + z - z_0 + z_3], \text{Log}[1 + z - z_0 + z_4], \text{Log}[1 + z - z_0 + z_5], \\ & \text{Log}[1 + z - z_0 + z_6], \text{Log}[1 + z - z_0 + z_7], \text{Log}[1 + z - z_0 + z_8], \\ & \text{Log}[1 + z - z_0 + z_9], \text{Log}[1 + z - z_0 + z_{10}] \} \end{aligned}$$

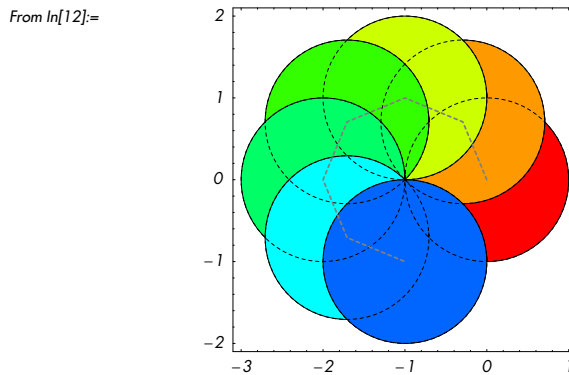
Now let us plug in explicit expansion points. We will encircle the point -1 (the branch point of **Log[1+z]**). Because the radius of convergence of the starting

power series is 1, we have to choose the next expansion point within a distance 1 from 0. Here is a possible choice for the z_j .

```
In[11]:= z_j_ = -1 + e $\frac{1}{8} j i 2 \pi$ ;
```

The picture below shows the expansion points together with their disks of convergence.

```
In[12]:= Show[Graphics[
  {Thickness[0.002], Table[{Hue[j/10], Disk[{Re[z_j], Im[z_j]}, 1],
    GrayLevel[0], Circle[{Re[z_j], Im[z_j]}, 1]}, {j, 0, 6}],
  GrayLevel[0], Dashing[{0.01, 0.01}],
  Table[Circle[{Re[z_j], Im[z_j]}, 1], {j, 0, 6}], {Thickness[0.005],
    GrayLevel[0.5], Line[Table[{Re[z_j], Im[z_j]}, {j, 0, 6}]]}],
  PlotRange -> All, AspectRatio -> Automatic, Frame -> True];
```



As the picture already suggests, the new expansion points lie inside the disks of convergence of the old ones.

```
In[13]:= Abs[z_1 - z_2] < 1
```

```
Out[13]:= True
```

Let us piecewise define a function **log** which is represented by the summed forms of the various series.

```
In[14]:= Do[log[j - 1, z_] = summedLogSeries[[j]], {j, 11}]
```

We look at the resulting Riemann surface by showing the values of the various **log** inside their disks of convergence.

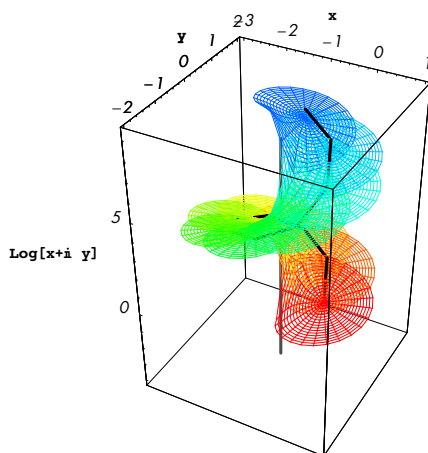
```
In[15]:= With[{rMax = 1 - 10-6, ppr = 16, ppφ = 24},
  Do[
    points[i] = Table[{Re[z_1 + r e $i \varphi$ ], Im[z_1 + r e $i \varphi$ ], Im[log[i, r e $i \varphi$ ]]},
      {r, 0., rMax, rMax/ppr}, {φ, 0, 2 π, 2 π/ppφ}], {i, 0, 10}]];
```

```

In[16]:= Show[Graphics3D[
  {{Thickness[0.001], Table[{Hue[0.06 i], Line /@ points[i],
    Line /@ Transpose[points[i]]}, {i, 0, 10}]},
  {Thickness[0.01], GrayLevel[0.3],
    Line[{{-1, 0, -4}, {-1, 0, 8}}]}, {Thickness[0.01],
    Line[Table[{Re[zi], Im[zi], Im[N[log[i, zi]]}, {i, 0, 10}]]}],
  Axes → True, PlotRange → All, BoxRatios → {1, 1, 1.5},
  AxesLabel → {StyleForm["x", "MB"],
    StyleForm["y", "MB"], StyleForm["Log[x+i y]", "MB"]}}];

```

From In[16]:=



After one round about the point -1 the function has changed (as expected) by $2\pi i$.

```

In[17]:= log[8, 0] - log[0, 0] // Simplify

```

```

Out[17]= 2 i π

```

This means that the function **log** is the analytic continuation of the built-in function **Log**.

■ The Riemann Surfaces of the Arc* Functions

Because of their practical importance, in this section we will construct pictures of the real and imaginary part as well as the absolute value of the inverse trigonometric and hyperbolic functions. Here is a table of some properties of these functions.

The second column gives the representation of the function using logarithms and square roots (which can be obtained using **TrigToExp**). Such a representation is particularly useful here because it is easy to continue the logarithm and the square root to the other sheet(s). The third column gives the branch points. All finite branch points are at the origin or at distance 1 along the real or imaginary axes. The fourth column gives the location of the branch cuts as intervals. This suggests accordingly subdividing the complex z -plane into sectors such that we do not encounter a branch cut inside any given sector. The last two columns give the real and imaginary part of the jump across a branch cut.

Function	Log Representation	Branch Points	Branch Cuts	Re Jump	Im Jump
ArcSin	$-i \log(i z + \sqrt{1 - z^2})$	$\{-1, 1, \infty\}$	$\{(-\infty, -1), (1, \infty)\}$	0	$2 \log(z + \sqrt{ z ^2 - 1})$
ArcSinh	$\log(z + \sqrt{z^2 + 1})$	$\{-i, i, \infty\}$	$\{(-\infty i, -i), (i, \infty i)\}$	$2 \log(z + \sqrt{ z ^2 - 1})$	0
ArcCos	$i \log(i z + \sqrt{1 - z^2}) + \frac{\pi}{2}$	$\{-1, 1, \infty\}$	$\{(-\infty, -1), (1, \infty)\}$	0	$2 \log(z + \sqrt{ z ^2 - 1})$
ArcCosh	$\log\left(z + \sqrt{\frac{z-1}{z+1}}(z+1)\right)$	$\{-1, 1, \infty\}$	$\{(-\infty, 1)\}$	0	$\frac{2\pi}{\arccos(z)} \quad z < -1$ $-1 \leq z \leq 1$
ArcTan	$\frac{1}{2} i (\log(z+1) - \log(1-z))$	$\{-i, i, \infty\}$	$\{(-\infty i, -i), [i, \infty i)\}$	0	π
ArcTanh	$\frac{1}{2} i (\log(1-iz) - \log(iz+1))$	$\{-1, 1, \infty\}$	$\{(-\infty, -1), (1, \infty)\}$	π	0
ArcCot	$-i \log\left((1 + \frac{1}{z}) / \sqrt{1 - \frac{1}{z^2}}\right)$	$\{-i, i\}$	$\{[-i, i]\}$	π	0
ArcCoth	$-\log\left((1 + \frac{i}{z}) \sqrt{1 + \frac{1}{z^2}}\right)$	$\{-1, 1\}$	$\{[-1, 1]\}$	0	π
ArcSec	$i \log\left(\sqrt{1 - \frac{1}{z^2}} + \frac{i}{z}\right) + \frac{\pi}{2}$	$\{-1, 1\}$	$\{(-1, 1)\}$	0	$\operatorname{arcsech}(z)$
ArcSech	$\log\left(\sqrt{\frac{1-z}{z+1}}\left(1 + \frac{1}{z}\right) + \frac{1}{z}\right)$	$\{-1, 0, 1, \infty\}$	$\{(-\infty, 0), (1, \infty)\}$	0	$\frac{2 \operatorname{arcsec}(z)}{2\pi} \quad z > 1$ $-1 \leq z \leq 1$
ArcCsc	$-i \log\left(\sqrt{1 - \frac{1}{z^2}} + \frac{i}{z}\right)$	$\{-1, 1\}$	$\{(-1, 1)\}$	0	$\operatorname{arcsech}(z)$
ArcCsch	$\log\left(\sqrt{1 + \frac{1}{z^2}} + \frac{1}{z}\right)$	$\{-i, i\}$	$\{(-i, i)\}$	$\operatorname{arcsech}(z)$	0

Table 1. Properties of inverse trigonometric and hyperbolic functions.

The function **rsfPicture** generates a picture of the real part, the imaginary part, and the absolute value of the sheets of the Riemann surface of the list **functions**.

```

In[18]:= rsfPicture[functions_, z_, opts___] :=
    With[{ε = 10-6},
        Module[{sheets, r, φ},
            sheets =
                Function[reImAbs, Function[f, Apply[ParametricPlot3D[
                    Evaluate[{r Cos[φ], r Sin[φ], reImAbs[f /. z → r ei φ]}],
                    ##, PlotPoints → {8, 12}, DisplayFunction → Identity] &,
                    Flatten[Table[{r, k + ε, (k + 1) - ε},
                        {φ, (j - 1) π / 2 + ε, j π / 2 - ε}], {k, 0, 1}, {j, 4}], 1],
                        {1}]] /@ Flatten[functions]] /@ {Re, Im, Abs};
            Show[GraphicsArray[
                Graphics3D[{EdgeForm[{GrayLevel[0], Thickness[0.0001]}],
                    #[[2]], Cases[#[[1]], _Polygon, ∞}], opts,
                    BoxRatios → {1, 1, 2}] & /@ Transpose[{sheets,
                        {SurfaceColor[Hue[0.38],
                            Hue[0.98], 1.92], SurfaceColor[Hue[0.02], Hue[0.31], 1.46],
                            SurfaceColor[Hue[0.45], Hue[0.88], 2.05]}]}]]]]]
    
```

Using the logarithmic representations given, we get the following pictures of parts of the Riemann surfaces of the **Arc*** functions. We analytically solve the square roots by taking into account the negative branch and the logarithms by $\log(z) \rightarrow \log(z + k \pi i)$, $k = \{0, \pm 1\}$. Some of the sheets will coincide and so

either the real or imaginary part will give a faithful representation of the Riemann surface.

A function which would produce the first arguments of the inputs below would be as follows.

```
In[19]:= continue[
  arcTrig : (ArcSin | ArcCos | ArcSec | ArcCsc | ArcTan | ArcCot |
    ArcSinh |
    ArcCosh | ArcSech | ArcCsch | ArcTanh | ArcCoth), z_] :=
Module[{squareRootCounter = 1, logCounter = 1},
  TrigToExp[arcTrig[z]] /.  $\sqrt{a}$  ->  $(-1)^{j_{\text{squareRootCounter}}++} \sqrt{a}$  /.
  Log[a_] -> Log[a] + 2  $\pi$  i klogCounter++]
```

Here are two examples of continued functions.

```
In[20]:= continue[ArcSin, z]
```

```
Out[20]= -i (Log[i z + (-1)j1  $\sqrt{1-z^2}$ ] + 2 i  $\pi$  k1)
```

```
In[21]:= continue[ArcTanh, z]
```

```
Out[21]=  $\frac{1}{2} (-\text{Log}[1-z] - 2 i \pi k_1) + \frac{1}{2} (\text{Log}[1+z] + 2 i \pi k_2)$ 
```

The function **arcTable** will generate a table of some sheets of the Riemann surface of the function **f**.

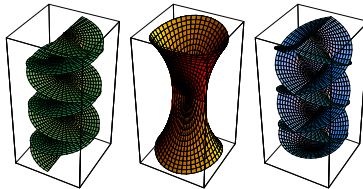
```
In[22]:= arcTable[f_, z_] :=
Module[{c = continue[f, z], js, ks},
  {js, ks} = Cases[c, #_,  $\infty$ ] & /@ {j, k};
  Flatten[
    Table[c, Evaluate[Sequence @@ ({#, 0, 1} & /@ js)],
      Evaluate[
        Sequence @@ ({#, -1, 1} & /@ ks)]]]]]
```

Here are pictures of the real and imaginary parts of all inverse trigonometric and hyperbolic functions.

```
In[23]:= rsfPictureArray[f_, z_] := rsfPicture[arcTable[f, z]]
```

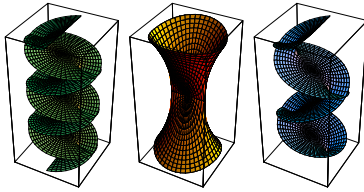
```
In[24]:= rsfPicture[arcTable[ArcSin, z], z];
```

From In[24]:=



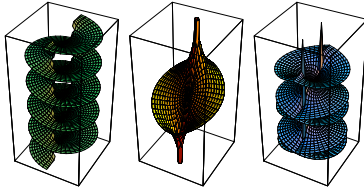
In[25]:= **rsfPicture**[arcTable[ArcCos,z],z];

From In[25]:=



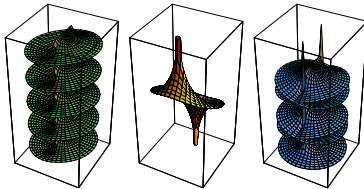
In[26]:= **rsfPicture**[arcTable[ArcTan,z],z];

From In[26]:=



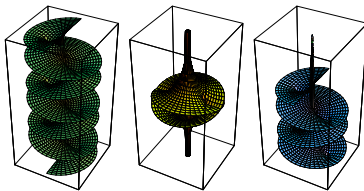
In[27]:= **rsfPicture**[arcTable[ArcCot,z],z];

From In[27]:=



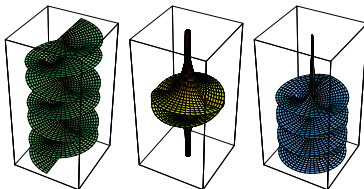
In[28]:= **rsfPicture**[arcTable[ArcSec,z],z];

From In[28]:=



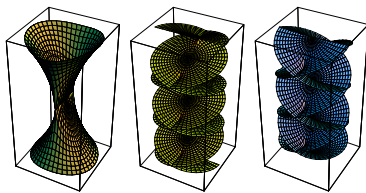
In[29]:= **rsfPicture**[arcTable[ArcCsc,z],z];

From In[29]:=



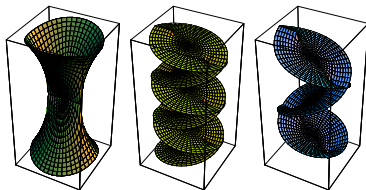
```
In[30]:= rsfPicture[arcTable[ArcSinh,z],z];
```

From In[30]:=



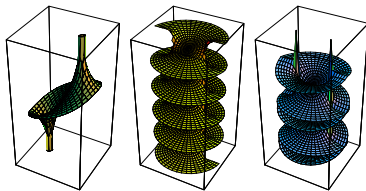
```
In[31]:= rsfPicture[arcTable[ArcCosh,z],z];
```

From In[31]:=



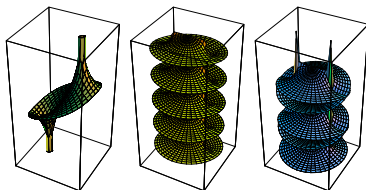
```
In[32]:= rsfPicture[arcTable[ArcTanh,z],z];
```

From In[32]:=



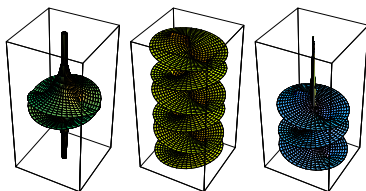
```
In[33]:= rsfPicture[arcTable[ArcCoth,z],z];
```

From In[33]:=



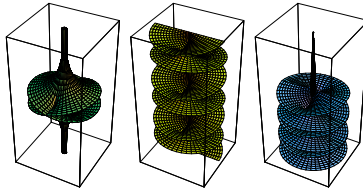
```
In[34]:= rsfPicture[arcTable[ArcSech,z],z];
```

From In[34]:=



```
In[35]:= rsfPicture[arcTable[ArcCsch,z],z];
```

From In[35]:=



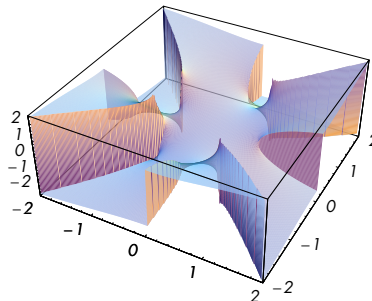
■ Obstacles in Dealing with More Complicated Functions

Let us now (try to) deal with a slightly complicated example, say $w(z) = \arcsin(\log(z^4 + z + 1))$.

```
In[36]:= w = ArcSin[Log[z^4 + z + 1]];
```

```
In[37]:= Plot3D[Evaluate[Im[w /. z -> x + i y]],
    {x, -2, 2}, {y, -2, 2}, PlotPoints -> 300, Mesh -> False];
```

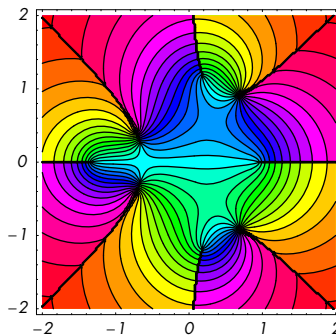
From In[37]:=



A contour plot nicely shows the location of the branch cuts.

```
In[38]:= ContourPlot[Evaluate[Im[w /. z -> x + i y]], {x, -2, 2}, {y, -2, 2},
    PlotPoints -> 200, ColorFunction -> Hue, Contours -> 30];
```

From In[38]:=

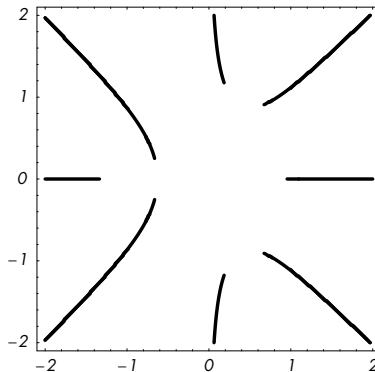


Numerically we can determine the branch cuts by looking for the lines in the complex plane where $z^4 + z + 1$, the argument of $\log$, equals a negative real

number and where $\arcsin(\log(z^4 + z + 1))$ equals a real number of absolute value greater than 1.

```
In[39]:= Module[{lines1, lineSegments1, goodLineSegments1,
  lines2, lineSegments2, goodLineSegments2},
  (* the lines of Im[z^4+z+1]==0 *)
  lines1 =
  Cases[Graphics[ContourPlot[Evaluate[Im[z^4 + z + 1 /. z -> x + i y]],
    {x, -2, 2}, {y, -2, 2}, PlotPoints -> 200, Contours -> {0},
    ContourShading -> False, DisplayFunction -> Identity]], _Line, ∞];
  lineSegments1 = Join @@ (Partition[#1[[1]], 2, 1] &) /@ lines1;
  (* the lines of Im[z^4+z+1]==0 and Re[z^4+z+1]>0 *)
  goodLineSegments1 = Select[DeleteCases[lineSegments1,
    _?(Function[{x, y}, Re[(x + i y)^4 + (x + i y) + 1] > 0] @@ #1 &),
    {2}], Length[#1] === 2 &];
  (* the lines of Im[Log[z^4+z+1]]==0 *)
  lines2 =
  Cases[Graphics[ContourPlot[Evaluate[Im[Log[z^4 + z + 1] /. z -> x + i y]],
    {x, -2, 2}, {y, -2, 2}, PlotPoints -> 200, Contours -> {0},
    ContourShading -> False, DisplayFunction -> Identity]], _Line, ∞];
  lineSegments2 = Join @@ (Partition[#1[[1]], 2, 1] &) /@ lines2;
  (* the lines of Im[Log[z^4+z+1]]==
    0 and -1<Re[Log[z^4+z+1]]<1 *)
  goodLineSegments2 = Select[DeleteCases[lineSegments2, _?
    (Function[{x, y}, -1 < Re[Log[(x + i y)^4 + (x + i y) + 1]] < 1] @@ #1 &),
    {2}], Length[#1] === 2 &];
  Show[Graphics[{GrayLevel[0], Thickness[0.01],
    Line /@ goodLineSegments1, Line /@ goodLineSegments2}],
    AspectRatio -> Automatic, Frame -> True];
```

From In[39]:=

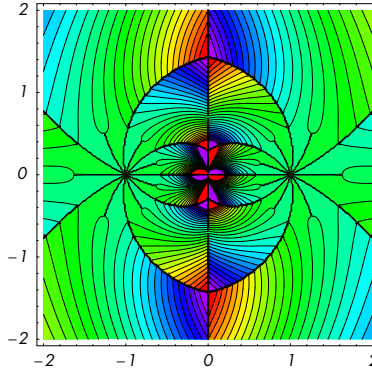


Looking at this picture it becomes obvious that even for this relatively simple function it would be a difficult undertaking to mesh the z -plane to exclude the branch cuts. For more complicated functions the problem would be, of course, still more difficult. That is why we will not mesh the plane to exclude the branch cuts, but just avoid the branch points.

Many other “simple”-looking functions will show a similar behavior—the branch cuts will have a quite complicated shape. Here is one more example that requires quite a bit of calculation to describe analytically the nice symmetric shape of the branch cuts.

```
In[40]:= ContourPlot[Im[ $\sqrt[3]{\text{Log}[(x + i y)^2] - 1}$ ], {x, -2, 2}, {y, -2, 2},
    PlotPoints → 250, ColorFunction → (Hue[0.78 #] &),
    Contours → 30, ContourStyle → {Thickness[0.0001]}];
```

From In[40]:=



For algebraic functions, using elimination techniques, it is possible in principle—although computationally quite expensive—to produce an implicit equation in $\text{Re}(z)$, $\text{Im}(z)$, and $\text{Re}(w)$ or $\text{Im}(w)$ and then use **ContourPlot3D** to display the function. However, for transcendental functions this method is rarely applicable because the elimination of $\text{Re}(w)$ or $\text{Im}(w)$ is algorithmically not feasible, except in very simple examples. Here we use $z = \tan(w)$ to get an implicit equation for $\text{Re}(z)$, $\text{Im}(z)$, and $\text{Re}(w)$.

```
In[41]:= zr + i zi == Tan[wr + i wi] // ComplexExpand
```

```
Out[41]:= i zi + zr ==  $\frac{\text{Sin}[2 wr]}{\text{Cos}[2 wr] + \text{Cosh}[2 wi]} + \frac{i \text{Sinh}[2 wi]}{\text{Cos}[2 wr] + \text{Cosh}[2 wi]}$ 
```

To eliminate $\text{Im}(w)$, we rewrite the terms containing $\text{Im}(w)$ in a rational form.

```
In[42]:= eqs = Eliminate[
    {zr ==  $\frac{\text{Sin}[2 wr]}{\text{Cos}[2 wr] + \text{Cosh}[2 wi]}$ ,  $\frac{i \text{Sinh}[2 wi]}{\text{Cos}[2 wr] + \text{Cosh}[2 wi]} == i zi$ } /.
    {Cosh[2 wi] →  $\frac{1}{2} \left( e + \frac{1}{e} \right)$ , Sinh[2 wi] →  $\frac{1}{2} \left( e - \frac{1}{e} \right)$ }, e]
```

```
Out[42]:=  $2 zr \text{Cos}[2 wr] \text{Sin}[2 wr] + (-1 + zi^2) \text{Sin}[2 wr]^2 == zr^2 (-1 + \text{Cos}[2 wr]^2)$ 
```

```
In[43]:= Simplify[Subtract@@eqs]
```

```
Out[43]:=  $\text{Sin}[2 wr] (2 zr \text{Cos}[2 wr] + (-1 + zi^2 + zr^2) \text{Sin}[2 wr])$ 
```

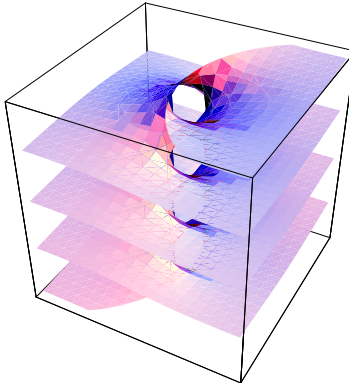
Now we have arrived at an implicit equation that can be visualized using **ContourPlot3D**.

```
In[44]:= Needs["Graphics`ContourPlot3D`"];

pic = ContourPlot3D[(2 zr Cos[2 wr] + (-1 + zi2 + zr2) Sin[2 wr]),
  {zr, -3, 3}, {zi, -3, 3}, {wr, -3, 3}, PlotPoints → 20,
  Axes → True, MaxRecursion → 0, DisplayFunction → Identity];

In[46]:= Show[Graphics3D[{EdgeForm[], Thickness[0.0001],
  SurfaceColor[Hue[0.82], Hue[0.22], 1.37],
  Cases[pic, _Polygon, ∞]}]]];
```

From In[46]:=



■ References

- [1] P. Henrici. *Applied and Computational Complex Analysis*, Vol. 1, New York: John Wiley & Sons, 1974.

Summary of Key Improvements in Mathematica 4.1

Wolfram Research Technical Staff

Mathematica 4.1 extends the technologies and advances pioneered by Wolfram Research, Inc. in *Mathematica* 4. Enhanced solvers, functions and an improved pattern matcher and compiler allow scientists and researchers to solve more complex problems more quickly. MathML and HTML integration helps academics and technical professionals share their work over the internet, while *J/Link* allows the design of innovative applications, combining the strengths of *Mathematica* and Java.

■ Major Improvements

- Enhanced *Mathematica* pattern matcher and compiler, increasing speed and minimizing memory consumption
- A new standard package for integration over inequality defined regions and piecewise functions
- Enhanced symbolic differential equation solvers
- Dramatic speed improvements in statistics functions
- Conversion between linear systems of equations and matrices
- Java integration with *J/Link* 1.1
- Improved MathML integration
- Support for saving in IBM techexplorer format
- Support for real-time manipulation of 3D graphics is now also available on Linux and Unix platforms
- Sound support under X
- Routines for plotting of graphics defined by inequalities
- Supports LinuxPPC and AlphaLinux
- New and faster import and export filters for Excel files, tabular data, compressed BMP, DXF, and STL

- Discrete trigonometric package
- Enhanced look and feel for the X front end

■ Enhanced Symbolic Differential Equation Solvers

The symbolic differential equation solvers in *Mathematica* 4.1 are completely redesigned and now cover almost all classical methods. These include a new implementation of Kovacic's algorithm for solving second-order linear homogeneous differential equations with rational function coefficients; powerful new methods and symmetry techniques for solving first-order nonlinear differential equations; improved recognition of differential equations for special functions; and powerful integrating factor methods for second-order nonlinear ordinary differential equations. Also, the methods for finding solutions of partial differential equations are upgraded significantly.

The improvements in **DSolve** are profound and cover nearly every class of differential equations handled by *Mathematica*. In addition to the new and revised methods, the new code has an enhanced parser, is much more modular, and is easier to maintain. The following examples cover just a fraction of the improvements, but they should give you an overview of the breadth of enhancements in this release.

□ Special Classes of Equations

These include all the well-known equations for special functions—for example, Airy and Bessel—as well as named classes of equations such as Riccati, Abel, and Euler. **DSolve** in *Mathematica* 4.1 can more readily identify and solve differential equations of this type.

Riccati Equation

$$\text{DSolve}[y'(x) == c x^m + b y(x)^2, y(x), x]$$

$$\{ \{ y(x) \rightarrow$$

$$\left(\sqrt{b} \sqrt{c} x^{\frac{m+2}{2}} \left(-2 J_{\frac{1}{m+2}-1} \left(\frac{2 \sqrt{b} \sqrt{c} x^{\frac{m+2}{2}}}{m+2} \right) + J_{\frac{m+1}{m+2}} \left(\frac{2 \sqrt{b} \sqrt{c} x^{\frac{m+2}{2}+1}}{m+2} \right) c_1 - \right. \right. \\ \left. \left. J_{-\frac{m+3}{m+2}} \left(\frac{2 \sqrt{b} \sqrt{c} x^{\frac{m+2}{2}}}{m+2} \right) c_1 \right) - J_{-\frac{1}{m+2}} \left(\frac{2 \sqrt{b} \sqrt{c} x^{\frac{m+2}{2}}}{m+2} \right) c_1 \right) / \\ \left(2 b x \left(J_{\frac{1}{m+2}} \left(\frac{2 \sqrt{b} \sqrt{c} x^{\frac{m+2}{2}}}{m+2} \right) + J_{-\frac{1}{m+2}} \left(\frac{2 \sqrt{b} \sqrt{c} x^{\frac{m+2}{2}}}{m+2} \right) c_1 \right) \right) \} \}$$

Hermite Polynomials

$$\text{DSolve}\left[n y(x) - \frac{2(b+ax)y'(x)}{a} + \frac{y''(x)}{a^2} == 0, y(x), x\right]$$

$$\{ \{ y(x) \rightarrow c_1 H_{\frac{n}{2}}(b+ax) + c_2 {}_1F_1\left(-\frac{n}{4}; \frac{1}{2}; (b+ax)^2\right) \} \}$$

Legendre Polynomials

```
DSolve[
  y'' (x) (1 - x^2)^2 - 2 x y' (x) (1 - x^2) + (v (v + 1) (1 - x^2) - m^2) y (x) == 0,
  y, x]
{{y -> Function[{x}, c1 P_v^m(x) + c2 Q_v^m(x)]}}
```

Mathieu Functions

```
DSolve[y'' (x) == 1/4 c^2 (a + q cos (d + c x)) y (x), y, x]
{{y -> Function[{x},
  c1 MathieuC(-a, q/2, 1/2 (d + c x)) + c2 MathieuS(-a, q/2, 1/2 (d + c x))]]}}
```

□ Special Classes of Solutions

These solutions include general methods for linear equations such as polynomial, rational, exponential (Liouvillian), and hypergeometric linear ordinary differential equations.

