

The *Mathematica*[®] Journal

Volume 8, Issue 3

Contents

Departments

Cover Image <i>by Sándor Kabai</i>	354
Staff and Contributors	355
Calendar of Events	356
News Bulletins	360
New Products	362
New in the Library	367
New Publications	368
Editor's Pick	375
Article Submissions to <i>The Mathematica Journal</i>	484
Classifieds	488

Tutorials

Tricks of the Trade <i>edited by Paul Abbott</i>	383
In and Out <i>edited by Paul Abbott</i>	396

Columns

Trott's Corner <i>by Michael Trott</i>	409
--	-----

Feature Articles

Leibniz: Implementing a Drag-and-Drop Calculator <i>by Joe Gregg</i>	433
Quantum Turing Machine Simulator <i>by Joachim Hertel</i>	440
Real Quaternions and Rotations <i>by Robert Piziak and Danny W. Turner</i>	458
Orthogonal Series Representation of Probability Density and Distribution Functions <i>by J. Todd Reinking</i>	473

The electronic version of this issue is available at www.mathematica-journal.com/issue/v8i3.

The online edition of *The Mathematica Journal* includes downloadable versions of all articles and packages in this issue.

Cover Image

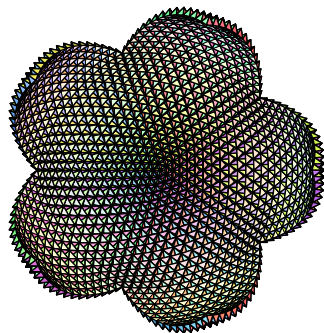
Sándor Kabai

3 Honvéd, Püspökladány
Hungary-4150
email: uniko@axelero.hu

■ Chestnut Shell

```
In[1]:= << Graphics`Polyhedra`  
In[2]:= << Graphics`Shapes`  
In[3]:= Show[  
  Graphics3D[Partition[Stellate[Geodesate[Icosahedron[], 15], 1.05],  
    675][[Range[11, 20]]], Boxed -> False, ViewPoint -> {0, 0, 2},  
  LightSources -> {{1, 0, 1}, RGBColor[1, 0.4, 0.4]}, {-1, 1, 1},  
    RGBColor[0.3, 1, 0.1]}, {-1, -1, 1}, RGBColor[0.2, 0.2, 0.9]}}];
```

From In[3]:=



About the Author

Sándor Kabai is a mechanical engineer specializing in aviation and manufacturing engineering. He is also a member of the Association of Users of Visualisation Software, Geometry and Space Research Team, an interdisciplinary group hosted by various universities in Hungary. Some of Mr. Kabai's work can be viewed at the following web sites.
members.tripod.com/vismath5/kabai/index.html
turing.mi.sanu.ac.yu/vismath/kabai1/index.html

Staff and Contributors

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

Managing Editor: Joe Grohens

Associate Editor: Greg Springer

Project Manager: Louise Holubek

Design Team: Jeremy Davis, John Lee, Al Navata

Compositor: Glenn Scholebo

Technical Specialist: Hormozd Gahvari

Archives and Web Production: Jean Buck

Web Development: Ankur Naik

Editorial Staff: George Beck, Catherine Brassac, Marian Peden

Customer Service: Matt Doman, Cheryl Miller

Columnists: Paul Abbott, Roman Maeder, Michael Trott

Contributors for This Issue: Joe Gregg, Joachim Hertel, Robert Piziak, J. Todd Reinking, Danny W. Turner

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 website 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.

© 2002 Wolfram Media, Inc. All rights reserved. No part of this publication may be reprinted or otherwise reproduced without the express permission of its publisher. No copyright claimed for U.S. government materials.

Mathematica, *MathLink*, and *MathSource* are registered trademarks of Wolfram Research, Inc. *Mathematica* is not associated with Mathematica Policy Research, Inc. or MathTech, Inc. All other product names mentioned are trademarks of their producers. Material in this publication 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.

February

February 5, 2002

Introduction to *Mathematica*

CANdienst

Amsterdam, Netherlands

Info: www.candienst.nl/english/cursussen/cursussendetail.asp?id=8

February 7–8, 2002

Numerical Methods for Pricing Financial Derivatives

CANdienst

Amsterdam, Netherlands

Info: www.candienst.nl/english/cursussen/cursussendetail.asp?id=25

February, 2002

A First Course in *Mathematica*

Wolfram Education Group

February 11–12, San Francisco, CA, USA

February 25–26, London, England

February 25–26, San Diego, CA, USA

Info: www.wolfram.com/services/education/courses/calendar.html

February 19–22, 2002

Hyper-Math: *Mathematica* for the Scientist and Engineer

Sigma Solutions

Tucson, AZ, USA

Info: www.sigmasolns.com/mathematica.html

February 25, 2002

webMathematica

CANdienst

Amsterdam, Netherlands

Info: www.candienst.nl/english/cursussen/cursussendetail.asp?id=35

February 26, 2002

***Mathematica* Introductory Training**

COMSOL AG

Zürich, Switzerland

Info: www.comsol.ch/content.php?si=19

February 26–27, 2002

Mathematica Intermediate Course

CANDiensten

Amsterdam, Netherlands

Info: www.candiensten.nl/english/cursussen/cursussendetail.asp?id=9

February 27–28, 2002

Advanced Training with Mathematica

COMSOL AG

Zürich, Switzerland

Info: www.comsol.ch/content.php?si=20

March

March, 2002

A First Course in Mathematica

Wolfram Education Group

March 4–5, Orlando, FL, USA

March 11–12, New Orleans, LA, USA

March 28–29, Austin, TX, USA

Info: www.wolfram.com/services/education/courses/calendar.html

March 12, 2002

Introduction to Mathematica

CANDiensten

Amsterdam, Netherlands

Info: www.candiensten.nl/english/cursussen/cursussendetail.asp?id=8

March 19–22, 2002

Hyper-Math: Mathematica for the Scientist and Engineer

Sigma Solutions

Tucson, AZ, USA

Info: www.sigmasolns.com/mathematica.html

April

April, 2002

A First Course in Mathematica

Wolfram Education Group

April 1–2, Boston, MA, USA

April 8–9, Seattle, WA, USA

April 15–16, New York, NY, USA

April 22–23, Washington, DC, USA

Info: www.wolfram.com/services/education/courses/calendar.html

April 11–12, 2002

Financial Modeling with *Mathematica*

COMSOL AG and Weber & Partner

Zürich, Switzerland

Info: www.weber-und-partner.com/training/finance/finmod/index.htm

April 17, 2002

Introduction to *Mathematica*

CANDiensten

Amsterdam, Netherlands

Info: www.candiensten.nl/english/cursussen/cursussendetail.asp?id=8

April 18–19, 2002

Financial Modeling with *Mathematica*

CANDiensten

Amsterdam, Netherlands

Info: www.candiensten.nl/english/cursussen/cursussendetail.asp?id=17

May

May, 2002

A First Course in *Mathematica*

Wolfram Education Group

May 2–3, Albuquerque, NM, USA

May 6–7, Portland, OR, USA

May 13–14, Baltimore, MD, USA

May 20–21, Chicago, IL, USA

May 20–21, San Francisco, CA, USA

May 29–30, Denver, CO, USA

Info: www.wolfram.com/services/education/courses/calendar.html

May 14, 2002

Introduction to *Mathematica*

CANDiensten

Amsterdam, Netherlands

Info: www.candiensten.nl/english/cursussen/cursussendetail.asp?id=8

May 28–29, 2002

***Mathematica* Intermediate Course**

CANDiensten

Amsterdam, Netherlands

Info: www.candiensten.nl/english/cursussen/cursussendetail.asp?id=9

June

June 11–12, 2002

Mastering *Mathematica*

Ritme Informatique

Paris, France

Info: www.ritme.com/formations/mathematica

June 13, 2002

Perfecting *Mathematica*

Ritme Informatique

Paris, France

Info: www.ritme.com/formations/mathematica

June 18, 2002

Introduction to *Mathematica*

CANdienst

Amsterdam, Netherlands

Info: www.candienst.nl/english/cursussen/cursussendetail.asp?id=8

June 27–28, 2002

A First Course in *Mathematica*

Wolfram Education Group

Atlanta, GA, USA

Info: www.wolfram.com/services/education/courses/calendar.html

News Bulletins

View more *Mathematica* news at www.wolfram.com/news.

***Mathematica* for Mac OS X Featured in Jobs's Keynote at Macworld Expo**

Mathematica for Mac OS X was one of the exciting new products featured during the opening keynote address at the 2002 Macworld Conference and Expo held in San Francisco, California. The complete story is available at www.wolfram.com/news/macexpo2002.html.

Stephen Wolfram Interview in *New Scientist*

In a recent issue of *New Scientist* magazine, writer Marcus Chown interviewed Stephen Wolfram, the developer of *Mathematica*, about his ongoing work on the edge of scientific discovery. The complete interview is available at www.newscientist.com/opinion/opinterview.jsp?id=ns230516.

***MathWorld*—Back Online**

MathWorld is a free online encyclopedia of mathematics that Eric Weisstein developed for students, educators, math enthusiasts, and researchers. The site is hosted and sponsored as a public service by Wolfram Research. *MathWorld* can be viewed at mathworld.wolfram.com.

***Mathematica* Puts New "Spin" on Cricket Bowling Action**

Mathematica user and New Zealand physicist Rene Ferdinands from the University of Waikato has been using *Mathematica* to investigate the dynamics of the bowling action in the game of cricket. Read the whole story at www.wolfram.com/news/cricket.html.

Quilt Design by *Mathematica*

Kenzou Nakamura, author of several books on *Mathematica* in design, has organized an art exhibit in Tokyo to show tapestries and a quilt designed with the use of *Mathematica*. Information and sample images can be found at www.asahi-net.or.jp/~SI4K-NKMR/inpaku.

Confessions of a *Mathematica* Addict

Associate Professor John Browne of Swinburne University of Technology is a self-described “*Mathematica* addict.” Read the story at www.wolfram.com/news/johnbrowne.html.

Mathematical Functions Site Offers “Encyclopedic” Resource

Recently released, Wolfram Research’s Mathematical Functions site was created as a resource for the educational, mathematical, and scientific communities. It contains the world’s most encyclopedic collection of information about mathematical functions. The site address is functions.wolfram.com.

New Products

webMathematica Now Shipping: Unique Product Brings Computation to the Web

Wolfram Research, Inc. announces that web*Mathematica*, the revolutionary product that merges the computational power of *Mathematica* with the convenience and accessibility of the web, is now shipping.

web*Mathematica* provides the long sought-after ability to quickly and easily add interactive calculations to the web, enabling users to:

- Build custom websites that provide specialized calculations
- Compute, visualize, and analyze data using a web browser
- Deliver sophisticated courseware and publish interactive books
- Provide active functionality for technical documentation that can be instantly updated from a centralized location
- Integrate computation into existing enterprise infrastructure

“web*Mathematica* changes the paradigm for delivering technical computations,” says Lars Hohmuth, Product Manager for web*Mathematica* at Wolfram Research. “Using web*Mathematica*, companies can rapidly create customized applications and deploy them over the web, using just standard web tools and a working knowledge of *Mathematica*.”

web*Mathematica* has generated unprecedented interest during its beta test with banks, engineering firms, and other institutions starting to phase it into their enterprisewide systems. Testers from 23 countries participated in the test phase.

“web*Mathematica* satisfies our long-standing ambition to bring truly interactive calculations to the web,” says Conrad Wolfram, Director of Strategic and International Business Development. “Judging by the dramatic way in which testers are adopting the technology, I’d say it fulfills their needs too.”

According to Cheng Wu of eFunda, “We are extremely impressed with web*Mathematica*’s performance, especially its rock-solid stability. In the race of bringing world-class mathematics software to the web, Wolfram Research is way ahead of its competition.”

A number of universities have begun creating web*Mathematica* applications, showing that the release of web*Mathematica* will also have a large impact on the academic and educational communities. “*Mathematica* is the tool of choice at hundreds of schools and universities worldwide. Now they can create web-based interactive distance education courses and supplemental class materials that take

advantage of *Mathematica*'s functionality, providing a richer learning experience," adds Hohmuth.

web*Mathematica* is built on Java servlets, making it compatible with any web server, servlet engine, or application server that supports the Servlet 2.0 or later API and giving developers a broad choice of platforms and web solutions to work from.

"Our adoption of modern web standards as the basis for web*Mathematica* makes its integration into existing web infrastructures virtually seamless, not to mention future-proof," states Tom Wickham-Jones, Director of Kernel Development at Wolfram Research and chief developer of web*Mathematica*. "If you want to put interactive computations and visualizations on your website, there's no question that web*Mathematica* is the easiest way to do it."

More information about web*Mathematica*, including working web*Mathematica* applications, is available.

web*Mathematica* is initially available for Windows 95/98/Me/NT/2000, Intel-based Linux platforms, and Solaris, and other platforms will follow soon. It comes in two license configurations: web*Mathematica Professional*, intended for development and deployment of sophisticated intranet and internet sites, and web*Mathematica Amateur*, designed for those who wish to showcase their work over the web. Subscribers to Wolfram Research's Premier Service will receive complimentary copies of web*Mathematica Amateur*.

More info

www.wolfram.com/products/webmathematica

Mathematica for Mac OS X

Mathematica for Mac OS X is now shipping. This fully native version of *Mathematica* for Mac OS X from Wolfram Research, Inc. offers users significant speed gains and vastly improved stability compared to running *Mathematica* on earlier Macintosh operating systems.

Mathematica users and industry insiders have eagerly anticipated the release of this version, one of the first serious technical applications to take full advantage of the extensive new features and technological advances in Mac OS X. "*Mathematica* for Mac OS X behaves beautifully. It's efficient, purposeful software... certainly one of the world's greatest applications, running on the greatest platform available. The product reminds one of a sleek Russian mink," says Richard Crandall, Apple Distinguished Scientist.

"The Mach 3.0 kernel and Unix-like foundation of Mac OS X allow this version of *Mathematica* to far surpass older Macintosh versions of *Mathematica* in speed, scalability, and the ability to handle calculations requiring open-ended amounts of memory. Mac OS X is the first true workstation operating system deployed as a personal-computer operating system," says Theodore Gray, cofounder of Wolfram Research and chief architect of the *Mathematica* user interface.

“We have been running *Mathematica* for Mac OS X internally for three years now and have worked closely with Apple to optimize its performance,” adds Gray. “*Mathematica* 4.1 running on Mac OS X has the makings of a remarkable environment for scientific computing. It is a no-compromise combination of speed, stability, capability, and ease.”

Mathematica is the system that top professionals in industry, research, and education turn to when they need to perform demanding calculations. *Mathematica* 4.1, the current release, includes greatly enhanced symbolic differential equation solvers and dramatic speed increases for statistical functions.

Professionals are not the only ones who use *Mathematica*. Like the Macintosh, *Mathematica* is popular in schools and on college campuses around the world. Students in engineering, mathematics, and other technical fields use *Mathematica* to expand their knowledge and to do their most serious number crunching. With *Mathematica* running on Mac OS X, they will be able to use the latest in Apple technology to make the most of their educational opportunities.

Read more about *Mathematica* for Mac OS X on the Apple website.

More info

www.apple.com/macosx/applications/mathematica

***Mathematica* 4.1 Supports Windows XP**

Wolfram Research is pleased to announce that Microsoft Windows XP has been added to the list of supported platforms for *Mathematica* 4.1. “For months, I’ve been running *Mathematica* 4.1 on a daily basis under Windows XP. Our experience is that it is an excellent and reliable system for running *Mathematica*. We’ve discovered no problems specific to the shipping versions of Windows XP,” says John Fultz, Manager of Front End Technologies at Wolfram Research. “Our Quality Assurance department has given *Mathematica* 4.1 under Windows XP a full workout with very satisfactory results,” he added.

Future versions of *Mathematica* will take full advantage of a variety of new Windows XP features, including themes and expanded internationalization support. *Mathematica*’s platform-independent, leading-edge design makes it easily compatible with most new software and emerging technologies.

***CalculationCenter* 2 Available in Fully Native Japanese Edition**

The first update to *CalculationCenter*, Wolfram Research’s new midmarket technical calculation software, is now available in a fully localized Japanese edition, with the English edition due to be released next month.

Combining algebraic operations with numerics, graphics, and a technical word processor, *CalculationCenter* provides an affordable, easy-to-use environment for engineers, scientists, and business analysts to do their everyday work and puts an

end to the compromise between quality and simplicity. An innovative point-and-click interface concept, smart plotting, and unique InstantCalculators combine to make it as easy as hand calculation and spreadsheet methods, but with much broader functionality.

Initially released in March 2001, *CalculationCenter* has been highly rated by *Macworld*, *Desktop Engineering*, and *PC Plus*, among others, and was well received by users. Drawing on the proven, state-of-the-art technology used in developing *Mathematica*, Wolfram Research's world-leading technical software, "*CalculationCenter* offers the quality, accuracy, and reliability of *Mathematica*, but for the less technically demanding or more occasional user," says Conrad Wolfram, Director of Strategic Development at Wolfram Research and one of the product's creators.

Key to *CalculationCenter*'s appeal is its ability to get users calculating quickly. "One of our goals was for the average user to be up and running—doing real calculations—in as little as 10 minutes," adds Wolfram. Several unique features make this possible:

- InstantCalculators guide users through every step of their calculations in plain terms and remain in the document for repeated use and future reference.
- Smart plotting selects not only the range of a plot but also the plot type and works whether the input is data or functions. Inequality plotting is new in Version 2.0.
- Traditional math input automatically interprets standard mathematical notation, using heuristics to resolve ambiguous input.
- Improved input and error-correction technology in Version 2.0 automatically suggests, if *CalculationCenter* cannot match the input, which function a user meant to enter.

CalculationCenter is designed to include the everyday capabilities that users need, with an emphasis on practical calculations. "With *CalculationCenter* we anticipated users' time constraints as well as their computing needs," states Jon McLoone of Wolfram Research, *CalculationCenter*'s key developer. "*CalculationCenter* has a great feature-set. But for practical calculations, the interface is more important. By putting our effort into intuitive interface design, we've taken the effort out of technical computing for our users."

CalculationCenter 2 has been enhanced with added features such as broad new statistics capabilities and support for the latest math-on-the-web standards. Further quality enhancements, including speed improvements and new import and export formats, have also been implemented. For native speakers, *CalculationCenter* 2 *Japanese Edition* offers a fully localized Japanese user interface—including menus, dialog boxes, and error and warning messages—and full translations of the accompanying online and print documentation. These features make it easy for Japanese users to work and create entire documents in the language with which they are most comfortable.

Containing hundreds of functions, many of them working with symbolic as well as numeric input and all with InstantCalculator support, *CalculationCenter* is the only tool needed to easily produce superior-quality technical reports that can be distributed by hard copy, via email, or on the web. Also ideal for organizations, *CalculationCenter* enables users of varying needs to share compatible documents and maintain the notebook document presentation style consistent with other Wolfram Research products.

New Mathematica Applications

UnRisk 1.5

Get fast, accurate pricing of derivatives in *Mathematica*.

More info

www.wolfram.com/products/applications/unrisk

Mathematica for ActiveX 1.0

This new product integrates *Mathematica* with Visual Basic environments.

More info

www.wolfram.com/products/applications/max

VAR+ 4.0

VAR+ 4.0 is a simulation-based software for risk analysis and strategic planning.

More info

www.wolfram.com/products/applications/var

MathModelica 2.0

Redefine desktop simulation with MathModelica 2.0.

More info

www.wolfram.com/products/applications/mathmodelica

Ruby/Mathematica

Ruby/Mathematica is a library for Ruby that provides an interface to the *MathLink* library.

More info

narayama.aist-nara.ac.jp/~daisu-ik/ruby/mathematica

New in the Library

Mathematica Explorations

Thanks to web $Mathematica$ technology, you can get a taste for $Mathematica$'s capabilities and solve some practical problems on the web at library.wolfram.com/explorations.

Talks from 2001 Mathematica Developer Conference

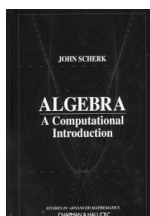
Details and transcriptions of select conference talks and presentations at the 2001 $Mathematica$ Developer Conference, held October 11–13 in Champaign, Illinois, are available online at library.wolfram.com/conferences/devconf2001.

MathSource

A number of new notebooks and packages have been added to $MathSource$.

- *Ray Tracing with Optica* (Henry Lamb, item 0211-633)
- *qmatrix: A Package for Quantum Information Theory* (Timo Felbinger, item 0211-644)
- *Graphical Analysis for Discrete Dynamical Systems* (Thomas LoFaro, item 0211-655)
- *Making Dimensionless Models* (Anton Antonov, item 0211-666)
- *Approximate Symbolic Solution of One-dimensional Integral Equation of the Second Kind* (Anry Nersessian, item 0211-677)
- *Ruby/Mathematica—Ruby Interface to MathLink* (Ikegami Daisuke, item 0211-688)
- *Spherical Density/Contour Plot* (Florian Berger, item 0211-699)
- *A Package of Bootstrap Algorithms for Mean, Simple Linear Regression Models and Correlation Coefficient* (Enis Siniksaran, item 0211-701)
- *A Monte Carlo Approach to the Permutation Test* (Enis Siniksaran, item 0211-712)
- *Graphics`Colors Palette* (Jack Calman, item 0211-723)
- *A package which defines a function that tries to find all roots of an equation between $xmin$ and $xmax$* (Ted Ersek, item 0211-734)

New Publications



Algebra: A Computational Introduction

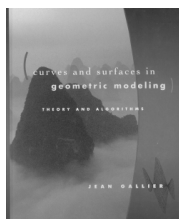
John Scherk, 2000, CRC Press, 319 pp., hardcover.

An algebra text for readers with a solid background in linear algebra. Each chapter builds on previous ones, from permutation groups, then linear groups, and finally abstract groups. *Mathematica* is integrated throughout the book, with numerous exercises and downloads available to provide references for complex problems encountered in the field.

Chapters include: Congruence | Permutations | Permutation Groups | Linear Groups | Groups | Subgroups | Symmetry Groups | Group Actions | Counting Formulas | Cosets | Sylow Subgroups | Simple Groups | Abelian Groups | Polynomial Rings | Symmetric Polynomials | Roots of Equations | Galois Groups | Quartics | The General Equation of the n th Degree | Solution by Radicals | Ruler-and-Compass Constructions | Appendix: *Mathematica* Commands

Available:

store.wolfram.com/view/book/ISBN1584880643.str



Curves and Surfaces in Geometric Modeling: Theory and Algorithms

Jean Gallier, 2000, Morgan Kauffman Publishers,
491pp., hardcover.

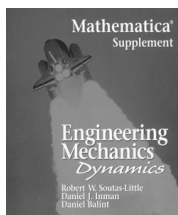
An introduction to geometric concepts and tools needed for solving problems of a geometric nature with a computer. It offers theoretical understanding of polynomial curves and surfaces. The focus of this book is on “blossoming”—the process of converting a polynomial to its polar form—as a natural, purely geometric explanation of the behavior of curves and surfaces. The book covers computer graphics and animation, robotics, virtual reality, geometric modeling and design, medical imaging, computer vision, and motion planning.

Chapters include: Introduction | Basics of Affine Geometry | Introduction to the Algorithmic Geometry of Polynomial Curves | Multiaffine Maps and Polar

Forms | Polynomial Curves as Bézier Curves | *B*-Spline Curves | Polynomial Surfaces | Subdivision Algorithms for Polynomial Surfaces | Polynomial Spline Surfaces and Subdivision Surfaces | Embedding an Affine Space in a Vector Space | Tensor Products and Symmetric Tensor Products | Appendices

Available:

store.wolfram.com/view/book/ISBN1558605991.str



Engineering Mechanics: Dynamics

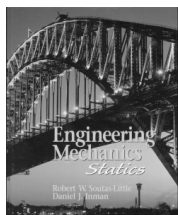
Robert W. Soutas-Little and Daniel J. Inman, 1999, Prentice Hall, 702 pp., hardcover.

An undergraduate textbook in mechanics that incorporates *Mathematica* to explore the basis of engineering design. Focuses on conceptually understanding mechanics and the transition from calculations to design concepts. The reader progresses from modeling free-body diagrams through writing the equations of motion and the constraint equations—and finally uses *Mathematica* to help solve numerical problems and investigate design parameters. The text enables the reader to solve problems for every angle or dimension.

Chapters include: Preface | Kinetics of a Particle | Kinetics of Particles | Work—Energy and Impulse—Momentum First Integrals of Motion | System of Particles | Kinematics of Rigid Bodies | Dynamics of Rigid Bodies in Plane Motion | Power, Work, Energy, Impulse, and Momentum of a Rigid Body | Three-Dimensional Dynamics of Rigid Bodies | Vibration | Appendix A: Mass Moment of Inertia | Appendix B: Vector Calculus and Ordinary Differential Equations | Dynamics Index Dictionary | Answers to Odd-Numbered Problems

Available:

store.wolfram.com/view/book/ISBN0132784092.str



Engineering Mechanics: Statics

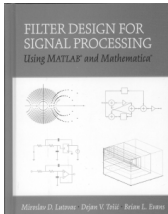
Robert W. Soutas-Little and Daniel J. Inman, 1999, Prentice Hall, 702 pp., hardcover.

Undergraduate text that focuses on the conceptual understanding of the mechanics of rigid bodies at rest or moving at constant velocity. *Mathematica* is used to model the concepts of forces and moments. Vector manipulations and matrix notation in statics are introduced and solved with *Mathematica*. Includes approximately one thousand problems and 95 worked sample problems.

Chapters include: Introduction | Vectors and Force Systems Acting on a Particle | Rigid Bodies: Equivalent Force Systems | Distributed Forces: Centroids and Center of Gravity | Equilibrium of Rigid Bodies | Analysis of Structures | Internal Forces in Structural Members | Friction | Moments of Inertia | Virtual Work | Appendix | Glossary | Problem Statements | Index

Available:

store.wolfram.com/view/book/ISBN0137690010.str



Filter Design for Signal Processing using MATLAB and *Mathematica*

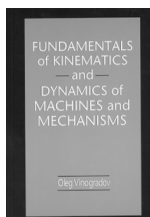
Miroslav D. Lutovac, Dejan V. Tosic, and Brian L. Evans,
2001, Prentice Hall, 756 pp., hardcover.

This book bridges the gap between filter theory and practice, presenting newly developed algorithms and designs. The emphasis is given to automating filter design in software, not to studying general filter theory. Students using this book are assumed to have a basic background in calculus, complex numbers, and differential equations. *Mathematica* and optimization routines are used to produce a new method of filter design, including ready-to-use filter design algorithms and implementations of the algorithms in *Mathematica*. A notable feature of the book is a detailed step-by-step exposure to the filter analysis by transform method, exemplified by self-contained *Mathematica* notebooks. Students can use these notebooks to automate symbolic filter analysis and design in software, to derive closed-form expressions, and to gain insight into the relevant filter design parameters and coefficients.

Chapters include: Signals | Systems | Transforms | Classic Analog Filter Design | Advanced Analog Filter Design Case Studies | Advanced Analog Filter Design Algorithms | Multicriteria Optimization of Analog Filter Designs | Classic Digital Filter Design | Advanced Digital Filter Design Case Studies | Advanced Digital Filter Design Algorithms | Multicriteria Optimization of Digital Filter Designs | Elliptic Functions | Elliptic Rational Function | Appendix A: Example *Mathematica* Notebooks

Available:

store.wolfram.com/view/book/ISBN0201361302.str



Fundamentals of Kinematics and Dynamics of Machines and Mechanisms

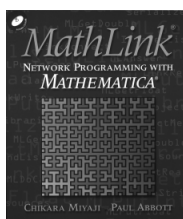
Oleg Vinogradov, 2000, CRC Press, 290 pp., hardcover.

An advanced text emphasizing the understanding and application of basic theoretical principles of kinematics and dynamics of machines and mechanisms. Details the analysis of planar mechanisms and provides an introduction to vibrations and rotordynamics. The text allows students to develop a perception of a mechanism as a whole and an understanding of it as an integral part of the design process. The integration of *Mathematica* software provides a chance to perform symbolic analysis, to plot the results, and to animate the motion. A CD-ROM packaged with the book contains *Mathematica*-based programs for suggested design projects. Basic knowledge of *Mathematica* is required.

Chapters include: Introduction | Kinematic Analysis of Mechanisms | Force Analysis of Mechanisms | Cams | Gears | Introduction to Linear Vibrations | Bibliography | Appendix | Index

Available:

store.wolfram.com/view/book/ISBN0849302579.str



MathLink: Network Programming with Mathematica

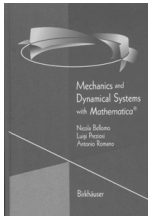
Chikara Miyaji and Paul Abbott, 2001, Cambridge University Press, 243 pp., softcover.

A programming tutorial giving a thorough explanation of *MathLink*, the mechanism enabling higher-level communication between *Mathematica* and external programs. Introduces basic concepts and explains the fundamentals of *MathLink* templates and its data-transfer model. Also provides source code for a wide range of practical applications, giving readers the opportunity to adapt the code to suit their own practical needs. Assumes prior knowledge of *Mathematica* and the C language.

Chapters include: Preliminaries | Connecting the Front End and the Kernel | Compiling AddTwo | Transfer Time Using *MathLink* | Debugging *MathLink* Programs | *TurtleGraphics* | Cellular Automata | **MovieDigitizer** | Object-Oriented Programming | Creating an Event-Driven Mechanism | Creating a Window Object | Window Object Applications | Writing an Interactive Graphics System | Interactive Geometry | Communications between *Mathematica* Sessions

Available:

store.wolfram.com/view/book/ISBN0521645980.str



Mechanics and Dynamical Systems with *Mathematica*

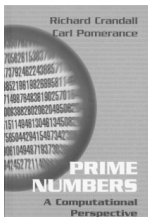
Nicola Bellomo, Luigi Preziosi, and Antonio Romano, 2000, Birkhäuser, 417 pp., hardcover.

A text for advanced students and practitioners in mechanics, scientific computing, physics, and mathematical modeling. Provides a systematic and unified treatment of mechanics and dynamical systems, addressing modeling, qualitative analysis, and simulations of physical systems using ordinary differential equations. *Mathematica* is used to present the scientific computational components in worked examples and end-of-chapter problems.

Chapters include: Models and Differential Equations | Models and Mathematical Problems | Stability and Perturbation Methods | Newtonian Dynamics | Rigid Body Dynamics | Energy Methods and Lagrangian Mechanics | Deterministic and Stochastic Models in Applied Sciences | Chaotic Dynamics, Stability, and Bifurcations | Discrete Models of Continuous Systems | Appendix I: Numerical Methods for Ordinary Differential Equations | Appendix II: Kinematics, Applied Forces, Momentum and Mechanical Energy | Appendix III: Scientific Programs

Available:

store.wolfram.com/view/book/ISBN081764007X.str



Prime Numbers: A Computational Perspective

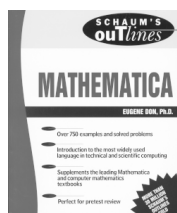
Richard Crandall and Carl Pomerance, 2001, Springer-Verlag, 545 pp., hardcover.

A reference for professionals interested in prime numbers and encryption, cryptography, factoring algorithms, elliptic curve arithmetic, and many more computational issues related to primes and factoring. The text focuses on the computational aspects of finding, testing, and characterizing prime numbers. The theoretically interesting, aesthetic, and practical aspects of primes are discussed. The reference provides theoretical explanations that explain the practical power of the computational algorithms. Detailed pseudocode and exercises that are designed to keep the student engaged and interested are also included.

Chapters include: Primes! | Number-Theoretical Tools | Recognizing Primes and Composites | Primality Proving | Exponential Factoring Algorithms | Subexponential Factoring Algorithms | Elliptic Curve Arithmetic | The Ubiquity of Prime Numbers | Fast Algorithms for Large-Integer Arithmetic | Appendix: Book Pseudocode | References | Index

Available:

store.wolfram.com/view/book/ISBN0387947779.str



Schaum's Outlines of Theory and Problems of Mathematica

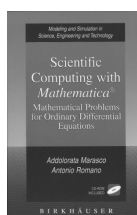
Eugene Don, 2001, McGraw-Hill, 342 pp., softcover.

An introduction to *Mathematica*, aimed at students, with each chapter focusing on a specific application of *Mathematica*. As a study guide it supplies a comprehensive, step-by-step approach to understanding the computer program. Numerous solved problems and examples demonstrate and reinforce ideas discussed in each chapter. *Mathematica* commands with examples are clearly classified according to type. The index of commands and applications expedites referencing.

Chapters include: Getting Acquainted | Basic Concepts | Lists | Two-Dimensional Graphics | Three-Dimensional Graphics | Equations | Algebra and Trigonometry | Differential Calculus | Integral Calculus | Multivariate Calculus | Ordinary Differential Equations | Linear Algebra

Available:

store.wolfram.com/view/book/ISBN007135719X.str



Scientific Computing with Mathematica: Mathematical Problems for Ordinary Differential Equations

Addolorata Marasco and Antonio Romano, 2001, Birkhäuser, 270 pp., hardcover.

An advanced applied mathematics textbook presenting methods for modeling scientific computations with *Mathematica*. Presents an overview of the essential subjects relative to ordinary differential equations. The main problems of ordinary differential equations are covered in an integrated fashion with numerous examples. Explains how to use the *Mathematica* package `ODE.m`, which allows solution of all of the theoretical problems described in the chapters.