Liouvillian-type Equations

```
DSolve[(1 + 2 x) y'''[x] + (-6 - 6 x + 4 x^2) y''[x] +
  (11 + 4 x - 12 x^2) y'[x] + (-6 + 8 x^2) y[x] == 0, y[x], x]
{{y[x] -> e^x C[1] + e^{2x} C[2] - 1/2 e^x sqrt(pi) C[3] Erf[x]}}
```

PFQ-type Equations

Mathematica 4.1 includes a new and enhanced module for finding PFQ-type solutions of homogeneous linear ODEs. With this new module, *Mathematica* is able to solve equations of the type $(b + c x) y(x) + a x y''(x) = 0$.

```
DSolve[(b + c x) y (x) + a x y'' (x) == 0, y (x), x]
{{y (x) -> 1/a (e^{(sqrt(c/a) - sqrt(-c/a)) x - sqrt(c/a) x} x c2
  1F1(-4 sqrt(c/a) a^3 - 2 (b - 2 a sqrt(c/a)) a^2, 1; 2; 2 sqrt(-c/a) x) +
  1/a (e^{(sqrt(c/a) - sqrt(-c/a)) x - sqrt(c/a) x} x c1 U(-4 sqrt(c/a) a^3 - 2 (b - 2 a sqrt(c/a)) a^2,
  2, 2 sqrt(-c/a) x))}}
```

This new module also fixes a number of recursion and other problems *Mathematica* had with equations such as the following.

$$\text{DSolve}\left[c y(x) \sqrt[5]{b+ax} + 5 y''(x) (b+ax) + 8 a y'(x) == 0, y, x\right]$$

$$\left\{y \rightarrow \text{Function}\left[\{x\}, \frac{6 a c_1 \cos\left(\frac{\sqrt{5} \sqrt{c} (b+ax)^{3/5}}{3a}\right)}{\sqrt{5} \sqrt{c} (b+ax)^{3/5}} + \frac{3 a c_2 \sin\left(\frac{\sqrt{5} \sqrt{c} (b+ax)^{3/5}}{3a}\right)}{\sqrt{5} \sqrt{c} (b+ax)^{3/5}}\right]\right\}$$

□ System-looking Independent Equations

Mathematica can now recognize if an apparent system of equations is in reality a set of independent equations as in the following example.

$$\text{DSolve}\{y'[x] == x y[x], u''[x] == x u'[x]\}, \{y[x], u[x]\}, x$$

$$\left\{y[x] \rightarrow e^{\frac{x^2}{2}} C[1], u[x] \rightarrow C[3] + \sqrt{\frac{\pi}{2}} C[2] \text{Erfi}\left[\frac{x}{\sqrt{2}}\right]\right\}$$

□ Homogenous Equations

$$\text{DSolve}\left[-a x - a y[x] y'[x] + y[x] \sqrt{1+y'[x]^2} == 0, y[x], x\right]$$

$$\left\{y[x] \rightarrow -\sqrt{\left(\frac{e^{2C[1]+2a^2C[1]}}{-e^{4C[1]}+a^2e^{4C[1]}} - \frac{a^2e^{2C[1]+2a^2C[1]}}{-e^{4C[1]}+a^2e^{4C[1]}} - (\sqrt{2}\sqrt{-1+a^2}e^{2C[1]}\right.}\right.$$

$$\left.\sqrt{(-e^{2(1+a^2)C[1]} - a^2e^{2(1+a^2)C[1]} - e^{2C[1]+2a^2C[1]} + a^2e^{2C[1]+2a^2C[1]})x}\right) /$$

$$(-e^{4C[1]}+a^2e^{4C[1]}) + \frac{e^{4C[1]}x^2}{-e^{4C[1]}+a^2e^{4C[1]}} - \frac{a^2e^{4C[1]}x^2}{-e^{4C[1]}+a^2e^{4C[1]}}\Bigg\},$$

$$\left\{y[x] \rightarrow \sqrt{\left(\frac{e^{2C[1]+2a^2C[1]}}{-e^{4C[1]}+a^2e^{4C[1]}} - \frac{a^2e^{2C[1]+2a^2C[1]}}{-e^{4C[1]}+a^2e^{4C[1]}} - (\sqrt{2}\sqrt{-1+a^2}e^{2C[1]}\right.}\right.$$

$$\left.\sqrt{(-e^{2(1+a^2)C[1]} - a^2e^{2(1+a^2)C[1]} - e^{2C[1]+2a^2C[1]} + a^2e^{2C[1]+2a^2C[1]})x}\right) /$$

$$(-e^{4C[1]}+a^2e^{4C[1]}) + \frac{e^{4C[1]}x^2}{-e^{4C[1]}+a^2e^{4C[1]}} - \frac{a^2e^{4C[1]}x^2}{-e^{4C[1]}+a^2e^{4C[1]}}\Bigg\},$$

$$\left\{y[x] \rightarrow -\sqrt{\left(\frac{e^{2C[1]+2a^2C[1]}}{-e^{4C[1]}+a^2e^{4C[1]}} - \frac{a^2e^{2C[1]+2a^2C[1]}}{-e^{4C[1]}+a^2e^{4C[1]}} + (\sqrt{2}\sqrt{-1+a^2}e^{2C[1]}\right.}\right.$$

$$\left.\sqrt{(-e^{2(1+a^2)C[1]} - a^2e^{2(1+a^2)C[1]} - e^{2C[1]+2a^2C[1]} + a^2e^{2C[1]+2a^2C[1]})x}\right) /$$

$$(-e^{4C[1]}+a^2e^{4C[1]}) + \frac{e^{4C[1]}x^2}{-e^{4C[1]}+a^2e^{4C[1]}} - \frac{a^2e^{4C[1]}x^2}{-e^{4C[1]}+a^2e^{4C[1]}}\Bigg\},$$

$$\left\{y[x] \rightarrow \sqrt{\left(\frac{e^{2C[1]+2a^2C[1]}}{-e^{4C[1]}+a^2e^{4C[1]}} - \frac{a^2e^{2C[1]+2a^2C[1]}}{-e^{4C[1]}+a^2e^{4C[1]}} + (\sqrt{2}\sqrt{-1+a^2}e^{2C[1]}\right.}\right.$$

$$\left.\sqrt{(-e^{2(1+a^2)C[1]} - a^2e^{2(1+a^2)C[1]} - e^{2C[1]+2a^2C[1]} + a^2e^{2C[1]+2a^2C[1]})x}\right) /$$

$$(-e^{4C[1]}+a^2e^{4C[1]}) + \frac{e^{4C[1]}x^2}{-e^{4C[1]}+a^2e^{4C[1]}} - \frac{a^2e^{4C[1]}x^2}{-e^{4C[1]}+a^2e^{4C[1]}}\Bigg\}$$

□ Reductions

These methods include reduction of order for linear equations and symmetry reduction for nonlinear equations, as well as integrating factors. In the following example, **DSolve** finds a solution to a second-order nonlinear ODE.

DSolve[$x (y[x] + 4) y'[x] - y[x]^2 - 2 y[x] - 2 x == 0, y[x], x]$

$$\left\{ \left\{ y[x] \rightarrow -4 + \frac{1}{x \left(-\frac{2}{-8x-2x^2} - \frac{1}{\sqrt{x} (4+x)^{3/2} \sqrt{-\frac{4}{4+x} + C[1]}} \right)} \right\}, \right. \\ \left. \left\{ y[x] \rightarrow -4 + \frac{1}{x \left(-\frac{2}{-8x-2x^2} + \frac{1}{\sqrt{x} (4+x)^{3/2} \sqrt{-\frac{4}{4+x} + C[1]}} \right)} \right\} \right\}$$

□ Mixed Methods

Mathematica has become much more adept at combining solvers to find solutions. The following examples show how methods for rational and exponential solutions, reduction of order, special functions, and hypergeometric solutions work together.

$$\text{DSolve} \left[\frac{(x^6 - 4x^5 + 9x^4 - 10x^3 + 4x^2 - 24x + 6) y(x)}{(x-1)^4 x^2} + \right. \\ \left. \frac{(x^6 - 2x^5 - 3x^4 + 10x^3 - 10x^2 + 30x - 8) y'(x)}{(x-1)^3 x^2} + \right. \\ \left. \frac{(-2x^3 + 5x^2 - 16x + 4) y''(x)}{(x-1)^2 x^2} + \frac{(4-x^2) y^{(3)}(x)}{(x-1)x} == 0, y, x \right] \\ \left\{ \left\{ y \rightarrow \text{Function} \left[\{x\}, \right. \right. \right. \\ \left. \frac{(-1+x) C[1]}{2(-4+x^2)} + \frac{(-1+x) \text{AiryAi}[x] C[2]}{2(-4+x^2)} + \frac{(-1+x) \text{AiryBi}[x] C[3]}{2(-4+x^2)} \right. \left. \left. \right] \right\} \right\} \\ \text{DSolve} [\\ -2(4x^3 - x^2 - 6x) y(x) + (12x^2 - x - 6) y'(x) - 6x y''(x) + y^{(3)}(x) == 0, \\ y(x), x] \\ \left\{ \left\{ y(x) \rightarrow e^{x^2} c_1 + \right. \right. \\ e^{x^2} c_2 \left(\frac{x \Gamma(\frac{1}{3}) {}_1F_2(\frac{1}{3}; \frac{2}{3}, \frac{4}{3}; \frac{x^3}{9})}{3 \cdot 3^{2/3} \Gamma(\frac{2}{3}) \Gamma(\frac{4}{3})} - \frac{x^2 \Gamma(\frac{2}{3}) {}_1F_2(\frac{2}{3}; \frac{4}{3}, \frac{5}{3}; \frac{x^3}{9})}{9 \sqrt[3]{3} \Gamma(\frac{4}{3}) \Gamma(\frac{5}{3})} \right) + \\ \left. e^{x^2} c_3 \left(\frac{\Gamma(\frac{2}{3}) {}_1F_2(\frac{2}{3}; \frac{4}{3}, \frac{5}{3}; \frac{x^3}{9}) x^2}{3 \cdot 3^{5/6} \Gamma(\frac{4}{3}) \Gamma(\frac{5}{3})} + \frac{\Gamma(\frac{1}{3}) {}_1F_2(\frac{1}{3}; \frac{2}{3}, \frac{4}{3}; \frac{x^3}{9}) x}{3 \sqrt[3]{3} \Gamma(\frac{2}{3}) \Gamma(\frac{4}{3})} \right) \right\} \right\}$$

□ Boundary Value Problems

DSolve can now calculate explicit solutions to boundary value problems if it can find a general solution to the differential equation.

```
DSolve[{y'[x] == (x - 1)x-1 (Log[x - 1] + 1), y[1] == 1}, y, x]
```

```
{y -> Function[{x}, (-1 + x)-1+x]}}
```

```
DSolve[{z''[x] == z[x], z[0] == 1, z[∞] == 0}, z[x], x]
```

```
DSolve::bvlm :
```

```
For some branches of the general solution, unable to compute the  
limit at the given points. Some of the solutions may be lost.
```

```
{{z[x] -> e-x}}
```

□ Partial Differential Equations

Solving partial differential equations is much harder and less understood than solving ordinary differential equations. In the general case it is even impossible to prove or disprove that partial differential equations have solutions. *Mathematica* 4.1 gives solutions for a number of partial differential equations that *Mathematica* 4.0 either could not handle or could not solve completely.

```
DSolve[b + a02 z(0,2)[x, y] + a11 z(1,1)[x, y] == 0, z[x, y], {x, y}]
```

```
{z[x, y] -> - $\frac{b y^2}{2 a02}$  + C[1] [a02 x - a11 y] + C[2] [x]}
```

■ Pattern Matcher and Compiler Improvements

Mathematica 4.0 introduced packed array technology, a much faster and more efficient way to operate on large data sets and matrices. *Mathematica* 4.1 adds new optimizations in the compiler and pattern matcher. This speeds up repetitive operations and calculations involving large arrays even further.

Most of the pattern matching and compiler improvements are fully transparent to the user, who will notice simply an overall performance improvement. To demonstrate the maximal effect, here is a somewhat artificial example:

First, a 10,000 element list is created, and then the first element gets extracted 10,000 times.

```
m = Range[10000];
```

```
time41 = Timing[Do[m /. {x_, __} -> x, {10000}]] [[1]]
```

```
0.51 Second
```

To simulate the performance in Version 4.0, unpack the 10,000 element packed array.

```
m1 = Developer`FromPackedArray[Range[10000]];
```

```
time40 = Timing[Do[m1 /. {x_, __} -> x, {10000}]] [[1]]
```

```
9.36 Second
```

Parsing through the packed array is about 18 times faster.

```
time40 / time41
```

```
18.3529
```

Mathematica 4.1 also improves the performance for many other pattern-matching operations where, in many cases the pattern matcher is hundreds or thousands of times faster—leading to faster program execution and performance gains across the board.

■ Integration Over Inequality Defined Regions and Integration of Piecewise Functions

The new standard package **Calculus`Integration`** extends **Integrate** and **NIntegrate** to handle multiple integration over regions implicitly defined by logical combinations of piecewise algebraic inequalities. It also extends **Integrate** to handle piecewise functions.

□ Examples

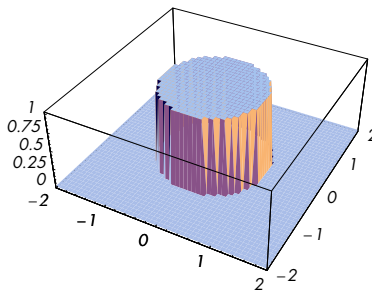
Integration Over Regions Defined by Inequalities

Load the package.

```
<< Calculus`Integration`
```

Plot the region over which to integrate.

```
Plot3D[Boole[x2 + y2 ≤ 1], {x, -2, 2},  
{y, -2, 2}, PlotPoints → 50, Mesh → False];
```



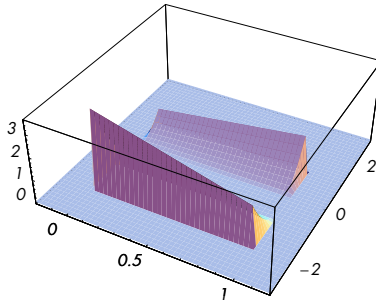
Integrate computes the area of the unit disk.

```
Integrate[Boole[x2 + y2 ≤ 1], {x, -2, 2}, {y, -2, 2}]
```

```
π
```

The areas do not have to be continuous but can instead be described by almost arbitrary complex Boolean operations.

```
Plot3D[(x y + y^2) Boole[0 ≤ x ≤ 1 ∧ (0 ≤ y ≤ 1 ∨ -2 ≤ y ≤ -1)],
  {x, -0.2, 1.2}, {y, -3, 3}, PlotPoints → 50,
  Mesh → False, PlotRange → All];
```



```
Integrate[(x y + y^2) Boole[0 ≤ x ≤ 1 ∧ (0 ≤ y ≤ 1 ∨ -2 ≤ y ≤ -1)],
  {x, -∞, ∞}, {y, -∞, ∞}]
```

$$\frac{13}{6}$$

Integrating Piecewise Functions

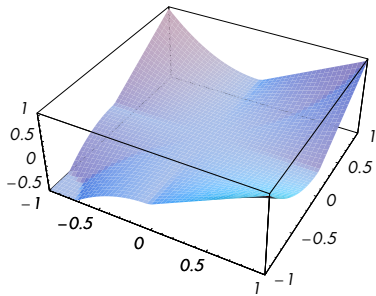
This integrates a **Max** function over a region.

```
Integrate[Max[x y^2, x^2 y] Boole[x^2 + y^2 ≤ 1], {x}, {y}]
```

$$\frac{1}{15} (4 - \sqrt{2})$$

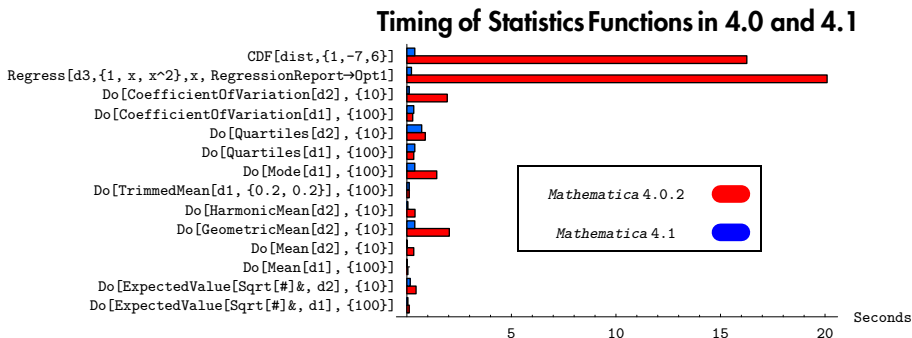
Piecewise functions can often have an irregular shape. For instance, this is the **Max** function you just saw.

```
Plot3D[Max[x y^2, x^2 y], {x, -1, 1},
  {y, -1, 1}, PlotPoints → 50, Mesh → False];
```



■ Optimized Statistical Algorithms

For large data sets, new optimizations in *Mathematica*'s statistical algorithms result in speed gains of 400 percent and more over *Mathematica* 4.0. These improvements span a large variety of operations from relatively simple ones like harmonic means to linear and nonlinear regression. For example, linear regression in *Mathematica* 4.1 is almost one hundred times faster than the same operation in *Mathematica* 4.0. Overall, a test suite of 15 commonly used statistical functions shows 4.1 being seven times faster than 4.0 on average.



■ Linear Expression to Matrix

Mathematica 4.1 introduces two new functions to convert between linear systems of equations and matrices. This functionality is handy for *Mathematica* users trying to solve large systems of equations or differential equations or wanting to construct matrices for a number of purposes, such as generating inputs for other numerical linear algebra code.

Example

The following code creates a small linear system.

```
A = DiagonalMatrix[Range[5]];
A[[-1]] = Reverse[Range[5]];
b = Range[6, 10];
vars = Table[x[i], {i, 5}];

MatrixForm[A.vars] - MatrixForm[b]
```

$$- \begin{pmatrix} 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{pmatrix} + \begin{pmatrix} x[1] \\ 2x[2] \\ 3x[3] \\ 4x[4] \\ 5x[1] + 4x[2] + 3x[3] + 2x[4] + x[5] \end{pmatrix}$$

The following command creates a sparse representation of the system $A.vars - b$.

```

sps = Developer`LinearExpressionToSparseMatrix[A.vars - b]

{{ {1, 1} → 1, {2, 2} → 2, {3, 3} → 3, {4, 4} → 4, {5, 1} → 5,
  {5, 2} → 4, {5, 3} → 3, {5, 4} → 2, {5, 5} → 1}, {6, 7, 8, 9, 10}}

```

The solution to $A.vars - b = 0$ can be found by `Developer`SparseLinear`Solve`.

```

Developer`SparseLinearSolve @@ sps

{6., 3.5, 2.66667, 2.25, -46.5}

```

This is equal to the solution given by `LinearSolve`.

```

LinearSolve[A, b] // N

{6., 3.5, 2.66667, 2.25, -46.5}

```

Why Are Sparse Linear Solvers Important?

Sparse linear solvers become important when users are trying to solve linear systems described by highly structured matrices that are common in many applications—such as finite differences used in finite element analysis. Sparse solvers introduced in *Mathematica* 4.0 brought great speed improvements for solving such systems. The following example shows some of the speedups possible using sparse instead of dense solvers.

First create a large sparse linear system. In this case, there are 2,000 equations in 2,000 variables.

```

RandomSparseMatrix[n_] := Join[Table[{j + 1, j} → Random[], {j, n - 1}],
  Table[{j, j} → 2 + Random[], {j, n}],
  Table[{j, j + 1} → Random[], {j, n - 1}]];

A = RandomSparseMatrix[2000];
b = Table[Random[], {2000}];

```

Solving this system using *Mathematica*'s `SparseLinearSolve` takes about $\frac{4}{100}$ seconds and takes up less than 2.3 megabytes of memory.

```

slsol = Timing[Developer`SparseLinearSolve[A, b];][[1]]

0.04 Second

slsmem = MaxMemoryUsed[]

2381968

```

The following small program converts the sparse matrix to dense form.

```

In[1]:= MakeSparse[m_] :=
Module[{xd, yd, mat},
  xd = Max[Part[m, All, 1, 1]];
  yd = Max[Part[m, All, 1, 2]];
  mat = Table[0, {xd}, {yd}];
  Map[(Part[mat, #[[1, 1]], #[[1, 2]]] = #[[2]]) &, m];
  mat // MatrixForm
];

```

```
c = MakeSparse[A];
```

The dimensions of the dense matrix are 2,000 by 2,000 elements.

```
Dimensions[c[[1]]]
```

```
{2000, 2000}
```

```
sol = Timing[LinearSolve[c[[1]], b];][[1]]
```

```
5.04 Second
```

SparseLinearSolve is about 126 times faster than **LinearSolve** for this example.

```
sol/slsol
```

```
126.
```

```
solmem = MaxMemoryUsed[]
```

```
82001048
```

And the dense operation takes up 34 times more memory than the sparse one.

```
solmem/slsmem // N
```

```
34.4258
```

■ *J/Link* and Web Integration

Java and the Internet are having an enormous impact on the way corporations, researchers, and developers think about software, business, and communications. The complete and transparent integration of *Mathematica* and Java via *J/Link* makes all the functionality of Java available to *Mathematica*, while also opening up *Mathematica* to Java programs.

Mathematica 4.1 expands MathML import/export capabilities. Users can directly convert typeset mathematical expressions to MathML and copy and paste them between MathML and *Mathematica* syntax. Those using MathML-compatible web browsers or plug-ins can now copy MathML code from their browsers, evaluate it in *Mathematica*, and paste the result back into HTML or XML pages.

□ *J/Link*

Mathematica 4.1 ships with the newest release of *J/Link*, a toolkit that integrates *Mathematica* and Java. It lets users call Java from *Mathematica* in a completely transparent way, and it allows them to use and control the *Mathematica* kernel from any Java program. For *Mathematica* users, *J/Link* makes the whole universe of existing and future Java classes an automatic extension to the *Mathematica* environment. For example, one can use standard Java classes to provide new interfaces to *Mathematica* or create GUI elements using AWT or Swing classes. For Java programmers, *J/Link* turns *Mathematica* into a scripting shell that lets

you experiment with, build, and test Java classes a line at a time. It also makes Java a powerful language for writing programs that use the computational services of *Mathematica*. You can find out more about *J/Link* at www.wolfram.com/solutions/mathlink/jlink/.

Examples of possible uses of *J/Link* are available at www.wolfram.com/solutions/mathlink/jlink/examples.html.

□ Improved MathML Integration

As one of the leaders in the development of the W3C MathML standard for displaying and reusing mathematics on the web, Wolfram Research provides a full implementation of MathML and full interpretability between *Mathematica* and MathML. This will enable *Mathematica* users to copy, paste, and evaluate formulas directly from web browsers, greatly facilitating cooperation and collaboration in both academic and corporate research.

MathML is intended to facilitate the use and reuse of mathematical and scientific content on the Web and in other applications, such as computer algebra systems, print typesetting, and voice synthesis. MathML can be used to encode the presentation of mathematical notation, for high-quality visual display, and mathematical content, for applications where the semantics plays more of a key role, as in scientific software or voice synthesis.

While *Mathematica* 4.0 had the ability to export notebooks and notebook expressions to HTML with additional MathML tags, these capabilities have been expanded to include the abilities to convert typeset mathematical expressions directly into MathML and to copy and paste between MathML and *Mathematica* syntax. This enables web designers and users of MathML-compatible web browsers or plug-ins to copy MathML code out of their web browser, evaluate it in *Mathematica*, and paste the result back into their XML or HTML pages.

Examples

Mathematica functions can be converted into MathML by wrapping the command **MathMLForm** around the function or by using Save As Special ▸ HTML +MathML, or by using Copy As ▸ MathML.

```
MathMLForm[Integrate[Sin[x/2], x]]
```

```
<math>
<mrow>
<mrow>
<mo>-</mo>
<mn>2</mn>
</mrow>
<mo>&it;</mo>
<mstyle>
<mspace width='0.3em' height='0.3ex' />
</mstyle>
<mo>&it;</mo>
<mrow>
<mi>cos</mi>
<mo>&af;</mo>
```

```

<mrow>
  <mo>(</mo>
  <mfrac>
    <mi>x</mi>
    <mn>2</mn>
  </mfrac>
  <mo>)</mo>
</mrow>
</mrow>
</mrow>
</math>

```

Select the output above, choose Copy As ▸ Plain Text, and paste the MathML back into *Mathematica*. You see that the MathML is rendered as valid *Mathematica* input in traditional form.

$$-2 \cos\left(\frac{x}{2}\right)$$

You can now change the typeset expression back to input form by selecting it and using Cell ▸ Convert To ▸ InputForm. You see it change to

```
-2*Cos[x/2]
```

The XML/MathML functions in *Mathematica* 4.1 allow users of MathML-compatible web browsers and the growing number of companies that use XML as their main file format to integrate *Mathematica* with their portals or XML-compliant software. With this, *Mathematica* takes another step toward becoming the world's primary technical communication software.

Wolfram Research has already built MathML-based production environments for the U.S. Patent Office and is working on numerous other MathML/XML projects, including a strategic partnership with IBM to integrate the strengths of *Mathematica* and techexplorer.

□ Support for IBM techexplorer

In a cooperative effort between IBM and Wolfram Research, *Mathematica* 4.1 now supports export to a T_EX-based format that works with IBM techexplorer. IBM techexplorer is a plug-in for Navigator and Internet Explorer, as well as an ActiveX control for applications such as Microsoft PowerPoint and Word. techexplorer enables the display of T_EX, L^AT_EX and MathML documents and the publishing of interactive scientific material on the Web. Creating web content is as easy as saving a *Mathematica* notebook in techexplorer format and viewing it in a web browser with the techexplorer plug-in. Users can also use the techexplorer *Mathematica* link to manipulate and evaluate typeset formulas in web documents directly in their web browser.

Together, techexplorer and *Mathematica* provide an easy and full-featured solution for publishers of scientific, engineering, and financial textbooks to convert their T_EX and L^AT_EX files into dynamic and interactive web content. They now have the ability to generate additional revenue streams by providing value-added services over pay-per-use or subscription-based sites using web *Mathematica* technology.

■ Improved Import and Export

In addition to the already strong import and export filters that make *Mathematica* one of the most interoperable software packages, *Mathematica* 4.1 now supports Excel spreadsheet format (.csv), compressed Windows device-independent bitmap (.bmp), AutoCAD drawing interchange format (.dxf), and STL stereolithography format (.stl), bringing the number of supported file formats to over 40.

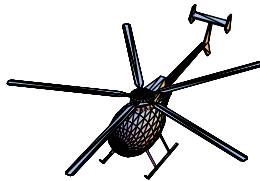
For even more advanced interoperability with other software, Wolfram Research and third party developers offer links to Microsoft Excel, LabView, Java and relational databases, as well as MathCode C++, a C++ code generator.

□ Example: AutoCAD Drawing Interchange Format (DXF)

The DXF format is intended to be used as a way to interchange three-dimensional models between CAD programs.

Prior to *Mathematica* 4.1, *Mathematica* had some degree of export capabilities through **Utilities`DXF`**. Support has been extended for exporting and added support for importing DXF files.

```
Show[dxfgraph = Import["hughes_500.dxf"],
      Boxed → False, PlotRange → All];
```



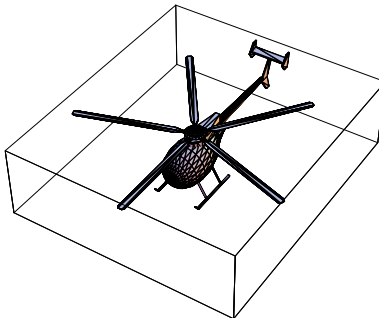
With **RealTime3D**, the imported graphic can be manipulated in real time.

The following line loads the **RealTime3D** package.

```
<< RealTime3D`
```

Turn the graphic below by dragging, zoom by **CTRL**-dragging.

```
Show[dxfgraph];
```



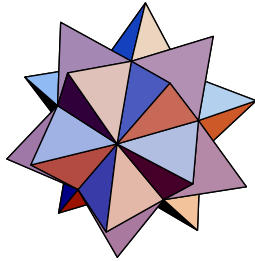
The following command resets the graphics to the *Mathematica* default.

```
<< Default3D'
```

Mathematica-generated graphics can now be exported as DXF.

```
<< Graphics`Polyhedra'
```

```
icos = Show[Stellate[Polyhedron[Icosahedron]], Boxed → False];
```



```
Export["stellated_icosahedron.dxf", icos]
```

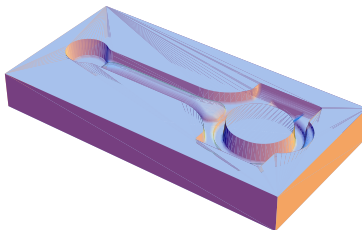
```
stellated_icosahedron.dxf
```

□ Example: Stereolithographic (STL)

The stereolithographic format (STL) is used to store three-dimensional descriptions of models. It is also a standard format for rapid prototyping.

This example demonstrates the import of a Helisys Conrod. This file was obtained from an STL repository.

```
Show[
  Import["conrod.stl"] /. ply_Polygon → {EdgeForm[], ply},
  Boxed → False,
  ViewPoint → {-2.210, -1.602, 2.000}
];
```



Again, for export demonstrations, you may want to use a familiar *Mathematica* image, then export it as STL, and display it in an STL viewer.

```
Export["stellated_icosahedron.stl", icos]
```

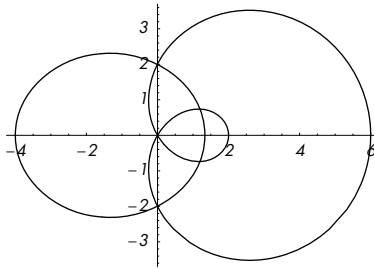
```
stellated_icosahedron.stl
```

□ Example: Windows Device-Independent Bitmap (BMP)

The specification for Windows device-independent bitmaps (BMP) allows for run-length encoded compression for non-true color bitmaps. This can result in a substantial decrease in the amount of disk space needed to store a bitmap.