Chapters include: Solutions of ODEs and Their Properties | Linear ODEs with Constant Coefficients | Power Series Solutions of ODEs and Frobenius Series | Poincaré's Perturbation Method | Problems of Stability | Stability: The Critical Case | Bifurcation in ODEs | The Lindstedt-Poincaré Method | Boundary-Value Problems for Second-Order ODEs | Rigid Body with a Fixed Point | How to Use the Package **ODE.m** | References | Index

Available:

store.wolfram.com/view/book/ISBN0817642056.str

EDITOR'S PICK

Limits, Uncertainty, and Randomness

Flip Phillips

Editor, The Mathematica Journal
flip@skidmore.edu

■ Introduction

Well, I guess it had to happen: all my hopes of finding truth and justice in science seem to have been dashed. Why? Well, let's just blame it on a symbol for now— Ω —it's easier that way.

"Fatalist!" you may shout. "Determinist!" I hurl your way in return.

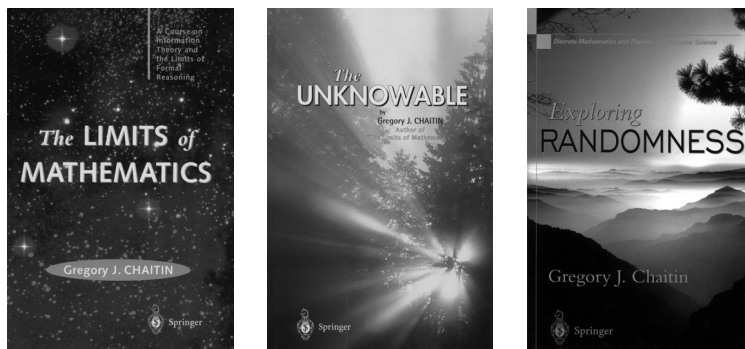
Intellectual punches are thrown, irrational numbers are produced, terms are factored. Pretty soon it's like the whole World Wrestling Federation has cut loose in the laboratory.

The problem as it turns out is that we're running into some real limits with what we can do with mathematics. As a foundation of all that we worship, we might have to deal with explaining some unfortunate characteristics of the chosen faith, at least as far as Greg Chaitin is concerned [1].

■ The Chaitin Trilogy

I was initially made aware of the work of Chaitin while reading John Horgan's excellent book, *The End of Science* [2]. In it, Horgan acts the part of a science reporter *cum* voyeur and, in the chapter "The End of Limitology," he attends a Santa Fe Institute conference titled "The Limits of Scientific Knowledge," a gathering that features, among its participants, Chaitin in the part of an *agent provocateur*. The way it reads, it appears to have been a superb encounter, full of the usual scientific angst and posturing that is most entertaining to observe. From this introduction it was only a matter of time before I started down the unknowable road myself.

Chaitin's latest three books [3, 4, 5] form a nice triangular base to support and explore the concepts underlying *algorithmic information theory* (AIT)—a clever blend of Gödel, Turing, and Shannon that Chaitin developed in his late teens, independent of the similar work of Kolmogorov and Solomonoff. Indeed, there are several prequel works [6, 7, 8], but this set of three volumes packages the material in a nice, quite digestible fashion—even for an easily addled cognitive scientist.



For the uninitiated, the general ideas of information theory revolve around representations—how much information do you need to represent a given thing? It goes beyond the mere semiotics of the situation by also looking at compression—how much can we compress something and still have the original thing? Does the information rely on the accuracy of the sender, receiver, or both? Can the compression be “lossy,” as in JPEG image compression, where the original information is discarded in favor of a visually adequate re-representation? Or must the compression be “lossless,” as in an algorithm that generates the digits of π ? Some of the areas of *coding theory* and *complexity theory* fall into play here as well.

From the viewpoint of a cognitive scientist, “information theory” is an all too familiar subject, having been one of the recent paradigm shifts (more recently overtaken by connectionism, which is just behaviorism revisited, albeit at a smaller scale) in my field of choice. If you’ve dialed a phone recently (or ever actually), you’ve had to deal with the concepts from the field, writ large. Claude Shannon of Bell Labs pretty much originated the field back when they were trying to figure out how much *stuff* the phone company could cram down their wires, the definition of *stuff* being basically that of information [9]. His work was wildly influential in many fields besides that of signals and systems and, to be sure, invaded cognitive science as well.

■ Elegance and Complexity

Chaitin and others refer to the minimal representation issue in AIT as “elegance”—the situation where we have an expression with the property that no smaller expressions yield the same result.

In cognitive science we're often interested in the *complexity* of a stimulus or system. Complexity in this particular case is roughly equivalent to the inverse of elegance, in an Occam's razor fashion. The (very, very rough) idea is that as something becomes more "complex," it becomes more difficult to do / harder to remember / harder to interpret / insert your limitological position here. It caught on quite well in the fields of attention, perception, and, oddly enough, aesthetics. Mathematician George Birkhoff believed that the aesthetic value of an object could be determined as a simple ratio of $\frac{O}{C}$ where O represents the amount of order in the object and C represents the amount of complexity [10]. The rather obvious problem with this ratio is that its two constituents are quite difficult to define in a truly operational and meaningful way. Complexity in this sense has been defined in an almost countless number of ways. One example from the world of shape is the number of vertices in a polygon.

```
In[1]:= Show[GraphicsArray[{Graphics[Rectangle[{0, 0}, {GoldenRatio, 1}]],  
Graphics[Polygon[Table[{Random[], Random[]}, {4}]]]}];
```

From In[1]:=



Two shapes, four vertices each. Is one more "complex" than the other? Depends how you look at it. Is one more aesthetically pleasing than the other? Again, perspective is everything. (See *Komar and Melamid's Scientific Guide to Art* for further details on [11].) I'm sure you'd agree that this probably isn't quite an adequate description of the situation. There have been coding-based approximations. In the above case, you *may* have a better chance of the measure working out if you described the rectangle in a raster-encoding sort of way, sort of like the following.

Thickness 1

Draw a black line from 0 to φ .

The economy is clear when you look at the metacode you'd need to encode the polygon on the right.

Draw a black line from .2 to .3.

Draw a black line from .21 to .32.

Draw a black line from .22 to .35.

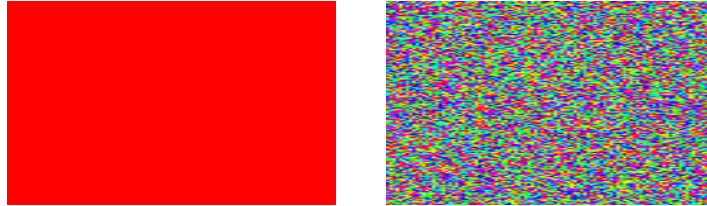
...

Draw a black line from 0 to 0.01.

But what about the following?

```
In[2]:= Show[GraphicsArray[{
  Graphics[RasterArray[
    Table[Hue[1], {IntegerPart[GoldenRatio*100]}, {100}]]],
  Graphics[RasterArray[Table[Hue[Random[]],
    {IntegerPart[GoldenRatio*100]}, {100}]]]]];
```

From In[2]:=



Where does the complexity lie now? Is it in the shape? It would depend on whether the shape included the ideas of color. If I were to turn them both into monochrome images and apply a little Gaussian blur to the right-hand image, as a near sighted person might see it, all of a sudden the difference in apparent complexity vanishes. We have two different potential measures depending on which representation we choose. This is why information theory has had tough times in cognitive science—it is difficult to determine it in a meaningful way.



But what of Chaitin and his ideas of *algorithmic* information theory? Luckily, it is a far better defined concept in the case of an algorithm—how many *symbols* do you need to define the algorithm? To simplify matters, we can define some arbitrary Turing machine that interprets some arbitrary symbol-system to give us a sort of starting point here.

■ How Do You Solve a Problem Like Ω ?

Chaitin's work focuses on the problems of mathematical "truth" as a convenient fiction. There are infinitely many possible mathematical facts, but, according to Chaitin, the underlying relationships among them are impossible to establish. This isn't good news for anyone interested in a "theory of everything," since, if the foundation is built on cottage cheese, the tower is going to be a bit tippy at best. Even worse, Chaitin's results demonstrate that not only is there no structure to the foundation of mathematics, the foundation is in fact *random*. Bad news, reductionists!

In the end it all comes down to, well, the end: Ω . Chaitin's constant, Ω , is best thought of as the probability that a given program p will halt when run on a universal Turing machine U . This halting probability can be empirically explored, which is my favorite part of this whole enterprise. The idea of treating mathematics as an *experimental* science rather than just relying on a pseudo-philosophical (and some might even say dogmatic) footing is particularly exciting to me. Of course, I'm biased anyway, since I do experiments for a living, but that aside, empiricism seems to have a bit to offer to mathematics, especially in the domain Chaitin is interested in pursuing.

□ Chaitin's LISP

To provide some tools to explore his point, Chaitin gives us a slightly modified LISP interpreter—implemented in C, as a Java applet, and in *Mathematica*. The *Mathematica* implementation uses the kernel as a sort of batch processor, defining `run[]` as a function that takes an input file name, the contents of which are evaluated and an output file created.

```
In[3]:= << lisp.m
```

To simplify input matters a bit, we can define the following function that will allow us to do our I/O directly from within *Mathematica*.

```
In[4]:= runexp[sexp_] := Module[{fp},
    fp = OpenWrite["cltemp.l"];
    WriteString[fp, sexp];
    Close[fp];
    run["cltemp"];
]
```

And here we run a simple expression.

```
In[5]:= runexp["cdr car'((a b)c d)"];
```

```
LISP Interpreter Run

cdr car'((a b)c d)

expression (cdr (car (' ((a b) c d))))
value      (b)

End of LISP Run
```

Chaitin's LISP has all of the usual features, but it also includes a set of functions for exploring information, including **bits**, **read-bit**, **read-exp**, and **try**. Furthermore, as demonstrated above, constraining the built-in functions to have a fixed number of parameters allows for input in M-expression form, resulting in a serious reduction in the number of parentheses necessary.

■ Searching for Ω

With these straightforward additions, Chaitin's LISP can be used in an empirical quest for Ω . One approach consists of a LISP expression implementing a universal Turing machine (UTM). This expression, along with an expression representing the program for the UTM, is fed to the LISP interpreter and tested to see if it halts within a given time limit. Since the UTM's program is simply a bit string of some fixed length, one can search through sets of possible "programs" (i.e., bit strings) and determine if they will halt within a given amount of time, according to the implementation of the UTM (again, in the form of a self-delimited bit string).

`In[6]:= run["utm"];`

LISP Interpreter Run

```
[[ An implementation of the UTM ]]
...
define (U p) cadr try no-time-limit 'eval read-exp p
...
define    U
value      (lambda (p) (car (cdr (try no-time-limit (' (eval
                    (read-exp))) p))))
...

[[ A run of the UTM on a bit string representing a LISP M-Expression ]]
...
(U bits 'cons x cons y cons z nil)

expression (U (bits (' (cons x (cons y (cons z nil)))))
value      (x y z)
...

End of LISP Run
```

From here it is a simple extension to try all possible strings of a given length to see if any of them will halt. Of course this is going to be a bit of an extended exercise should we want to test all possible strings of *all* possible lengths. Yet, we trundle on nonetheless by computing the n th lower bound of Ω . To get this lower bound, we take all programs of length n , run them for some given time, and look at which ones halt. As $n \rightarrow \infty$ then we converge on Ω , albeit very slowly indeed. But let's take a look at a few n anyway.

$ln[7]:=$ **run["omega"];**

```
LISP Interpreter Run

[[[[ Omega in the limit from below! ]]]]

define (all-bit-strings-of-size k)
  if = 0 k ' (())
  (extend-by-one-bit (all-bit-strings-of-size - k 1))
  ...
define (count-halt p)
  if atom p 0
  +
  if = success car try t 'eval read-exp car p
  1 0
  (count-halt cdr p)
  ...
define (omega t) cons (count-halt (all-bit-strings-of-size t))
  cons /
  cons ^ 2 t
  nil

[[ Now, run it for a few values of t ]]

(omega 0)
value      (0 / 1)

(omega 1)
value      (0 / 2)

(omega 2)
value      (0 / 4)

(omega 3)
value      (0 / 8)

(omega 8)
value      (1 / 256)

End of LISP Run
```

As it turns out, the various Ω_n form a monotonically increasing series of rational numbers as n increases. The fun thing here is that you can actually *try* it. You can compute a lower bound for Ω at least for some given program size. To give some of the punch line away, the bits that make up the actual value of Ω turn out to be unpredictable; they're *random*! Therefore they are incompressible. You can't make a program that will generate them that is any smaller than the bits themselves (plus you need a few bits to do the generation itself). The most elegant representation of Ω turns out to be the most complex as well.

■ Randomness Rules

Chaitin concludes that, since there is no structure to Ω , the very foundations of number theory are at risk to be washed away. Obviously there are well-formed relationships between some mathematical ideas, some things *do* follow from others, but it would seem that these relationships are the exception rather than the rule.

So, what does all of this mean? To paraphrase Horgan, are we reaching the “end of mathematics?” It seems certain that we’re not reaching any sort of absolute limit, but perhaps it is time for a shift to a more empirical position. We’re finally reaching the point in the field of computation where we can really start to tackle some very large problems. Of course, this situation will only get better over time, subject to some limit of Moore’s Law. And who is to say that the foundations underlying this and other “laws” aren’t random as well?

The paths of math and physics diverged over the past century but now seem to be converging again. The more empirical approach, already well established in physics, appears to be necessary for math as well. It seems like the thing to do is grab some of Chaitin’s work, crank up *Mathematica*, and start doing experiments. Q.E.D.

■ References

- [1] Gregory J. Chaitin home page, www.cs.auckland.ac.nz/CDMTCS/chaitin/
- [2] J. Horgan, *The End of Science*, New York: Addison-Wesley, 1996.
- [3] G. J. Chaitin, *Exploring Randomness*, London: Springer-Verlag, 2001.
- [4] G. J. Chaitin, *The Unknowable*, London: Springer-Verlag, 1999.
- [5] G. J. Chaitin, *The Limits of Mathematics*, London: Springer-Verlag, 1998.
- [6] G. J. Chaitin, *Information-Theoretic Incompleteness*, Singapore: World Scientific, 1992.
- [7] G. J. Chaitin, *Information, Randomness & Incompleteness*, 2nd ed., Singapore: World Scientific, 1990.
- [8] G. J. Chaitin, *Algorithmic Information Theory*, Cambridge: Cambridge University Press, 1987.
- [9] C. E. Shannon, “A Mathematical Theory of Communication,” *Bell System Technical Journal*, **27**, July and October, 1948 pp. 379–423 and 623–656.
- [10] G. D. Birkhoff, *Aesthetic Measure*, Cambridge: Harvard University Press, 1933.
- [11] J. Wypijewski, ed., *Painting by Numbers: Komar and Melamid’s Scientific Guide to Art*, New York: Farrar, Straus, and Giroux, 1997.

Tricks of the Trade

Edited by Paul Abbott

*Department of Physics
University of Western Australia
Nedlands WA 6009, 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.

■ Numerical Differentiation Using Fourier

Jing B. Wang

*University of Western Australia
Nedlands WA 6009, Australia
wang@physics.uwa.edu.au*

A common numerical task is to evaluate the derivatives of a data set given on a numerical mesh. For example, in simulating quantum wavepacket scattering using the time-dependent Schrödinger's equation, the dominant computational effort is in the evaluation of the second-order spatial derivatives of system wavefunctions [1, 2]. In the five-point finite-difference method (FD5), the second derivative of a data set $(f_1, f_2, \dots, f_m, \dots, f_{N-1}, f_N)$ is given by

$$\left. \frac{d^2 f}{dx^2} \right|_m = \frac{1}{12 \Delta x^2} (-f_{m-2} + 16 f_{m-1} - 30 f_m + 16 f_{m+1} - f_{m+2}), \quad (1)$$

where Δx is the grid spacing. If higher accuracy is required, the seven-point finite-difference method (FD7) can be applied

$$\left. \frac{d^2 f}{dx^2} \right|_m = \frac{1}{180 \Delta x^2} (2 f_{m-3} - 27 f_{m-2} + 270 f_{m-1} - 490 f_m + 270 f_{m+1} - 27 f_{m+2} + 2 f_{m+3}). \quad (2)$$

However, finite-difference methods are based on *local approximations* to the derivative operator, which brings about error. Also, the convergence with decreasing grid spacing is slow.

A more accurate and computationally efficient scheme [3], based on the discrete Fourier transform (DFT), uses the fact that if

$$f(x_1, x_2, \dots, x_L) = \int \int \dots \int F(k_1, k_2, \dots, k_L) e^{2\pi i (k_1 x_1 + k_2 x_2 + \dots + k_L x_L)} dk_1 dk_2 \dots dk_L,$$

for a function $F(k_1, k_2, \dots, k_L)$, then the partial derivatives are

$$f^{(n_1, n_2, \dots, n_L)}(x_1, x_2, \dots, x_L) = \int \int \dots \int ((2\pi i k_1)^{n_1} (2\pi i k_2)^{n_2} \dots (2\pi i k_L)^{n_L}) F(k_1, k_2, \dots, k_L) e^{2\pi i (k_1 x_1 + k_2 x_2 + \dots + k_L x_L)} dk_1 dk_2 \dots dk_L. \quad (3)$$

It follows that the derivatives can be evaluated by firstly calculating $F(k_1, k_2, \dots, k_L)$ using the DFT and secondly taking the inverse DFT of the product $(2\pi i k_1)^{n_1} (2\pi i k_2)^{n_2} \dots (2\pi i k_L)^{n_L} F(k_1, k_2, \dots, k_L)$.