Mathematica 4.1 supports run-length encoding in both import and export operations. Here is an example that demonstrates a decrease in the file size.

```
<< Graphics`Graphics`
polgrph = PolarPlot[{4/(2 + Cos[t]), 4 Cos[t] - 2},
  {t, 0, 2 Pi}];
```



Since this is a black-and-white image, it makes a good candidate for RLE compression. Here the graphic is exported with and without compression, respectively.

```
Export[
  "polar_graph_norle.bmp",
  polgrph,
  "BMP",
  ConversionOptions -> {"RLECompression" -> False}
]

polar_graph_norle.bmp

Export[
  "polar_graph_rle.bmp",
  polgrph,
  "BMP",
  ConversionOptions -> {"RLECompression" -> True}
]

polar_graph_rle.bmp
```

Comparing the two file sizes we see that compression reduces the size of the graphic by a factor of 12.

```
Apply[Divide,
  Map[FileByteCount,
    {"polar_graph_norle.bmp", "polar_graph_rle.bmp"}
  ]] // N

12.2658
```

□ Example: Comma-Separated Value (CSV)

Most spreadsheets support exporting of data to a comma delimited format (CSV) so that the data can be read into other applications. *Mathematica* 4.1 includes support for importing and exporting CSV data. Under prior versions, users had to write their own code to read and store this data.

Here is an example of a typical CSV format file.

```
!! "WWACCS.CSV"
```

```
Additional Charges - International 1998 Rate,  
Bill Duty/Tax & Shipping Chgs to Shipper,15.00  
Certificate of Origin,10.00  
Currency Conversion Rate,0.75% (of amount converted)  
Delivered Duty Paid, V.A.T. Unpaid,$15.00  
Dimensional Weight Factor - International,166  
Express Plus Surcharge,$40.00  
Extended Area Surcharge,Greater of $20 or $0.20 per pound  
Freight Collect Refusal - International,$10.00  
Shipper's Export Declaration,$10.00  
Ship Notification Request,$0.75 each
```

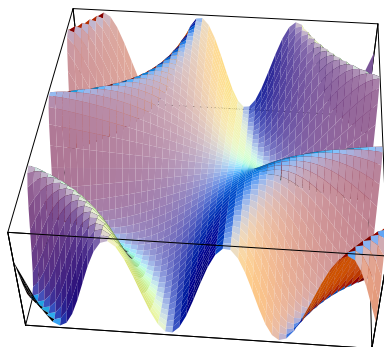
■ Graphics and Sound

□ RealTime3D Support for X

Mathematica 4.1 introduced the **RealTime3D** package for generating and manipulating three-dimensional graphics interactively using OpenGL. In *Mathematica* 4.1 this package has been ported to all supported Unix platforms using the OpenGL or Mesa graphics API.

```
<< RealTime3D`
```

```
Plot3D[Sin[x y], {x, - $\pi$ ,  $\pi$ }, {y, - $\pi$ ,  $\pi$ }, PlotPoints  $\rightarrow$  50, Mesh  $\rightarrow$  False];
```



```
<< Default3D`
```

□ Sound Support Under X

Mathematica 4.1 adds sound support on most Linux and Unix platforms, closing one of the last gaps toward full platform independence. Now almost all *Mathematica* users can hear sound signals before, during, and after they analyze them.

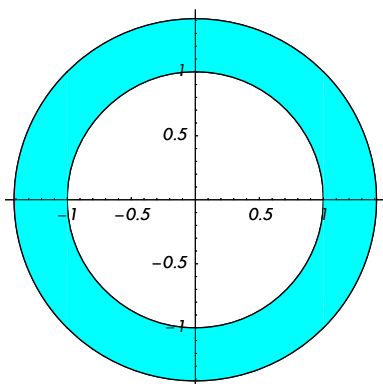
□ Inequality Graphics Package

Graphics`InequalityGraphics` offers new functions to plot two- and three-dimensional graphics that are defined by inequalities.

Examples

```
<< Graphics`InequalityGraphics`
```

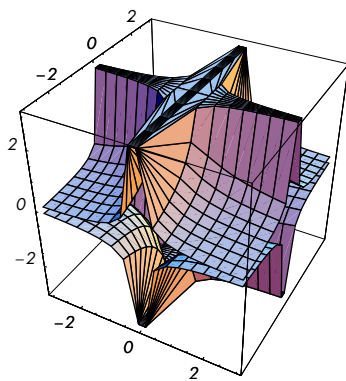
```
InequalityPlot[ $1 \leq x^2 + y^2 \leq 2$ , {x, - $\infty$ ,  $\infty$ }, {y, - $\infty$ ,  $\infty$ }];
```



```
<< Default3D`
```

```
absgraph =
```

```
InequalityPlot3D[Abs[x y z] < 1, {x, -3, 3}, {y, -3, 3}, {z, -3, 3}];
```



■ License Management and Platform Support

Now Supports LinuxPPC and AlphaLinux

Mathematica 4.1 brings the world's leading technical computation and communication software to these exciting and growing platforms. AlphaLinux is currently the fastest platform running *Mathematica* 4.1. With the Parallel Computing Toolkit and *Mathematica* 4.1 combined, the Alpha-based clusters, popular in academic settings, can easily solve large-scale problems.

Network Mathematica

All versions of 4.1 fully support *Network Mathematica* and *MathLM*, the *Mathematica* license manager. *Network Mathematica* offers the easiest, most flexible, and most cost-effective way to buy, deploy, and administer *Mathematica* and other Wolfram Research products throughout organizations. System administrators can easily deploy and administer *Mathematica* licenses in a central location, greatly simplifying system-administration tasks. Using *MathLM*, *Network Mathematica* also allows concurrent use, making *Mathematica* available to many more users at significant savings over the cost of single-user copies for occasional users.

■ Major Enhancements

□ Notebook and User Interface

Enhanced error and help message system

Enhanced traditional typesetting formats

New function:

MathMLForm

□ System and Program Interface

Compiler enhancements with higher optimization levels

Enhanced pattern-matching performance

Increased support for very long-running or very large computations

Additional import and export formats including: Excel spreadsheet format (CSV) and compressed Windows device-independent bitmap (BMP)

New functions:

Ordering

Nand

Nor

Xor

□ Numerical Computation

Optimized algorithms for statistical computation

Numericalization of **Root** objects

ContinuedFraction improvements

□ Algebraic Computation

Greatly enhanced symbolic differential equation solving

Support for multiple integration over inequality defined regions

Implementation of **GCD** by Chinese remainders

Expand improvements

Evaluation of **Roots** for polynomials with algebraic number coefficients

Simplify enhancements

Enhanced **Solve** for large linear systems

InequalitySolve improvements

□ **Graphics and Sound**

Optimized import of tabular data

Support for import and export of DXF and STL

InequalityGraphics for plotting of logical combinations of inequalities in 2D and 3D

RealTime3D now also supported under X

Sound support under X

□ **Lists and Matrices**

New functions:

LinearExpressionToMatrix

LinearExpressionToSparseMatrix

□ **Miscellaneous**

New standard package:

FourierTrig

MathLink revision 8

64-bit timestamp

J/Link

Secure switch

■ **Appendix A: Comments on Additional Enhancements**

□ **Variable Analysis for Solve**

Solve now performs variable analysis before trying to solve systems involving large numbers of variables.

The new code also has a special case to handle diagonal systems since they come up often in internal uses—for example, from **NDSolve**.

The following line of code generates 10,000 unique variable names.

```
allvars = Table[Unique[x], {10000}];
```

This code snippet creates a system of n equations. To make it easier to see, start out with $n = 2$.

```

n = 2;
vars = Take[allvars, n];
eqns = Map[
  Function[# Random[Integer, {1, n}] == Random[Integer, {-n, n}]], vars]

{2 x$15 == 1, 2 x$16 == -2}

```

For a more realistic example, *Mathematica* 4.1 solves a system of 2,500 equations, which is equivalent to a 2,500 by 2,500 diagonal matrix in 0.17 seconds, about 130 times faster than *Mathematica* 4.0.

```

n = 2500;
vars = Take[allvars, n];
eqns = Map[Function[
  # Random[Integer, {1, n}] == Random[Integer, {-n, n}]], vars];

Timing[Solve[eqns, vars];]

{0.171 Second, Null}

```

Solve also shows improvements for dense systems; in the following case, Version 4.1 is about 15 times faster than 4.0.

```

n = 250;
vars = Take[allvars, n]; b = Table[Random[], {n}];
A = Table[Random[], {n}, {n}];
eqns = Thread[A.vars == b];

Timing[Solve[eqns, vars];]

{1.572 Second, Null}

```

□ Ordering

Ordering[list] gives the position at which each element of list appears in **Sort**[list].

Here is a list.

```
dd = {5, 3, 9, 6, 7, 4};
```

This is the permutation that will put **dd** into sorted order.

```
Ordering[dd]

{2, 6, 1, 4, 5, 3}
```

□ Nand

The character $\bar{\wedge}$ can now be entered as **nand** or **\[Nand]**. **Nand**[$e_1, e_2, \dots$] is equivalent to **Not**[**And**[$e_1, e_2, \dots$]].

```
LogicalExpand[a b]

! a || ! b

```

□ Nor

The character $\bar{\vee}$ can now be entered as `nor` or `\[Nor]`. `Nor[e1, e2, ...]` is equivalent to `Not[Or[e1, e2, ...]]`.

```
LogicalExpand[a b]
```

```
! a && ! b
```

□ Numericalization of Root Objects

Numericalization of `Root` objects to large precisions has been speeded up significantly.

The following operation, which took 5.6 seconds using Version 4.0, now takes 0.26 seconds.

```
N[Root[#^20 - 5 # - 11 &, 17], 1000]; // Timing
```

```
{0.26 Second, Null}
```

□ LinearAlgebra`FourierTrig`

The discrete sine and cosine transforms are real variants of the discrete Fourier transform, which are widely applicable from signal processing to the numerical solution of partial differential equations.

FourierCos[{a₁, a₂, ..., a_{n+1}}] discrete cosine transform

FourierSin[{a₁, a₂, ..., a_{n+1}}] discrete sine transform

Discrete Transforms

There are several different conventions used to define discrete transforms. In *Mathematica*, the discrete cosine transform of b_s of a list a_r of length $n + 1$ is taken to be

$$\sqrt{\frac{2}{n}} \left(\frac{a_1}{2} + \frac{1}{2} (-1)^{s-1} a_{n+1} + \sum_{j=2}^n \frac{\cos((j-1)(s-1)\pi) a_j}{n} \right)$$

and the discrete sine transform of a list a_r of length $n - 1$ is taken to be

$$\sqrt{\frac{2}{n}} \sum_{j=1}^{n-1} \sin\left(\frac{j s \pi}{n}\right) a_j$$

This loads the package.

```
<< LinearAlgebra`FourierTrig`
```

Here is some sample data.

```
Table[1 - 2 Abs[x - .5], {x, .1, .9, .1}]
```

```
{0.2, 0.4, 0.6, 0.8, 1., 0.8, 0.6, 0.4, 0.2}
```

Here is the discrete sine transform of the data.

```
Chop[FourierSin[%]]
```

```
{1.82747, 0, -0.216981, 0, 0.0894427, 0, -0.0563317, 0, 0.0458432}
```

If this were the initial profile of, for example, a plucked string attached at both ends, this shows that only the odd modes (harmonics) are excited.

Mathematica has normalized the discrete sine and cosine transforms so that they are each their own inverses.

```
FourierSin[%]
```

```
{0.2, 0.4, 0.6, 0.8, 1., 0.8, 0.6, 0.4, 0.2}
```

□ Simplify Enhancements

A number of improvements have been made to **Simplify**, mainly concerning assumptions, logical operations, and trigonometric functions.

```
Simplify[ $e^{\frac{2i(1-n^2+4n)2\pi}{n}}$ , n ∈ Integers]
```

```
 $e^{\frac{4i\pi}{n}}$ 
```

```
Simplify[Cos[z]2 + Sin[z]2 == 1 && Cos[2 z] + 2 Sin[z]2 == 1]
```

```
True
```

```
expr =  $\frac{\left(\frac{w}{(1+w)^2}\right)^r (1+2r)!}{(1+w)^2 r!^2}$ ;
```

```
Simplify[expr, {0 < w < ∞, r ∈ Integers, r > 0}]
```

```
 $\frac{w^r (1+w)^{-2(1+r)} (1+2r)!}{r!^2}$ 
```

```
Simplify[ArcTan[-2 - x, 0], x < 0 && x > -1]
```

```
π
```

```
Simplify[Implies[a, a]]
```

```
True
```

```
Simplify[Csc[(t - n) π] Sin[(t + n) π], n ∈ Integers]
```

```
1
```

```
Simplify[ $e^{\frac{in\pi}{2} + x}$ , n ∈ Integers]
```

```
 $i^n e^x$ 
```

Mathematica 4.1 began shipping November 27, 2000. For more information or to download this notebook, go to

www.wolfram.com/products/mathematica/newin41/.

BioEqCalc

*A Package for Performing Equilibrium Calculations
on Biochemical Reactions*

David L. Akers and Robert N. Goldberg

Biotechnology Division

National Institute of Standards and Technology

Gaithersburg, Maryland 20899, USA

Email addresses: dla@cs.brown.edu; robert.goldberg@nist.gov.

A *Mathematica* package *BioEqCalc.m* has been developed for treating complex equilibria in aqueous solutions. The package is geared towards the treatment of biochemical systems and as such yields information on the molalities and mole fractions of the species in the solution, the activity coefficients of the species and the activity of the solvent (H_2O), values of apparent equilibrium constants K' , standard transformed Gibbs free energies $\Delta_r G'^o$ and standard transformed enthalpies of reaction $\Delta_r H'^o$, calorimetric enthalpies of reaction $\Delta_r H(cal)$, and changes in the binding $\Delta_r N(X)$ of ligands such as $X = H^+$ and Mg^{+2} . Constraints such as constant pH, pX, and ionic strength I can be introduced into the calculations. The effects of ionic strength are accounted for in the computation. The chemical equilibrium problem is defined by input of the data in a convenient chemical format. Two- and three-dimensional plots of the apparent equilibrium constant as a function of temperature, pX, and ionic strength can be produced.

■ Introduction

□ Thermodynamic Background

A fundamental problem in biochemical thermodynamics pertains to the treatment of complex biochemical equilibria in aqueous solution. The problem is interesting because it involves the coupling of several different types of equilibria with ligands such as H^+ and Mg^{+2} , as well as the effects of temperature and ionic strength. The effects caused by changes in these variables can dramatically affect the position of equilibrium. The problem is important because the reactions that are dealt with form the basis of metabolism and energy utilization in living systems [1].

Traditional methods [2–4] for performing these calculations involve first the derivation of analytic expressions for the fractions of the various species in solution. Since these fractions must be introduced into the computer codes, some programming effort is required for each new problem. However, a more general

approach has been implemented by Alberty [5] and by Alberty and Krambeck [6] in which the chemical equilibrium problem is formulated in terms of the conservation matrix for the system of reactions [7, 8]. A Newton–Raphson minimization of the Gibbs free energy is then used to solve for the molalities of the species.

The package described here builds upon these earlier efforts by including the effects of ionic strength in the computation and by allowing for the input of the data in a convenient chemical format. Thus, the transpose of the stoichiometric number matrix ν^T is constructed directly from the chemical equations. The package allows one to calculate the molalities and mole fractions of the species in the solution, the activity coefficients of the species and the activity of the solvent (H_2O), values of apparent equilibrium constants K' , standard transformed Gibbs free energies $\Delta_r G'^{\circ}$ and standard transformed enthalpies of reaction $\Delta_r H'^{\circ}$, calorimetric enthalpies of reaction $\Delta_r H(\text{cal})$, and changes in the binding $\Delta_r N(X)$ of ligands such as $X = \text{H}^+$ and Mg^{+2} . The package allows for the calculation of these thermodynamic quantities as functions of pX (e.g., pH and pMg), ionic strength, and temperature. It is also capable of treating systems involving several overall biochemical reactions at a specified temperature, ionic strength, and pX. Thus, the package allows for the convenient calculation of the quantities that are of primary interest to biochemical thermodynamics. Although designed for biochemical applications, this package can also be used for the study of other complex systems at equilibrium.

The reader wishing to gain additional background on the thermodynamics of biochemical reactions is referred to [9] and the references cited therein. The terminology, symbols, and units used for the physical quantities in the present article adhere generally to IUPAC recommendations [9, 10] (see glossary).

Mathematica [11] has several features that are useful for the implementation of a computer program suitable for the above purposes. Firstly, the capabilities to conveniently perform symbolic manipulations and matrix calculations are particularly important. Also, the Newton–Raphson minimization used by Krambeck and Alberty [6] has been implemented in a very brief but elegant *Mathematica* module. This module has also proven to be extremely powerful and rugged in its ability to provide reliable values of the molalities of the species in solution at equilibrium. The capability of *Mathematica* to carry out equilibrium calculations with arbitrary precision will also almost certainly prove valuable in dealing with some classes of problems. Finally, the capability to conveniently integrate graphics with calculations in a notebook format is a significant convenience to the user.

□ Treatment of Activity Coefficients

In these calculations, the activity coefficients of the species are represented by the extended Debye–Hückel equation:

$$\ln \gamma_i = - \frac{A_m z_i^2 I^{1/2}}{1 + B I^{1/2}}, \quad (1)$$

where γ_i and z_i are, respectively, the activity coefficient and the signed charge of species i . Values of the Debye–Hückel constant A_m at various temperatures are

computed by using a relatively simple analytic equation [12]. The ionic strength I is equal to $1/2 \sum_i m_i z_i^2$, where m_i is the molality of species i . The “ion-size” parameter B has been set equal to $1.6 \text{ kg}^{1/2} \text{ mol}^{-1/2}$, but this value can be changed by the user.

The calculated values of the activity coefficients are used to adjust the values of the (standard) equilibrium constants, which are input as data and pertain to $I = 0$, to the actual ionic strength of the solution. Values of the standard molar enthalpies of reaction $\Delta_r H^\circ$ at $I = 0$ are also adjusted to the appropriate ionic strength. The activity of water $a(\text{H}_2\text{O})$ is calculated by using a Gibbs–Duhem integration [4]. Following an initial guess for the ionic strength I , this quantity is calculated at each cycle of the equilibrium calculations until it has converged to within 1 part in 10^4 . The standard state used in the thermodynamic calculations is that of the hypothetical ideal solution, namely $\gamma_i \rightarrow 1$ as $\sum_i m_i \rightarrow 0$. As a consequence of the Gibbs–Duhem equation, $a(\text{H}_2\text{O})$ also $\rightarrow 1$ as $\sum_i m_i \rightarrow 0$.

□ Overview of Package

The package consists of 35 modules, each of which performs a specific function. However, not all of the modules need to be directly accessed. Documentation for use of *BioEqCalc* is primarily by way of example, solving generic problems that users can copy and adapt for their own use. In these examples, the names of functions will be denoted in the text in **bold** with **[]** following the name of the function; the names of variables or parameters used by the functions will also be in **bold**. These functions are also defined with usage statements in the package *BioEqCalc*. The package also uses several global variables which are summarized below.

In many of the calculations that follow, adenosine 5'-triphosphate (ATP) (see Figure 1) is used as an example. This is done for several reasons. First, it is an excellent example of a biochemical reaction in that it involves the binding of H^+

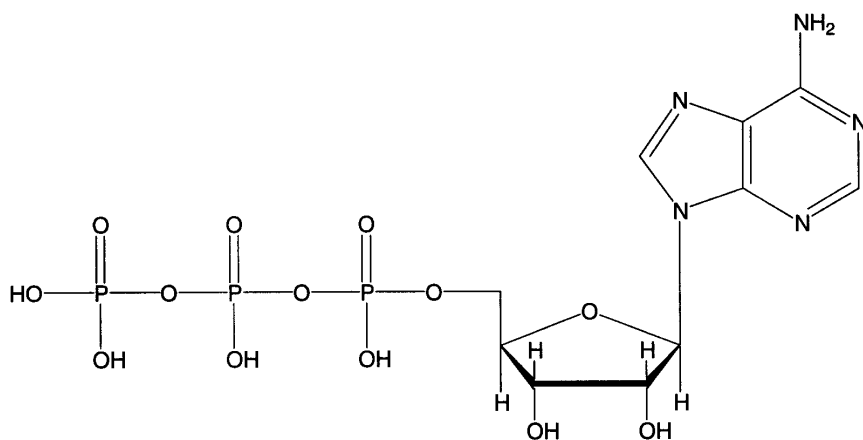


Figure 1.

and metal ions such as Mg^{+2} . Thus, the thermodynamic quantities are functions of pH, pMg, and ionic strength. Also, ATP is of fundamental biochemical importance since it can be viewed as “energy currency” in biochemical reactions and in metabolism. Finally, some carefully done calculations involving ATP are in the literature [13]. The results of these calculations can be used to check the accuracy of the results obtained herein. For this reason, the data used in the examples that follow are also taken from Alberty and Goldberg [13].

■ A Simple Chemical Equilibrium Calculation: Phosphoric Acid

The initial step is to load the *BioEqCalc* package.

```
In[1]:= << BioEqCalc`
```

□ Example 1.

The first step in a chemical equilibrium calculation is to define the chemical reactions comprising the system of reactions. This is done by defining the variable **system**.

```
In[2]:= system =
      "H3P04 = H+ + H2P04- ,
      H2P04- = H+ + HP04-2 ,
      HP04-2 = H+ + P04-3 ,
      H2O = H+ + OH-";
```

Note that double quotes are used at the beginning and end of these four reactions and that commas serve to separate the reactions from each other. It is essential to allow at least one space between the equal signs and the species as well as between the species themselves. Charges are written as a part of the chemical species and are specified as in the above example (i.e., use **-2** and not **2-**). In the absence of a charge, the charge is assumed to be zero.

Next, the function **makeInput[]** is used to extract essential information from the variable **system**.

```
In[3]:= makeInput[system];
```

The system is described by 4 reactions and the following 9 species:

No.	Species
1	H+
2	H2O
3	H2P04-
4	H3P04
5	HP04-2
6	OH-
7	P04-3
8	Na+
9	Cl-

Although not shown in this output, the transpose of the stoichiometric number matrix ν^T and the vectors containing the species and the charges of the species have been automatically generated. These can be displayed.

```
In[4]:= Print[" $\nu^T$ =", MatrixForm[nt]];
Print[species];
Print[charge];
```

$$\nu^T = \begin{pmatrix} 1 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

```
{H+, H2O, H2PO4-, H3PO4, HP04-2, OH-, PO4-3, Na+, Cl-}
```

```
{1, 0, -1, 0, -2, -1, -3, 1, -1}
```

Note that the ions Na^+ and Cl^- have been automatically added to the list of species. This has been done so that appropriate amounts of these species can be added to the solution later, if necessary, to maintain electrical neutrality. In the above example, H_2O was present as a reactant. If this was not the case, it would also have been added to the list of species in the system.

The next step is to provide the data needed to define the problem. This is done by giving the following parameters to the function `reactions[]`.

nt	The transpose of the stoichiometric number matrix. This parameter is generated by <code>makeInput[]</code> .
species	The vector containing the species. This parameter is generated by <code>makeInput[]</code> .
charge	The vector containing the charges of the species. This parameter is generated by <code>makeInput[]</code> .
k0	The equilibrium constants (dimensionless) or pKs (dimensionless) at $I=0$ for the reactions defined by the variable system .
pKorK	Since it is often convenient to use pKs rather than equilibrium constants, the variable pKorK is used to specify whether an entry in k0 is an equilibrium constant or a pK. The convention used is that a 0 as an entry in the pKorK vector implies that the corresponding entry in the k0 vector is an equilibrium constant; a 1 implies that it is a pK .

ΔH	The standard molar enthalpy changes (J mol^{-1}) at $I=0$ for the reactions defined by the variable system .
ΔCP	The standard molar heat - capacity changes ($\text{J K}^{-1} \text{mol}^{-1}$) at $I=0$ for the reactions defined by the variable system .
pseudo	The matrix used to define the pseudoisomer groups in the system. An entry of 1 specifies that a species belongs to a specified pseudoisomer group; otherwise 0 should be used.
reactants	The vector used to define the biochemical reactants in the system.
ntprime	The transpose of the apparent stoichiometric number matrix.
molalinit	The initial molalities (mol kg^{-1}) of the species.
temperature	The thermodynamic temperature (K).
Istr	The ionic strength (mol kg^{-1}) can either be fixed by specifying a value for it or one can allow the codes to solve for its value. By setting the value of Istr to -1 , the value of the ionic strength will be solved for as a part of the problem. If Istr is set to a positive value, it will be constrained to that value.
constrainedSpecies	The vector used to designate the species whose activities are held constant.
constpXa	The vector containing the values of the pX_a s corresponding to the entries in constrainedSpecies .

In the following example, **constrainedSpecies** has no entries and **constpXa** is also empty. Note that **nt**, **species**, and **charge** are generated by **makeInput[]**. The parameters **reactants**, **pseudo**, and **ntprime** have been left empty. The output should be used to check the accuracy of the data that has been entered.

```

In[7]:= k0 = {2.15, 7.21, 12.34, 13.995};
pKorK = {1, 1, 1, 1};
ΔH = {-7.9, 3.0, 15.0, 55.84} * 1000.;
ΔCP = {-128., -220., -242., -224.};
Istr = -1;
pseudo = {};
reactants = {"";
ntprime = {};
molalinit = {0., 55.508, 0.1, 0., 0.1, 0, 0., 0.3, 0};
temperature = 310.15;
constrainedSpecies = "";
constpXa = {};
reactions[nt, species, charge,
  k0, pKorK, ΔH, ΔCP, pseudo, reactants,
  ntprime, molalinit, temperature, Istr, constrainedSpecies, constpXa]

```

The system is described by 4 reactions and 9 species.

No.	Species	m(initial)/(mol kg ⁻¹)
1	H+	0.
2	H2O	55.508
3	H2PO4-	0.1
4	H3PO4	0.
5	HPO4-2	0.1
6	OH-	0
7	PO4-3	0.
8	Na+	0.3
9	Cl-	0

Initial charge balance = 0

No.	Reaction	K	pK	ΔH°/(kJ mol ⁻¹)	ΔC _p °/(J mol ⁻¹ K ⁻¹)
1	H3PO4 = H+ + H2PO4-		2.15	-7.9	-128.
2	H2PO4- = H+ + HPO4-2		7.21	3.	-220.
3	HPO4-2 = H+ + PO4-3		1.234 × 10 ¹	15.	-242.
4	H2O = H+ + OH-		1.3995 × 10 ¹	55.84	-224.

No pseudoisomer groups have been identified.

There are no constrained species.

The ionic strength has not been constrained.

T = 310.15 K

The solution of the equilibrium problem is obtained by using **ionicEquilib[]**. All of the parameters have already been defined with the exception of the following.

output A value of 1 will give a full display of output. Any other value will suppress displayed output.

The vector containing the molalities m of the species and the ionic strength I at equilibrium are calculated.

```
In[20]:= output = 1;
answer = ionicEquilib[nt, species, charge,
  k0, pKorK, molalinit, temperature, constspeciesid,
  constpKa, Istr, ΔH, ΔCP, output];
Print["m = ", answer];
Print["I = ", Istr, " mol kg-1"];
```

No.	Species	$m(\text{initial})/(\text{mol kg}^{-1})$	$m(\text{final})/(\text{mol kg}^{-1})$
1	H ⁺	0.	2.86568×10^{-7}
2	H ₂ O	5.5508×10^1	5.5508×10^1
3	H ₂ PO ₄ ⁻	$1. \times 10^{-1}$	9.99974×10^{-2}
4	H ₃ PO ₄	0.	2.1779×10^{-6}
5	HP ₄ O ₄ ⁻²	$1. \times 10^{-1}$	9.99985×10^{-2}
6	OH ⁻	0	1.7449×10^{-7}
7	PO ₄ ⁻³	0.	1.8981×10^{-6}
8	Na ⁺	$3. \times 10^{-1}$	$3. \times 10^{-1}$
9	Cl ⁻	0	0

Ionic Strength = $0.400004 \text{ mol kg}^{-1}$

$m = \{2.86568 \times 10^{-7}, 55.508, 0.0999974, 2.1779 \times 10^{-6},$
 $0.0999985, 1.7449 \times 10^{-7}, 1.8981 \times 10^{-6}, 0.3, 0\}$

$I = 0.400004 \text{ mol kg}^{-1}$

■ Speciation of Adenosine 5'-triphosphate

□ Example 2.

ATP binds both H⁺ and metal ions such as Mg⁺² and presents a more complex example than the phosphoric acid example. Again, the system must be defined and `makeInput[]` is used to obtain the transpose of the stoichiometric number matrix ν^T , the species vector, and the charge vector.

```
In[24]:= system =
  "H2ATP-2 = H+ + HATP-3,
  HATP-3 = H+ + ATP-4,
  MgHATP- = Mg+2 + HATP-3,
  MgATP-2 = Mg+2 + ATP-4,
  Mg2ATP = Mg+2 + MgATP-2,
  TrisH+ = Tris + H+";
makeInput[system];
Print["νT = ", MatrixForm[nt]];
Print[species];
Print[charge];
```

The system is described by 6 reactions and the following 13 species:

No.	Species
1	ATP-4
2	H+
3	H2ATP-2
4	HATP-3
5	Mg+2
6	Mg2ATP
7	MgATP-2
8	MgHATP-
9	Tris
10	TrisH+
11	H2O
12	Na+
13	Cl-

$$v^T = \begin{pmatrix} 0 & 1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \end{pmatrix}$$

{ATP-4, H+, H2ATP-2, HATP-3, Mg+2, Mg2ATP,
MgATP-2, MgHATP-, Tris, TrisH+, H2O, Na+, Cl-}

{-4, 1, -2, -3, 2, 0, -2, -1, 0, 1, 0, 1, -1}

Again the parameters used by **reactions[]** must be defined. However, this time **"ATP"** is a biochemical reactant (i.e., it represents the sums of the molalities of the various ATP species) and as such is the entry for the parameter **reactants**. The vector **pseudo** specifies which of the species belong to the ATP pseudoisomer group. The values of pX_a for the species H^+ and Mg^{+2} have been constrained to be 7.0 and 3.0, respectively. In this case the transpose of the apparent stoichiometric number matrix $(v')^T$ as specified in **ntprime** should be set to **{{1}}**.

```
In[29]:= k0 = {2.089*10-5, 2.512*10-8,
           2.344*10-4, 6.607*10-7, 2.042*10-3, 8.47*10-9};
pKorK = {0, 0, 0, 0, 0, 0};
ΔH = {15.0, -6.3, -16.9, -22.9, -10.8, 47.48}*1000.;
ΔCP = {-63., -126., -188., -126., -251., -50.};
pseudo = {{1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0}};
reactants = {"ATP"};
ntprime = {{1}};
molalinit = {0.1, 0, 0, 0, 0.1, 0, 0, 0, 0.1, 0, 55.508, 0.2, 0.1};
temperature = 310.15;
Istr = -1;
constrainedSpecies = "H+,Mg+2";
constpXa = {7.0, 3.0};
```

```

output = 1;
reactions[nt, species, charge,
  k0, pKorK, ΔH, ΔCP, pseudo, reactants,
  ntprime, molalinit, temperature, Istr, constrainedSpecies, constpXa];

```

The system is described by 6 reactions and 13 species.

No.	Species	m(initial)/(mol kg ⁻¹)
1	ATP-4	0.1
2	H+	0
3	H2ATP-2	0
4	HATP-3	0
5	Mg+2	0.1
6	Mg2ATP	0
7	MgATP-2	0
8	MgHATP-	0
9	Tris	0.1
10	TrisH+	0
11	H2O	55.508
12	Na+	0.2
13	Cl-	0.1

Initial charge balance = -0.1

No.	Reaction	K	pK	ΔH°/(kJ mol ⁻¹)	ΔC _p °/(J mol ⁻¹ K ⁻¹)
1	H2ATP-2 = H+ + HATP-3	2.089 × 10 ⁻⁵		15.	-63.
2	HATP-3 = ATP-4 + H+	2.512 × 10 ⁻⁸		-6.3	-126.
3	MgHATP- = HATP-3 + Mg+2	2.344 × 10 ⁻⁴		-16.9	-188.
4	MgATP-2 = ATP-4 + Mg+2	6.607 × 10 ⁻⁷		-22.9	-126.
5	Mg2ATP = Mg+2 + MgATP-2	2.042 × 10 ⁻³		-10.8	-251.
6	TrisH+ = H+ + Tris	8.47 × 10 ⁻⁹		47.48	-50.

The pseudoisomer group(s) are:

ATP: ATP-4 H2ATP-2 HATP-3 Mg2ATP MgATP-2 MgHATP-

The constraints are:

p(H_a) = 7.

p(Mg_a+2) = 3.

The ionic strength has not been constrained.

T = 310.15 K

As before, `ionicEquilib[]` is used to obtain the molalities m of the species at equilibrium.

```
In[43]:= ionicEquilib[nt, species, charge, k0, pKorK, molalinit, temperature,
constspeciesid, constpXa, Istr, ΔH, ΔCP, output];
```

No.	Species	$m(\text{initial})/(\text{mol kg}^{-1})$	$m(\text{final})/(\text{mol kg}^{-1})$
1	ATP-4	$1. \times 10^{-1}$	3.68203×10^{-3}
2	H+	0	1.46278×10^{-7}
3	H2ATP-2	0	6.49509×10^{-7}
4	HATP-3	0	1.14175×10^{-3}
5	Mg+2	$1. \times 10^{-1}$	4.57846×10^{-3}
6	Mg2ATP	0	1.0882×10^{-2}
7	MgATP-2	0	8.39857×10^{-2}
8	MgHATP-	0	3.07802×10^{-4}
9	Tris	$1. \times 10^{-1}$	1.07883×10^{-2}
10	TrisH+	0	8.92117×10^{-2}
11	H2O	5.5508×10^1	5.5508×10^1
12	Na+	$2. \times 10^{-1}$	$2. \times 10^{-1}$
13	Cl-	$1. \times 10^{-1}$	1.11935×10^{-1}

Ionic Strength = $0.412451 \text{ mol kg}^{-1}$

The mole fractions x within the ATP pseudoisomer group can be obtained by using the function `moleFractions[]`. All of the parameters have already been specified.

```
In[44]:= answer = moleFractions[nt, species,
charge, k0, pKorK, molalinit, temperature,
constspeciesid, constpXa, Istr, ΔH, ΔCP, pseudo, output];
Print["x = ", answer];
```

The mole fractions of the species within their pseudoisomer groups are:

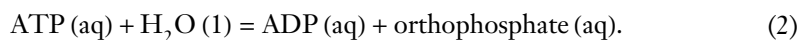
No.	Species	$x(\text{final})$
1	ATP-4	0.0368204
2	H+	0
3	H2ATP-2	6.49509×10^{-6}
4	HATP-3	0.0114175
5	Mg+2	0
6	Mg2ATP	0.10882
7	MgATP-2	0.839857
8	MgHATP-	0.00307802
9	Tris	0
10	TrisH+	0
11	H2O	0
12	Na+	0
13	Cl-	0

$x = \{0.0368204, 0, 6.49509 \times 10^{-6}, 0.0114175,$
 $0, 0.10882, 0.839857, 0.00307802, 0, 0, 0, 0, 0\}$

■ Hydrolysis of Adenosine 5'-triphosphate

□ Example 3.

We are now in a position to consider a typical example of an overall biochemical reaction, namely the hydrolysis of ATP to adenosine 5'-diphosphate (ADP) and orthophosphate:



As before, the system of reactions must be specified in terms of the pertinent species.

```
In[46]:= system =
"ATP-4 + H2O = ADP-3 + HP04-2 + H+,
H2ATP-2 = H+ + HATP-3,
HATP-3 = H+ + ATP-4,
MgHATP- = Mg+2 + HATP-3,
MgATP-2 = Mg+2 + ATP-4,
Mg2ATP = Mg+2 + MgATP-2,
H2ADP- = H+ + HADP-2,
HADP-2 = H+ + ADP-3,
MgHADP = Mg+2 + HADP-2,
MgADP- = Mg+2 + ADP-3,
H2P04- = H+ + HP04-2,
MgHP04 = Mg+2 + HP04-2,
TrisH+ = Tris + H+";
makeInput[system];
```

The system is described by 13 reactions and the following 21 species:

No.	Species
1	ADP-3
2	ATP-4
3	H+
4	H2ADP-
5	H2ATP-2
6	H2O
7	H2P04-
8	HADP-2
9	HATP-3
10	HP04-2
11	Mg+2
12	Mg2ATP
13	MgADP-
14	MgATP-2
15	MgHADP
16	MgHATP-
17	MgHP04
18	Tris
19	TrisH+
20	Na+
21	Cl-

As in the previous examples, one must provide the necessary data for the function `reactions[]`. However, instead of directly inputting the parameters `reactants`, `ntprime`, and `pseudo` one can use the function `makeInputBR[]` to obtain these parameters. Here are the parameters needed by `makeInputBR[]`.

bioreaction	Contains the list of overall biochemical reactions. These reactions are specified in a manner similar to specifying the chemical reactions (see the parameter system). Note, that biochemical reactants do not have charges and therefore none should be attached to them.
pseudoisomers	Specifies the species that comprise the various pseudoisomer groups.

The function `makeInputBR[]` returns the parameters `reactants`, `ntprime`, and `pseudo`. Therefore, it should be used before using the function `reactions`. In this example, there is only one overall biochemical reaction. However, four pseudoisomer groups are specified, one for each biochemical reactant (including H_2O).

```
In[48]:= k0 = {0.2946, 2.089*10-5, 2.512*10-8, 2.344*10-4, 6.607*10-7,
          2.042*10-3, 4.365*10-5, 6.607*10-8, 3.162*10-3, 2.239*10-5,
          6.026*10-8, 1.95*10-3, 8.47*10-9};
pKorK = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
ΔH = {-20.5, 15.0, -6.3, -16.9, -22.9, -10.8,
       17.6, -5.6, -12.5, -19.0, 3.6, -12.2, 47.48}*1000.;
ΔCP = {-283., -63., -126., -188., -126., -251., -63.,
       -126., -251., -188., -226., -3., -50.};
molalinit = {0, 0.01, 0, 0, 0, 55.508, 0, 0, 0, 0,
             0.00550, 0, 0, 0, 0, 0, 0.001, 0, 0.20706, 0.17906};
temperature = 298.15;
constrainedSpecies = "H+,Mg+2";
constpXa = {7.0, 3.0};
Istr = -1;
output = 1;

bioreaction = "ATP + H2O = ADP + phosphate";
pseudoisomers =
"ATP = ATP-4 + HATP-3 + H2ATP-2 + MgATP-2 + Mg2ATP + MgHATP-,
ADP = ADP-3 + HADP-2 + H2ADP- + MgADP- + MgHADP,
phosphate = HP04-2 + H2P04- + MgHP04,
H2O = H2O";

makeInputBR[bioreaction, pseudoisomers];
reactions[nt, species, charge,
          k0, pKorK, ΔH, ΔCP, pseudo, reactants,
          ntprime, molalinit, temperature, Istr,
          constrainedSpecies, constpXa];
```

```
Print["(ν')T=", MatrixForm[ntprime]];
Print["pseudo=", MatrixForm[pseudo]];
Print["reactants = ", reactants];
```

The system is described by 13 reactions and 21 species.

No.	Species	m(initial)/(mol kg ⁻¹)
1	ADP-3	0
2	ATP-4	0.01
3	H+	0
4	H2ADP-	0
5	H2ATP-2	0
6	H2O	55.508
7	H2PO4-	0
8	HADP-2	0
9	HATP-3	0
10	HP04-2	0
11	Mg+2	0.0055
12	Mg2ATP	0
13	MgADP-	0
14	MgATP-2	0
15	MgHADP	0
16	MgHATP-	0
17	MgHP04	0
18	Tris	0.001
19	TrisH+	0
20	Na+	0.20706
21	Cl-	0.17906

Initial charge balance = -0.001

No.	Reaction	K	pK	ΔH ^o /(kJ mol ⁻¹)	ΔC _p ^o /(J mol ⁻¹ K ⁻¹)
1	ATP-4 + H2O = ADP-3 + H+ + HP04-2	2.946 × 10 ⁻¹		-20.5	-283.
2	H2ATP-2 = H+ + HATP-3	2.089 × 10 ⁻⁵		15.	-63.
3	HATP-3 = ATP-4 + H+	2.512 × 10 ⁻⁸		-6.3	-126.
4	MgHATP- = HATP-3 + Mg+2	2.344 × 10 ⁻⁴		-16.9	-188.
5	MgATP-2 = ATP-4 + Mg+2	6.607 × 10 ⁻⁷		-22.9	-126.
6	Mg2ATP = Mg+2 + MgATP-2	2.042 × 10 ⁻³		-10.8	-251.
7	H2ADP- = H+ + HADP-2	4.365 × 10 ⁻⁵		17.6	-63.
8	HADP-2 = ADP-3 + H+	6.607 × 10 ⁻⁸		-5.6	-126.
9	MgHADP = HADP-2 + Mg+2	3.162 × 10 ⁻³		-12.5	-251.
10	MgADP- = ADP-3 + Mg+2	2.239 × 10 ⁻⁵		-19.	-188.
11	H2PO4- = H+ + HP04-2	6.026 × 10 ⁻⁸		3.6	-226.
12	MgHP04 = HP04-2 + Mg+2	1.95 × 10 ⁻³		-12.2	-3.
13	TrisH+ = H+ + Tris	8.47 × 10 ⁻⁹		47.48	-50.

The pseudoisomer group(s) are:

```
ADP:   ADP-3   H2ADP-   HADP-2   MgADP-   MgHADP
ATP:   ATP-4   H2ATP-2   HATP-3   Mg2ATP   MgATP-2   MgHATP-
H2O:   H2O
phosphate:  H2PO4-   HP04-2   MgHP04
```

The overall biochemical reaction(s) are:



The constraints are:

$$p(\text{H}^+)_{\text{a}} = 7.$$

$$p(\text{Mg}^{+2})_{\text{a}} = 3.$$

The ionic strength has not been constrained.

$$T = 298.15 \text{ K}$$

$$(\nu')^T = \begin{pmatrix} 1 & -1 & -1 & 1 \end{pmatrix}$$

$$\text{pseudo} = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

reactants = {ADP, ATP, H₂O, phosphate}

Again, **ionicEquilib[]** can be used to calculate the molalities of the species, the ionic strength, and the mole fractions of the species within the pseudoisomer groups. In this example the values of the activity coefficients γ , the activity of the water $a(\text{H}_2\text{O})$, the transpose of the stoichiometric number matrix ν^T , and the transpose of the apparent stoichiometric number matrix $(\nu')^T$ are also displayed.

```
In[65]:= ionicEquilib[nt, species, charge, k0, pKorK, molalinit, temperature,
constspeciesid, constpXa, Istr, ΔH, ΔCP, output];
moleFractions[nt, species,
charge, k0, pKorK, molalinit, temperature,
constspeciesid, constpXa, Istr, ΔH, ΔCP, pseudo, output];
Print["γ=", γList];
Print["activity of H2O = ", activityH2O];
Print["νT = ", MatrixForm[nt]];
Print["(ν')T = ", MatrixForm[ntprime]];
```

No.	Species	m(initial)/(mol kg ⁻¹)	m(final)/(mol kg ⁻¹)
1	ADP-3	0	2.08426×10^{-3}
2	ATP-4	$1. \times 10^{-2}$	1.04417×10^{-11}
3	H+	0	1.37654×10^{-7}
4	H2ADP-	0	5.6059×10^{-7}
5	H2ATP-2	0	4.29868×10^{-15}
6	H2O	5.5508×10^1	5.5508×10^1
7	H2PO4-	0	3.57616×10^{-3}
8	HADP-2	0	6.38261×10^{-4}
9	HATP-3	0	4.43835×10^{-12}
10	HPO4-2	0	5.62102×10^{-3}
11	Mg+2	5.5×10^{-3}	3.59053×10^{-3}
12	Mg2ATP	0	4.65666×10^{-11}
13	MgADP-	0	7.2207×10^{-3}
14	MgATP-2	0	3.4142×10^{-10}
15	MgHADP	0	5.62183×10^{-5}
16	MgHATP-	0	1.46875×10^{-12}
17	MgHPO4	0	8.02826×10^{-4}
18	Tris	$1. \times 10^{-3}$	5.79644×10^{-5}
19	TrisH+	0	9.42036×10^{-4}
20	Na+	2.0706×10^{-1}	2.0706×10^{-1}
21	Cl-	1.7906×10^{-1}	1.85614×10^{-1}

Ionic Strength = $0.231286 \text{ mol kg}^{-1}$

The mole fractions of the species within their pseudoisomer groups are:

No.	Species	x(final)
1	ADP-3	0.208425
2	ATP-4	0.0258239
3	H+	0
4	H2ADP-	0.000056059
5	H2ATP-2	0.0000106314
6	H2O	1.
7	H2PO4-	0.357616
8	HADP-2	0.0638261
9	HATP-3	0.0109768
10	HPO4-2	0.562102
11	Mg+2	0
12	Mg2ATP	0.115167
13	MgADP-	0.722071
14	MgATP-2	0.844389
15	MgHADP	0.00562183
16	MgHATP-	0.00363246
17	MgHPO4	0.0802826
18	Tris	0
19	TrisH+	0
20	Na+	0
21	Cl-	0

$\gamma = \{0.0563499, 0.0060168, 0.726458, 0.726458, 0.27851, 1, 0.726458, 0.27851, 0.0563499, 0.27851, 0.27851, 1, 0.726458, 0.27851, 1, 0.726458, 1, 1, 0.726458, 0.726458, 0.726458\}$

activity of H₂O = 0.99342

$$\nu^T = \begin{pmatrix} 1 & -1 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 \end{pmatrix}$$

$$(\nu')^T = (1 \quad -1 \quad -1 \quad 1)$$

The apparent equilibrium constant K' for the overall biochemical reaction can now be calculated by using the function **apparentEqX[]**. The output that is displayed also includes the total molalities of the biochemical reactants, the values of pX_a for the constrained species (H^+ and Mg^{+2} in this example), the temperature T , and the ionic strength I . All of the parameters needed by **apparentEqX[]** have already been obtained.

```
In[71]:= k' = apparentEqX[nt, species,
      charge, k0, pKorK, molalinit, temperature,
      constspeciesid, constpXa, Istr, ΔH, ΔCP, pseudo,
      reactants, ntprime, output];
Print["K' = ", k'];
```

$$\begin{pmatrix} \text{Reactant} & m/(\text{mol kg}^{-1}) \\ \text{ADP} & 0.01 \\ \text{ATP} & 4.0434 \times 10^{-10} \\ \text{H2O} & 55.508 \\ \text{phosphate} & 0.01 \end{pmatrix}$$

$$\begin{pmatrix} \text{Reaction} & K' \\ 1 & 2.4732 \times 10^5 \end{pmatrix}$$

$$pX_a(H^+) = 7.$$

$$pX_a(Mg^{+2}) = 3.$$

$$T = 298.15 \text{ K}$$

$$I = 0.231286 \text{ mol kg}^{-1}$$

$$K' = \{247317.\}$$

To calculate the value of K' at $I = 0.25 \text{ mol kg}^{-1}$, we set **Istr** to that value.

```
In[73]:= Istr = 0.25;
k' = apparentEqX[nt, species,
  charge, k0, pKorK, molalinit, temperature,
  constspeciesid, constpXa, Istr, ΔH, ΔCP, pseudo,
  reactants, ntprime, output];
Print["K' = ", k'];
```

$\left(\begin{array}{cc} \text{Reactant} & \text{m}/(\text{mol kg}^{-1}) \\ \text{ADP} & 0.01 \\ \text{ATP} & 3.9981 \times 10^{-10} \\ \text{H}_2\text{O} & 55.508 \\ \text{phosphate} & 0.01 \end{array} \right)$
$\left(\begin{array}{cc} \text{Reaction} & K' \\ 1 & 2.5012 \times 10^5 \end{array} \right)$
$\text{pX}_a(\text{H}^+) = 7.$
$\text{pX}_a(\text{Mg}^{+2}) = 3.$
$T = 298.15 \text{ K}$
$I = 0.25 \text{ mol kg}^{-1}$
$K' = \{250120.\}$

Alberty and Goldberg [13] calculated the standard transformed Gibbs free energy $\Delta_r G'^o = -32.49 \text{ kJ mol}^{-1}$ for biochemical reaction (2), the hydrolysis of ATP(aq) to {ADP(aq) + orthophosphate(aq)}, at $T = 298.15 \text{ K}$, at $\text{pH}_m = 7.0$, $\text{pMg}_m = 3.0$, and $I = 0.25 \text{ mol kg}^{-1}$. We wish to see if we obtain the same value. To do this, we first obtain the values of the activity coefficients γ of H^+ (species 3) and of Mg^{+2} (species 11) from the parameter **γList** and then calculate the values of pH_a and pMg_a that correspond to the respective values of pH_m and pMg_m . These values are subsequently used in the parameter **constpXa**.

```
In[76]:= Print["pHa = ", -Log[10, 10.^-7 * γList[[3]]]];
Print["pMga = ", -Log[10, 10.^-3 * γList[[11]]]];
```

$\text{pHa} = 7.14185$
$\text{pMga} = 3.56739$

The function **apparentEqX[]** is next used to calculate K' and then the function **gibbsEnergy[]** is used to calculate the standard transformed Gibbs free energy $\Delta_r G'^o$. Here are the parameters used by **gibbsEnergy[]**.

temperature	The thermodynamic temperature (K).
K'	The value of the apparent equilibrium constant (dimensionless). Note that since K' is a list (a list of one in this case), the appropriate element in that list has to be specified.

```
In[78]:= output = 0;
constpXa = {7.14185, 3.56739};
K' = apparentEqX[nt, species,
  charge, k0, pKorK, molalinit, temperature,
  constspeciesid, constpXa, Istr, ΔH, ΔCP,
  pseudo, reactants, ntprime, output];
Print["K' = ", K'[1]];
Print["I = ", Istr, " mol kg-1"];
Print["pMgm = ", pXm[11], " pHm = ", pXm[3]];
Print["pMga = ", pXa[11], " pHa = ", pXa[3]];
Print["ΔrG'o = ",
  gibbsEnergy[temperature, K'[1] * 0.001, " kJ mol-1"]];
```

$K' = 489166.$

$I = 0.25 \text{ mol kg}^{-1}$

$pMg_m = 3. \quad pH_m = 7.$

$pMg_a = 3.56739 \quad pH_a = 7.14185$

$\Delta_r G'^o = -32.4757 \text{ kJ mol}^{-1}$

Thus, the value calculated for $\Delta_r G'^o$ is in excellent agreement with the result ($\Delta_r G'^o = -32.49 \text{ kJ mol}^{-1}$) obtained by Alberty and Goldberg [13].

■ Graphics

□ Example 4.

Plots of the apparent equilibrium constant K' as a function of temperature T , pH, pMg, and ionic strength I can be obtained by using the function **graphic2D[]**. All of the parameters needed by **graphic2D[]** have already been defined and specified with the following exceptions.

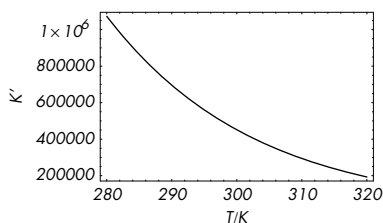
vary	Specifies what is to be varied. "T" specifies the temperature T ; "H+" specifies pH_a ; "Mg+2" specifies pMg_a ; and "I" specifies the ionic strength I .
minmax	Specifies the range for the variation of the specified variable.
npts	The number of points to be calculated, this will form the basis of the smoothed curve. If the curve is not smooth enough, the value of npts should be increased.

```

In[86]:= temperature = 298.15;
constpXa = {7.14185, 3.56739};
Istr = 0.25;
vary = "T";
minmax = {280, 320};
npts = 5;
graphic2D[nt, species, charge, k0, pKorK, molalinit, temperature,
constspeciesid, constpXa, Istr, ΔH, ΔCP, pseudo, reactants,
ntprime, vary, minmax, npts];

```

From In[86]:=

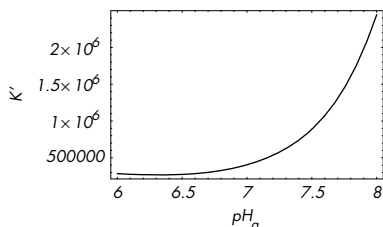


```

In[93]:= vary = "H+";
minmax = {6, 8};
npts = 10;
graphic2D[nt, species, charge, k0, pKorK, molalinit, temperature,
constspeciesid, constpXa, Istr, ΔH, ΔCP, pseudo, reactants,
ntprime, vary, minmax, npts];

```

From In[93]:=

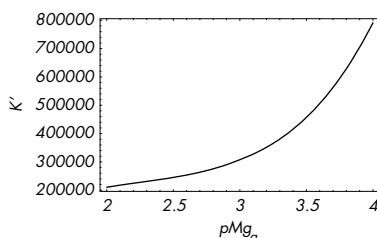


```

In[97]:= vary = "Mg+2";
minmax = {2, 4};
npts = 5;
graphic2D[nt, species, charge, k0, pKorK, molalinit, temperature,
constspeciesid, constpXa, Istr, ΔH, ΔCP, pseudo, reactants,
ntprime, vary, minmax, npts];

```

From In[97]:=

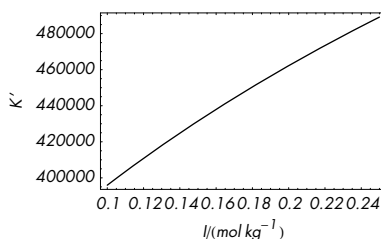


```

In[101]:= vary = "I";
minmax = {0.1, 0.25};
npts = 5;
graphic2D[nt, species, charge, k0, pKorK, molalinit, temperature,
constspeciesid, constpXa, Istr, ΔH, ΔCP, pseudo, reactants,
ntprime, vary, minmax, npts];

```

From In[101]:=



The function **graphic3D[]** can be used to produce three-dimensional plots of the apparent equilibrium constant K' as a function of any two of the following independent variables: temperature T , pH, pMg , and ionic strength I . The use of this function is essentially the same as that of **graphic2D[]** except that one must specify the following parameters.

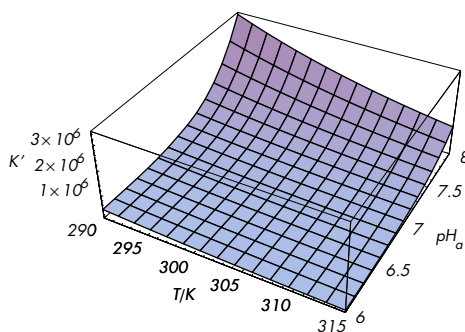
vary1	Specifies one of the independent variables. " T " specifies the temperature T ; " H+ " specifies pH_a ; " Mg+2 " specifies pMg_a ; and " I " specifies the ionic strength I .
vary2	Specifies the second of the independent variables. " T " specifies the temperature T ; " H+ " specifies pH_a ; " Mg+2 " specifies pMg_a ; and " I " specifies the ionic strength I .
minmax1	Specifies the range for the variation of the first of the specified variables.

minmax2	Specifies the range for the variation of the second of the specified variables.
npts1	The number of points to be used in calculating variations due to the first of the independent variables.
npts2	The number of points to be used in calculating variations due to the second of the independent variables.

Note that if the surfaces are not smooth enough, the value(s) of **npts1** and/or **npts2** should be increased.

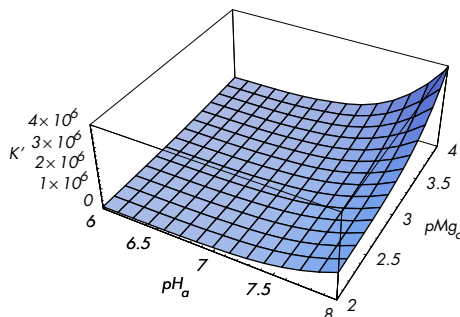
```
In[105]:= vary1 = "T"; minmax1 = {290., 315.}; npts1 = 5;
vary2 = "H+"; minmax2 = {6.0, 8.0}; npts2 = 5;
graphic3D[nt, species, charge, k0, pKorK, molalinit, temperature,
constspeciesid, constpXa, Istr, ΔH, ΔCP, pseudo, reactants,
ntprime, vary1, vary2, minmax1, minmax2, npts1, npts2];
```

From In[105]:=



```
In[112]:= vary1 = "H+"; minmax1 = {6.0, 8.0}; npts1 = 5;
vary2 = "Mg+2"; minmax2 = {2.0, 4.0}; npts2 = 5;
graphic3D[nt, species, charge, k0, pKorK, molalinit, temperature,
constspeciesid, constpXa, Istr, ΔH, ΔCP, pseudo, reactants,
ntprime, vary1, vary2, minmax1, minmax2, npts1, npts2];
```

From In[112]:=

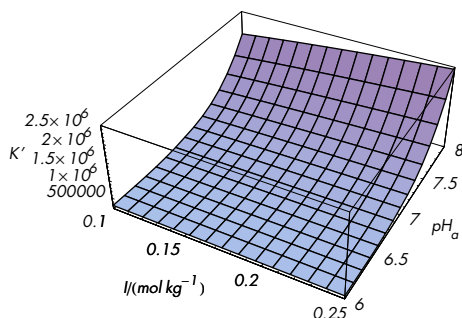


```

In[119]:= vary1 = "I"; minmax1 = {0.1, 0.25}; npts1 = 5;
          vary2 = "H+"; minmax2 = {6., 8.}; npts2 = 5;
          graphic3D[nt, species, charge, k0, pKorK, molalinit, temperature,
                    constspeciesid, constpXa, Istr, ΔH, ΔCP, pseudo, reactants,
                    ntprime, vary1, vary2, minmax1, minmax2, npts1, npts2];

```

From In[119]:=



■ Additional Capabilities of *BioEqCalc*

Note: Additional Capabilities of *BioEqCalc* are described in the online version of this article. See www.mathematica-journal.com/issue/v8i1/features/akers/.

■ Conclusion and Future Extensions of *BioEqCalc*

This article has demonstrated how one can use the package *BioEqCalc* to treat equilibrium problems in biochemical thermodynamics. This has been done primarily by use of examples involving several types of problems that can be of interest. Several extensions of this package can be envisioned. One possible extension could involve the capability to produce contour plots which would show the apparent equilibrium constant K' , changes in binding of species X $\Delta_r N(X)$ and standard molar transformed enthalpy changes $\Delta_r H'^o$ as functions of T , pX , and I . It is also possible to make use of more advanced treatments of nonideality. This could involve the Pitzer formalism [15] and/or the mean spherical approximation [16]. The current package deals only with equilibria occurring in a single condensed phase. Thus, the treatment could be extended to include several coexisting condensed phases as well as the gas phase. Additionally, the equilibrium data in the current package are input as either equilibrium constants or as pK s. Clearly, there are some classes of problems where it is more convenient to input values for the standard molar Gibbs free energies of formation $\Delta_f G^o$. Finally, there is a real need to assemble the necessary data for many key biochemical reactions and to have this accessible in a form that is particularly convenient for biochemists.

■ Global Variables

activityH2O	The activity of the water.
B	The ion - size parameter; this parameter has been set equal to $1.6 \text{ kg}^{1/2} \text{ mol}^{-1/2}$, but can be changed by the user.
charge	Vector of the charges on the species.
chlorideIndex	Index of Cl^- .
constpXa	The pX_a values of those species whose activities are held constant.
constspeciesid	The indexes of those species whose activities are held constant.
H2OIndex	Index of the species H_2O .
Istr	Ionic strength.
nt	The transpose of the stoichiometric number matrix.
ntprime	The transpose of the apparent stoichiometric number matrix.
pseudo	The matrix that defines the pseudoisomers.
dhpar	The Debye–Hückel parameters.
pXm	$\text{pX}_m = -\log_{10} \{m(X)/m^o\}$, where $m(X)$ is the molality of species X and $m^o = 1 \text{ mol kg}^{-1}$.
pXa	$\text{pX}_a = -\log_{10} \{a(X)\}$, where $a(X)$ is the activity of species X.
R	The gas constant.
reactants	List of biochemical reactants.
sodiumIndex	Index of Na^+ .
species	List of species.
γList	Vector of the activity coefficients of the species.
ideal	"Y" specifies ideal behavior; any other value uses the extended Debye–Hückel equation (1) to calculate the activity coefficients.