For N data points of a one-dimensional function $f(x)$, separated by uniform spacing Δx , where $x \in [0, (N-1)\Delta x]$, the continuous Fourier integral (3) can be approximated using the DFT

$$F(k_j) = \sum_{m=0}^{N-1} f(x_m) e^{-2\pi i k_j x_m \Delta x},$$

where $x_m = m \Delta x$. The corresponding inverse DFT is

$$f(x_m) = \sum_{j=0}^{N-1} F(k_j) e^{2\pi i x_m k_j \Delta k},$$

where $\Delta k = 1/(N \Delta x)$ and $k_j = j \Delta k$.

For any sampling interval Δx , the maximum frequency in its Fourier transform is $k_{\max} = 1/(2 \Delta x)$, which is the Nyquist critical frequency [4]. In other words, the valid frequency components are contained in $F(k_j)$ with $j \in [0, N/2]$, while $F(k_j) = F^*(k_{N-j})$ for $j \in [N/2 + 1, N-1]$, where $*$ indicates complex conjugation. Consequently, the inverse Fourier transform given by equation (3) is discretized to a good approximation as

$$f^{(n)}(x_m) = \sum_{j=0}^{N-1} (2\pi i \tilde{j} \Delta k)^n F(k_j) e^{2\pi i x_m k_j \Delta k}, \quad (4)$$

where $\tilde{j} = j$ for $j \in [0, N/2]$ and $\tilde{j} = j - N$ for $j \in [N/2 + 1, N-1]$. Similarly, for a two-dimensional data set of $N_1 \times N_2$ points, the discrete form of equation (3) is

$$f^{(n_1, n_2)}(x_{m_1}, x_{m_2}) = \sum_{j_1=0}^{N_1-1} \sum_{j_2=0}^{N_2-1} (2\pi i \tilde{j}_1 \Delta k_1)^{n_1} (2\pi i \tilde{j}_2 \Delta k_2)^{n_2} F(k_{j_1}, k_{j_2}) e^{2\pi i (x_{m_1} k_{j_1} + x_{m_2} k_{j_2}) \Delta k_1 \Delta k_2}.$$

The DFT scheme provides a *global representation* of the derivative operator and is highly accurate for functions with *periodic boundary conditions*.

For one-dimensional data sets, implementation of equation (4) is straightforward. The kernel $(2\pi i \tilde{j} \Delta k)^n = \left(\frac{2\pi i \tilde{j}}{N \Delta x}\right)^n$ is

$$In[1]:= \text{Kernel}[n_, m_, \Delta x_] := \text{RotateLeft}\left[\frac{2\pi i \text{Range}\left[\left\lceil \frac{1-m}{2} \right\rceil, \left\lfloor \frac{m}{2} \right\rfloor\right]}{m \Delta x}\right]^n, \left\lfloor \frac{m-1}{2} \right\rfloor\right]$$

and the n th discrete derivative of the data set, **data**, with spacing Δx , reads

$$In[2]:= \text{DFTDerivative}[n_][\text{data}][\Delta x_] := \\ \text{Re}[\text{Fourier}[\text{InverseFourier}[\text{data}] \text{Kernel}[n, \text{Length}[\text{data}], \Delta x]]]$$

(Since there must be constant spacing, Δx , between points, one could use the number of points as a parameter instead.)

The following code is included to handle derivatives of order zero, required for the generalization to higher dimensions.

$$In[3]:= \text{Kernel}[0, m_, \Delta x_] := \text{Table}[1, \{m\}]$$

The syntax of **DFTDerivative** is based on that of **Derivative**. **Re** is included to eliminate the small imaginary component that can arise during computation. Although the derivation refers to the interval $x \in [0, (N-1)\Delta x]$, the implementation works for *any* interval $[a, b]$ with spacing Δx .

For the Gaussian data set,

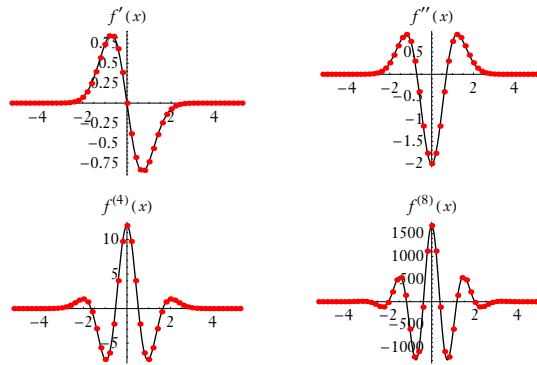
$$In[4]:= \{\text{xs}, \text{data}\} = \text{Table}\left[\{x, e^{-x^2}\}, \{x, -8, 8, \Delta x = 0.2\}\right]^T;$$

the results of **DFTDerivative** $[n]$ for $n = 2^m = 1, 2, 4, 8$ are overlaid upon the exact value of the derivative.

$$In[5]:= \text{Table}\left[\text{Plot}\left[\text{Evaluate}\left[\frac{\partial^{2^m} e^{-x^2}}{\partial x^{2^m}}\right], \{x, -5, 5\}, \text{PlotLabel} \rightarrow f^{(2^m)}(x), \right. \right. \\ \left. \left. \text{PlotRange} \rightarrow \text{All}, \text{Epilog} \rightarrow \{\text{Hue}[1], \text{AbsolutePointSize}[2], \right. \right. \\ \left. \left. \text{Point} /@ \{\text{xs}, \text{DFTDerivative}[2^m][\text{data}][\Delta x]\}^T, \right. \right. \\ \left. \left. \text{DisplayFunction} \rightarrow \text{Identity}\right], \{m, 0, 3\}\right];$$

```
In[6]:= Show[GraphicsArray[Partition[%, 2]]];
```

From In[6]:=

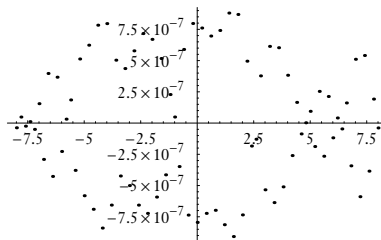


Here, `DFTDerivative[9][data][Δx]` is compared to the exact value.

```
In[7]:= ListPlot[
```

```
{xs, DFTDerivative[9][data][Δx] - Function[x, Evaluate[ $\frac{\partial^9 e^{-x^2}}{\partial x^9}$ ]]/@xs}^T,
PlotRange -> All];
```

From In[7]:=



These figures show that `DFTDerivative[n]` is extremely accurate even for high n and large Δx .

To compare the numerical error in evaluating the second derivatives using the FD5, FD7, and DFT methods, and to examine the relative convergence of these methods as a function of grid spacing, Δx , we first implement equations (1) and (2) using `ListCorrelate`. `ListCorrelate[kernel, list, j]` forms the cyclic correlation in which the j th element of `kernel` is aligned with each element in `list`. Since the central element of $\{-1, 16, -30, 16, -1\}$ is -30 , we have $j = 3$ for **FD5**.

```
In[8]:= FD5[data_][Δx_] :=  $\frac{1}{12 \Delta x^2}$  ListCorrelate[{-1, 16, -30, 16, -1}, data, 3]
```

Similarly, $j = 4$ for **FD7**.

```
In[9]:= FD7[data_][Δx_] :=
```

$$\frac{1}{180 \Delta x^2} \text{ListCorrelate}[\{2, -27, 270, -490, 270, -27, 2\}, \text{data}, 4]$$

When Δx is reasonably large, say $\Delta x = 0.2$, the DFT scheme is several orders of magnitude more accurate than the FD5 and FD7 methods.

```
In[10]:= TableForm[Function[Δx, Module[{xs, f, d2}, {xs, f, d2} =
    Table[Evaluate[{x, e-x2, ∂(x,2) e-x2}], {x, -20., 20, Δx}]T;
    ScientificForm[#, 2] & /@ {Δx, Max[d2 - FD5[f][Δx]], Max[
        d2 - FD7[f][Δx]], Max[d2 - DFTDerivative[2][f][Δx]]}] /@
    {0.01, 0.02, 0.05, 0.1, 0.2, 0.5}, TableHeadings →
    {None, {"Δx", "FD5", "FD7", "DFT"}}]
```

```
Out[10]//TableForm=
```

Δx	FD5	FD7	DFT
$1. \times 10^{-2}$	1.3×10^{-8}	5.1×10^{-11}	7.6×10^{-11}
$2. \times 10^{-2}$	2.1×10^{-7}	1.9×10^{-10}	1.2×10^{-11}
$5. \times 10^{-2}$	8.3×10^{-6}	4.7×10^{-8}	2.2×10^{-12}
$1. \times 10^{-1}$	1.3×10^{-4}	2.9×10^{-6}	5.8×10^{-13}
$2. \times 10^{-1}$	$2. \times 10^{-3}$	1.7×10^{-4}	2.9×10^{-14}
$5. \times 10^{-1}$	6.2×10^{-2}	2.5×10^{-2}	6.6×10^{-5}

There is a limitation to the DFT scheme because the Fourier transformation implies periodic boundary conditions in which the last point of the data set needs to be connected smoothly to the starting point. For Gaussian wavepackets, which rapidly approach zero far from the “center” of the wavepacket, such a boundary condition is clearly satisfied since the wavepackets have essentially zero values at the boundaries. This scheme works equally well for functions with periodic boundaries.

The code is easily extended to *arbitrary* dimensions, with possibly different grid spacings for each dimension, using **Outer**.

```
In[11]:= Kernel[n_List, m_List, Δx_List] :=
    Outer[Times, Sequence @@ Apply[Kernel, {n, m, Δx}]T, 1]
```

```
In[12]:= DFTDerivative[n_][data_][Δx_] :=
    Re[Fourier[InverseFourier[data] Kernel[{n}, Dimensions[data], {Δx}]]] /;
    Length[{n}] == Length[{Δx}] == Length[Dimensions[data]]
```

For a two-dimensional Gaussian data set,

```
In[13]:= data = Table[e-x2-(y-3)2, {x, xmin = -8, xmax = 8, Δx = 0.2},
    {y, ymin = -9, ymax = 13, Δy = 0.4}];
```

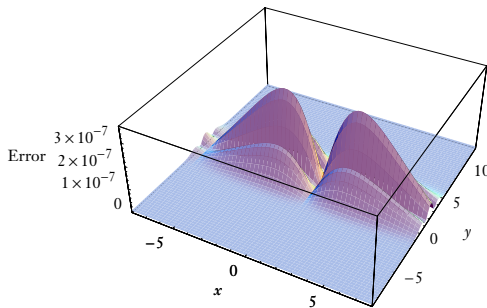
the difference between the exact derivative, $\frac{\partial^3 e^{-x^2-(y-3)^2}}{\partial x^2 \partial y}$, and the DFT scheme is everywhere less than 4×10^{-7} .

```

In[14]:= ListPlot3D[
  |
  DFTDerivative[2, 1][data][Δx, Δy] - Table[Evaluate[ $\frac{\partial^3 e^{-x^2 - (y-3)^2}}{\partial x^2 \partial y}$ ],
    {x, xmin, xmax, Δx}, {y, ymin, ymax, Δy}] |,
  PlotRange → All, Mesh → False, MeshRange → {
    {xmin, xmax},
    {ymin, ymax}
  },
  AxesLabel → {x, y, "Error"}];

```

From In[14]:=



The DFT representation of the derivative operator is *global*, giving it improved accuracy over *local* representations. Moreover, the calculation of higher-order derivatives using the DFT scheme requires little extra computing in contrast to, for example, the finite-difference methods. The code is simple, its structure transparent, and the generalization to arbitrary dimension is straightforward.

Robert Knapp (rknapp@wolfram.com) comments:

In applications it is important to cache the computation of **Kernel**. If the data consists of machine numbers, improved efficiency is possible if **Kernel** is a packed array. Doing this takes more work (and some type-detecting code), but it can make a huge difference in performance.

Code for automatic computation of numerical derivatives using an extension of these ideas is planned for incorporation into a future version of *Mathematica*.

■ Series

Series expansions can be used to solve a wide range of equations including differential equations, functional equations, transcendental equations, and integral equations. Here are two examples.

□ Ideal Fermi Gas

The *parametric form* of the equation of state for a spinless ideal Fermi gas is

$$\frac{P}{kT} = \frac{1}{\lambda^3} f_{5/2}(z), \quad \frac{1}{v} = \frac{1}{\lambda^3} f_{3/2}(z) \quad (1)$$

where $f_n(z)$ can be determined in closed form.

$$\text{In}[1]:= f_{n-}(z_-) = \sum_{i=1}^{\infty} \frac{(-1)^{i+1} z^i}{i^n}$$

$$\text{Out}[1]= -\text{Li}_n(-z)$$

The *virial expansion* of the equation of state is defined to be

$$\frac{Pv}{kT} = \sum_{l=1}^{\infty} a_l \left(\frac{\lambda^3}{v} \right)^{l-1} \quad (2)$$

where a_l is called the l th *virial coefficient*. Combining equations (1) and (2) yields

$$\frac{f_{5/2}(z)}{f_{3/2}(z)} = \sum_{l=1}^{\infty} a_l f_{3/2}(z)^{l-1} \quad (3)$$

The virial coefficients can be obtained by direct expansion of equation (3) into a series in z . Adding the term $O[z]^n$ to an expression *coerces* it into a series up to order z^n . For example,

$$\text{In}[2]:= f_{3/2}(z) + O[z]^3$$

$$\text{Out}[2]= z - \frac{z^2}{2\sqrt{2}} + O(z^3)$$

shows that the leading term of $f_{3/2}(z)^{l-1}$ is z^{l-1} (in this case, obvious from the definition of $f_n(z)$). Hence, to obtain the first four virial coefficients, we truncate the sum in equation (3) at $l = 4$ and expand into a series up to order z^4 .

$$\text{In}[3]:= \frac{f_{5/2}(z)}{f_{3/2}(z)} - \sum_{l=1}^4 a_l f_{3/2}(z)^{l-1} + O[z]^4 == 0 // \text{Simplify}$$

$$\begin{aligned} \text{Out}[3]= & (1 - a_1) + \frac{1}{8} (\sqrt{2} - 8a_2)z + \left(\frac{a_2}{2\sqrt{2}} - a_3 - \frac{2}{9\sqrt{3}} + \frac{1}{16} \right) z^2 + \\ & \left(-\frac{a_2}{3\sqrt{3}} + \frac{a_3}{\sqrt{2}} - a_4 - \frac{7}{36\sqrt{6}} + \frac{1}{32\sqrt{2}} + \frac{3}{32} \right) z^3 + O(z^4) == 0 \end{aligned}$$

The virial coefficients, a_l , are most easily extracted using **SolveAlways**.

```
In[4]:= SolveAlways[%, z] // First // Sort // Simplify // TableForm
```

```
Out[4]//TableForm=
```

$$a_1 \rightarrow 1$$

$$a_2 \rightarrow \frac{1}{4\sqrt{2}}$$

$$a_3 \rightarrow \frac{1}{8} - \frac{2}{9\sqrt{3}}$$

$$a_4 \rightarrow \frac{1}{192} (18 + 15\sqrt{2} - 16\sqrt{6})$$

□ Series Solution of Equations

In many situations, the summation symbol, $\sum$, is not intended to imply explicit summation but is just a notational device. The *Einstein summation convention*, where summation over repeated indices is implied rather than explicitly stated, for example, $\sum_{i=1}^n a_i b^i \rightarrow a_i b^i$, can often be used to simplify the algebra. Interestingly, this convention translates into a general principle for analyzing and evaluating integrals and products of sums in *Mathematica*. Just focus on the *summand* (i.e., the general term of the sum) and drop the summation symbol.

For example, to determine the series solution to the differential equation $x y''(x) + y'(x) + y(x) = 0$, with initial condition $y(0) = 1$, first substitute in the generic term of the series solution $y(x) = \sum_{n=0}^{\infty} a_n x^n$, which is $a_n x^n$.

```
In[5]:= x y''(x) + y'(x) + y(x) == 0 /. y -> Function[x, a_n x^n] // ExpandAll
```

```
Out[5]= n^2 a_n x^{n-1} + a_n x^n == 0
```

The rule $y \rightarrow \text{Function}[x, a_n x^n]$ is used so that both $y(x)$ and its derivatives are computed. Next, use *pattern matching* to find the recurrence relation satisfied by the a_n .

```
In[6]:= % /. c_ x^{n+k_} -> (c /. n -> n - k - 1) // Simplify
```

```
Out[6]= a_n n^2 + a_{n-1} == 0
```

This works by looking for the pattern $c_ x^{n+k_}$ (which means anything $c_$ multiplied by $x^{n+k_}$ where $k_$ means anything or 0), and replacing it with c where n is reduced by $k+1$ and x^{n+k} is omitted. This is, essentially, what a human does.