■ Glossary

Symbol	Name	Unit
$a(X)$	activity of species X	dimensionless
A_m	Debye–Hückel constant	dimensionless
B	parameter in the Debye–Hückel equation (see equation 1)	dimensionless
I	ionic strength	mol kg^{-1}
K	equilibrium constant	dimensionless
K'	apparent equilibrium constant	dimensionless
$m(X)$	molality of species X	mol kg^{-1}
m°	standard molality (1 mol kg^{-1})	mol kg^{-1}
P	pressure	Pa
$\text{p}X_a^\dagger$	$-\log_{10} a(X)$	dimensionless
$\text{p}X_m^\dagger$	$-\log_{10} \{m(X)/m^\circ\}$	dimensionless
R	gas constant ($8.31451 \text{ J K}^{-1} \text{ mol}^{-1}$)	$\text{J K}^{-1} \text{ mol}^{-1}$
T	thermodynamic temperature	K
x	mole fraction	dimensionless
z	signed charge	dimensionless
γ	activity coefficient	dimensionless
$\Delta_r C_p^\circ$	standard molar heat-capacity change for a reaction	$\text{J K}^{-1} \text{ mol}^{-1}$
$\Delta_r G^\circ$	standard molar Gibbs free energy change for a chemical reaction	J mol^{-1}
$\Delta_r G'^\circ$	standard molar transformed Gibbs free energy change for a biochemical reaction	J mol^{-1}
$\Delta_r H^\circ$	standard molar enthalpy energy change for a chemical reaction	J mol^{-1}
$\Delta_r H'^\circ$	standard molar transformed enthalpy change for a biochemical reaction	J mol^{-1}

Symbol	Name	Unit
$\Delta_r H$ (cal)	calorimetrically determined enthalpy of reaction	J mol^{-1}
$\Delta_r N(\text{X})$	change in binding of species X in a biochemical reaction at specified conditions	dimensionless
ν^T	the transpose of the stoichiometric number matrix	dimensionless
$(\nu')^T$	the transpose of the apparent stoichiometric number matrix	dimensionless

[†]There is a distinction made between pX based on molality $[-\log_{10} \{m(\text{X})/m^\circ\}]$, where $m^\circ = 1 \text{ mol kg}^{-1}$ and that based on activity $[-\log_{10} a(\text{X})]$.

■ Acknowledgment

We thank Drs. Robert A. Alberty (Massachusetts Institute of Technology) and Fred Krambeck (Mobil Research and Development Company) for providing two of the modules used in this package and for their helpful discussions.

■ References

- [1] H. A. Krebs, H. L. Kornberg, and K. Burton, *A Survey of the Energy Transformations in Living Matter*, Berlin: Springer-Verlag, 1957.
- [2] R. A. Alberty, "Effect of pH and metal ion concentration on the equilibrium hydrolysis of adenosine triphosphate to adenosine diphosphate," *Journal of Biological Chemistry*, **243**, 1968 pp. 1337–1343.
- [3] R. A. Alberty, "Standard Gibbs free energy, enthalpy, and entropy changes as a function of pH and pMg for several reactions involving adenosine phosphates," *Journal of Biological Chemistry*, **244**, 1969 pp. 3290–3302.
- [4] R. N. Goldberg and Y. B. Tewari, "Thermodynamics of the disproportionation of adenosine 5'-diphosphate to adenosine 5'-triphosphate and adenosine 5'-monophosphate I: Equilibrium model," *Biophysical Chemistry*, **40**, 1991 pp. 241–261.
- [5] R. A. Alberty, "Determination of the seven apparent equilibrium constants for the binding of oxygen by hemoglobin from measured fractional saturations," *Biophysical Chemistry*, **63**, 1997 pp. 119–132.
- [6] R. A. Alberty and F. J. Krambeck, "Calculation of equilibrium compositions of reaction systems including cases where the chemical potential of a species is specified," submitted to *Mathematica in Education and Research*.
- [7] M. Klein, "Practical treatment of coupled gas equilibrium," in *Physical Chemistry*, Vol. 1 (H. Eyring, W. Jost, and D. Henderson, eds.), New York: Academic, 1971.

- [8] W. R. Smith and R. W. Missen, *Chemical Reaction Equilibrium Analysis*, New York: Wiley, 1982.
- [9] R. A. Alberty, A. Cornish-Bowden, Q. H. Gibson, R. N. Goldberg, G. G. Hammes, W. Jencks, K. F. Tipton, R. Veech, H. V. Westerhoff, and E. C. Webb, "Recommendations for nomenclature and tables in biochemical thermodynamics," *Pure and Applied Chemistry*, **66**, 1994 pp. 1641–1666. Also published in *European Journal of Biochemistry*, **240**, 1996 pp. 1–14.
- [10] I. Mills, T. Cvitas, K. Homann, N. Kallay, and K. Kuchitsu, *Quantities, Units, and Symbols in Physical Chemistry*, 2nd ed., Oxford: Blackwell, 1993.
- [11] S. Wolfram, *The Mathematica Book*, 3rd ed., Champaign, IL: Wolfram Media and Cambridge, UK: Cambridge University Press, 1996.
- [12] E. C. W. Clarke and D. N. Glew, "Evaluation of Debye–Hückel limiting slopes for water between 0 and 150 °C," *Journal of the Chemical Society, Faraday Transactions I*, **76**, 1980 pp. 1911–1916.
- [13] R. A. Alberty and R. N. Goldberg, "Standard thermodynamic formation properties for the adenosine 5'-triphosphate series," *Biochemistry*, **31**, 1992 pp. 10610–10615.
- [14] R. A. Alberty and R. N. Goldberg, "Calorimetric determination of the standard transformed enthalpy of a biochemical reaction at specified pH and pMg," *Biophysical Chemistry*, **47**, 1993 pp. 213–223.
- [15] K. S. Pitzer, "Ion interaction approach: Theory and data correlation," in *Activity Coefficients in Electrolyte Solutions*, 2nd ed. (K. S. Pitzer, ed.), Boca Raton, FL: CRC, 1991.
- [16] J.-P. Simonin, O. Bernard, and L. Blum, "Real ionic solutions in the mean spherical approximation: 3. Osmotic and activity coefficients for associating electrolytes in the primitive model," *Journal of Physical Chemistry B*, **102**, 1998 pp. 4411–4417.

■ Additional Material

Package: **BioEqCalc.m**

BioEqCalc uses *Mathematica* version 4.0 and later.

About the Authors

David L. Akers has recently completed his baccalaureate degree in mathematics and computer science at Brown University. He is now employed at Silicon Graphics, Inc. His work on *BioEqCalc* was done while a summer student at the National Institute of Standards and Technology.

Robert N. Goldberg is a research chemist in the Biotechnology Division at the National Institute of Standards and Technology. His interests include chemical thermodynamics, calorimetry, equilibrium measurements, and the properties of solutions. He is the author or coauthor of approximately 110 scientific publications.

■ Copyright and Disclaimer

This is an official publication of the National Institute of Standards and Technology and is not subject to copyright. Certain commercial items are identified in this paper to specify the procedures adequately. Such identification is not intended to imply recommendation or endorsement by the National Institute of Standards and Technology, nor is it intended to imply that the items identified are necessarily the best available for the purpose.

Employing Complex Kinetic Diagrams to Understand the Belousov–Zhabotinskii Reaction

Scot T. Martin

Division of Engineering and Applied Sciences

Pierce Hall

Harvard University

Cambridge, MA 02138

In contrast to many physical phenomena such as a swinging pendulum, chemical reactions must approach equilibrium monotonically. However, far from equilibrium, concentrations of chemical species can oscillate during the time course of the reaction. This interesting branch of kinetics is exemplified by the Belousov–Zhabotinskii reaction. After an induction period, the color of the solution changes between clear and yellow with a period of two minutes for several hours. In this article, a general, convenient chemical kinetics solver is employed in conjunction with complex kinetic diagrams to understand the coupling of chemical reactions that yield chemical oscillations.

■ Introduction

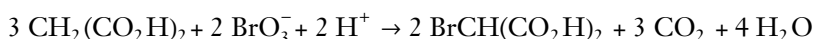
Chemical kinetics is the discipline concerning the time rate of change of the concentration of chemical species through the course of a chemical reaction. The typical approach to chemical kinetics problems is first to note the elementary steps and then to express the time rate of change of the concentration of each chemical species as a sum of its sources and sinks. The result is a system of coupled, often nonlinear, differential equations. This set of equations is subsequently

simplified by appropriate approximations, and the reduced set of equations is solved analytically or numerically, as appropriate. For example, for the reaction $A \rightarrow B$, we can write $\frac{d[A]}{dt} = -k[A]$ where k is the rate constant. The solution for $A[t]$ readily follows as $A_0 e^{-kt}$ where A_0 is the initial concentration of A .

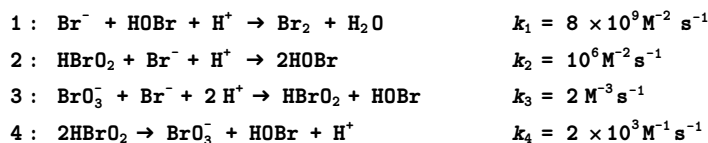
The chemical kinetics of the reaction $A \rightarrow B$ are thus readily understood, and an analytical solution is easily obtained. Many chemical reactions are much more complex, and an understanding of the coupled differential equations is more difficult. A fascinating chemical reaction is the Belousov–Zhabotinskii reaction, discovered first in 1958 and again in 1964. Prior to its discovery, esteemed chemists argued that oscillating chemical reactions could not occur. In the Belousov–Zhabotinskii reaction, when the proper concentrations of bromate, malonic acid, sulfuric acid, and cerium ions are mixed together, the color of the solution oscillates from yellow to clear with a period of about two minutes for several hours. How can this reaction be understood? Many fundamental chemical measurements have been made over the last three decades to obtain the rate constants of the elementary steps, and twenty-one important steps have been identified. In addition, the resulting system of coupled differential equations has been numerically integrated to reproduce the observed oscillations. However, obtaining an intuitive understanding of this complex system is difficult for the novice. One successful approach in the past has been through chaos theory, invoking an Oregonator model. In this paper, a general chemical kinetics solver for *Mathematica* is introduced. Then, the powerful numerical engine and graphic capabilities of *Mathematica* are employed to understand the Belousov–Zhabotinskii reaction.

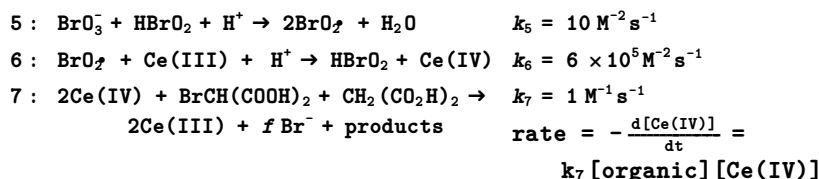
■ The Belousov–Zhabotinskii Reaction

The reactants include malonic acid, sodium bromate, and sulfuric acid, which are mixed together at initial concentrations far from equilibrium. Energy is released by the oxidation of malonic acid by bromate. The net reaction is as follows:



The direct chemical reaction between bromate and malonic acid is kinetically slow. Thus, cerium ions at much lower concentrations are added to the reaction mixture. The cerium ions catalyze the overall reaction by acting as an electron relay between malonic acid and bromate. During the course of the reaction, the color of the solution oscillates between clear and yellow with a period of about two minutes because the dominant chemical speciation of cerium oscillates between the +3 and +4 oxidation states. The oscillatory reaction can be represented in 21 steps, but the following 6 elementary steps and 1 empirical step retain the essential features of the system [4]:





To simulate this system by numerical methods, we write a rate equation for each reactant. The list of nine reactant species includes: Br^- , HOBr , H^+ , HBrO_2 , BrO_3^- , $\text{BrO}_2^\bullet$, Ce(III) , Ce(IV) , and “organic” (namely malonic acid). We then specify initial conditions and seek a numerical solution.

■ Kinetic Equations

A *Mathematica* notebook accompanying this article, *Kinetics.nb*, contains the instructions to process the kinetic input and generate the graphs. To translate the chemical equations into a system of coupled differential equations, the user interface proceeds by identifying the chemical species, by providing initial conditions and the integration time, and by specifying each kinetic step. Every place a user should change information to specify the kinetic system is shown in blue. (Editor’s note: Colors may be seen in the online version of this article. Here the blue is replaced by gray.) The detailed procedure is as follows.

□ Names of Species in Kinetic Analysis

```
species = {"Br-", "HOBr", "H+", "HBrO2",
           "BrO3-", "BrO2.", "Ce(III)", "Ce(IV)", "organic"};
```

□ Initial Concentrations (M) of Chemical Species

```
conditions = {0.000625, 10-6, 2, 10-6, 0.0625, 10-6, 10-6, 0.002, 0.275} ;
```

□ Length of Time (sec) Over Which to Obtain Kinetic Solution

```
integrationtime = 20*60;
```

□ Elementary Rate Equation for Each Species

The entry of the kinetic equations is now shown by employing reaction 2 as an example, as follows.



The following entries are made for this elementary step:

```
{2, 4, loss, {4, 1, 3}, 1, 106},
{2, 1, loss, {4, 1, 3}, 1, 106},
{2, 3, loss, {4, 1, 3}, 1, 106},
{2, 2, production, {4, 1, 3}, 2, 106}
```

The first line is interpreted as follows: “In reaction 2, species #4 (i.e., HBrO_2) is lost according to a rate law including species #4 (HBrO_2), #1 (Br^-), and #3 (H^+)

with a rate constant of 10^6 .” The corresponding differential equation is as follows:

$$2: d[\text{HBrO}_2]/dt = -10^6 [\text{HBrO}_2][\text{Br}^-][\text{H}^+]$$

This entry is one term in the complete expression for $d[\text{HBrO}_2]/dt$. Reactions 3 to 6 also contribute to $d[\text{HBrO}_2]/dt$.

The six entries in each sublist correspond to the following format:

1. reaction number
2. species number for time-derivative
3. “loss” or “production” of species
4. {first species, second species, ... and so on} of kinetic rate equation
5. rate multiplier [account for stoichiometry]
6. rate constant

Reactions that do not fit this mold (e.g., surface reactions) can be manually adjusted in the kinetic equations later.

The complete set of entries corresponding to reactions 1 to 7 of the Belousov–Zhabotinskii chemical mechanism are shown below.

```

rates = {{1, 1, "loss", {1, 2, 3}, 1, 8 × 109},
           {1, 2, "loss", {1, 2, 3}, 1, 8 × 109},
           {1, 3, "loss", {1, 2, 3}, 1, 8 × 109},
           {2, 4, "loss", {4, 1, 3}, 1, 106},
           {2, 1, "loss", {4, 1, 3}, 1, 106},
           {2, 3, "loss", {4, 1, 3}, 1, 106},
           {2, 2, "production", {4, 1, 3}, 2, 106},
           {3, 5, "loss", {5, 1, 3, 3}, 1, 2},
           {3, 1, "loss", {5, 1, 3, 3}, 1, 2},
           {3, 3, "loss", {5, 1, 3, 3}, 2, 2},
           {3, 4, "production", {5, 1, 3, 3}, 1, 2},
           {3, 2, "production", {5, 1, 3, 3}, 1, 2},
           {4, 4, "loss", {4, 4}, 1, 2 × 103},
           {4, 5, "production", {4, 4}, 1, 2 × 103},
           {4, 2, "production", {4, 4}, 1, 2 × 103},
           {4, 3, "production", {4, 4}, 1, 2 × 103},
           {5, 5, "loss", {5, 4, 3}, 1, 10},
           {5, 4, "loss", {5, 4, 3}, 1, 10},
           {5, 3, "loss", {5, 4, 3}, 1, 10},
           {5, 6, "production", {5, 4, 3}, 2, 10},
           {6, 6, "loss", {6, 7, 3}, 1, 6 × 105},
           {6, 7, "loss", {6, 7, 3}, 1, 6 × 105},
           {6, 3, "loss", {6, 7, 3}, 1, 6 × 105},
           {6, 4, "production", {6, 7, 3}, 1, 6 × 105},
           {6, 8, "production", {6, 7, 3}, 1, 6 × 105},
           {7, 8, "loss", {9, 8}, 1, 1},
           {7, 9, "loss", {9, 8}, 1, 1},
           {7, 7, "production", {9, 8}, 1, 1},
           {7, 1, "production", {9, 8}, 0.5 f, 1}};

```

□ Special Instructions

There is occasionally the need for special instructions to be appended to the elementary rate equations. For example, in the Belousov–Zhabotinskii reaction, there is the parameter f . What value should it take?

```
specialinstructions = {f → 2};
```

□ Input Verification

After providing this input to *Mathematica*, the user verifies the interpretation. The species numbers, labels, and initial concentrations are displayed, as follows:

```
Table[{i, species[[i]], conditions[[i]]}, {i, totalspecies}] //
TableForm
```

1	Br ⁻	0.000625
2	HOBr	$\frac{1}{1000000}$
3	H ⁺	2
4	HBrO ₂	$\frac{1}{1000000}$
5	BrO ₃ ⁻	0.0625
6	BrO ₂ •	$\frac{1}{1000000}$
7	Ce(III)	$\frac{1}{1000000}$
8	Ce(IV)	0.002
9	organic	0.275

The individual kinetic terms are also displayed, as shown below for sorting by species. A complete kinetic term for the time rate of change of a species is the sum of the individual terms.

Species		
1	d[Br ⁻]/dt =	0.5 f × 1. [organic] [Ce(IV)]
1	d[Br ⁻]/dt =	-1.e6 [HBrO ₂] [Br ⁻] [H ⁺]
1	d[Br ⁻]/dt =	-2. [BrO ₃ ⁻] [Br ⁻] [H ⁺] [H ⁺]
1	d[Br ⁻]/dt =	-8.e9 [Br ⁻] [HOBr] [H ⁺]
2	d[HOBr]/dt =	2000. [HBrO ₂] [HBrO ₂]
2	d[HOBr]/dt =	2. [BrO ₃ ⁻] [Br ⁻] [H ⁺] [H ⁺]
2	d[HOBr]/dt =	2 × 1.e6 [HBrO ₂] [Br ⁻] [H ⁺]
2	d[HOBr]/dt =	-8.e9 [Br ⁻] [HOBr] [H ⁺]
3	d[H ⁺]/dt =	-10. [BrO ₃ ⁻] [HBrO ₂] [H ⁺]
3	d[H ⁺]/dt =	-1.e6 [HBrO ₂] [Br ⁻] [H ⁺]
3	d[H ⁺]/dt =	2000. [HBrO ₂] [HBrO ₂]
3	d[H ⁺]/dt =	-2 × 2. [BrO ₃ ⁻] [Br ⁻] [H ⁺] [H ⁺]
3	d[H ⁺]/dt =	-600000. [BrO ₂ •] [Ce(III)] [H ⁺]
3	d[H ⁺]/dt =	-8.e9 [Br ⁻] [HOBr] [H ⁺]
4	d[HBrO ₂]/dt =	-10. [BrO ₃ ⁻] [HBrO ₂] [H ⁺]
4	d[HBrO ₂]/dt =	-1.e6 [HBrO ₂] [Br ⁻] [H ⁺]
4	d[HBrO ₂]/dt =	-2000. [HBrO ₂] [HBrO ₂]
4	d[HBrO ₂]/dt =	2. [BrO ₃ ⁻] [Br ⁻] [H ⁺] [H ⁺]
4	d[HBrO ₂]/dt =	600000. [BrO ₂ •] [Ce(III)] [H ⁺]

5	$d[\text{BrO}_3^-]/dt =$	$-10. [\text{BrO}_3^-] [\text{HBrO}_2] [\text{H}^+]$
5	$d[\text{BrO}_3^-]/dt =$	$2000. [\text{HBrO}_2] [\text{HBrO}_2]$
5	$d[\text{BrO}_3^-]/dt =$	$-2. [\text{BrO}_3^-] [\text{Br}^-] [\text{H}^+] [\text{H}^+]$
6	$d[\text{BrO}_2\cdot]/dt =$	$2 \times 10. [\text{BrO}_3^-] [\text{HBrO}_2] [\text{H}^+]$
6	$d[\text{BrO}_2\cdot]/dt =$	$-600000. [\text{BrO}_2\cdot] [\text{Ce(III)}] [\text{H}^+]$
7	$d[\text{Ce(III)}]/dt =$	$1. [\text{organic}] [\text{Ce(IV)}]$
7	$d[\text{Ce(III)}]/dt =$	$-600000. [\text{BrO}_2\cdot] [\text{Ce(III)}] [\text{H}^+]$
8	$d[\text{Ce(IV)}]/dt =$	$-1. [\text{organic}] [\text{Ce(IV)}]$
8	$d[\text{Ce(IV)}]/dt =$	$600000. [\text{BrO}_2\cdot] [\text{Ce(III)}] [\text{H}^+]$
9	$d[\text{organic}]/dt =$	$-1. [\text{organic}] [\text{Ce(IV)}]$

■ Chemical Oscillations

After the user enters the chemical system as shown above, the subsequent steps of setting up the differential equations and numerically solving the coupled equations are fully automated. After the numerical engine has finished the computations, the user is presented with graphs of the solutions, as shown in Figures 1 and 2. Following an initial induction period, the oscillatory behavior of chemical intermediates is apparent in Figure 1. In particular, $[\text{Ce(IV)}]$ oscillates from nearly zero to 2 mM, with accompanying changes from clear to yellow. Figure 2 shows that the concentrations of the reactants are much higher than those of the chemical intermediates, and the concentrations monotonically decrease during the course of the reaction. There are some periods when the overall reaction rate is apparently slow (i.e., the nearly flat regions in Figure 2) and other periods when the reaction rate appears to accelerate.

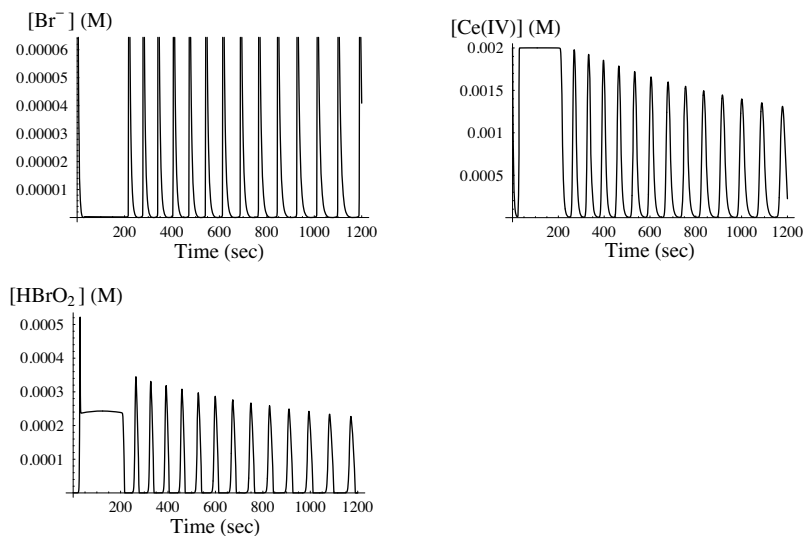


Figure 1. The concentrations of Br^- , HBrO_2 , and Ce(IV) as a function of time from 0 to 1200 sec. After an initial induction period, an oscillatory behavior with a period of about 2 minutes is observed.

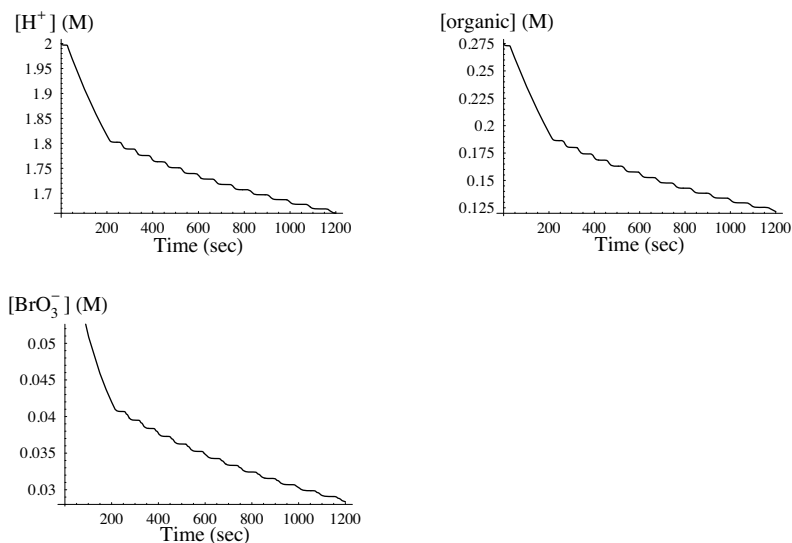


Figure 2. The concentrations of H^+ , HBrO_3^- , and organic as a function of time from 0 to 1200 sec. The concentrations of these reactant species decrease monotonically through the course of the reaction, proceeding stepwise through periods of slow and fast reaction rates.

The following two conditions must be satisfied for a reaction to oscillate.

1. The concentrations of reactants and products must be far from their equilibrium conditions.
2. There must be chemical feedback in the chemical mechanism (implying a complex kinetic mechanism).

At sufficiently long time periods (i.e., when reactant and product concentrations are sufficiently close to equilibrium), the chemical system reverts to a single mode and all chemical species approach equilibrium concentrations monotonically. Throughout the course of the reaction, the free energy of the system declines as the extent of reaction continues.

■ Complex Kinetic Diagram

A *complex kinetic diagram* is introduced in Figure 3 to understand the interactions that result in the oscillations shown in Figure 1. The complex kinetic diagram shows a snapshot at 360 sec of the rates of reactions leading to the production of chemical species. The right-hand side of the diagram is a bar chart indicating the percent change in the concentration of the nine chemical species per unit time. On the left-hand side of this diagram, the fastest rate of all seven reactions is shown to be $6.2 \times 10^{-6} \text{ M s}^{-1}$, and that value represents unity. All other rates are scaled to that value. The nine black circles shown each have a radius of 1. The red circles indicate net production of a species while the blue circles indicate a

net loss of a species. The rate of loss or production is proportional to the radius of the circle shown (i.e., as a fraction of unity). We can say, for example, that $d[\text{H}^+]/dt$ is being lost at $6.2 \times 10^{-6} \text{ M s}^{-1}$ whereas Ce(III) is being produced at approximately 75% of this rate. The dashed lines show connections between products and reactants. The product side of the line is indicated by the heavy black mark. Thus, in this scheme, $\text{BrO}_2^\bullet$ is *potentially* produced from BrO_3^- , HBrO_2 , and H^+ . In turn, $\text{BrO}_2^\bullet$ produces Ce(IV) and HBrO_2 . How much $\text{BrO}_2^\bullet$ is *actually* being produced? The length of the red line covering the heavy black line indicates the rate at which a species is actually being produced. It is apparent that very little $\text{BrO}_2^\bullet$ is being produced. In contrast, Ce(III) is rapidly being produced from organic and Ce(IV) . We can also see that HOBr is being produced rapidly from Br^- , which is likewise being produced rapidly from Ce(IV) . Very little is producing Ce(IV) , and its net loss rate (circle in blue) is seen to be substantial.

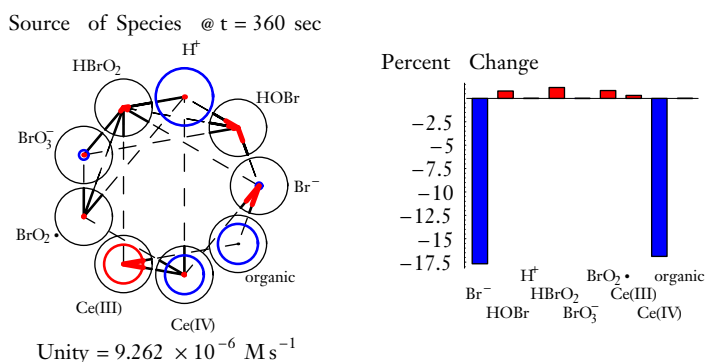


Figure 3. Complex kinetic diagram showing a snapshot of the Belousov–Zhabotinskii reaction at 360 sec. See further explanation in the text.

In Figure 4, the concentrations of Ce(IV) , Br^- , and HBrO_2 are overlaid through several chemical cycles. The drop in $[\text{HBrO}_2]$ is seen to be followed by a rapid increase in $[\text{Br}^-]$. Similarly, the rise in $[\text{Ce(IV)}]$ is coincident with the rise in the $[\text{HBrO}_2]$. A specific timepoint (e.g., 360 sec) can be selected and analyzed in greater detail, as was done in Figure 3. Timepoints of 309, 325, 332, and 342 sec are also selected and shown as complex kinetic diagrams in Figure 5. Figure 4 shows that these timepoints cover the initiation, peak, and quiescence of one chemical cycle.

The *Mathematica* notebook is highly automated. Subsequent to the numerical solutions obtained in Figures 1 and 2, the user can execute the following command to display a complex kinetics diagram at a given time point. The user enters the parameters shown in blue, that is, `drawrates[1,2,3,4,5]` where the first parameter is the timepoint and the other parameters are options not employed in this article but discussed in the notebook *Kinetics.nb*.

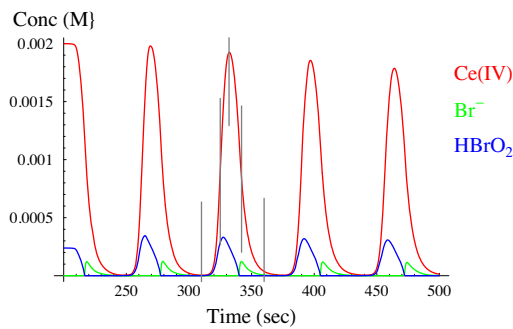


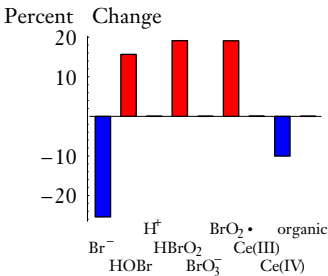
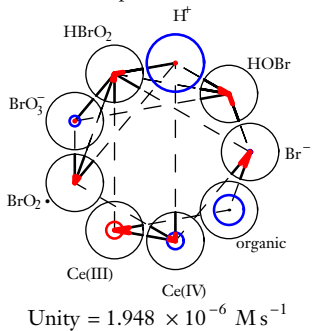
Figure 4. Concentrations of Ce(IV), Br⁻, and HBrO₂ from 200 to 500 sec. The gray bars are shown at timepoints of 309, 325, 332, 342, and 360 sec and correspond to the complex kinetics diagrams shown in Figures 3 and 5.

```
assem = Graphics[drawrates[325, 1, True, "production", True],
  TextStyle -> {FontFamily -> "Times", FontSize -> 10},
  AspectRatio -> 1, PlotRange -> {{-r, r}, {-r, r}}];

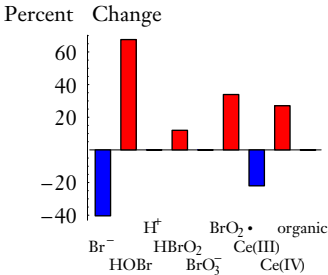
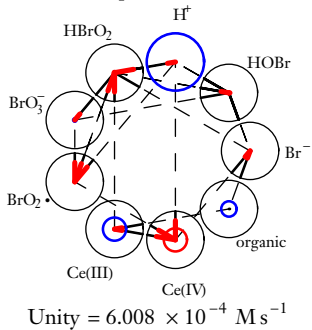
percentbar = makepercentbar[325] ;

Show[GraphicsArray[{assem, percentbar}],
  GraphicsSpacing -> 0, ImageSize -> {72 5, 72 3}]
```

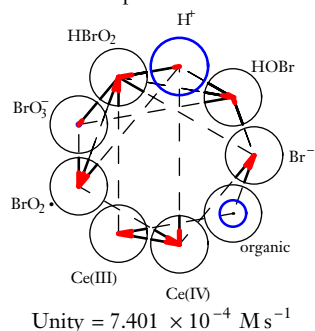
Source of Species @ t = 309 sec



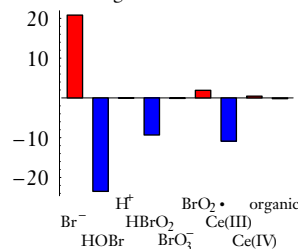
Source of Species @ t = 325 sec



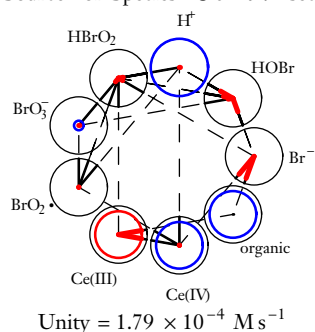
Source of Species @ t = 332 sec



Percent Change



Source of Species @ t = 342 sec



Percent Change

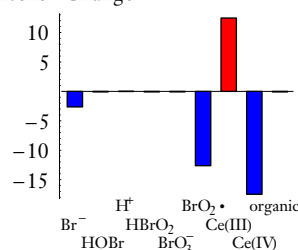
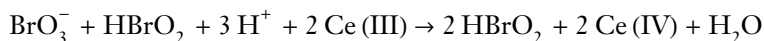


Figure 5. Complex kinetic diagrams of the Belousov–Zhabotinskii reaction at 309, 325, 332, and 342 sec. The timepoint at 360 sec is shown in Figure 3.

■ Understanding the Chemical Oscillations

The complex kinetics diagram can be employed to understand the oscillatory behavior shown in Figure 1. At the beginning of a chemical cycle, [HBrO₂] and [Br⁻] are low, as shown at 309 sec in Figure 4. Although the overall reaction is slow (1.9×10^{-6} M s⁻¹ in Figure 5), reactions 5 and 6 together combine to make an autocatalytic sequence for HBrO₂, as follows.



[HBrO₂] is thus increasing at an accelerating rate. The production of HBrO₂ from BrO₂•, H⁺, and Ce(III) (reaction 6) is shown by the red lines surrounding HBrO₂ in Figure 5. In turn, the red lines surrounding BrO₂• indicate production from BrO₃⁻, HBrO₂, and H⁺ (reaction 5). [HBrO₂] and [BrO₂•] are increasing by 20% per second, as shown in the bar chart. By 325 sec, [HBrO₂] has dramatically increased by the autocatalytic process (Figure 4). Accompanying the increase in [HBrO₂], [BrO₂•] is high enough that reaction 6 is depleting Ce(III)

at about 20% per second, as shown in the bar chart. Consequently, Ce(IV) is forming (Figure 4). However, a limit is being approached because the disproportionation reaction (4) limits $[\text{HBrO}_2]$, as indicated by the red line representing production of BrO_3^- from HBrO_2 . The continuing increase of $[\text{Ce(IV)}]$ leads to the formation of Br^- via reaction 7, as shown by the 20% per second increase in $[\text{Br}^-]$ at 332 sec. At this point, the overall reaction rate has increased to $7.4 \times 10^{-4} \text{ M s}^{-1}$. The formation of Br^- leads to the depletion of HBrO_2 via reaction 2. This relationship is apparent in Figure 4 and in the bar chart at 332 sec in Figure 5. The red lines indicate the formation of HOBr from the reactants for reaction 2 (i.e., Br^- , HBrO_2 , and H^+). The removal of HBrO_2 shifts the system away from reactions 5, 6, and 7, as shown by the absence at 342 sec of red lines corresponding to these reactions (cf. 325 sec). $[\text{Ce(IV)}]$ continues to fall (Figures 4 and 5), and the red lines show the accompanying production of Ce(III) and Br^- . Overall, however, $[\text{Br}^-]$ is decreasing via reactions 1 through 3. When Br^- is depleted to trace levels, the overall reaction rate is slow (viz. 360 sec in Figure 3). The low concentration of Br^- keeps reaction 2 slow while allowing HBrO_2 to begin forming via reaction 3. $[\text{HBrO}_2]$ and $[\text{Br}^-]$ are again low (cf. 309 sec), and the chemical cycle then repeats.

■ Conclusions

Reaction 7 is an empirical relationship including a factor, f , which relates the number of Br^- ions released by the reduction of two Ce(IV) ions and the concomitant oxidation of malonic acid and its brominated derivatives. The concentration of brominated derivatives depends on the time course of the reaction via the production of Br_2 in reaction 1. Thus, f also depends on time. To account for this complete mechanism, the seven reactions employed in this paper must be expanded to 21 elementary steps. As employed, however, f plays a pivotal role. If f is much below 2, then an insufficient amount of Br^- is produced to remove HBrO_2 completely. In this case, the steady-state $[\text{HBrO}_2]$ is high and depends upon the competition between an autocatalytic cycle (reactions 5 and 6) and a second-order disproportionation reaction (4). When f is much above 2, then $[\text{Br}^-]$ is sufficient at all times that reaction 2 is important. This sink for HBrO_2 competes with an autocatalytic cycle (reactions 5 and 6). The result is a low steady-state $[\text{HBrO}_2]$. Chemical oscillations occur when f is near 2 because the dominant chemical mode alternates.

■ Acknowledgment

We are grateful for support received from the NSF Atmospheric Chemistry Program and the American Chemical Society Petroleum Research Fund (ACS-PRF# 32580-G9).

■ References

- [1] W. Braun, J. T. Herron, and D. K. Kahaner, "Acuchem: A computer program for modeling complex chemical reaction systems," *International Journal of Chemical Kinetics*, **20**, 1988 p. 51.
- [2] R. J. Field and H.-D. Försterling, "On the oxybromine chemistry rate constants with cerium ions in the Field–Koeroes–Noyes mechanism of the Belousov–Zhabotinskii reaction: The equilibrium $\text{HBrO}_2 + \text{BrO}_3^- + \text{H}^+$," *Journal of Physical Chemistry*, **90**, 1986 p. 5400.
- [3] B. F. Gray, "Kinetics of oscillatory reactions," in *Reaction Kinetics: Specialist Periodical Reports*, (ed. P. G. Ashmore), London: The Chemical Society, 1974.
- [4] P. Gray and S. K. Scott, *Chemical Oscillations and Instabilities*, New York: Oxford, 1990.
- [5] M. Guzman, P. Ortega, and L. Vera, "A computer calculation method in chemical kinetics," *Journal of Chemical Education*, **73**, 1996 p. 921.
- [6] J. J. Tyson, *The Belousov–Zhabotinskii Reaction*, Berlin: Springer, 1976.

About the Author

Professor Martin earned a B.S. in chemistry from Georgetown University (1991) and a Ph.D. in environmental/physical chemistry from Caltech (1995). He was supported by a Department of Defense National Defense Science and Engineering Graduate Fellowship. After his Ph.D., he completed postdoctoral work at MIT in atmospheric chemistry. During that time, he was supported by a National Oceanic and Atmospheric Administration postdoctoral fellowship in global climate change. He began as Assistant Professor of Aquatic and Atmospheric Chemistry at The University of North Carolina at Chapel Hill in January 1997. In 1998, he was awarded a CAREER grant from the National Science Foundation Atmospheric Chemistry Program, and in 1999 he received a Presidential Early Career Award for Scientists and Engineers (PECASE). He began as an Associate Professor of Environmental Chemistry at Harvard University in July 2000.

■ Additional Material

Kinetics.nb

A Language to Solve Finite Element Problems

Finite Element Analysis for Dummies

Naci M. Dai

(Formerly of Carleton University)

The Object People Inc.

Ottawa, Ontario, Canada

nacidai@acm.org

■ Introduction

The finite element method (FEM) is used extensively in science and engineering. The relative ease of establishing the governing equations for analysis of continuum and structures, as well as the compact and well-defined linear algebra and matrix operations that are associated with it, makes FEM an attractive computational tool. Plenty has been said and written about FEM; for that you should see, among many others, Bathe, Irons, or Zienkiewicz [1,2,3].

Unfortunately, the same reasons that make FEM an attractive mathematical tool make it a challenging programming task. Quite often programming FEM is hard, involving a wide variety of software engineering and numerical analysis dilemmas. It is intimidating to program FEM from scratch, and often it is hard to customize commercial FEM products for new and specific problems.

I wrote a framework to provide an intuitive programming language by abstracting FEM operations to a level that is very close to the mathematical notations used in expressing the governing equations. The inner complexity of these operations are hidden, thus the framework provides a useful tool for prototyping FEM programs.

If you are able to understand FEM in terms of the following integral, which describes the physical equations in a form such as

$$\int_{\Omega} \ell(x, t) d\Omega$$

where $\ell(x, t)$ is an integrable operator in space and time, you should be able to program this equation with a simple statement such as:

```
physicalIntegrate[domain, operator]
```

A simplified list of functionality in a typical FEM program can be given as follows.

1. *Linear algebra.* The practical implementation of finite element analysis usually involves extensive use of matrix algebra. These include algebraic operations such as addition, subtraction, inner products, and more complex operations such as matrix inversion, solutions of systems of equations, matrix decomposition (e.g., QR, LU, LLT), change of basis, and transformations.
2. *Computations of the finite element integrals.* FEM transforms partial differential equations (PDEs) into integral equations. The evaluation of integrals is an important task in finite element analysis. The results of the computations of these integrals are usually referred to as the “element integrals” or the “element matrices/vectors.” The principal idea is to achieve a relationship between the physical fields such as displacements, velocities, stresses, temperatures, magnetic charges, and so on, and discrete approximations of these fields. We do this by using a finite dimensional function space defined by basis functions (also known as the trial functions, interpolation functions, or shape functions). These functions provide an effective and efficient means of evaluating the element integrals.
3. *Global operations.* Practical implications of FEM are probably most obvious at the 10,000 foot level. Simple element-level integrals are combined to describe the global behavior of an arbitrarily complex problem. Local behavior and even some singularities can be captured by special elements. The assembly of these small elements describes the global problem. The global operations include assembly, imposition of essential and natural boundary conditions, solutions of the linear/nonlinear systems of algebraic equations, time integration of ordinary differential equations (ODEs), and so on.

The framework will be demonstrated in the following sections. It is assumed that the reader has some familiarity with FEM, linear algebra, and PDEs. The initial section introduces the framework, followed by FEM element integrals. The second section describes and demonstrates the global FEM operations. Finally, the last section studies an example implementation of the Laplacian operator.

■ A Look at the Fem Package

The framework is a collection of libraries that provide functions for doing finite element analysis. It was implemented using an object-oriented extension of *Mathematica*’s internal programming language, *Classes*, by Roman Maeder. There is a complementary version which uses a math library written in C++ for solving larger problems. Usually, after prototyping a problem completely in *Mathematica*, I automatically generated a C++ equivalent. The logical structure of an FEM

problem is common in both languages. The framework can be accessed easily during a *Mathematica* session.

```
In[1]:= Needs["Fem"];
```

The primary objects in the framework are called **Fem**, **FemNode**, and **FemElement**. **Fem** provides the context which captures the global problem's structure and provides access to global operations such as assembly, display, and so on. It contains a finite element mesh, which is an unstructured mesh that is a set of elements and a set of nodes. Nodes represent points in space. Elements define a topological relationship between the nodes. **FemNode** abstracts the finite element node and contains the point in some space that it represents. **FemElement** abstracts the element-level operations such as the element integrals and matrices. **FemElement** contains an ordered list of nodes that defines the topology of the element. **FemElement** also provides the basis functions for the element and rules of element quadrature, that is, Gaussian integration.

FemElement is an abstract class; there should not be any instances of it. It provides the behavior common to all elements. The subclasses of the **FemElement** provide the behavior for specific element types. A typical class hierarchy for **FemElement** follows.

```
FemElement
  TimeElement
    TwoNodeLineTime
  SpaceTimeElement
    FourNodeSpaceTimeQuad
  SpatialElement
    TwoNodeLine
    ThreeNodeLine
    FourNodeLine
    EightNodeBrick
    FifteenNodeBrick
    TwentySevenNodeBrick
    FourNodeTetrahedron
    TenNodeTetrahedron
    FourNodeQuad
    NineNodeQuad
    ThreeNodeTriangle
    SixNodeTriangle
```

■ Let's Do Some Element Integrals

Most functions of elements can be defined in either the physical (global) element or a reference (local) element. The change of basis transformation, the Jacobian matrix, provides the mapping between derivatives of the physical and reference element. Similarly, basis functions can be defined in physical or the reference space. I will consider basis functions in the reference space only.

Most of the entities of importance to an element are *element matrix functions*, which are matrix/vector-valued functions defined on the element. They can be

expressed as being defined either on the reference element E or on the physical element e . I will use the following notation: the variable ξ ranges over the reference element E , and the variable x ranges over the physical element e . The correspondence between them is denoted by $x(\xi)$, its inverse by $\xi(x)$, and the Jacobian of the map $x(\xi)$ is written $J(\xi)$. In fact, $J(\xi)$ is a matrix for each ξ , and so $J(\xi)$ is one of the matrix-valued functions of the element. In simple terms this says that, if you can express something on a nicely shaped simple element, then the Jacobian will help you translate into the very ugly-looking complex physical counterpart.

Imagine the physical variable x that is defined on the physical space. The basis functions of the element e are defined as $R(\xi) = [\varphi_1(\xi), \varphi_2(\xi), \dots, \varphi_m(\xi)]$, where m is the number of nodes of the element, the map $x(\xi)$ is written as:

$$x(\xi) = \sum_{i=1}^m \varphi_i(\xi) \xi_i$$

For example, $T(x)$ is a temperature field defined as $T(x) = x^2 + 3x + 4$ (*just imagine that!*), and the nodal temperatures T_i are $T(x_i)$. Using one-dimensional element ($\varphi_1(\xi) = 1/2(1 - \xi)$, $\varphi_2(\xi) = 1/2(1 + \xi)$), the mapping $T(\xi)$ is obtained as $T(\xi) = 1/2(1 - \xi)T_1 + 1/2(1 + \xi)T_2$. In the following example with $x_1 = 12.5$, $x_2 = 30.5$, $T_1 = 98.875$, and $T_2 = 512.875$, the temperature $T(x(\xi))$ varies linearly in the element and can be plotted as:

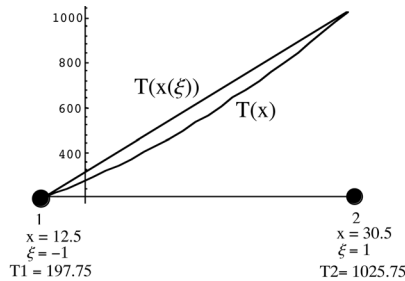


Figure 1.

Let's define this element and its nodes using the following framework.

```
In[2]:= node1 = new[FemNode, 1, {12.5}];
        node2 = new[FemNode, 2, {30.5}];
```

These statements create two nodes, **node1** and **node2**, with the coordinates **{12.5}** and **{30.5}**, and IDs **1** and **2** respectively. The **element** is defined as:

```
In[4]:= element = new[TwoNodeLine, {node1, node2}];
```

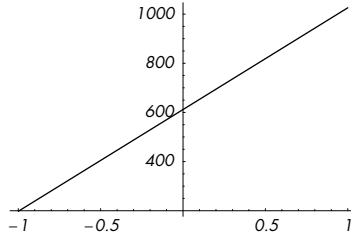
This statement creates a **TwoNodeLine** element whose topology is defined by the ordered list of nodes **{node1, node2}**. The **TwoNodeLine** element is a subclass of the **FemElement** class. The next two statements create the interpolated temperature map $T(\xi)$ and plot it in the reference element for the points $\xi \in (-1, 1)$.

```
In[5]:= T = interpolate[element, {197.75, 1025.75}]
```

```
Out[5]= 98.875 (1 - r) + 512.875 (1 + r)
```

```
In[6]:= Plot[T, {r, -1, 1}];
```

From In[6]:=



The Jacobian operator $\mathcal{J}$ of the element relates the physical coordinate derivatives to the reference coordinate derivatives:

$$\frac{\partial}{\partial \xi} = \mathcal{J} \frac{\partial}{\partial x} \quad \text{and} \quad \frac{\partial}{\partial x} = \mathcal{J}^{-1} \frac{\partial}{\partial \xi}$$

Using the same one-dimensional element above, the Jacobian of the element can be computed by:

```
In[7]:= J = jacobian[element]
```

```
Out[7]= {{9.}}
```

It is also possible to do all of these operations symbolically. Using the same problem as before, but with symbolic variables for coordinates $\{a, b\}$ and temperatures (T_a and T_b) at the nodes 1 and 2 respectively, the next example is obtained.

```
In[8]:= node1 = new[ FemNode, 1, {a}];
```

```
In[9]:= node2 = new[ FemNode, 2, {b}];
```

```
In[10]:= element = new[ TwoNodeLine, {node1, node2}];
```

```
In[11]:= T = interpolate[element, {Ta, Tb}]
```

```
Out[11]= 1/2 (1 - r) Ta + 1/2 (1 + r) Tb
```

```
In[12]:= J = jacobian[element]
```

```
Out[12]= {{1/2 (-a + b)}}
```

A more comprehensive list of the element matrix operations for the **FemElement** class follows.

1. **jacobian**. The reference Jacobian, $\mathcal{J}(\xi)$. This function returns an element matrix of size $M \times N$, where M is the dimension of the reference space and N is the dimension of the physical space. For example,

$$\mathcal{J} = \begin{bmatrix} \partial x / \partial r & \partial y / \partial r & \partial z / \partial r \\ \partial x / \partial s & \partial y / \partial s & \partial z / \partial s \\ \partial x / \partial t & \partial y / \partial t & \partial z / \partial t \end{bmatrix}$$

`J = jacobian[element]`

`JInv = inverseJacobian[element]`

represents a Jacobian with a three-dimensional reference space (r, s, t) and three-dimensional physical space (x, y, z) .

2. **shapeFunctions.** The nodal scalar basis matrix $R(\xi)$. This function returns an $M \times 1$ matrix, where M is the number of nodal basis functions.

$$R(\xi) = \begin{bmatrix} \varphi_1(\xi) \\ \varphi_2(\xi) \\ . \\ . \\ \varphi_m(\xi) \end{bmatrix}_{M \times 1}$$

`R = shapeFunctions[element]`

3. **gradient.** The gradient of $R(\xi)$ in the reference space is an $N \times M$ matrix, where N is the dimension of the reference element and M is the number of nodal basis functions.

$$\nabla_{\xi} R(\xi) = [\nabla \varphi_1(\xi) \quad \nabla \varphi_2(\xi) \quad \dots \quad \nabla \varphi_m(\xi)]_{N \times M}$$

`GradR = gradient[element]`

4. **physicalGradient.** The gradient of $R(\xi)$ in the physical space is an $N \times M$ matrix, where N is the dimension of the physical element and M is the number of nodal basis functions.

$$\nabla_x R(\xi) = \mathcal{J}(\xi)^{-1} \nabla_{\xi} R(\xi)$$

`GradR = physicalGradient[element]`

5. **interpolate.** This returns the map $x(\xi) \sum_{i=1}^m \varphi(\xi)_i \xi_i$ for an element. It requires a nodal function as an argument. The nodal function is either a discrete list of scalars $M \times 1$, where M is the number of nodal basis functions, or a pure function.

`In[13]:= interpolate[element, {a,b}]`

$$\text{Out[13]} = \frac{1}{2} a (1 - r) + \frac{1}{2} b (1 + r)$$

`In[14]:= interpolate[element, Sin[r]]`

$$\text{Out[14]} = \frac{1}{2} (1 - r) \text{Sin}[r] + \frac{1}{2} (1 + r) \text{Sin}[r]$$

6. **physicalIntegrate.** In general, a system of continuum mechanics equations that form the basis of the physical phenomenon are described.

These equations are then discretized using one of the standard finite element procedures such as the Galerkin formulation, least squares method, and so on.

The resulting equations, also called the governing FEM equations, are generally provided in the form of inner products in terms of other functions of the element, such as the basis functions, gradients, and so on, state variables in the problem, and material properties. The most common inner product that is utilized in FEM is the element integration.

A typical example is the Laplacian operator $\nabla \cdot \nabla \psi$, where ψ is a scalar field. FEM discretization of the Laplacian operator can be written as the following inner product:

$$\nabla \cdot \nabla \psi \Rightarrow \langle \nabla \psi, \nabla \varphi_i \rangle$$

where $\{\varphi_j \mid j = 1, \dots, m\}$ is the finite dimensional space test function space.

With some manipulation and using the standard Galerkin FEM discretization the Laplacian operator is written as the integral

$$\nabla \cdot \nabla \psi = 0$$

$$\hat{\psi}(\xi) = \varphi_j \tilde{\psi}_j$$

where $\tilde{\psi}_j$ is a scalar value defined at each node

$$\nabla \cdot \nabla \hat{\psi}(\xi) = \varepsilon$$

where ε is the error introduced by the approximation.

The standard Galerkin FEM suggests

$$\int_{\Omega} \varphi_j (\nabla \cdot \nabla \hat{\psi}(\xi)) d\Omega = 0$$

$$\int_{\Omega} \varphi_j (\nabla \cdot \nabla \varphi_j \tilde{\psi}_j) d\Omega = 0$$

Applying the divergence theorem

$$\begin{aligned} \int_{\Omega} \nabla \varphi_i \nabla \varphi_j \tilde{\psi}_j - \int_{\nabla \Omega} \varphi_i \mathbf{n} \cdot \nabla \varphi_j &= 0 \\ \left(\int_{\Omega} \nabla \varphi_i \nabla \varphi_j \right) \tilde{\psi}_j &= 0 \end{aligned}$$

This can be evaluated by the following integral

$$K = \int_{\Omega_{\text{element}}} \nabla R^T \nabla R d\Omega_{\text{element}}$$

where Ω_{element} is the subdomain of the element in the physical space.

In the reference space the integral can be evaluated as:

$$K_E = \int_E [\mathcal{J}(\xi)^{-1} \nabla_{\xi} R(\xi)]^T [\mathcal{J}(\xi)^{-1} \nabla_{\xi} R(\xi)] \text{Det}(\mathcal{J}(\xi)) d\xi$$

physicalIntegrate is a function provided by the package. It does the element integration in the reference space. The function **physicalIntegrate** internally

performs the $\int_E [\] \text{Det}(\mathcal{J}(\xi)) d\xi$ part of the integral. Therefore, the only part of the expression that the programmer must provide is $[\mathcal{J}(\xi)^{-1} \nabla_{\xi} R(\xi)]^T [\mathcal{J}(\xi)^{-1} \nabla_{\xi} R(\xi)]$. **physicalIntegrate** accepts this expression as an argument. The expression can be any function of the element. For example, if the expression for $[\mathcal{J}(\xi)^{-1} \nabla_{\xi} R(\xi)]^T [\mathcal{J}(\xi)^{-1} \nabla_{\xi} R(\xi)]$ is

```
In[15]:= LaplacianIntegrand[e_] :=
Module[{GradR},
  GradR = physicalGradient[e];
  (Transpose[GradR] . GradR)
];
```

then the integral $K_E = \int_E [\mathcal{J}(\xi)^{-1} \nabla_{\xi} R(\xi)]^T [\mathcal{J}(\xi)^{-1} \nabla_{\xi} R(\xi)] \text{Det}(\mathcal{J}(\xi)) d\xi$ is

```
In[16]:= Laplacian[e_] := physicalIntegrate[e, LaplacianIntegrand];
```

The use of the Laplacian operator, **Laplacian[e_]**, can be demonstrated by an example with a two-dimensional element. In the following example a new element class called the **LaplacianElement** is created as a subclass of the **FourNodeQuad** element. **FourNodeQuad** is a subclass of the **FemElement**.

```
In[17]:= Class[LaplacianElement, FourNodeQuad,
  {},
  {}];
```

Let's define an instance of the Laplacian element by the following setup.

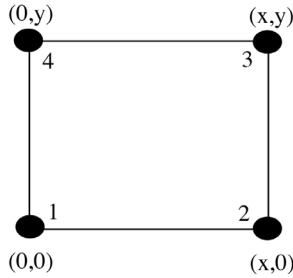


Figure 2.

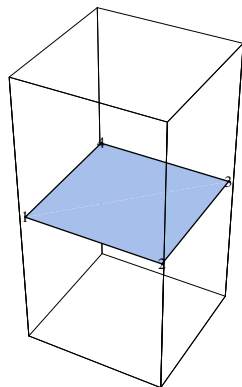
```
In[18]:= n1 = new[FemNode, 1, {0,0}];
n2 = new[FemNode, 2, {x,0}];
n3 = new[FemNode, 3, {x,y}];
n4 = new[FemNode, 4, {0,y}];

In[22]:= element = new[LaplacianElement, {n1,n2,n3,n4}];
lap = Laplacian[element];
```

If x and y were set to $x = 1$, $y = 1$ as in the following example,

```
In[24]:= Show[Graphics3D[display[element] /. {x→1,y→1}]];
```

```
From In[24]:=
```



then the Laplacian operator for the element is:

```
In[25]:= MatrixForm[lap /. {x→1,y→1}]
```

```
Out[25]//MatrixForm=
( 0.666667  -0.166667  -0.333333  -0.166667 )
( -0.166667  0.666667  -0.166667  -0.333333 )
( -0.333333  -0.166667  0.666667  -0.166667 )
( -0.166667  -0.333333  -0.166667  0.666667 )
```

■ Going Global

Element-level operations establish the local equations governing the behavior of the element in the subdomain Ω_{element} of the element. However, the behavior of the system, which is the assembly of all the elements, is defined on the domain $\Omega \approx \prod \Omega_{\text{element}}$, where $\prod$ is the assembly operator.

In practical terms this amounts to an assembly of the local element integrals into a global system of equations. The assembly operation is best described with an example. Let's create a mesh with a set of **LaplacianElements** as described in the previous section.

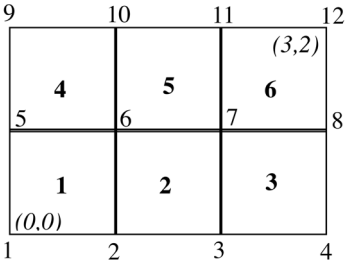


Figure 3.

Three rows and four columns of nodes describe this mesh. These nodes are created as a regular grid with the following statements.

```
In[26]:= rows = 3;
        cols = 4;
        nuds = Flatten[Table[new[FemNode, i + (j - 1) * cols, {i - 1, j - 1}],
            {j, 1, rows}, {i, 1, cols}]];
```

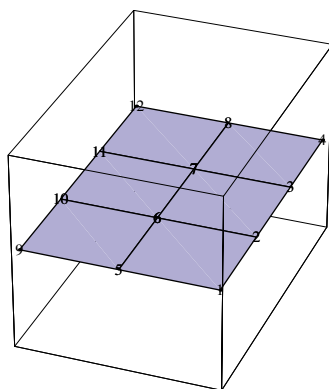
Next, two rows and three columns of elements on the regular grid are described as:

```
In[29]:= els = Flatten[Table[
        new[LaplacianElement,
            {nuds[[i + (j - 1) * cols]],
            nuds[[i + 1 + (j - 1) * cols]],
            nuds[[i + 1 + j * cols]],
            nuds[[i + j * cols]]}],
        {j, 1, rows - 1}, {i, 1, cols - 1}]];
```

Let's create a mesh describing this geometry and the topology of elements. We have created 12 nodes and 6 elements in two lists called **nuds** and **els** respectively. The mesh is an instance of the class called **Fem**.

```
In[30]:= mesh = new[Fem, nuds, els];
        Show[display[mesh], ViewPoint -> {-7.091, -2.682, 4.449}];
```

From In[30]:=



Using the element-level matrices, the global system of equations can be described for the Laplacian equation on this grid. This involves computing the element integrals and assembling them into the global system of equations. The assembly operation is performed by the method called **assembleblock** in the **Fem** class. This method requires a matrix-valued element function such as **Laplacian[e_]** which was defined above. It will then loop over the elements in the mesh and assemble the matrices returned by the function passed in as the argument into the global matrix, K . In this example the global matrix K is a 12×12 matrix.

```
In[32]:= K = assembleblock[mesh, Laplacian, 1];
MatrixForm[K]
```

```
Out[33]//MatrixForm=
```

0.666667	-0.166667	0	0	-0.166667	-0.333333	0	0	0	0	0	0
-0.166667	1.333333	-0.166667	0	-0.333333	-0.333333	-0.333333	0	0	0	0	0
0	-0.166667	1.333333	-0.166667	0	-0.333333	-0.333333	-0.333333	0	0	0	0
0	0	-0.166667	0.666667	0	0	-0.333333	-0.166667	0	0	0	0
-0.166667	-0.333333	0	0	1.333333	-0.333333	0	0	-0.166667	-0.333333	0	0
-0.333333	-0.333333	-0.333333	0	-0.333333	2.666667	-0.333333	0	-0.333333	-0.333333	-0.333333	0
0	-0.333333	-0.333333	-0.333333	0	-0.333333	2.666667	-0.333333	0	-0.333333	-0.333333	-0.333333
0	0	-0.333333	-0.166667	0	0	-0.333333	1.333333	0	0	-0.333333	-0.166667
0	0	0	0	-0.166667	-0.333333	0	0	0.666667	-0.166667	0	0
0	0	0	0	-0.333333	-0.333333	-0.333333	0	-0.166667	1.333333	-0.166667	0
0	0	0	0	0	-0.333333	-0.333333	-0.333333	0	-0.166667	1.333333	-0.166667
0	0	0	0	0	0	-0.333333	-0.166667	0	0	-0.166667	0.666667

The Laplacian field on this grid can be calculated for a given set of boundary conditions. If the boundary conditions are defined as $\psi = x$ at $x = 0$ and $x = 3$ ($\psi_{x=0} = 0$, $\psi_{x=3} = 3$), the exact solution of this problem exists and is defined by $\psi(x, y) = x$.