The solution to this recurrence relation can be obtained using the **DiscreteMath`RSolve`** package. Alternatively, since

$$a_n = \left(\frac{a_n}{a_{n-1}} \right) \left(\frac{a_{n-1}}{a_{n-2}} \right) \cdots \left(\frac{a_1}{a_0} \right) a_0 = a_0 \prod_{i=1}^n \left(\frac{a_i}{a_{i-1}} \right)$$

then determining a_n in terms of a_{n-1} ,

```
In[7]:= Solve[%, a_n] // First
```

```
Out[7]= {a_n -> -\frac{a_{n-1}}{n^2}}
```

leads to general solution of the recurrence relation.

$$\text{In}[8]:= a_{m_} = \prod_{n=1}^m \left(\frac{a_n}{a_{n-1}} /. \% \right)$$

$$\text{Out}[8]= \frac{(-1)^m}{m!^2}$$

Here we have put $a_0 = 1$, corresponding to the initial condition $y(0) = 1$. Finally, the series solution, $y(x) = \sum_{n=0}^{\infty} a_n x^n$, is computed in closed form.

$$\text{In}[9]:= \sum_{n=0}^{\infty} a_n x^n$$

$$\text{Out}[9]= \mathcal{J}_0(2\sqrt{x})$$

Of course, **DSolve** can solve this differential equation directly.

$$\text{In}[10]:= \text{DSolve}[x y''(x) + y'(x) + y(x) == 0, y(0) == 1], y(x), x]$$

$$\text{Out}[10]= \{\{y(x) \rightarrow \mathcal{J}_0(2\sqrt{x})\}\}$$

■ Files Palette

Mathematica has a plethora of commands for working with files. For example,

$$\text{In}[1]:= \text{SetDirectory}[\$TopDirectory]$$

$$\text{Out}[1]= \text{Spike:Mac OS 9:Applications:Mathematica 4.1}$$

$$\text{In}[2]:= \text{FileNames}[]$$

$$\text{Out}[2]= \{:\text{AddOns}, :\text{Configuration}, :\text{Documentation}, \\ :\text{Executables}, :\text{Mathematica}, :\text{Registration}, :\text{SystemFiles}\}$$

returns a list of file names in the top-level directory in which *Mathematica* system files are installed. Note that the location and the form of file and directory names vary among computer systems.

Here is the (short) list of notebooks that are in subdirectories of the **SystemFiles** directory, up to three levels down.

$$\text{In}[3]:= \text{Short}[\text{fullpath} = \text{FileNames}["*.nb", "\text{SystemFiles}", 3], 8]$$

$$\text{Out}[3]//\text{Short}=$$

$$\{:\text{SystemFiles:FrontEnd:Palettes:AlgebraicManipulation.nb}, \\ :\text{SystemFiles:FrontEnd:Palettes:BasicCalculations.nb}, \\ :\text{SystemFiles:FrontEnd:Palettes:BasicInput.nb}, \ll 24 \gg, \\ :\text{SystemFiles:FrontEnd:TextResources:AboutBoxStudent.nb}, \\ :\text{SystemFiles:FrontEnd:TextResources:AboutBoxTrial.nb}\}$$

Referencing a file using its full path is cumbersome. The name of the file can be extracted by **StringDrop**, using **DirectoryName** to obtain the directory name from the specification for a file.

```

In[4]:= PathToFileName[file_String] :=
        StringDrop[file, StringLength[DirectoryName[file]]]

In[5]:= PathToFileName /@ fullpath

Out[5]= {AlgebraicManipulation.nb, BasicCalculations.nb, BasicInput.nb,
        BasicTypesetting.nb, CompleteCharacters.nb, InternationalCharacters.nb,
        NotebookLauncher.nb, ArticleClassic.nb, ArticleModern.nb,
        Classic.nb, Classroom.nb, Default.nb, Demo.nb, DemoText.nb,
        HelpBrowser.nb, HTML.nb, NaturalColor.nb, NotepadMono.nb,
        Notepad.nb, PastelColor.nb, PrimaryColor.nb, Report.nb,
        Textbook.nb, TutorialBook.nb, AboutBox.nb, AboutBoxReader.nb,
        AboutBoxSecured.nb, AboutBoxStudent.nb, AboutBoxTrial.nb}

```

Functions like **ReadList** are most often used for processing text and data from external files. In some cases, however, you may find it convenient to use these same functions to process strings. You can do this by using the function **StringToStream**, which opens an input stream that takes characters not from an external file, but instead from a string (see 2.11.10 of *The Mathematica Book*).

```

In[6]:= str = StringToStream[StringJoin @@ fullpath]

Out[6]= InputStream[String, 18]

```

Then **ReadList** can be used to extract data from the stream, and convert it to a list. For example, we extract the names of all the palettes in subdirectories of the **SystemFiles** directory using **Cases**.

```

In[7]:= palettes = Cases[ReadList[str, Word,
        RecordLists → True, WordSeparators → $PathnameSeparator,
        RecordSeparators → ".nb"], {_, _, "Palettes", _}]

Out[7]= {SystemFiles FrontEnd Palettes AlgebraicManipulation
        SystemFiles FrontEnd Palettes BasicCalculations
        SystemFiles FrontEnd Palettes BasicInput
        SystemFiles FrontEnd Palettes BasicTypesetting
        SystemFiles FrontEnd Palettes CompleteCharacters
        SystemFiles FrontEnd Palettes InternationalCharacters
        SystemFiles FrontEnd Palettes NotebookLauncher }

```

Using **RecordSeparators → ".nb"** produces a separate record for each file, and has the side effect of shortening the file name by dropping the **.nb** extension. Also, breaking the path into individual components eliminates the need for a (machine-dependent) delimiter character and makes further manipulation easy, since the individual elements of the path can be addressed as elements of a list.

Failure to close streams after you are done with them will result in a slowdown in the kernel's performance so the stream should be closed after the operation is complete.

```

In[8]:= Close[str]

Out[8]= String

```

It is quite easy to produce a button that, when clicked, opens a specified file in a subdirectory of the system files directory (**\$TopDirectory**).

```
In[9]:= ButtonAction [{path___, filename_}] := ButtonBox[filename,
  ButtonFunction -> (NotebookOpen [#2] &),
  ButtonData -> FrontEndFileName[{TopDirectory, path},
    Evaluate[StringJoin[filename, ".nb"]]], Active -> True]
```

Machine-dependent issues with paths are minimized by using **FrontEndFileName** to construct the file name when the button is clicked. This approach is strongly encouraged for users who wish to make their work easily transportable to other systems. Clicking the button

```
In[10]:= ButtonAction [palettes[[1]]] // DisplayForm
```

```
Out[10]//DisplayForm=
```

AlgebraicManipulation

will open the **AlgebraicManipulation.nb** palette in the front end under *any* operating system.

A straightforward extension is to generate a palette with buttons for each file in a directory.

```
In[11]:= DisplayForm [GridBox[Partition [ButtonAction /@ palettes, 2, 2, {1, 1}, ""],
  RowSpacings -> 0, ColumnSpacings -> 0]]
```

```
Out[11]//DisplayForm=
```

AlgebraicManipulation	BasicCalculations
BasicInput	BasicTypesetting
CompleteCharacters	InternationalCharacters
NotebookLauncher	

Partition is used here to produce a palette of manageable size with two items on each row. The following simple example demonstrates how this is done.

```
In[12]:= Partition [Range[7], 2, 2, {1, 1}]
```

```
Out[12]= 
$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 1 \end{pmatrix}$$

```

Padding with an empty string, "", is used to make the palette rectangular without repetition.

```
In[13]:= Partition [Range[7], 2, 2, {1, 1}, ""]
```

```
Out[13]= 
$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & \end{pmatrix}$$

```

■ Smooth Interpolation

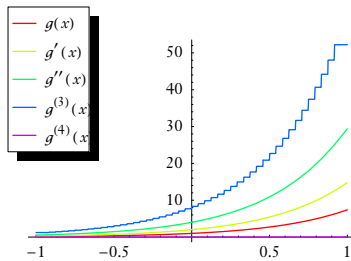
Robert Knapp

Wolfram Research
rknapp@wolfram.com

In general, **InterpolatingFunction** objects are continuous but not smooth, as in the following example.

```
In[1]:= f(x_) = e2x;
In[2]:= g = FunctionInterpolation[f(x), {x, -1, 1}]
Out[2]= InterpolatingFunction[(-1. 1.), <>]
In[3]:= << Graphics`
In[4]:= Plot[Evaluate[Table[g(n)(x), {n, 0, 4}], {x, -1, 1},
  PlotRange → All, PlotStyle → Table[Hue[ $\frac{n}{5}$ ], {n, 0, 4}],
  PlotLegend → Table[g(n)(x), {n, 0, 4}], LegendPosition → {-1, 0}];
```

From In[4]:=



It can be clearly seen that these **InterpolatingFunction** objects are piecewise polynomial functions.

If the derivatives of the function are also available, one can use **FunctionInterpolation** as follows.

```
In[5]:= h = FunctionInterpolation[{f(x), f'(x), f''(x)}, {x, -1, 1}]
Out[5]= InterpolatingFunction[(-1. 1.), <>]
```

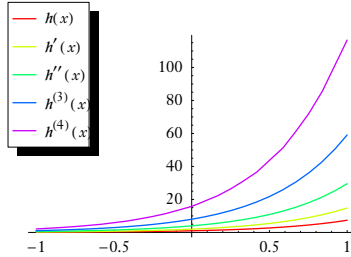
The resulting **InterpolatingFunction** object is continuous and smooth.

```

In[6]:= Plot[Evaluate[Table[h(n)(x), {n, 0, 4}]], {x, -1, 1},
           PlotRange → All, PlotStyle → Table[Hue[ $\frac{n}{5}$ ], {n, 0, 4}],
           PlotLegend → Table[h(n)(x), {n, 0, 4}], LegendPosition → {-1, 0}];

```

From In[6]:=



In general, the derivatives of an **InterpolatingFunction** object are continuous up to the minimum order which is contained in its data.

■ References

- [1] N. Balakrishnan, C. Kalyanaraman, and N. Sathiyamurthy, "Time-dependent quantum mechanical to reactive scattering and related processes," *Physics Reports*, **280**, 1997 pp. 80–144.
- [2] J. B. Wang and T. T. Scholz, "Time-dependent approach to scattering Chebyshev expansion and FFT," *Physical Review A*, **57**, 1998 pp. 3554–3559.
- [3] R. Kosloff, "Time-dependent quantum mechanical methods for molecular dynamics," *Journal of Physical Chemistry*, **92**, 1988 pp. 2087–2100.
- [4] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in Fortran 90*, Cambridge: Cambridge University Press, 1996.

In and Out

Edited by Paul Abbott

*Department of Physics
University of Western Australia
Nedlands WA 6009, 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.

■ Inline Equations

Q: When entering text into a Text cell, if I enter a subscripted variable (say $b_{\text{CTRL-}[]0}$), it appears as an inline cell, b_0 , in Courier font. I assume that this is the default for inline cells. How can I change my settings so that it appears as b_0 ?

A: P. J. Hinton (paulh@wolfram.com) writes: There are two different editing environments in the notebook front end. The “text editor” is used for handling regular text (**TextData**), possibly with some style enrichments such as modifications in font family, color, weight, and so on. Heading and plain text cells use the “text editor” by default. The “text editor” is also used when **InputForm** cells are being modified. The “math editor” is used to manipulate anything in the notebook that is represented as a two-dimensional typeset structure (**BoxData**). If a cell style uses either the **StandardForm** or **TraditionalForm** format types, then the “math editor” will be used to modify the contents. The front end switches between editing environments based on settings that normally reside in your notebook’s style sheet.

When you attempt to enter a subscript in a plain text cell, the front end handles this by creating an inline cell and switching from the text editor to the math editor. The appearance of the space occupied by the inline cell is set by a cell style in your style sheet known as **InlineCellEditing**. It is applied to the contents of the inline cell if your text cursor resides within the boundaries of the cell. By default, this color is `RGBColor[1, 0.749996, 0.8]`, which produces that pink background color.

Inline cells do not immediately obey the formatting settings of the encapsulating cell. Instead, the settings are inherited from one of the **FormatType** style cells that is in your notebook's style sheet. Just which **FormatType** is applied depends on the setting of the option. You can query this value by evaluating the following expression in the notebook you are using.

```
In[1]:= "TextInline" /. (CommonDefaultFormatTypes /.
      Options[EvaluationNotebook[], CommonDefaultFormatTypes])
```

```
Out[1]= TraditionalForm
```

In most situations, **TraditionalForm** will be returned. However, you may be using a notebook that uses **StandardForm** instead. That would account for the use of Courier in your subscripting.

If you open a notebook's style sheet with the Format ▸ Edit StyleSheet menu command and then expose the cell group that has the heading **FormatType Styles**, you will see where all the mischief comes into play.

Note that you can open a notebook's style sheet by executing the following command.

```
In[2]:= FrontEndTokenExecute["EditStyleDefinitions"]
```

FrontEndTokenExecute allows you to send from the kernel any command that the front end can execute. These commands include both menu items and control sequences. See Section 2.10.4 of *The Mathematica Book*. **TraditionalForm** typically does not have an explicit font setting, so it inherits the default for the notebook.

```
In[3]:= FontFamily /. Options[EvaluationNotebook[], FontFamily]
```

```
Out[3]= Times
```

On most systems, this will return Times or the system look-alike for this font. However, **StandardForm** *does* have a default font choice in most style sheets—Courier. If your notebook is using **StandardForm** as the **FormatType** for inline cells, you could probably do one of two things to fix your problem. The first is to simply change the value of **DefaultInlineFormatType** to **TraditionalForm** by using the menu command Cell ▸ Default Inline Format Type ▸ TraditionalForm. A less desirable alternative is to need to modify the **FontFamily** option that appears in the prototype cell for **StandardForm**. The reason that this is discouraged is that it will affect other cell styles which use the **StandardForm** prototype, including Input and Output style cells.

■ Pop-up Window Button

Q: I am working with text cells that include parenthetical remarks. Instead of using parentheses, I would like to configure a button, embedded in the text cells, so

that when you click on it, it opens up a small pop-up window containing the parenthetical remark. How can I do this?

A: P. J. Hinton (paulh@wolfram.com) writes: You can define a button style that can do this. Here is an example of text with a hidden parenthetical remark located **here**. Clicking **here** opens up a small pop-up window containing the parenthetical remark.

How does this work? A style named **PopupButton** has been added to the style sheet of this notebook:

Prototype for Style: "PopupButton":

PopupButton

If you unformat this cell, you will see that its **ButtonFunction** opens a new notebook, which itself contains a **NotebookClose** button.

```
ButtonFunction:>(NotebookPut[
Notebook[{
  Cell[ #2, "Text"],
  Cell[
    BoxData[
      ButtonBox[ "Dismiss",
        ButtonFunction :> NotebookClose[
          ButtonNotebook[ ],
          Interactive -> False],
        Active -> True]
    ], "Text"]},
  ShowCellBracket -> False,
  WindowFrame -> "Frameless",
  WindowElements -> None,
  WindowFrameElements -> None,
  WindowTitle -> None,
  WindowSize -> {90, 80}]
]&)
```

To add a new parenthetical remark:

1. Select the text that will be used to anchor the pop-up.
2. Click on the front end menu command sequence Input > Create Button > **PopupButton**. The text will turn red.
3. Now drag-select the text from left to right and then open the Option Inspector dialog (Format > Option Inspector). Check that the scope is selection, and then locate the option **ButtonData**.
4. Enter the text you want displayed in the pop-up window as a string.
5. Close the Option Inspector dialog.

■ Output Window

Q: Is there any way to separate input and output? I want to input code in one window and show the results in another.

A: P. J. Hinton (paulh@wolfram.com) writes: Here is a very simple implementation of what you describe using notebook manipulation functions. First, define a function named **externalOutput** that acts as a wrapper to **NotebookWrite**. This function needs to cope with **Null** output,

```
In[1]:= externalOutput[Null] := Return []
```

allow for recursive calls,

```
In[2]:= externalOutput[externalOutput] := Return []
```

handle graphics,

```
In[3]:= externalOutput[expr : (_Graphics | _Graphics3D |
      _SurfaceGraphics | _ContourGraphics | _DensityGraphics)] :=
Module[{$DisplayFunction = Identity}, NotebookWrite[$outputNotebook,
  Cell[GraphicsData["PostScript", DisplayString[expr]], "Graphics"];
  NotebookWrite[$outputNotebook,
    Cell[BoxData[ToBoxes[expr]], "Output"]]]
```

and all other output.

```
In[4]:= externalOutput[expr_] := NotebookWrite[
  $outputNotebook, Cell[BoxData[ToBoxes[expr]], "Output"]]
```

Entering

```
In[5]:= $DisplayFunction = Identity;
```

prevents graphics from being displayed in the notebook with **Input** style cells.