The boundary conditions describe the values of the Laplacian field at the nodes 1, 5, 9 and 4, 8, 12. The values of the Laplacian field for the remainder of the nodes must be computed. The global system of equations is given as:

$$K[\psi] = [0]$$

The condensed form of the equations, after the prescribed values multiply their columns, are moved to the right-hand side of the equations, and written as:

$$K_{\text{unknowns}} [\psi_{\text{unknowns}}] = -K[\psi_{\text{prescribed}}]$$

The resulting equations can be solved with:

```
In[34]:= nodalvalues = {{0},{0},{0},{3},{0},{0},{0},{3},{0},{0},{0},{3}};
unknownnodes = {2,3,6,7,10,11};
Kcondensed = K[[unknownnodes, unknownnodes]];
load = (K . nodalvalues)[[unknownnodes, 1]];
LinearSolve[Kcondensed, -load]
```

```
Out[38]= {1., 2., 1., 2., 1., 2.}
```

These are the exact answers for the nodes {2, 3, 6, 7, 10, 11} respectively. However, the *Fem* package provides another convenience for imposing boundary conditions and solving the resulting linear system. I can redefine the same problem as:

```
In[39]:= fixednodes = {{1,4,5,8,9,12}};
fixedvalues = {{0,3,0,3,0,3}};
b = Table[0, {i,1,12}, {j,1,1}];
linearSolve[mesh, K, b, fixednodes, fixedvalues]
```

```
Out[42]= {0, 1., 2., 3, 0, 1., 2., 3, 0, 1., 2., 3}
```

■ A More Interesting Example

Let's solve a larger and more interesting Laplacian problem: a simple sink and a source on a rectangular grid. It can be defined with the following boundary conditions and mesh.

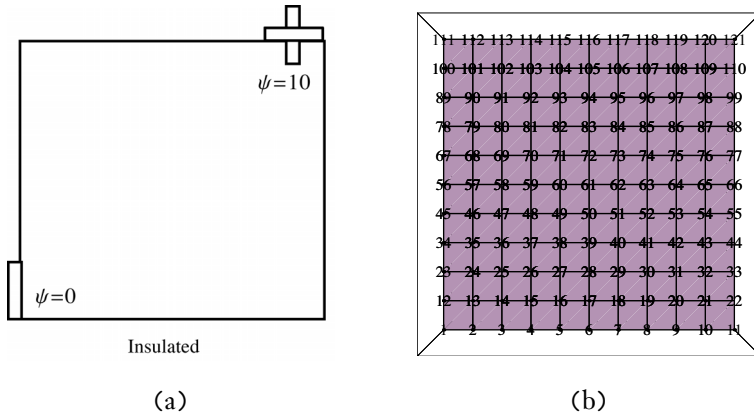


Figure 4. A two-dimensional Laplacian problem: (a) Problem description and (b) the FEM mesh. © Copyright Naci M. Dai.

The mesh in Figure 4(b) is formed by the following regular grid:

```
In[51]:= rows = 11;
cols = 11;
nods = Flatten[Table[new[FemNode, i + (j - 1) * cols, {i - 1, j - 1}],
  {j, 1, rows}, {i, 1, cols}]];
els = Flatten[Table[new[LaplacianElement,
  {nods[[i + (j - 1) * cols]], nods[[i + 1 + (j - 1) * cols]],
  nods[[i + 1 + j * cols]], nods[[i + j * cols]]}],
  {j, 1, rows - 1}, {i, 1, cols - 1}]];
mesh = new[Fem, nods, els];
Show[display[mesh], ViewPoint → {0, 0, 0.9}];

mesh = new[Fem, nods, els];
globalMatrix = assembleblock[mesh, Laplacian, 1];
```

Each node in the domain has only one degree-of-freedom (dof); therefore it has the same index as the node number. Providing the fixed dofs and their values as below, we solve the linear system of equations:

```
In[59]:= fixeddof = {{1, 12, 23, 119, 120, 121}};
fixedvalues = {{0, 0, 0, 10, 10, 10}};
globalLoad = Table[0, {i, 1, 121}, {j, 1, 1}];
ans =
  linearSolve[mesh, globalMatrix, globalLoad, fixeddof, fixedvalues];
```

The FEM domain is represented by a regular grid. The result can be plotted with the built-in function **ListContourPlot**.

```

In[63]:= Needs["Graphics`Graphics3D`"];
Needs["Graphics`Legend`"];

HueMod[h_] := Hue[1- 0.8*h];

ShowLegend[
ListContourPlot[Partition[ans, cols], Contours→19,
ContourSmoothing→Automatic,
ColorFunction→HueMod,
DisplayFunction→Identity],
{HueMod, 10, "0", " 10",
LegendPosition→{1.1,-0.4}}];

```

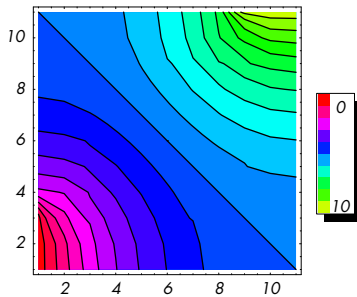


Figure 5. The contour plot for the scalar field approximated by the FEM solution.
© Copyright Naci M. Dai.

■ The Gory Details

It is time to look under the hood. The full source code is available in the file `Fem.m`. Interested users may consult the `::usage` messages for a detailed description of the arguments and implementation of the operations. It is necessary to introduce some of the basic terminology and the syntax of object oriented programming in *Mathematica* before the description of the package itself.

Object oriented programming (OOP) combines the data and functional aspects of a program into one integrated view. An OOP is described as interactions of *objects* in a collective manner. An *object* encapsulates the representation (*data/instance variables*) with the *operations* that provide its behavior. A *class* is the meta-description of an object. It is a template from which the *instances* of objects are created. Inheritance is an important aspect of OOP and is a mechanism for code reuse. New classes can be derived from existing classes. A *subclass* inherits all the allowed operations and representation of its *superclass*. It can modify and override some of the behavior of its superclass and extend its representation.

Mathematica is not an OOP language. Maeder [4] has provided a framework for Smalltalk-like [5] OOP in *Mathematica*, which makes the *Fem* package possible. It is the best of both worlds: OOP and *Mathematica* all in one. OOP in *Mathematica*

is provided by the package named *Classes*. In Maeder's framework, a new class can be described as:

```
Class[class, superclass, variables, methods]
```

where

<i>class</i> :	the symbolic name for the class,
<i>superclass</i> :	the symbolic name of the super class,
<i>variables</i> :	list of symbols that correspond the instance variables of a class formally described as: {_Symbol...},
<i>methods</i> :	list of methods {m1, m2, m3, ...} of the class. Each method is a pair {name, body}. The name of the method can be any symbol and the body of the method can be any <i>Mathematica</i> expression. Formally, a list of methods is described as {{_Symbol, _}...}{{}}.

□ The Package

The *Fem* package requires the standard *Mathematica* packages *Classes*, *Gaussian Quadrature*, and *MatrixManipulation*.

```
BeginPackage["Fem`","Classes`",
  "NumericalMath`GaussianQuadrature`","LinearAlgebra`MatrixManipulation`"]
```

Many FEM computations are done in the reference map of an element. A set of symbols is reserved to represent the dimensions of the reference space. The notation is generalized for arbitrary 1D, 2D, 3D, and 4D space and time coordinates. **r**, **s**, and **t** are reserved for the Cartesian reference coordinate frames corresponding to the space elements in 1D (**r**), 2D (**r,s**), and 3D (**r,s,t**). The time dimension is referred to as **thau**.

```
Unprotect[ref, r, s, t, thau];
Clear[r, s, t, thau];
ref={r, s, t, thau};
Protect[ref, r, s, t, thau];
```

FemNode Class

The class **FemNode** represents the node in an FEM problem. The **FemNode** is a simple class that represents a global degree of freedom and a position in space.

```
Class[FemNode, Object, {pos,localId}, ...
```

The instantiation method **new** requires two arguments, **i** and **location**, which are the integer node ID and the list that represents the coordinate of the node in any dimension.

```
{new,
  Function[{i,location},
    new[super];
    setCoordinate[self, location];
    setId[self, i];
    self]]}
```

Here is the protocol that provides the get-set operations for the **FemNode** class.

```
{coordinate, pos&}
{id, localId&}
{setId, Function[{i}, localId = i]}
{setCoordinate, Function[{p}, pos = p]}
```

The protocol to support graphics operations includes two methods. **asPoint** returns the coordinate of the node as a 3D point. **displayNode** returns a graphics list that contains a **Point** and **Text** object corresponding to the location of the node and its ID.

```
{asPoint,
  Module[{pstn,i},
    If[ Length[pos] < 3,
      pstn = Table[0, {i,1,3}];
      Do[pstn[[i]] = pos[[i]], {i,1,Length[pos]}];
      Return[pstn],
      Return[pos]]]&}
{displayNode,
  {Point[asPoint[self]],
   Text[FontForm[InputForm[id[self]],
    {"Times-Roman",14}],asPoint[self]]}&}
```

FemElement Class

This is the abstract class that defines the behavior of an element. There should not be any instances of this class. The subclasses of **FemElement** define the behavior of the specific elements. The subclass of the **FemElement** must implement at least four methods.

1. *functions*. The method that returns the basis functions for an element in a list. Basis functions must be functions of (**r**, **s**, **t**, and **thau**).
2. *gausspoints*. The list of Gauss points for the element. This list is used for the Gaussian quadrature. The list contains a list of coordinates for the Gauss points. Each coordinate is a list of n numbers, where n is the dimension of the reference space for this element.
3. *gaussweights*. The list of Gauss weights corresponding to the Gauss points. This is a list of numbers. *gausspoints* and *gaussweights* must be present; otherwise no element function that performs **physicalIntegrate** will yield valid results.
4. *dimension*. The dimension of the reference space for the element.

```
Class[FemElement, Object, {nodz}, ...
```

The instantiation method **new** requires one argument **n** which is the list that contains the nodes of the element. The order of the nodes is important because it represents the topology of the element.

```
{new,
  Function[{n},
    new[super];
    setnodes[self, n];
    self]}
```

Get-set operations for the **FemElement** class are:

```
{dimension, 0}&(*Default-Subclass must implement*)
{setnodes, Function[{c}, nodz = c]}
{nodes, nodz&}
{numberOfNodes, Length[functions[self]]&}
{getlocations, (coordinate[#]& /@ nodz)&}
```

Methods for basis functions are:

```
{functions, {1}&(*Default-Subclass must implement*)
{shapeFunctions, (List[#]& /@ functions[self])&}
{gradient, Module[{i, grad},
  grad = Table[{D[#, ref[[i]]]& /@ functions[self]},
    {i, 1, dimension[self]}];
  Return[Simplify[grad]]&}
{physicalGradient,
  (inverseJacobian[self] . gradient[self])&}
{interpolate, Function[{f}, Plus@@(f*functions[self])]}
```

Methods that compute the Jacobian, its inverse, and its determinant for the element are:

```
{jacobian, Module[{rdim, pdim, pts, xyzt, jac, i},
  rdim = dimension[self]; (*Dimension of reference space*)
  pts = Transpose[getlocations[self]];
  pdim = Length[pts]; (*Dimension of physical space*)
  xyz = Table[Plus@@(functions[self]*pts[[i]]),
    {i, 1, pdim}];
  jac = Table[{D[#, ref[[i]]]& /@ xyz},
    {i, 1, rdim}];
  Return[jac]]
{inverseJacobian, (Inverse[jacobian[self]])&}
{detJacobian, (Det[jacobian[self]])&}
```

The following methods implement Gaussian quadrature for the element. Integration over the reference element is done by evaluating the element function at each Gauss point and returning the sum of products of these matrices with the weight and determinant of the Jacobian at those Gauss points.

```
{gausspoints, {}}&(*Default-Subclass must implement*)
{gaussweights, {}}&(*Default-Subclass must implement*)
{physicalIntegrate,
  Function[{integ}, Module[{f, i, detJ, gw, gp, lcs, num},
    f = integ[self];
    gw = gaussweights[self];
    num = Length[gw];
    gp = gausspoints[self];
    lcs = Take[{ref, dimension[self]}];
    detJ = detJacobian[self];
    Sum[gw[[i]] (detJ /. Thread[lcs->gp[[i]]])
      (f /. Thread[lcs->gp[[i]]]),
    {i, 1, num}]]]
```

Graphics support is provided by the **display** function, which returns a list of polygons that represent the element and its nodes.

```
{display, (Module[{l,i,polys,p},
  l = (List[displayNode[#]]& /@ nodes[self]);
  polys = asPolygon[self];
  p = Table[Polygon @ (asPoint[#]& /@ polys[[i]])),
    {i,1,Length[polys] }];
  Join[l, p]]&}
```

Fem Class

The class that represents the global FEM problem and the mesh is called the **Fem** class. It holds the list of nodes and elements in the problem.

```
Class[ Fem, Object, {nodz, elmts}, ...
```

The instantiation method **new** requires two arguments, **n** and **e**, which are the list of nodes and elements respectively.

```
{new,
  Function[
    {n,e},
    new[super];
    setnodes[self,n];
    setelements[self,e];
    self]}
```

The protocol that provides the get-set operations for the **Fem** class is:

```
{elements, elmts&}
{nodes, nodz&}
{setnodes, Function[{c}, nodz = c]}
{setelements, Function[{e}, elmts = e]}
```

The mesh of the problem can be visualized. The method **display** returns a **Graphics3D** object that can be displayed.

```
{display, (Graphics3D[display[#]]& /@ elmts)&}
```

The element matrices and vectors describe governing equations that correspond to the local degrees of freedom in the elements. The assembly protocols provide methods to join the local element's dof matrix to the global matrix. Block assembly assumes that each degree of freedom at a node of the element is assembled as one block. The global matrix is organized similarly. The assembly function requires an argument that is an element function that returns a matrix.

```
{assembleblock,
  Function[{elementFunction, dofNode},
    Module[{nd,n,m,em,ng,gi,gj,li,lj,mg,d,i,j,k,nelz},
      n = Length[nodz]*dofNode;
      nelz = Length[elmts];
      gm = ZeroMatrix[n]; (*Global Matrix*)
      {ng,mg} = Dimensions[gm]/dofNode;
      Do[(*Loop over the elements*)
        em = elementFunction [elmts[[k]]];
```

```

nd = nodes[elmts[[k]]];
{n,m} = Dimensions[em]/dofNode;
Do[(*Assemble local matrix em into gm*)
  gi = id[nd[[i]]] + ((d-1)*ng);
  gj = id[nd[[j]]] + ((d-1)*mg);
  li = i + ((d-1)*n);
  lj = j + ((d-1)*m);
  gm[[gi, gj]] += em[[li, lj]];
  , {d,1,dofNode}, {i,1,n}, {j,1,m} ,
  {k,1,nelz}];
gm
]]}

```

linearSolve solves the linear system of equations $Kx = b$ and returns the answer x as a vector. In addition to solving the linear system of equations, the function can also impose the boundary conditions defined by the arguments **fixed** and **fixedvalues**, which correspond to the fixed degrees of freedom in the problem.

```

{linearSolve,
  Function[{K,b,fixed, fixedvalues},
    Module[{freedof,fixeddof, known,
      dof,n, Kremoved,Kcondensed,
      load,bcondensed,x,answer},
      dof = Length[fixed];
      n = Dimensions[K][[1]]/dof;
      fixeddof = Flatten[Table[((i-1)n + #)& /@ fixed[[i]],
        {i,1,dof}]];
      freedof = Complement[Range[1, dof*n], fixeddof];
      known = Flatten[fixedvalues];
      Kremoved = Transpose[ Transpose[K][[fixeddof]]];
      Kcondensed = K[[freedof, freedof]];
      bcondensed = b[[freedof, {1}]];
      answer = Table[0, {i,1,n*dof}];
      (answer[[#]] = known[[Position[fixeddof, #][[1, 1]]]]&
        /@ fixeddof;
      load = bcondensed -
        ((Kremoved . (List/@known))[[freedof]]);
      x = Flatten[LinearSolve[Kcondensed, load]];
      (answer[[#]] = x[[Position[freedof, #][[1, 1]]]]&
        /@ freedof;
      answer]]}

```

□ Implementing Subclasses of FemElement

The **FemElement** class is an abstract class. It defines some behavior without defining implementation details of a real FEM element, (e.g., **methods**, **dimension**, **function**, **gausspoints**, **asPolygon**). The minimum set of methods that must be implemented by a *concrete* subclass (a class that can have instances) of the **FemElement** are: (1) *functions*, (2) *gausspoints*, (3) *gaussweights*, and (4) *dimension*. In addition to these methods, if the method **asPolygon** is implemented, the element can be displayed.

Additional abstract classes are also provided for convenience. These are: **SpatialElement**, **TimeElement**, and **SpaceTimeElement**. **SpatialElement** provides some additional structure for classifying elements that represent a spatial subdomain. **TimeElement** is a subdomain of time. The **SpaceTimeElement** class describes a subdomain in space-time. **SpaceTimeElement** and **TimeElements** override the methods that compute Gauss, its inverse, and so on. They also provide additional methods for gradients of the basis functions with respect to space and time coordinates.

SpatialElement

This is a shadow subclass of **FemElement**.

```
Class[SpatialElement, FemElement, {}, {}];
```

TimeElement

The Jacobian of the **TimeElement** has the derivatives of the basis functions with respect to the time dimension only.

```
Class[TimeElement, FemElement, {}, {
  {jacobian, Module[{dim, pts},
    dim = dimension[self];
    pts = Transpose[getlocations[self]];
    jac = ZeroMatrix[dim];
    jac[[dim, dim]] = (D[#, thau] & /@
      (Plus@@(functions[self]*pts[[dim]])))&);
    Return[jac] ]&}]
```

SpaceTimeElement

The Jacobian of the **SpaceTimeElement** has the derivatives of the basis functions with respect to space and time dimensions. **SpaceTimeElement** also provides methods for spatial and time derivatives of basis functions.

```
Class[SpaceTimeElement, FemElement, {}, {
  {inverseJacobian, Module[{invjac, i, jac, coltime, dim, sdim},
    jac = jacobian[self];
    dim = dimension[self];
    sdim = Range[1, dim-1];
    invjac = Inverse[jac[[sdim, sdim]]];
    coltime = -invjac . jac[[dim, sdim]];
    invjac = AppendRows[invjac, Transpose[{coltime}]];
    invjac = AppendColumns[invjac, ZeroMatrix[1, dim]];
    invjac[[dim, dim]] = 1/jac[[dim, dim]];
    Simplify[invjac]]&,
  {detJacobian, Module[{jac, dim, sdim},
    jac = jacobian[self];
    dim = dimension[self];
    sdim = Range[1, dim-1];
    (jac[[dim, dim]] Det[jac[[sdim, sdim]]])&},
  {detJacobianSpatial, Module[{jac, sdim},
    jac = jacobian[self];
    sdim = Range[1, dimension[self]-1];
    Det[jac[[sdim, sdim]]]&},
  {shapeFunctionsSpatial, Module[{fcs, numnodz},
```

```

numnodz = numberOfNodes[self];
fcs = functions[self];
Return[Simplify[
  Partition[fcs, 1][[Range[1, Length[fcs]/2]]]]
/. thau->0]]&},
{gradientSpatial, Module[{i, grad},
  grad = Table[(D[#, ref[[i]]]& /@ functions[self]),
    {i, 1, dimension[self]-1}];
  Return[Simplify[grad]]]&},
{gradientTime, Module[{i, grad},
  grad = {D[#, thau]& /@ functions[self]};
  Return[Simplify[grad]]]&}

```

□ Concrete Subclasses of the FemElement

A typical class hierarchy for the **FemElement** was provided earlier. The description of only one of these classes is included here for completeness. It demonstrates the stages of implementing a subclass of the **FemElement**. The 3D trilinear brick element **EightNodeBrick** with a two point Gaussian quadrature rule can be implemented as follows.

```

npoint = 2;
gaussquad = GaussianQuadratureWeights[npoint, -1, 1, 50];
gausspointsbrick = Partition[Flatten[Table[
  {gaussquad[[i, 1]], gaussquad[[j, 1]], gaussquad[[k, 1]]},
  {i, 1, npoint}, {j, 1, npoint}, {k, 1, npoint}]], 3];
gaussweightsbrick = Flatten[Table[
  gaussquad[[i, 2]] gaussquad[[j, 2]] gaussquad[[k, 2]],
  {i, 1, npoint}, {j, 1, npoint}, {k, 1, npoint}]];
Class[ EightNodeBrick, SpatialElement, {}, {
  {dimension, 3&},
  {functions, {1/8(1+r)(1+s)(1+t),
    1/8(1-r)(1+s)(1+t),
    1/8(1-r)(1-s)(1+t),
    1/8(1+r)(1-s)(1+t),
    1/8(1+r)(1+s)(1-t),
    1/8(1-r)(1+s)(1-t),
    1/8(1-r)(1-s)(1-t),
    1/8(1+r)(1-s)(1-t)}&},
  {gausspoints, gausspointsbrick&},
  {gaussweights, gaussweightsbrick&},
  {asPolygon, Module[{faces, i, j},
    faces = {{6, 7, 3, 2},
      {7, 8, 4, 3},
      {7, 6, 5, 8},
      {8, 5, 1, 4},
      {5, 6, 2, 1},
      {4, 1, 2, 3}};
    Table[(nodes[self])[[faces[[i, j]]]],
      {i, 1, 6}, {j, 1, 4}]]&}
  }];

```

■ Additional Material

1. `Fem.m`
2. `Classes.m`

■ Acknowledgments

This work originated while I was completing my graduate studies at Carleton University. It is due to wonderful mentors like Prof. John Goldak and Dr. Edward Hughes that many clueless grad students regularly produce inspiring work. I also have to thank many of the fellow students at Carleton University who have endured my whims and failures as I tried to teach FEM using this work.

■ References

- [1] K.J. Bathe, *Finite Element Procedures in Engineering Analysis*, Englewood Cliffs, NJ: Prentice-Hall, 1982.
- [2] B. Irons, *Techniques of Finite Elements*, Chicester, England: E. Horwood, 1980.
- [3] O. C. Zienkiewicz, *The Finite Element Method*, 4th ed., London: McGraw-Hill, 1991.
- [4] R. Maeder, *The Mathematica Programmer*, Cambridge: AP Professional, 1994.
- [5] A. Goldberg, *Smalltalk-80*, 2nd ed., Addison-Wesley, 1989.

Linear and Morphological Image Processing

Mariusz Jankowski

University of Southern Maine
mjkcc@usm.maine.edu
www.usm.maine.edu/~mjkcc

Version 4 of *Mathematica* includes a number of enhancements especially relevant to the signal/image processing community. The introduction of an efficient internal representation for machine-precision numerical data, addition of import/export capabilities covering most signal and image interchange formats, and most importantly, the addition of a fast convolution operator have finally made it possible to use *Mathematica* as an effective platform for signal and image processing. In this article we take a close look at convolution and present a number of image processing operations that have a convolution-like implementation. This includes not only the better-known linear filters but also binary and grayscale morphological operators.

■ 1. Introduction

This article was prompted in part by a recent message posted to MathGroup that inquired about *Mathematica*'s functionality in the area of signal and image processing. Having successfully used *Mathematica* to teach courses in signal and image processing and as the developer of the *Digital Image Processing* package, a recent addition to *Mathematica*'s Applications Library, I responded with a short and enthusiastic summary of the relevant improvements in Version 4. Indeed, without these enhancements the *Digital Image Processing* package would not have been possible; its performance would simply not have been competitive with most major image processing products on the market. Conversely, the latest improvements make *Mathematica* an effective platform for signal and image processing.

In this article we take a close look at convolution in general and the new function **ListConvolve** in particular. Convolution is a key operation in most signal and image processing applications. Many useful image processing operations are

implemented as convolutions with finite impulse response (FIR) filters. Examples of image smoothing, unsharp masking, and gradient edge detection will be presented. Interestingly, the fundamental morphological operators of erosion and dilation may also be realized with convolution-like algorithms. We will show **ListConvolve**-based realizations of binary and grayscale erosion and dilation, and demonstrate edge detection using these fundamental operators.

■ 2. Convolution

One of the most widely used signal/image processing operations is filtering. In many practical cases the filtering operation is linear shift-invariant (LSI) and may be described by the following relation [1]:

$$y[n] = \sum_{i=-\infty}^{\infty} x[i] b[n-i] = \sum_{i=-\infty}^{\infty} b[i] x[n-i], \quad (2.1)$$

where $b[n]$ is the so-called impulse response of the filter, $x[n]$ is the input, and $y[n]$ is the output of the system. Equation (2.1) is called the convolution sum formulation of an LSI system. A single output value is computed by summing over all point-wise products of one sequence with a time-reversed and shifted version of the other.

An important category of discrete-time systems exhibits a finite impulse response (FIR), so $b[n]$ has finite length. It is common to define $b[n]$ on the interval $0 \leq n \leq N-1$. In practice, the sequence $x[n]$ is also finite in length, so that $y[n]$ can be computed in finite time. Consider two sequences $x[n]$ and $b[n]$, of lengths M and N , respectively, and defined for positive values of n only. Equation (2.1) then reduces to the following finite summation:

$$y[n] = \sum_{i=0}^{M-1} x[i] b[n-i] = \sum_{i=0}^{N-1} b[i] x[n-i], \quad 0 \leq n < M+N-1 \quad (2.2)$$

The output sequence $y[n]$ has length $L = M+N-1$. Equation (2.2) is called a linear convolution.

It is instructive to evaluate the sums in Equation (2.2) given some input sequence $x[n]$ and impulse response sequence $b[n]$. For demonstration purposes, we use two symbolic sequences.

```
In[1]:= (sig = Table[x1, {i, 0, 5}];
         fir = Table[h1, {i, 0, 2}];)
```

This returns a linear convolution of the two sequences using **ListConvolve**.

```
In[2]:= ListConvolve[fir, sig, {1, -1}, 0] // TableForm
```

```
Out[2]//TableForm=
```

```
h0 x0
h1 x0 + h0 x1
h2 x0 + h1 x1 + h0 x2
h2 x1 + h1 x2 + h0 x3
h2 x2 + h1 x3 + h0 x4
h2 x3 + h1 x4 + h0 x5
h2 x4 + h1 x5
h2 x5
```

The output sequence may be subdivided into three subranges. The first two output samples represent a transient portion of the output signal due to a partial overlap of nonzero values of the sequences $x[n]$ and $b[n]$. The middle four samples represent a steady state portion of the output sequence (nonzero values of $x[n]$ and $b[n]$ are fully overlapped). The last two output samples represent a second transient portion of the output sequence, which again is due to a partial overlap of the two sequences. The lengths of these transients are always given by $N - 1$, where $N = \text{Length}[b]$. A result that excludes the transient samples is sometimes sufficient or desirable.

```
In[3]:= ListConvolve[fir, sig] // TableForm
```

```
Out[3]//TableForm=
```

$$h_2 x_0 + h_1 x_1 + h_0 x_2$$

$$h_2 x_1 + h_1 x_2 + h_0 x_3$$

$$h_2 x_2 + h_1 x_3 + h_0 x_4$$

$$h_2 x_3 + h_1 x_4 + h_0 x_5$$

Two additional convolution formulations can be found in the signal processing literature. The first is called periodic convolution and defines a convolution operation for periodic sequences. Assuming two sequences $\tilde{x}[n]$ and $\tilde{b}[n]$ have periods M and N , respectively (i.e., $\tilde{x}[n] = \tilde{x}[n + M]$ and $\tilde{b}[n] = \tilde{b}[n + N]$), we define periodic convolution of the two sequences as

$$\tilde{y}[n] = \sum_{i=0}^{M-1} \tilde{x}[i] \tilde{b}[n-i] = \sum_{i=0}^{N-1} \tilde{b}[i] \tilde{x}[n-i], \quad (2.3)$$

where $0 \leq n < \min(M, N) - 1$. The output sequence $\tilde{y}[n]$ is also periodic with period $\min(N, M)$. Equation (2.3) looks very much like linear convolution and involves the summation of the products of one sequence with the time-reversed and shifted copy of the other. However, since the sequences in Equation (2.3) are periodic, shifts of such sequences produce results different from shifts of finite-duration signals (see Figure 2.1).

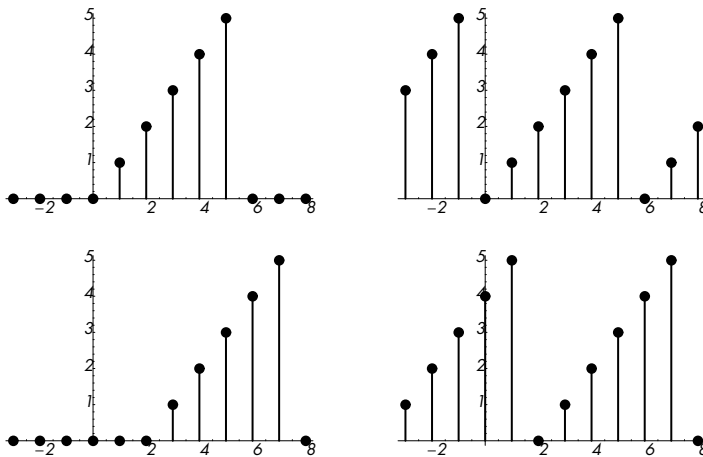


Figure 2.1. Demonstration of a right shift of an aperiodic (left) and periodic (right) signal.

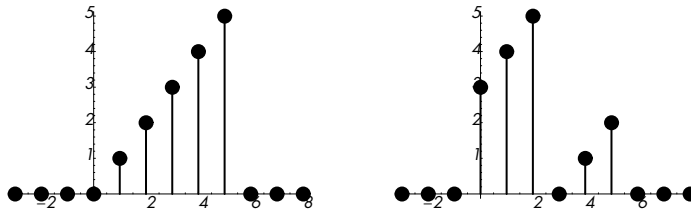


Figure 2.2. Example of a circular shift of a sequence of length $N = 6$.