To use **externalOutput**, create a global variable named **\$outputNotebook** which points to the notebook where output is to be placed.

```
In[6]:= $outputNotebook = NotebookCreate[];
```

Setting **\$Post** to use **externalOutput** means that all expressions returned by the kernel will be fed to the **externalOutput** function.

```
In[7]:= $Post = externalOutput
```

Here are three tests.

```
In[8]:= 2 + 2
```

```
In[9]:= Plot[sin(x), {x, 0, 2π}]
```

```
In[10]:= 2 + 2;
```

All output, including graphics, is displayed in the external notebook. Any expression that returns **Null** does not create an **Output** style cell.

See also “Exporting Graphics,” *TMJ*, 7 (2), 1998 p. 130 and the Tech Support column, *TMJ*, 7(2), 1998 pp. 145–146.

■ Why the Beep?

Q: How can I save the message from the **Why the Beep?** dialog to the clipboard?

A: Lou D’Andria (lou@wolfram.com) writes: You can specify that messages appear in the **Messages** window (or in the notebook itself)—from which you can copy it to the clipboard—by setting the appropriate suboption of **MessageOptions** (either **WarningAction** or **ErrorAction**) to include **PrintToConsole** (respectively **PrintToNotebook**).

1. Open the Option Inspector dialog (Format ▷ Option Inspector). Check that the scope is global (if you want the change to apply to all notebooks), and then locate **MessageOptions** by typing in “message.”
2. Modify **WarningAction** to include **WarningAction** or **ErrorAction**.
3. Modify **ErrorAction** to include **WarningAction** or **ErrorAction**.

■ Associativity

Q: I have defined the operator $\oplus$ in the following way:

$$\text{In}[1]:= \text{x_} \oplus \text{y_} := \frac{x \, y}{x + y}$$

When I type in

$$\text{In}[2]:= 1 \oplus 2$$

$$\text{Out}[2]= \frac{2}{3}$$

I get the correct result. But when I try to compute

$$\text{In}[3]:= 1 \oplus 2 \oplus 3$$

$$\text{Out}[3]= 1 \oplus 2 \oplus 3$$

the input is returned. How can I specify the operator in a general way?

A: Hartmut Wolf (hwolf@debis.com) writes: If you consult Section A.2.7 of *The Mathematica Book*, you will find that the operator $\oplus$ is fully associative. Definitions respecting associativity will work without problem. Noting that $\frac{x y}{x+y} = \frac{1}{1/x+1/y}$, an associative definition that works for any number of arguments is

In[4]:= **Clear[CirclePlus]**

In[5]:= **CirclePlus[x_]:=** $\frac{1}{\text{Plus}@@\frac{1}{\{x\}}}$

In[6]:= **1** $\oplus$ **2**

Out[6]= $\frac{2}{3}$

In[7]:= **1** $\oplus$ **2** $\oplus$ **3**

Out[7]= $\frac{6}{11}$

Another natural extension of the original definition is the sequence

$$x \oplus y = \frac{x y}{x + y}, \quad x \oplus y \oplus z = \frac{x y z}{x + y + z}, \dots$$

This will, in general, lead to *different* results than the definition above. An associative definition for an arbitrary number of arguments is immediate.

In[8]:= **CirclePlus[x_]:=** $\frac{\text{Times}[x]}{\text{Plus}[x]}$

In[9]:= **1** $\oplus$ **2**

Out[9]= $\frac{2}{3}$

In[10]:= **1** $\oplus$ **2** $\oplus$ **3**

Out[10]= 1

David Park (djmp@earthlink.net) writes: Either use parentheses, since your definition only accommodates two arguments,

In[11]:= **Clear[CirclePlus]**

In[12]:= **x_** $\oplus$ **y_** := $\frac{x y}{x + y}$

In[13]:= **1** $\oplus$ (**2** $\oplus$ **3**)

Out[13]= $\frac{6}{11}$

or extend the definition of $\oplus$ to more than two arguments by adding the (right-association) rule

In[14]:= **x_** $\oplus$ **y_** $\oplus$ **z_** := **x** $\oplus$ (**y** $\oplus$ **z**)

Now the desired result is obtained without using parentheses.

$$\text{In}[15]:= 1 \oplus 2 \oplus 3$$

$$\text{Out}[15]= \frac{6}{11}$$

Note that $x \oplus y = \frac{x y}{x + y}$ is the addition formula for capacitors in series or resistors in parallel.

$$\text{In}[16]:= R_1 \oplus R_2 \oplus R_3 \text{ // Together}$$

$$\text{Out}[16]= \frac{R_1 R_2 R_3}{R_1 R_2 + R_3 R_2 + R_1 R_3}$$

$$\text{In}[17]:= C_1 \oplus C_2 \oplus C_3 \oplus C_4 \text{ // Together}$$

$$\text{Out}[17]= (C_1 C_2 C_3 C_4) / (C_1 C_2 C_3 + C_1 C_4 C_3 + C_2 C_4 C_3 + C_1 C_2 C_4)$$

As another example, the following approach makes it straightforward to implement the velocity addition formula of special relativity for an arbitrary number of velocities, where c is the speed of light.

$$\text{In}[18]:= \text{Clear}[\text{CirclePlus}]$$

$$\text{In}[19]:= \mathbf{u_} \oplus \mathbf{v_} := \frac{\mathbf{u} + \mathbf{v}}{1 + \frac{\mathbf{u} \mathbf{v}}{c^2}}$$

$$\text{In}[20]:= \mathbf{u_} \oplus \mathbf{v_} \oplus \mathbf{w_} := \mathbf{u} \oplus (\mathbf{v} \oplus \mathbf{w})$$

Relativistic addition of c to c gives c .

$$\text{In}[21]:= c \oplus c$$

$$\text{Out}[21]= c$$

Ordinary addition of the following three velocities exceeds c .

$$\text{In}[22]:= \frac{c}{2} + \frac{c}{3} + \frac{c}{5}$$

$$\text{Out}[22]= \frac{31 c}{30}$$

However, relativistic addition leads to a result less than c .

$$\text{In}[23]:= \frac{c}{2} \oplus \frac{c}{3} \oplus \frac{c}{5}$$

$$\text{Out}[23]= \frac{4 c}{5}$$

In special relativity, the **rapidity parameter** ξ is defined by $\xi = \frac{\beta}{|\beta|} \tanh^{-1}(|\beta|)$, where $\beta = \frac{v}{c}$, and v is the relative velocity between the moving reference frames. The benefit of the rapidity representation is the simple formula for the addition of rapidities (in the same spatial direction).

$$\text{In}[24]:= c \tanh(\xi_1) \oplus c \tanh(\xi_2) \text{ // Simplify}$$

$$\text{Out}[24]= c \tanh(\xi_1 + \xi_2)$$

■ Complex Vectors

Q: For the complex vector

```
In[1]:= v = {1, i};
```

with inner product (defined using **AngleBracket**; see Section 3.10.4 of *The Mathematica Book*),

```
In[2]:= <x_, y_> := x†.y
```

and norm (defined using **DoubleBracketingBar**; see Section 3.10.2 of *The Mathematica Book*),

```
In[3]:= ||x_|| := Sqrt[<x, x>]
```

where conjugation is implemented using **SuperStar** (see Section 3.10.2 of *The Mathematica Book*),

```
In[4]:= x† := x /. Complex[a_, b_] -> Complex[a, -b]
```

the normalized vector is

```
In[5]:= v / ||v||
```

```
Out[5]:= { 1/√2, i/√2 }
```

However, using the package

```
In[6]:= << LinearAlgebra`Orthogonalization`
```

why do I get the following error?

```
In[7]:= Normalize[v]
```

Normalize::zeromag : The magnitude of {1, i} was zero under the inner product Dot.

```
Out[7]:= {1, i}
```

A: Jürgen Tischer (jtischer@col2.telecom.com.co) writes: Evaluating

```
In[8]:= Options[Normalize]
```

```
Out[8]:= {InnerProduct -> Dot}
```

shows that **Dot** is used as the **InnerProduct** and

```
In[9]:= v.v
```

```
Out[9]:= 0
```

explains the error message. Using

```
In[10]:= Normalize[v, InnerProduct → AngleBracket]
```

$$\text{Out[10]} = \left\{ \frac{1}{\sqrt{2}}, \frac{i}{\sqrt{2}} \right\}$$

or

```
In[11]:= SetOptions[Normalize, InnerProduct → AngleBracket];
```

```
In[12]:= Normalize[v]
```

$$\text{Out[12]} = \left\{ \frac{1}{\sqrt{2}}, \frac{i}{\sqrt{2}} \right\}$$

gives the desired result.

■ Radial Basis Function

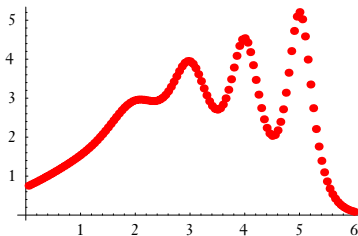
Q: Given a numeric data set, how does one go about finding a reasonable fitting function when you have no underlying theoretical equation to work with?

A: Consider the following data set.

```
In[1]:= data = Table[Evaluate[{x, Sum[i sech(i (i - x)), {i, 1, 5}]}], {x, 0.05, 6, 0.05}];
```

```
In[2]:= lp = ListPlot[data, PlotStyle → {Hue[1], AbsolutePointSize[3]}];
```

From In[2]:=



For this example, we know the (exact) functional form of the fitting function and could use the **Statistics`NonlinearFit`** package to determine the best fit. However, assume that we are just given the data set. Tony Roberts (aroberts@usq.edu.au) suggests fitting with the *radial basis function*,

$$\text{In[3]} := f(\mathbf{r}_-) = r^2 \log(r);$$

To do this, we need to solve for the coefficients z_j in the *linear equations* $y_i = \sum_{j \neq i} f(r_{i,j}) z_j$, where $r_{i,j}$ is the distance between the i th and the j th data point, $r_{i,j} = |x_j - x_i|$. We can extend the sum to include $j = i$ by noting that

```
In[4]:= f(0 | 0.) =  $\lim_{r \rightarrow 0} f(r)$ 
```

```
Out[4]= 0
```

The syntax `0 | 0.` uses **Alternatives** to handle both exact and approximate zeros.

In matrix form, the equations read $\mathcal{F} \cdot \mathbf{z} = \mathbf{y}$, where $\mathcal{F}_{i,j} = f(r_{i,j})$. The matrix $\mathcal{F}$ is easily constructed using **Outer**.

```
In[5]:= {xs, ys} = dataT;
```

```
In[6]:=  $\mathcal{F} = \text{Outer}[f(\#2 - \#1) \&, \text{xs}, \text{xs}];$ 
```

The vector $\mathbf{z} = \mathcal{F}^{-1} \cdot \mathbf{y}$ can be computed efficiently using **LinearSolve**.

```
In[7]:= z = LinearSolve[ $\mathcal{F}$ , ys];
```

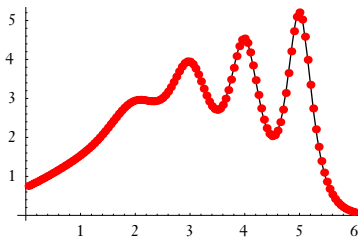
Finally, the fitted function is $y(x) = \sum_j z_j f(r_{x,j})$, where $r_{x,j}$ is the distance from any point x to the j th data point. Implementing this function as

```
In[8]:= y(x_) = (f(|x - #|) & /@ xs).z;
```

we plot the fitted function together with the data set.

```
In[9]:= Plot[y(x), {x, 0, 6}, Epilog -> First[lp]];
```

From In[9]:=



This method is robust and is simple to code. However, it can be expensive to compute if you have a lot of data points.

■ Permutations, Signature, and Trace

In “Antisymmetrizing Functions,” *TMJ*, 2 (1), 1992 p. 45, **Permutations** and **Signature** were used in the construction of totally antisymmetric functions, that is, functions that change sign under the interchange of any pair of arguments. Jorma Virtamo (jorma.virtamo@vtt.fi) suggested the following improved code.

```
In[1]:= Antisymmetrize[f_] :=
```

```
With[{p = Permutations[f]}, Signature[f] p.Signature /@ p]
```

The **Signature** of a permutation is $(-1)^n$, where n is the number of transpositions of pairs of elements that must be composed to build up the permutation. By including **Signature[f]** in the definition of **Antisymmetrize** initially, cyclic and anticyclic permutations are distinguishable.

In[2]:= **Antisymmetrize**[$f(a, b, c)$]

Out[2]= $f(a, b, c) - f(a, c, b) - f(b, a, c) + f(b, c, a) + f(c, a, b) - f(c, b, a)$

In[3]:= **Antisymmetrize**[$f(b, a, c)$]

Out[3]= $-f(a, b, c) + f(a, c, b) + f(b, a, c) - f(b, c, a) - f(c, a, b) + f(c, b, a)$

In “Calculus with Anticommuting Variables,” *TMJ*, 8 (1), 2001 pp. 38–40, ordered determinants are computed using dynamic programming and Laplace expansion. For an interesting discussion of determinants of matrices with noncommutative entries, see [1]. Here is an alternative functional implementation, which is an extension of the code for **Antisymmetrize**.

In[4]:= **||m_?MatrixQ|| /; Equal @@ Dimensions[m] :=**
With[{p = Permutations[Range[Length[m]]],
(Tr[m^T[[#1]], CircleTimes] & /@ p).Signature /@ p}

For example, the ordered determinant of a general 3×3 matrix

In[5]:= $M = \text{Table}[a_{i,j}, \{i, 3\}, \{j, 3\}]$

Out[5]=
$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix}$$

is

In[6]:= **||M||**

Out[6]= $a_{1,1} \otimes a_{2,2} \otimes a_{3,3} - a_{1,1} \otimes a_{2,3} \otimes a_{3,2} - a_{1,2} \otimes a_{2,1} \otimes a_{3,3} +$
 $a_{1,2} \otimes a_{2,3} \otimes a_{3,1} + a_{1,3} \otimes a_{2,1} \otimes a_{3,2} - a_{1,3} \otimes a_{2,2} \otimes a_{3,1}$

The code uses the observation that (ordered) terms in the determinant, say $a_{1,2} \otimes a_{2,3} \otimes a_{3,1}$, appear as *diagonal elements* in a permutation of the rows of the *transposed* matrix.

In[7]:= $M^T[[\{2, 3, 1\}]]$

Out[7]=
$$\begin{pmatrix} a_{1,2} & a_{2,2} & a_{3,2} \\ a_{1,3} & a_{2,3} & a_{3,3} \\ a_{1,1} & a_{2,1} & a_{3,1} \end{pmatrix}$$

Tr[*list*, *f*] finds a generalized trace, combining terms with *f* instead of **Plus**.

In[8]:= **Tr**[% , CircleTimes]

Out[8]= $a_{1,2} \otimes a_{2,3} \otimes a_{3,1}$

As an aside, Robert Knapp (rknapp@wolfram.com) writes: Here are two user-defined functions for computing the trace of a matrix.

In[9]:= **tr1**[A_] := $\sum_{i=1}^{\text{Length}[A]} A[[i, i]]$;

```

In[10]:= tr2[A_] := Plus @@ Transpose[A, {1, 1}];
In[11]:= A = Table[Random [], {500}, {500}];
In[12]:= Do[tr1[A], {1000}] // Timing // First
Out[12]= 5.62 Second
In[13]:= Do[tr2[A], {1000}] // Timing // First
Out[13]= 1.25 Second

```

Note that **Tr** is much, much faster.

```

In[14]:= Do[Tr[A], {1000}] // Timing // First
Out[14]= 0.05 Second

```

The timings here are small, but if you have an application that requires repeatedly computing a trace, it makes a difference.

Second, **Tr** generalizes to multiple dimensions.

```

In[15]:= Tr[{a, b}]
Out[15]= a + b
In[16]:= Tr[{a, b}, {c, d}]
Out[16]= a + d
In[17]:= Tr[{{a, b}, {c, d}}, {{e, f}, {g, b}}]
Out[17]= a + b

```

One can also use **Tr** to project out the list of diagonal elements.

```

In[18]:= Tr[{a, b}, {c, d}], List
Out[18]= {a, d}

```

Here is a cute way to use it to speed up something very common.

```

In[19]:= mean1[x_] := 
$$\frac{\text{Apply}[\text{Plus}, x]}{\text{Length}[x]}$$

In[20]:= mean2[x_] := 
$$\frac{\text{Tr}[x]}{\text{Length}[x]}$$

In[21]:= x = Flatten[A];
In[22]:= mean1[x] // Timing
Out[22]= {0.77 Second, 0.50025}
In[23]:= mean2[x] // Timing
Out[23]= {0.01 Second, 0.50025}

```

The thing to note here is that **Tr**[*list of numbers*] does the same thing as **Apply**[**Plus**, *list of numbers*], but is much faster for packed arrays.

■ References

- [1] H. Aslaksen, "Quaternionic determinants," *The Mathematical Intelligencer*, **18**, 1996 pp. 56–65.

Visualization of Riemann Surfaces IIc

Compositions of Elementary Transcendental Functions

Michael Trott

*Wolfram Research, Inc.
mtrott@wolfram.com*

This is the next to last installment of the second series on the visualization of Riemann surfaces of compositions of elementary transcendental functions. In this installment we will develop and implement a general procedure for the calculation of the Riemann surfaces of compositions of elementary functions.

■ Outline of the General Construction

The previous installment showed that the main obstacles in constructing a smooth surface of all the sheets are the branch cuts of radicals, **Root**-objects, **arctrig**, and **archyperbolic** functions with nontrivial arguments. To avoid discontinuities we would have to restrict carefully the regions in which we calculate the function values. On the other hand, branch cuts can be chosen arbitrarily to a large extent, and only the locations of the branch points represent an inherent property and are well defined (i.e., branch cuts have to connect branch points, but how exactly a branch cut is chosen in the principle sheet between the branch points does not matter—at least as long as we do not form knotted ones). That is why we will completely avoid using closed-form formulas and branch cuts. Instead we will use **NDSolve** to calculate smooth patches inside branch-point-free regions of the complex z -plane by solving smooth differential equations that have smooth solutions. Using this treatment we completely *avoid* dealing with branch cuts (which do not exist on a Riemann surface anyway). The patches will be chosen so that the branch points (which must be treated carefully, as two or more sheets are glued together there) are always at their vertices. Having the branch points as vertices of the patches also allows a good spatial resolution near the branch points. This spatial resolution is important because $w(z)$ often varies

most near branch points. Using branch points as the vertices of the patches will result in a smooth surface, self-intersecting in 3D, that represents the Riemann surface of the function.

To construct the Riemann surface, we will proceed with the following partially symbolic, partially numerical steps.

1. All branch points come from either arc trig functions, logarithms, or radicals. For a uniform treatment, we convert all arc trig functions into a representation using only logarithms (the function **TrigToExp** comes in handy here). We then extract all log-terms, radicals, and **Root**-objects and determine their branch points. These are the (potential) branch points of the function under consideration. The branch points will be determined using **Solve**. (This means that if **Solve** cannot find all the roots for certain equations, not all branch points will be known, so we cannot generate a picture of the corresponding Riemann surface. In principle one could use numerical root finding techniques, but making sure that all branch points are calculated is a highly nontrivial issue and outside the scope of this article.)
2. We divide the complex z -plane radially and azimuthally into sectors generated by the outer product of the radial and azimuthal coordinates of the branch points, so that every sector has no interior branch points.
3. We derive a set of coupled nonlinear ordinary differential equations describing $w'_j(z)$ as a rational function in z and the $w_k(z)$ for all radical and log expressions.
4. We generate some possible sheets of the function. For radicals and **Root**-objects we generate all possible sheets, and for logarithmic terms we use n (by default we will use three sheets) out of the infinitely many sheets.
5. Starting quite near the branch points, we solve the differential equation radially outward for all sheets of every sector. Here it is important to avoid calculating sheets more than once. (A trivial case in which one might calculate more sheets than needed would be the uncollapsed sum $f(z) = (\sqrt{z} - 1)^2 + (\sqrt{z} + 1)^2$, a function that has only one sheet, not $2 \times 2 = 4$ sheets.)
6. Starting from the radial solutions of the differential equations constructed in step 3, we solve the differential equation azimuthally inside the sectors constructed in step 2. These solutions result in parametrized parts (parametrized by the radius r and the angle φ) of all sheets for each sector.
7. We generate polygons from the parametrizations for all sheets and all sectors.
8. We display all polygons together. (After we have finished, we might cut holes into the polygons for better visibility of the inner parts of a Riemann surface.)

■ Implementing the General Construction

```
In[1]:= Off[General::spell]; Off[General::spell1];
$HistoryLength = 0;
$DefaultFont = {"WriCMTT9", 6};
```

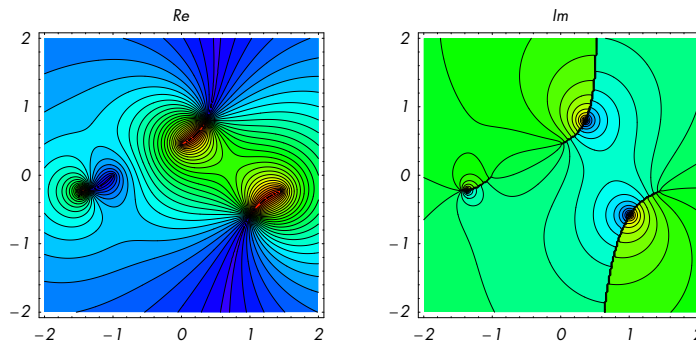
Let us demonstrate the steps outlined above using the following not-too-simple function.

```
In[4]:= wExample = ArcTan[ $\sqrt{\frac{z^3 - 2z + i}{z + 1}}$ ];
```

A contour plot shows that **wExample** has a complicated branch cut structure. As mentioned in the previous installment, in a contour plot the branch cuts show up as clusters of contour lines.

```
In[5]:= Show[GraphicsArray[
  ContourPlot[Evaluate[#][wExample] /. z -> x + i y], {x, -2, 2},
    {y, -2, 2}, PlotPoints -> 200, ColorFunction -> (Hue[0.7 #] &),
    Contours -> 30, ContourStyle -> {{Thickness[0.00001]}},
    DisplayFunction -> Identity, PlotLabel -> #] & /@ {Re, Im}]];
```

From In[5]:=



We canonicalize all functions by writing them explicitly in logarithms. This rewriting allows for a more uniform treatment of all functions. We will not have to deal separately with all of the individual **Arc*** functions. At the same time, we do not want meromorphic functions like **Sin** to be rewritten as a sum of exponentials, because potentially this could multiply the number of logarithmic terms. Instead we apply the function **TrigToExp** only to the **Arc*** functions.

```
In[6]:= ArcTrigFunctionQ[f_] :=
  MatchQ[f, ArcSin | ArcCos | ArcTan | ArcCot | ArcSec | ArcCsc |
    ArcSinh | ArcCosh | ArcTanh | ArcCoth | ArcSech | ArcCsch]

In[7]:= canonicalizeFunction[w_] :=
  w /. f_?ArcTrigFunctionQ[x_] -> (Evaluate[TrigToExp[f[#]]] &)[x])
```

Here are two examples of canonicalized functions.

```
In[8]:= canonicalizeFunction@ArcSin[ArcSin[z]]
```

$$\text{Out[8]} = -i \log \left[\log \left[i z + \sqrt{1 - z^2} \right] + \sqrt{1 + \log \left[i z + \sqrt{1 - z^2} \right]^2} \right]$$

```
In[9]:= wcan = canonicalizeFunction[wExample]
```

$$\text{Out[9]} = \frac{1}{2} i \log \left[1 - i \sqrt{\frac{i - 2z + z^3}{1 + z}} \right] - \frac{1}{2} i \log \left[1 + i \sqrt{\frac{i - 2z + z^3}{1 + z}} \right]$$

This representation shows the source of the three poles visible in the above contour plot. They are caused by the arguments of the logarithms when the arguments become zero. This event happens at exactly three z -values.

$$\text{In[10]} := \text{Solve} \left[1 + i \sqrt{\frac{z^3 - 2z + i}{1 + z}} == 0 \mid \mid 1 + i \sqrt{\frac{z^3 - 2z + i}{1 + z}} == 0, z \right] // N$$

$$\text{Out[10]} = \{ \{z \rightarrow 1.00996 - 0.580274 i\}, \{z \rightarrow 0.35895 + 0.798905 i\}, \{z \rightarrow -1.36891 - 0.218631 i\} \}$$

The next step is to determine all potential branch points. (I say *potential* branch points because in some situations they might turn out not to be branch points, but including them does not hurt in any way.) To do this we select all z -dependent log terms and power terms (with fractional exponents). The function **multiValuedTerms** returns a list of all multivalued parts of the function under consideration. We are looking for the multivalued functions of the form $\ln(f(z))$, $f(z)^{\frac{p}{q}}$, $f(z)^{g(z)}$, and $\text{Root}[f(z), j]$.

```
In[11]:= multiValuedTerms[w_, z_] :=
  Union[Cases[canonicalizeFunction[w],
    Log[_?(MemberQ[#, z, {0, ∞}] &)] |
    Power[_?(MemberQ[#, z, {0, ∞}] &),
      _Rational | _?(MemberQ[#, z, {0, ∞}] &)] |
    Root[_?(MemberQ[#, z, {0, ∞}] &), __], {0, ∞}]]
```

Here are two examples of the result of applying **multiValuedTerms**.

```
In[12]:= multiValuedTerms[z^z + 3^z + (2 z + 1)^Sin[z], z]
```

$$\text{Out[12]} = \{z^z, (1 + 2z)^{\sin[z]}\}$$

```
In[13]:= multiValuedTerms[Root[-1 + 2 z # - #^5 &, 3] + Log[z] - Log[z]^Log[z], z]
```

$$\text{Out[13]} = \{\log[z], \log[z]^{\log[z]}, \text{Root}[1 - 2z\#1 + \#1^5 \&, 3]\}$$

Here are the multivalued terms of our example function.

```
In[14]:= multiValuedTerms[wExample, z]
```

$$\text{Out[14]} = \left\{ \sqrt{\frac{i - 2z + z^3}{1 + z}}, \log \left[1 - i \sqrt{\frac{i - 2z + z^3}{1 + z}} \right], \log \left[1 + i \sqrt{\frac{i - 2z + z^3}{1 + z}} \right] \right\}$$

The branch points are then given as the zeros and infinities of these expressions, because a rational function can map the branch point at infinity to a finite z -value. (An example of this situation is $w(z) = \sqrt{\frac{z+1}{z-1}}$, where at $z = +1$ the ∞ -

branch point of the square root function reappears.) We equate the denominator and the numerator independently to 0. When calling **Solve**, error messages may be generated (either because the equations could not be solved at all or because not all branch points could be found). If messages were generated, we give up.

```
In[15]:= branchPoints[w_, z_] :=
  If[TrueQ[And @@ (NumericQ /@ DeleteCases[#, z | _DirectedInfinity])],
    DeleteCases[#, z | _DirectedInfinity], $Failed] & [Check[
    Union[Flatten[branchPointsF[#, z] & /@ multiValuedTerms[w, z]],
    $Failed,
    (* catch the following messages from Solve *)
    Solve::tdep, Solve::ifun]]
```

The branch points of $\ln(f(z))$ are the values of z such that $f(z) = 0$ or $f(z) = \infty$.

```
In[16]:= branchPointsF[Log[f_], z_] := z /. Solve[f == 0 || 1/f == 0, z]
```

The branch points of $f(z)^{\frac{p}{q}}$ and $f(z)^{g(z)}$ are the values of z such that $f(z) = 0$ or $f(z) = \infty$.

```
In[17]:= branchPointsF[Power[f_, _], z_] := z /. Solve[f == 0 || 1/f == 0, z]
```

The branch points of **Root**[$f(z)$, j] are the values of z such that $f(z)(w)$ has a multiple root with respect to w . (Here $f(z)$ is a pure function.)

```
In[18]:= branchPointsF[Root[f_, _], z_] :=
  Module[{w, p}, p = f[w]; (z /. Solve[Resultant[p, D_w p, w] == 0, z])]
```

Here are some examples of **branchPoints** at work. In the first example the $z = 0$ branch point originates from the **Log** term, and the other five branch points come from the **Root**-object.

```
In[19]:= branchPoints[Root[-1 + 2 z # - #^5 &, 3] + Log[z], z]
```

```
Out[19]= {0, -5/4 (-1/2)^(3/5), 5/(4 2^(3/5)), -5(-1)^(1/5)/(4 2^(3/5)), 5(-1)^(2/5)/(4 2^(3/5)), 5(-1)^(4/5)/(4 2^(3/5))}
```

We encourage the reader to carefully examine the following values for the potential branch points and to note from which functions they arise.

```
In[20]:= branchPoints[z^z + 3^z + (2 z + 1)^(Sin[z]), z]
```

```
Out[20]= {-1/2, 0}
```

```
In[21]:= branchPoints[1/(z + (z^(3/2) - (z - 1)^(-3/2))^2), z]
```

```
Out[21]= {0, 1}
```

```
In[22]:= branchPoints[Root[#^5 + #^3 - 2 Log[z] # + 7 &, 3], z]
```

```
Out[22]= {0, e^(Root[-7508417-88200 #1-392000 #1^2+128 #1^3+2048 #1^4+8192 #1^5 &, 1]),
  e^(Root[-7508417-88200 #1-392000 #1^2+128 #1^3+2048 #1^4+8192 #1^5 &, 2]),
  e^(Root[-7508417-88200 #1-392000 #1^2+128 #1^3+2048 #1^4+8192 #1^5 &, 3])}
```

```
In[23]:= branchPoints[Log[z^(1/3)], z]
```

```
Out[23]= {0}
```

In[24]:= **branchPoints**[z^z , z]

Out[24]= {0}

In[25]:= **branchPoints**[**Root**[-1 + 2 z # - #⁵ &, 3] + $\frac{z}{4}$, z]

Out[25]= $\left\{-\frac{5}{4}\left(-\frac{1}{2}\right)^{3/5}, \frac{5}{4 \cdot 2^{3/5}}, -\frac{5(-1)^{1/5}}{4 \cdot 2^{3/5}}, \frac{5(-1)^{2/5}}{4 \cdot 2^{3/5}}, \frac{5(-1)^{4/5}}{4 \cdot 2^{3/5}}\right\}$

The following function is meromorphic in z and does not have any branch points.

In[26]:= **branchPoints**[$\frac{\text{Sin}[z]}{z^3 - 1} + \sqrt{\pi} e^z$, z]

Out[26]= {}

We get the following branch points for our example function **wExample**.

In[27]:= **bps** = **branchPoints**[**wExample**, z]

Out[27]= $\left\{-1, \frac{\left(\frac{1}{2}\left((-9-9i)+\sqrt{-12+162i}\right)\right)^{1/3}}{3^{2/3}} + \frac{1}{\left(\frac{3}{2}\left((-9-9i)+\sqrt{-12+162i}\right)\right)^{1/3}},$
 $-\frac{1-i\sqrt{3}}{2^{2/3}\left(3\left((-9-9i)+\sqrt{-12+162i}\right)\right)^{1/3}} -$
 $\frac{\left(\frac{1}{2}\left((-9-9i)+\sqrt{-12+162i}\right)\right)^{1/3}(1+i\sqrt{3})}{2 \cdot 3^{2/3}},$
 $-\frac{\left(\frac{1}{2}\left((-9-9i)+\sqrt{-12+162i}\right)\right)^{1/3}(1-i\sqrt{3})}{2 \cdot 3^{2/3}} -$
 $\frac{1+i\sqrt{3}}{2^{2/3}\left(3\left((-9-9i)+\sqrt{-12+162i}\right)\right)^{1/3}},$
 $-2(-1)^{5/6}\left(\frac{2}{3(-9+\sqrt{177})}\right)^{1/3} + \frac{(-1)^{1/6}\left(\frac{1}{2}(-9+\sqrt{177})\right)^{1/3}}{3^{2/3}},$
 $(-1)^{5/6}(1+i\sqrt{3})\left(\frac{2}{3(-9+\sqrt{177})}\right)^{1/3} -$
 $\frac{(-1)^{1/6}(1-i\sqrt{3})\left(\frac{1}{2}(-9+\sqrt{177})\right)^{1/3}}{2 \cdot 3^{2/3}},$
 $(-1)^{5/6}(1-i\sqrt{3})\left(\frac{2}{3(-9+\sqrt{177})}\right)^{1/3} -$
 $\left.\frac{(-1)^{1/6}(1+i\sqrt{3})\left(\frac{1}{2}(-9+\sqrt{177})\right)^{1/3}}{2 \cdot 3^{2/3}}\right\}$

We transform all exact numeric quantities into approximate floating point numbers. All of the further work is mainly numerical in nature anyway.

```
In[28]:= (bps = bps // N[#, 30] &) // N
```

```
Out[28]= {-1., 1.00996 - 0.580274 i, 0.35895 + 0.798905 i, -1.36891 - 0.218631 i,
  1.46771 - 0.226699 i, -1.46771 - 0.226699 i, 0. + 0.453398 i}
```