The other convolution formulation is called circular convolution [1]. By analogy with linear convolution, the formulation applies to finite-length signals, but all shifts are “circular” as in periodic convolution (see Figure 2.2). Circular convolution is particularly important, since it arises from a discrete Fourier transform (DFT) implementation of the convolution operation. It is well known that given three sequences of length N , $x_1[n]$, $x_2[n]$, $x_3[n]$, and their respective N -point DFTs, $X_1[k]$, $X_2[k]$, $X_3[k]$, and further given that

$$X_3[k] = X_1[k] X_2[k] \quad (2.4)$$

it can be shown that the following convolution relationship exists between the sequences:

$$\begin{aligned} y[n] &= \sum_{i=0}^{N-1} x[i] b[(n-i)_N] \\ &= \sum_{i=0}^{N-1} b[i] x[(n-i)_N], \quad 0 \leq n \leq N-1 \end{aligned} \quad (2.5)$$

where $x[(n-i)_N]$ represents a circular shift of the sequence $x[n]$. A well-known algorithm called the fast Fourier transform (FFT) allows a fast computation of the DFT, and thus a fast implementation of convolution. Circular and linear convolutions yield different transient samples and identical steady-state samples.

Circular convolution is equivalent to periodic convolution if we consider sequences $x[n]$ and $b[n]$ to correspond to single periods of their respective infinite periodic extensions $\tilde{x}[n]$ and $\tilde{b}[n]$, such that

$$\begin{aligned} x[n] &= \tilde{x}[n], \quad 0 \leq n \leq N-1 \\ b[n] &= \tilde{b}[n], \quad 0 \leq n \leq N-1 \end{aligned} \quad (2.6)$$

We calculate the circular convolution of the two example sequences and compare with linear convolution.

```
In[4]:= ListConvolve[fir, sig, {1, 1}] // TableForm
```

```
Out[4]//TableForm=
```

```
h0 x0 + h2 x4 + h1 x5
h1 x0 + h0 x1 + h2 x5
h2 x0 + h1 x1 + h0 x2
h2 x1 + h1 x2 + h0 x3
h2 x2 + h1 x3 + h0 x4
h2 x3 + h1 x4 + h0 x5
```

An examination of the last result confirms that circular and linear convolutions yield the same steady-state samples. Note that the last four output samples are the same as the steady-state outputs resulting from linear convolution of the two sequences. It is frequently desirable to have the steady-state samples appear centered in the output list. This is achieved by setting the *klist* elements k_L and k_R equal to the index of the center element of **fir** (see **ListConvolve** for details).

```
In[5]:= ListConvolve[fir, sig, {2, 2}] // TableForm
```

```
Out[5]//TableForm=
```

```
h1 x0 + h0 x1 + h2 x5
h2 x0 + h1 x1 + h0 x2
h2 x1 + h1 x2 + h0 x3
h2 x2 + h1 x3 + h0 x4
h2 x3 + h1 x4 + h0 x5
h0 x0 + h2 x4 + h1 x5
```

FIR systems play an especially important role in 2D systems. In digital image processing, many useful operators (i.e., edge detectors) are linear with finite spatial dimensions. The 2D linear convolution sum takes the following form:

$$\begin{aligned} y[n_1, n_2] &= \sum_{i=0}^{N_1-1} \sum_{j=0}^{N_2-1} b[i, j] \cdot x[n_1 - i, n_2 - j] \\ &= \sum_{i=0}^{M_1-1} \sum_{j=0}^{M_2-1} x[i, j] \cdot b[n_1 - i, n_2 - j], \end{aligned} \quad (2.7)$$

for $0 \leq n_1 \leq L_1 - 1$, $0 \leq n_2 \leq L_2 - 1$, where M_1, M_2 are the dimensions of $x[n_1, n_2]$, N_1, N_2 are the dimensions of $b[n_1, n_2]$, and $L_1 = N_1 + M_1 - 1$, $L_2 = N_2 + M_2 - 1$ are the dimensions of $y[n_1, n_2]$. The properties of the convolution operator defined in Equation (2.7) are straightforward extensions of the 1D linear convolution operator.

In two dimensions, circular convolution is given by

$$\begin{aligned} y[n_1, n_2] &= \sum_{i=0}^{N_1-1} \sum_{j=0}^{N_2-1} b[i, j] \cdot x[(n_1 - i)_{N_1}, (n_2 - j)_{N_2}] \\ &= \sum_{i=0}^{N_1-1} \sum_{j=0}^{N_2-1} x[i, j] \cdot b[(n_1 - i)_{N_1}, (n_2 - j)_{N_2}], \end{aligned} \quad (2.8)$$

for $0 \leq n_1 \leq N_1 - 1$, $0 \leq n_2 \leq N_2 - 1$.

■ 3. Linear Filtering

Many useful image processing operations may be implemented by filtering with a selected FIR filter [2, 3, 4]. This is known as linear filtering and finds application in edge detection, image sharpening, noise reduction, and image matching. According to Equations (2.2), (2.3), (2.7), and (2.8) the output image is obtained by a convolution operation. The choice of filter depends on the desired outcome. Here we demonstrate a number of standard FIR filters commonly used in operations such as blurring, unsharp masking, and edge detection.

We first consider so-called smoothing filters that return the local average of the samples of a signal. Such an operation tends to reduce the local variability of sample values and thus decreases signal noise and smoothes the signal. However, averaging also smears image edges resulting in a blurred picture. The simplest such filter returns a matrix with constant coefficients equal to the inverse of the filter's dimensions. This FIR filter is commonly known as a moving-average filter or box filter.

Here is a 3×3 moving-average filter.

```
In[6]:= (fir = 1/9. Table[1, {3}, {3}]) // MatrixForm
```

```
Out[6]//MatrixForm=
```

$$\begin{pmatrix} 0.111111 & 0.111111 & 0.111111 \\ 0.111111 & 0.111111 & 0.111111 \\ 0.111111 & 0.111111 & 0.111111 \end{pmatrix}$$

The averaging property of the moving-average filter may be verified by substituting for $b[i, j]$ in Equation (2.8) the values of the filter. It is easily seen that the output $y[n, m]$ is equal to an average of the sample values in the nine-element neighborhood of sample $x[n, m]$.

$$\begin{aligned} y[n_1, n_2] &= \frac{1}{9} \sum_{i=0}^2 \sum_{j=0}^2 x[n_1 - i, n_2 - j] \\ &= \frac{1}{9} (x[n_1, n_2] + x[n_1, n_2 - 1] + \cdots + x[n_1 - 2, n_2 - 2]) \end{aligned} \quad (3.1)$$

Averaging operations are useful in removing noise from a signal, however, they also tend to blur the signal, due to smoothing of sharp transitions in image intensity (i.e., edges).

This loads an example image.

```
In[7]:= (img = Import[ToFileName[{$HomeDirectory}, "Books.tif"]];
         books = img[[1, 1]]);
```

This sets some useful display options for **ListDensityPlot**.

```
In[8]:= SetOptions[ListDensityPlot, Mesh -> False, FrameTicks -> False,
                 ImageSize -> Dimensions[books] + 32, AspectRatio -> Automatic];
```

This shows the example image.

```
In[9]:= ListDensityPlot[books];
```

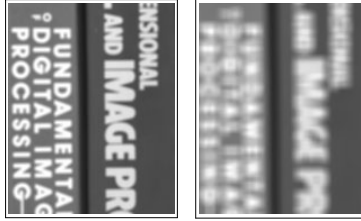
From In[9]:=



Here we demonstrate image smoothing with two moving-average filters of different dimensions. Note that the degree of smoothing is directly related to the dimensions of the moving-average filter.

```
In[10]:= Show[
  GraphicsArray[
    ListDensityPlot[#, DisplayFunction -> Identity] & /@
    {
      ListConvolve[fir, books],
      ListConvolve[
        1/81. Table[1, {9}], {9}], books
    ]
  ]];
```

From In[10]:=



A standard technique to improve image quality is to enhance the visibility of edges by using a technique called unsharp masking [2]. Unsharp masking introduces overshoots in the vicinity of an edge, effectively simulating a well-documented human visual phenomenon known as the Mach band effect. Mach bands are nonexistent overshoots that we tend to perceive in the vicinity of sharp brightness transitions in a picture. This linear operation may be implemented by convolving an image with an unsharp filter. In unsharp masking, the original image is blurred and a fraction of the unsharp image is subtracted from the original. This can be expressed as

$$g[n, m] = a f[n, m] - b f_L[n, m] \quad (3.2)$$

where $f[n, m]$ is the original image and $f_L[n, m]$ is the lowpass filtered image, a and b are positive constants with $a > b$, and $g[n, m]$ is the processed image. Here are some typical impulse responses of unsharp masking filters.

$$\frac{1}{9} \begin{pmatrix} -1 & -1 & -1 \\ -1 & -1 + 9a & -1 \\ -1 & -1 & -1 \end{pmatrix}, \begin{pmatrix} 1 & -2 & 1 \\ -2 & 5 & -2 \\ 1 & -2 & 1 \end{pmatrix}, \frac{1}{7} \begin{pmatrix} -1 & -2 & -1 \\ -2 & 19 & -2 \\ -1 & -2 & -1 \end{pmatrix} \quad (3.3)$$

The parameter α is a real constant in the range $0.5 \leq \alpha \leq 1.5$. Here we define a 5×5 unsharp filter.

```
In[11]:= unsharp = ReplacePart[-Table[1, {5}], {5}] / 25., 2. - 1/25., {3, 3}];
```

Here we demonstrate unsharp masking. First we blur the original **books** image.

```
In[12]:= tmp = ListConvolve[fir, books];
```

Here we show the result. Note the somewhat improved quality of the unsharp masked image compared to the blurred copy.

```
In[13]:= Show[
  GraphicsArray[ListDensityPlot[#, DisplayFunction -> Identity] & /@
    {tmp, ListConvolve[unsharp, tmp]}]];

```

From In[13]:=



Edge detection is a common image processing task [2, 3, 4]. It is well known that edges are important to mammalian vision. In typical images, edges characterize object boundaries and therefore are useful for segmentation, registration, and identification of objects in a scene. The most commonly used edge detection methods are implemented as linear filtering operations with FIR filters that approximate directional derivatives. Specifically, the so-called gradient magnitude of some continuous function $f(x, y)$ is defined as

$$|\nabla f(x, y)| = \sqrt{(\partial_x f(x, y))^2 + (\partial_y f(x, y))^2} \quad (3.4)$$

The orientation of an edge is sometimes of interest. The gradient orientation is given by

$$\angle \nabla f(x, y) = \text{ArcTan}(\partial_y f(x, y) / \partial_x f(x, y)) \quad (3.5)$$

Local maxima of the gradient magnitude identify edges in $f(x, y)$. In the case of a digital image, the derivatives $\partial_x f$ and $\partial_y f$ are easily obtained by convolving the image with two FIR differentiators. Here is one frequently used filter pair known as the Sobel edge detector.

```
In[14]:= MatrixForm /@ {sobel = {{-1, 0, 1}, {-2, 0, 2}, {-1, 0, 1}},
  {{-1, -2, -1}, {0, 0, 0}, {1, 2, 1}} / 4.}

```

$$\text{Out[14]} = \left\{ \begin{pmatrix} -0.25 & 0 & 0.25 \\ -0.5 & 0 & 0.5 \\ -0.25 & 0 & 0.25 \end{pmatrix}, \begin{pmatrix} -0.25 & -0.5 & -0.25 \\ 0 & 0 & 0 \\ 0.25 & 0.5 & 0.25 \end{pmatrix} \right\}$$

This returns the two-directional derivatives of the example image using the Sobel filters.

```
In[15]:= {gc = ListConvolve[sobel[[1]], books];
  gr = ListConvolve[sobel[[2]], books];}

```

Here we compute the gradient magnitude and edge direction.

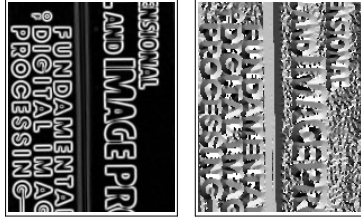
```
In[16]:= {mag = Sqrt[gr^2 + gc^2];
  dir = ArcTan[gr + 0.0001, gc];}

```

This displays the edge magnitude and the orientation.

```
In[17]:= Show[GraphicsArray[
  ListDensityPlot[#, DisplayFunction -> Identity] & /@ {mag, dir}]];
```

From In[17]:=



■ 4. Binary and Grayscale Morphological Operators

Mathematical morphology provides an approach to the processing of digital images that is based on the spatial structure of objects in a scene [2, 4, 5, 6, 7]. Unlike the linear operators discussed so far, morphological operators modify the shape of pixel groupings instead of their amplitude. The fundamental operators of binary morphology are erosion and dilation. These operators are typically defined using set-theoretic concepts, but here we restrict our discussion to their matrix formulations. In analogy with the linear operators discussed in the previous section, binary morphological operators may be implemented using convolution-like algorithms, with the fundamental operations of addition and multiplication replaced by logical OR and AND.

To simplify the notation and without loss of generality we assume all arrays have square dimensions. With A as an $N \times N$ binary-valued image, B as the typically small structuring element of dimensions $2M + 1 \times 2M + 1$, and $C = A \oplus B$ as the dilation result, the matrix formulation is as follows:

$$C[n_1, n_2] = \bigcup_{i=-M}^M \bigcup_{j=-M}^M B[i, j] \cap A[n_1 - i, n_2 - j] \quad (4.1)$$

with $0 \leq n_1, n_2 \leq N - 1$, and where $\bigcup$ and $\cap$ are the logical OR (union) and logical AND (intersection) operators, respectively. For the special case of a 3×3 structuring element ($M = 1$) we can write Equation (4.1) explicitly as

$$\begin{aligned} C[n_1, n_2] = & (B[-1, -1] \cap A[n_1 + 1, n_2 + 1]) \cup \\ & (B[-1, 0] \cap A[n_1 + 1, n_2]) \cup (B[-1, 1] \cap A[n_1 + 1, n_2 - 1]) \\ & \bigcup (B[0, -1] \cap A[n_1, n_2 + 1]) \cup (B[0, 0] \cap A[n_1, n_2]) \cup \\ & (B[0, 1] \cap A[n_1, n_2 - 1]) \\ & \bigcup (B[1, -1] \cap A[n_1 - 1, n_2 + 1]) \cup \\ & (B[1, 0] \cap A[n_1 - 1, n_2]) \cup (B[1, 1] \cap A[n_1 - 1, n_2 - 1]) \end{aligned} \quad (4.2)$$

Equations (4.1) and (4.2) resemble the familiar convolution sum formulation with OR replacing addition and AND replacing multiplication. Thus the structuring element plays an analogous role in morphological signal processing to the

FIR filter in LSI signal processing. This realization leads quickly to an implementation of erosion using **ListConvolve**.

Here we define an example array A .

$$\text{In}[18]:= \mathbf{A} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix};$$

The matrix formulation in Equation (4.1) assumes an odd-sized structuring element with the origin centered in the array. Therefore, we define a 3×3 array B with an L-shaped arrangement of 1s as follows.

$$\text{In}[19]:= \mathbf{B} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix};$$

The binary dilation of image A by structuring element B , namely $A \oplus B$, is given by the following.

```
Out[20]:= ListConvolve[B, A, {2, 2}, 0, BitAnd, BitOr] // MatrixForm
```

```
Out[20]//MatrixForm=
```

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Erosion is a morphological dual to dilation. The structuring element may again be viewed as a morphologic filter that is scanned across an image. As in the case of dilation, erosion may be defined by the following correlation-like formulation. The erosion of image A with structuring element B , denoted $C = A \ominus B$, has the following matrix formulation:

$$C[n_1, n_2] = \bigcap_{i=-M}^M \bigcap_{j=-M}^M \overline{B}[i, j] \cup A[n_1 + i, n_2 + j] \quad (4.3)$$

with $0 \leq n_1, n_2 \leq N-1$. Operators $\cup$ and $\cap$ are the logical OR (union) and logical AND (intersection) operators, respectively, and $\overline{B}$ is the point-wise complement of structuring element B . In effect, erosion results in a value of 1 at position (r, c) in C if the spatial arrangement of 1s in B is a subset of the arrangement of 1s in A . In particular, for the special case of a 3×3 structuring element ($M = 1$) of all 1s, we can write Equation (4.3) explicitly as

$$\begin{aligned} C[n_1, n_2] = & (\overline{B}[-1, -1] \cup A[n_1 - 1, n_2 - 1]) \cap \\ & (\overline{B}[-1, 0] \cup A[n_1 - 1, n_2]) \cap (\overline{B}[-1, 1] \cup A[n_1 - 1, n_2 + 1]) \cap \\ & (\overline{B}[0, -1] \cup A[n_1, n_2 - 1]) \cap (\overline{B}[0, 0] \cup A[n_1, n_2]) \cap \\ & (\overline{B}[0, 1] \cup A[n_1, n_2 + 1]) \cap (\overline{B}[1, -1] \cup A[n_1 + 1, n_2 - 1]) \cap \\ & (\overline{B}[1, 0] \cup A[n_1 + 1, n_2]) \cap (\overline{B}[1, 1] \cup A[n_1 + 1, n_2 + 1]) \end{aligned} \quad (4.4)$$

Here is the binary erosion of array A by structuring element B , namely $A \ominus B$.

```
In[21]:= ListCorrelate[1 - B, A, {2, 2}, 1, BitOr, BitAnd] // MatrixForm
```

```
Out[21]//MatrixForm=
```

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Here we demonstrate the operations of binary erosion and dilation. First we threshold the example **books** image using a threshold value of $t = 165$ to obtain a binary array.

```
In[22]:= tmp = Map[If[# < 165, 0, 1] &, books, {2}];
```

This shows the original and the binary images.

```
In[23]:= Show[GraphicsArray[
  ListDensityPlot[#, DisplayFunction -> Identity] & /@ {books, tmp}]];
```

From In[23]:=



This defines a square structuring element.

```
In[24]:= S = {{0, 0, 0}, {0, 1, 1}, {0, 1, 1}};
```

Here we show the results of dilation and erosion.

```
In[25]:= (dila = ListConvolve[S, tmp, {2, 2}, 0, BitAnd, BitOr] ;
erod = ListCorrelate[1 - S, tmp, {2, 2}, 1, BitOr, BitAnd] ;
Show[
  GraphicsArray[ListDensityPlot[#, DisplayFunction -> Identity] & /@
    {erod, dila}]];
```

From In[25]:=



Note how dilation increases the white regions in the image, hence the use of the word dilation for this operator. The operation of erosion, on the other hand, decreases or shrinks the area occupied by white pixels.

Binary morphological concepts may be extended to grayscale images. The common approach is to use a grayscale extremum operation over a small neighbor-

hood defined by the structuring element. A typically small structuring element is scanned over an image and, at each position, the maximum of point-by-point sums of the image and the structuring element is computed. This may again be implemented as a convolution-like operation. For notational convenience we assume arrays with square dimensions. Then, given a $2M+1 \times 2M+1$ structuring element B and an $N \times N$ image array A , the dilation of A by B may be defined as

$$C[n_1, n_2] = \text{Max}_{i,j} [A[n_1 - i, n_2 - j] + B[i, j]]. \quad (4.5)$$

for all i, j such that $-M \leq i, j \leq M$, with $0 \leq n_1, n_2 \leq N-1$. For example, with $M=1$ and $B[i, j] = 0$ for all i, j , we get

$$\begin{aligned} C[n_1, n_2] = \text{Max} [& \{A[n_1+1, n_2+1], A[n_1+1, n_2], \\ & A[n_1+1, n_2-1], A[n_1, n_2+1], A[n_1, n_2], A[n_1, n_2-1], \\ & A[n_1-1, n_2+1], A[n_1-1, n_2], A[n_1-1, n_2-1]\}] \end{aligned} \quad (4.6)$$

Dilation tends to grow the white regions of an image. If the structuring element has positive values, the resulting image tends to be brighter. Grayscale erosion is the dual of grayscale dilation; the structuring element values are subtracted from the image during the scanning process and a minimum of each set of values is returned. This may be implemented as a correlation-like operation where the extremum operator **Min** and subtraction replace multiplication and addition, respectively. Given a $2M+1 \times 2M+1$ structuring element B and an $N \times N$ image array A , the dilation of A by B may be defined as

$$C[n_1, n_2] = \text{Min}_{i,j} [A[n_1 + i, n_2 + j] - B[i, j]] \quad (4.7)$$

for all i, j such that $-M \leq i, j \leq M$, with $0 \leq n_1, n_2 \leq N-1$. Erosion tends to shrink the white regions of an image. If the structuring element has positive values, the resulting image tends to be darker.

Here we show an example of a morphological gradient operator. The morphological boundary $\beta(A)$ of image A (i.e., the edges) is defined as

$$\beta(A) = (A \oplus B) - (A \ominus B) \quad (4.8)$$

In Equation (4.8) symbols $\ominus$ and $\oplus$ are used to represent grayscale erosion and grayscale dilation, respectively.

This defines a uniform 3×3 structuring element.

```
In[26]:= S = {{1, 1, 1}, {1, 1, 1}, {1, 1, 1}};
```

Here is the result of a morphological gradient operation on the example image using a uniform structuring element of 1s. We use **ListConvolve** and **ListCorrelate**: **late** to implement grayscale dilation and erosion, respectively.

```
In[27]:= tmp = ListConvolve[S, books, {2, 2}, 0, Plus, Max] -  
ListCorrelate[-S, books, {2, 2}, 255, Plus, Min];
```

This shows the original image and its morphological gradient.

```
In[28]:= Show[GraphicsArray[
  ListDensityPlot[#, DisplayFunction -> Identity] & /@ {books, tmp}]];
```

From In[28]:=



■ References

- [1] A. V. Oppenheim, A. S. Willsky, and S. H. Navab, *Discrete Time Signal Processing*, Englewood Cliffs, NJ: Prentice Hall, 1989.
- [2] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, Reading, MA: Addison-Wesley, 1992.
- [3] A. K. Jain, *Fundamentals of Digital Image Processing*, Englewood Cliffs, NJ: Prentice Hall, 1989.
- [4] W. K. Pratt, *Digital Image Processing*, 2nd ed., New York: John Wiley & Sons, 1991.
- [5] R. M. Haralick, S. R. Sternberg, and X. Zhuang, "Image analysis using mathematical morphology," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **PAMI-9**, 1987 pp. 532–550.
- [6] R. M. Haralick and L. G. Shapiro, *Computer and Robot Vision*, Vol. 1, Reading, MA: Addison-Wesley, 1992.
- [7] J. Serra, *Image Analysis and Mathematical Morphology*, New York: Academic Press, 1982.

Classifieds

Classified advertising is provided for *Mathematica* users. Rates are available at www.mathematica-journal.com/advertising.

Industrial Electromagnetism is a *Mathematica* application package, designed for performing analyses in all fields of electrodynamics. It provides innovative techniques for solving electromagnetic problems. You can create, change, and optimize parametric mathematical models quickly and intuitively.

More info: www.wolfram.com/products/applications/electromagnetism

Industrial Optimization is a *Mathematica* application package, designed for solving a wide range of optimization problems. It provides well-tried optimization algorithms as well as modern techniques. Each function can additionally be customized by various options.

More info: www.wolfram.com/products/applications/optimization

Custom Calculus Solutions Instantly. Automatic step-by-step derivatives and integrals with each step explained. Long multiplication and division of polynomials, too.

More info: www.calc101.com

Actuarial Models and Interactive Modeling with ACTEX. ACTEX is a series of study guides used to prepare for the Society of Actuaries (SOA) exams. ACTEX is the only approved interactive study guide for the SOA exams. ACTEX is a reference on Actuarial Models and the Modeling Process. And, ACTEX is powered by *Mathematica*.

More info: www.actexamdriver.com

MathRing is a network of *Mathematica*-related web sites allowing users to share information, tips, notebooks, and more. Commercial sites and academic sites are welcome. Anyone who has created a *Mathematica*-related web page that they would like to share with other *Mathematica* users can join MathRing.

More info: www.e-notebooks.com/rings/mathring

arXiv.org, the e-print server hosted by Los Alamos National Laboratory, now accepts *Mathematica* notebook submissions. Authors can submit notebooks as supplemental materials or as an entire article. arXiv.org is a fully automated electronic archive and distribution server for research papers in physics and related disciplines, including mathematics, nonlinear sciences, computational linguistics, and neuroscience. In the year 2000, an estimated 13 million preprints were downloaded from arXiv and its mirror sites.

More info: www.wolfram.com/solutions/publishing/guidearxiv.html

Article Submissions to *The Mathematica* *Journal*

The Mathematica Journal publishes articles and notes on all aspects of *Mathematica* and related subjects. Its goal is to inform and challenge the community of *Mathematica* users and to enable readers at all levels of proficiency to use *Mathematica* more effectively. A high standard of *Mathematica* programming, using clarity and conciseness in writing, is valued. *The Mathematica Journal* reserves the right to edit contributions, with changes subject to the authors' approval.

The Mathematica Journal also publishes *Mathematica* notebooks and packages, as well as programs in other languages that can be used to complement *Mathematica*. All programs are accompanied by a note or article. Programs distributed through the journal can be used freely and duplicated for non-commercial purposes.

Some of the categories of material published in *The Mathematica Journal* are listed below.

- Reports of applications of *Mathematica* to particular fields in education, research, or business. The level of exposition should be such that readers from other fields can grasp the essence of the application, the reasons why *Mathematica* is being used, and the possible relevance to other fields of the techniques being employed.
- Descriptions of programs—*Mathematica* packages and notebooks, or *Mathematica*-related programs in other languages—whether or not accompanied by the code itself. The programs should be exemplary in their implementation and thoroughly tested.
- *Mathematica* programming advice. This can range from notes about little-known programming tricks to tutorials focusing on a general problem or technique. The main test of applicability here is how many readers are likely to benefit from the ideas, but even a relatively esoteric application can aid in the teaching of good programming principles.
- Discussion of issues beyond *Mathematica*. Essays, columns and surveys on subjects that do not involve *Mathematica* directly may be considered, if they are likely to be of general interest to the *Mathematica* community. Possible subjects include symbolic computation, numerical computation,

computer-based education, computer graphics, computer art, and so forth. Articles of this type should be particularly well written and well argued, and their relevance to the *Mathematica* community should be clear.

- General informational articles, profiles of exemplary *Mathematica* users or applications, interviews, and news related to the *Mathematica* community will also be sought for publication online on a frequent basis.

To submit a contribution, send the material—together with a cover letter stating that it is being submitted for publication in *The Mathematica Journal*—in one of the following formats:

Hard Copy and/or 3.5" Diskettes by Mail

Send to:

Editor

The Mathematica Journal

PO Box 6359

Champaign, IL 61826-6359 USA

Articles, packages, graphics, notebooks and programs submitted for publication can be submitted in Macintosh or DOS format.

Submissions will not be returned. Do not send proprietary or confidential information.

Email

Articles and other material may be sent by email to editor@mathematica-journal.com. Do not email articles or attachments that exceed 1MB.

Web

Articles, and graphics, and other files may be placed on the Internet for downloading by *The Mathematica Journal*.

■ Manuscript Submission Guidelines

Manuscripts must be in English. Manuscripts should be submitted as properly formatted notebooks made with *Mathematica* Version 3 or later. Notebooks should make good use of text, sections, and inline and displayed equations (if required), and should use the *Mathematica* style sheet “TMJv4Style.nb”. This style sheet is available at www.mathematica-journal.com/submissions/, together with an authoring palette.

Manuscripts should present a significant application of *Mathematica*. Manuscripts under consideration by or submitted simultaneously to other journals will not be considered.

Remember to include:

1. the title. If the title has more than 50 characters, submit an alternative short title;
2. your name(s), affiliation, postal and electronic mail address;
3. an abstract of at most 150 words.

The remaining material should be included in the following order:

1. text of the article or note;
2. footnotes in order of citation (avoid footnotes if possible);
3. references (see below);
4. a list of additional electronic material, including, if appropriate, information on the necessary environment (machine, operating system, and so on);
5. a biography of each author, limited to 60 words per author, and 180 words overall.

References should be numbered in the text in order of appearance. In the listing at the end of the article, each reference should include full information: author or institution, full title, publisher, city, and year (for books, manuals, etc.); or full journal name, volume, year, and page range (for papers). References to software and hardware should contain complete manufacturer's or distributor's names and addresses. See format in a current issue.

All references in the bibliography should be cited in the text, or accompanied by comments stating their relevance.

Avoid footnotes if possible. If used, they should be numbered sequentially and clearly marked in the text.

Please observe the following guidelines for text and embedded code:

- Use of the word *Mathematica* should be kept to a minimum, since the reference is almost always clear from the context. It is especially discouraged in titles.
- Technical terms unlikely to be known to non-specialists should be italicized the first time they are used, and, unless the first use is self-explanatory, they should be defined or explained then. Italics should not be used for emphasis, except very occasionally.
- *Mathematica* words, symbols or expressions should appear in a font distinct from that used for text. The use of a fixed-pitch font, in which all characters have the same width, is strongly encouraged. Use the author's palette and *The Mathematica Journal* style sheet.
- References to *Mathematica* functions should not be followed by `[]`, unless what is meant is that the function is being invoked with no arguments.

- Lines of displayed code, whether from packages or interactive sessions, should be no longer than 60 characters.
- Do not use *Mathematica* comments in interactive sessions; instead, alternate explanatory text with input/output pairs. It is especially important to explain the commands in a long interactive session as you go along, rather than all at once at the beginning or at the end of the session. As a rule, notebooks should be cast into this same format: explanatory text alternating with code.
- Always mention which version of *Mathematica* you are running.

■ Guidelines for *Mathematica* Packages

Packages are programs written in *Mathematica*, to be used on their own or integrated with the user's code.

The single most important consideration in designing a *Mathematica* package for distribution is that it must be readily usable by others. Clarity is the goal and a good test of the clarity of your design is the ease with which it can be documented. Ideally, the purpose and use of each symbol visible to the user should be simple enough to explain in its usage string.

The more simply your design can be described, the more likely people are to be able to understand and use what you have built. In addition, such designs are often more general and therefore usable in a wider range of circumstances.

The book *Programming in Mathematica*, Third Edition, by Roman Maeder (Addison-Wesley, 1996) provides good pointers on programming style.