Now that we have found the finite branch points, we divide the complex z -plane into sectors between them by using a tensor product structure formed by the radial and azimuthal positions of the branch points. We use a polar coordinate system for doing this. (Of course, one could also use a Cartesian one—we leave this to the interested reader.) The sectors returned from **sectors** are strictly inside the areas bounded by either the potential branch points (achieved by adding/subtracting a small ε to all coordinate values at branch points) or regular points. For a graphical representation of the Riemann surface, which is the only goal here, one will usually not see the resulting little gaps.

```
In[29]:= sectors[branchPoints_] :=
Module[{ε = 10-5, rMax, rList, φList},
  rMax = If[# == 0., 1, 14/10 #] &[
    Max[Abs[rList = Abs /@ N[branchPoints, 25]]]];
  (* the different distances *)
  rList = Union[Prepend[Append[rList, rMax], ε],
    SameTest → (Abs[#1 - #2] < 2 ε &)];
  (* the different angles *)
  φList =
    Union[If[# == 0., (* bignum zero *) 0 `` 25, Arg[#]] & /@ branchPoints,
      SameTest → (Abs[#1 - #2] < ε &)];
  φList = Append[φList, First[φList] + 2 π];
  (* the different sectors *)
  N[#, 25] &@Flatten[
    Table[{rList[[i]] + ε, φList[[j]] + ε, rList[[i + 1]] - ε, φList[[j + 1]] - ε},
      {i, Length[rList] - 1}, {j, Length[φList] - 1}], 1]]
```

These are the sectors originating from the example function.

```
In[30]:= (s1 = sectors[bps]) // Length
```

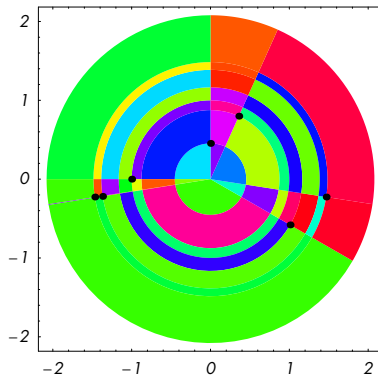
```
Out[30]= 49
```

Let us visualize the sectors together with the branch points by coloring each sector with a randomly chosen color. The function **sectorPolygon** generates 2D polygons representing the individual sectors.

```
In[31]:= sectorPolygon[{r1_, φ1_, r2_, φ2_}] :=
  With[{pp = 30},
    Polygon[Join[
      Table[r1 {Cos[φ ], Sin[φ ]}, {φ, φ1, φ2, (φ2 - φ1) / pp}],
      Table[r {Cos[φ2], Sin[φ2]}, {r, r1, r2, (r2 - r1) / pp}],
      Table[r2 {Cos[φ ], Sin[φ ]}, {φ, φ2, φ1, (φ1 - φ2) / pp}],
      Table[r {Cos[φ1], Sin[φ1]}, {r, r2, r1, (r1 - r2) / pp}]]]]
```

```
In[32]:= Show[Graphics[{
  {Hue[Random[]], sectorPolygon[#]} & /@ s1,
  PointSize[0.02], Point[{Re[#], Im[#]}] & /@ bps}],
  PlotRange -> All, Frame -> True, AspectRatio -> Automatic];
```

From In[32]:=



Now we must consider all the multivalued terms in our function $w(z)$ to properly continue them analytically. Be aware that the sector boundaries have nothing in common with the branch cuts of the original functions; only the branch points coincide. The actual continuation will be carried out in this way.

- a. Radicals will be continued according to $\sqrt[q]{\xi(z)} \Rightarrow e^{\frac{p}{q} 2\pi i} \sqrt[q]{\xi(z)}$, $0 \leq p \leq q - 1$.
- b. Logarithms will be continued according to $\log(\xi(z)) \Rightarrow \log(\xi(z)) + 2k\pi i$, $-\infty < k < \infty$. We will use only a finite number of sheets—typically three—later on for the graphics.
- c. Powers of the form $\xi(z)^{\eta(z)}$ will be continued by rewriting them using logarithms as $\xi(z)^{\eta(z)} \Rightarrow e^{\ln(\xi(z))\eta(z)}$. This reduces this case to case b.
- d. **Root** objects will be continued according to $\mathbf{Root}[f, j] \Rightarrow \mathbf{Root}[f, i]$, $1 \leq i \leq \deg_x f(x)$.

To differentiate the various multivalued terms, we give each of them an “index,” like **Power**₁[..., 1/3] or **Log**₂[...]. This index allows us to keep track of them uniquely later on. This is important because equal-looking terms could give rise to different sheets, like in the example $w(z) = \sqrt{z} + \sin(\sqrt{z})$. The two square roots are unrelated and each one generates two branches, so that the resulting Riemann surface has four sheets.

```

In[33]:= numberMultiValuedTerms[w_, z_] :=
Module[{root, function, ω},
Block[{i = 1},
MapAll[Which[MatchQ[#, Log[_?(MemberQ[#, z, {0, ω}] &)]],
(* the Log terms *) Logi++ @@ #,
MatchQ[#, Power[_?(MemberQ[#, z, {0, ω}] &), _Rational]],
(* the rational power terms *) Poweri++ @@ #,
MatchQ[#, Power[_?(MemberQ[#, z, {0, ω}] &),
_?(MemberQ[#, z, {0, ω}] &)]],
(* the f[z]^g[z] terms *) Exp[Logi++ [#[[1]]] #[[2]]],
MatchQ[#, root[_?(MemberQ[#, z, {0, ω}] &), _]],
(* the Root terms *) rooti++ @@ #,
True, #] &,
(* some preprocessing for Root objects *)
w /. Root[f_, _] := root[f, Exponent[f[ω], ω]] /.
(* neutralizing the Function in Root *) Function → function] //.
(* some postprocessing for Root objects *)
{function → Function, root → Root}
]]

```

Here are all multivalued terms numbered in **wExample**.

```

In[34]:= wnum = numberMultiValuedTerms[wcan, z]

```

$$\begin{aligned}
\text{Out[34]} = & \frac{1}{2} i \text{Log}_2 \left[1 - i \text{Power}_1 \left[\frac{i - 2z + z^3}{1 + z}, \frac{1}{2} \right] \right] - \\
& \frac{1}{2} i \text{Log}_4 \left[1 + i \text{Power}_3 \left[\frac{i - 2z + z^3}{1 + z}, \frac{1}{2} \right] \right]
\end{aligned}$$

The numbering for two other examples is as follows.

```

In[35]:= numberMultiValuedTerms[Sin[z] + Root[#7 z - z + 1 &, 2] +  $\sqrt[5]{z}$  + Log[2 z], z]

```

$$\text{Out[35]} = \text{Sin}[z] + \text{Log}_2[2z] + \text{Power}_1 \left[z, \frac{1}{5} \right] + \text{Root}_3[1 - z + z \#1^7 \&, 7]$$

```

In[36]:= numberMultiValuedTerms[zz + 3z + (2 z + 1)Sin[z], z]

```

$$\text{Out[36]} = 3^z + e^{z \text{Log}_1[z]} + e^{\text{Sin}[z] \text{Log}_2[1 + 2z]}$$

Now we extract all indexed terms from the function and collect them in a list.

```

In[37]:= powerAndLogTerms[w_, z_] :=
Module[{powerTerms, logTerms, sortedPowerAndLogTerms},
powerTerms = Cases[w, Power[_], {0, ω}];
logTerms = Cases[w, Log[_], {0, ω}];
rootTerms = Cases[w, Root[_], {0, ω}];
Sort[Join[powerTerms, logTerms, rootTerms], #1[[0, 2]] < #2[[0, 2]] &]]

```

```

In[38]:= plts = powerAndLogTerms[wnum, z]

```

$$\begin{aligned}
\text{Out[38]} = & \left\{ \text{Power}_1 \left[\frac{i - 2z + z^3}{1 + z}, \frac{1}{2} \right], \text{Log}_2 \left[1 - i \text{Power}_1 \left[\frac{i - 2z + z^3}{1 + z}, \frac{1}{2} \right] \right], \right. \\
& \left. \text{Power}_3 \left[\frac{i - 2z + z^3}{1 + z}, \frac{1}{2} \right], \text{Log}_4 \left[1 + i \text{Power}_3 \left[\frac{i - 2z + z^3}{1 + z}, \frac{1}{2} \right] \right] \right\}
\end{aligned}$$

Each of the numbered multivalued functions is now replaced by a symbolic function $w_i(z)$. After rewriting them into a nicer form, we can deal with them numerically later on inside **NDSolve**.

```
In[39]:= substitutions[powerAndLogTerms_, wFunc_, w_, z_] :=
Module[{ws = (# -> w[[0, 2]] [z]) & /@ powerAndLogTerms},
{Last /@ FoldList[Append[#1, #2 /. #1] &, {First[ws]}, Rest[ws]],
ws, wFunc /. ws}]
```

The result of **substitutions** is a list with three elements. The first element is a list of the replacements of the multivalued functions by the w_i . The second element is a similar list without the recursive back-substitution of the w_i . The third element is the original function expressed through the w_i s. For our example function application of **substitutions** results in the following six substitutions.

```
In[40]:= subs = substitutions[plts, wnum, w, z]
Out[40]= {{Power1[ $\frac{i-2z+z^3}{1+z}$ ,  $\frac{1}{2}$ ] -> w1[z], Log2[1 - i w1[z]] -> w2[z],
Power3[ $\frac{i-2z+z^3}{1+z}$ ,  $\frac{1}{2}$ ] -> w3[z], Log4[1 + i w3[z]] -> w4[z]},
{Power1[ $\frac{i-2z+z^3}{1+z}$ ,  $\frac{1}{2}$ ] -> w1[z], Log2[1 - i Power1[ $\frac{i-2z+z^3}{1+z}$ ,  $\frac{1}{2}$ ]] ->
w2[z], Power3[ $\frac{i-2z+z^3}{1+z}$ ,  $\frac{1}{2}$ ] -> w3[z],
Log4[1 + i Power3[ $\frac{i-2z+z^3}{1+z}$ ,  $\frac{1}{2}$ ]] -> w4[z]},  $\frac{1}{2} i w2[z] - \frac{1}{2} i w4[z]$ }
```

We now generate differential equations from the last list of substitutions. Doing this seems like a lot of work, considering that we already know the solution of these differential equations—these are just the values at the other sheets—but it is unavoidable. Although we know the location of the branch points, we do not know the detailed location of the branch cuts. So our solution might not represent the intended sheet. The branch cuts might form complicated patterns inside the sectors calculated above, and it would be very difficult or virtually impossible to construct the “correct” sheet purely symbolically. (Consider $\log(aComplicatedPolynomial)$). The branch points are just the zeros of $aComplicatedPolynomial$, but the branch cuts will typically wind through the individual sectors as complicated curves.) The resulting system of differential equations is then of the form $w'_j(z) = b_j(w_1(z), \dots, w_n(z), z)$ where all of the b_j are now meromorphic in all arguments, and no multivaluedness is present anymore in the solution of the differential equation system for the given initial conditions.

We obtain the differential equation for $w(z) = \ln(f(z))$ by using straightforward differentiation.

```
In[41]:= toODE[w : (Log[_] -> w_), z_] := {w,  $\partial_z w == Together[\frac{\partial_z \xi}{\xi}]$ }
```

We use implicit differentiation to obtain the differential equation, meromorphic in w and z for $w(z) = f(z)^{\frac{p}{q}}$.

```
In[42]:= toODE[ω : (Power[f_, Rational[p_, q_]] → w_), z_] :=
      {ω, ∂z w == Together[ $\frac{p}{q} w \frac{\partial_z f}{f}$ ]}
```

We obtain the differential equation for $w(z) = \text{Root}[f(z), j]$ by implicit differentiation of $f(z)(w(z)) = 0$. Again $f(z)$ is a pure function. Observe the parentheses in $(\partial_z \xi)[w]$. First we differentiate the pure function ξ which parametrically depends on z with respect to z , and then we apply the resulting pure function $\partial_z \xi$ to the argument w .

```
In[43]:= toODE[ω : (Root[ξ_, d_] → w_), z_] := {ω, ∂z w == Together[- $\frac{(\partial_z \xi)[w]}{\partial_z \xi[w]}$ ]}
```

Here are the differential equations corresponding to **subs**.

```
In[44]:= (odes1 = toODE[#, z] & /@ subs[1]) // TableForm
```

```
Out[44]//TableForm=
```

$\text{Power}_1\left[\frac{i-2z+z^3}{1+z}, \frac{1}{2}\right] \rightarrow w_1[z]$	$w'_1[z] == \frac{((-2-i)+3z^2+2z^3)w_1[z]}{2(1+z)(i-2z+z^3)}$
$\text{Log}_2[1-iw_1[z]] \rightarrow w_2[z]$	$w'_2[z] == \frac{w'_1[z]}{i+w_1[z]}$
$\text{Power}_3\left[\frac{i-2z+z^3}{1+z}, \frac{1}{2}\right] \rightarrow w_3[z]$	$w'_3[z] == \frac{((-2-i)+3z^2+2z^3)w_3[z]}{2(1+z)(i-2z+z^3)}$
$\text{Log}_4[1+iw_3[z]] \rightarrow w_4[z]$	$w'_4[z] == \frac{w'_3[z]}{-i+w_3[z]}$

Here the differential equation corresponding to a **Root**-object is calculated.

```
In[45]:= toODE[Rooti[#5-2z+#z2-1 &, 5] → W[z], z]
```

```
Out[45]= {Rooti[#5-2z+#z2-1 &, 5] → W[z], W'[z] == - $\frac{2(-1+zW[z])}{z^2+5W[z]^4}$ }
```

The right-hand sides of the last system of differential equations often still contain the derivatives of some of the $w_i(z)$. We recursively insert the corresponding right-hand sides to get a system of differential equations with rational right-hand sides that can be solved directly by **NDSolve**.

```
In[46]:= derivativeFreeRhs[odes_] :=
      Apply[Equal, Last /@ Transpose[{First /@ odes,
      Last /@ FoldList[Append[#1, #2 /. #1] &, {First[#]}, Rest[#]] &[
      Apply[Rule, Last /@ odes, {1}]]}], {1}]
```

```
In[47]:= (odes2 = derivativeFreeRhs[odes1]) // TableForm
```

```
Out[47]//TableForm=
```

$w'_1[z] == \frac{((-2-i)+3z^2+2z^3)w_1[z]}{2(1+z)(i-2z+z^3)}$
$w'_2[z] == \frac{((-2-i)+3z^2+2z^3)w_1[z]}{2(1+z)(i-2z+z^3)(i+w_1[z])}$
$w'_3[z] == \frac{((-2-i)+3z^2+2z^3)w_3[z]}{2(1+z)(i-2z+z^3)}$
$w'_4[z] == \frac{((-2-i)+3z^2+2z^3)w_3[z]}{2(1+z)(i-2z+z^3)(-i+w_3[z])}$

Let us look at the resulting system of differential equations for two more examples. Here is the differential equation system corresponding to $w(z) = \left(\sqrt[3]{z}\right)^{\sqrt[3]{z}}$. The function generates the system of differential equations in the form used to visualize Riemann surfaces.

```
In[48]:= makeOde[f_, z_] :=
  Function[ω, MapAt[derivativeFreeRhs[toODE[#, z] & /@#] &,
    substitutions[powerAndLogTerms[#, z], #, w, z] &[
      numberMultiValuedTerms[ω, z]], 1]] [f]
```

```
In[49]:= makeOde[( $\sqrt[3]{z}$ ) $\sqrt{z}$ , z]
```

```
Out[49]= {{w'_1[z] ==  $\frac{w_1[z]}{5z}$ , w'_2[z] ==  $\frac{1}{z}$ },
  {Power_1[z,  $\frac{1}{5}$ ] → w_1[z], Log_2[z] → w_2[z], e $\frac{1}{3} w_1[z] w_2[z]$ }}
```

Here are the systems of differential equations that corresponds to some more complicated functions.

```
In[50]:= makeOde[zz + 3z + (2z + 1)Sin[z] - Sin[z]4 +
  Root[#7 z - z + 1 &, 2]2 +  $\sqrt[5]{z}$  +  $\frac{\sqrt{\text{Log}[2z]}}{\sqrt{z} - 2}$ , z]
```

```
Out[50]= {{w'_1[z] ==  $\frac{w_1[z]}{5z}$ , w'_2[z] ==  $\frac{1}{z}$ , w'_3[z] ==  $\frac{2}{1+2z}$ , w'_4[z] ==  $\frac{w_4[z]}{2z}$ ,
  w'_5[z] ==  $\frac{1}{z}$ , w'_6[z] ==  $\frac{w_6[z]}{2z w_5[z]}$ , w'_7[z] ==  $\frac{1 - w_7[z]^7}{7z w_7[z]^6}$ },
  {Power_1[z,  $\frac{1}{5}$ ] → w_1[z], Log_2[z] → w_2[z], Log_3[1 + 2z] → w_3[z],
  Power_4[z,  $\frac{1}{2}$ ] → w_4[z], Log_5[2z] → w_5[z],
  Power_6[Log_5[2z],  $\frac{1}{2}$ ] → w_6[z], Root_7[1 - z + z #17 &, 7] → w_7[z]},
  3z + ez w_2[z] + eSin[z] w_3[z] - Sin[z]4 + w_1[z] +  $\frac{w_6[z]}{-2 + w_4[z]} + w_7[z]^2$ }
```

```
In[51]:= makeOde[( $\frac{1}{z}$ ) $\frac{1}{z}$ , z]
```

```
Out[51]= {{w'_1[z] ==  $-\frac{1}{z}$ , w'_2[z] ==  $-\frac{1 - w_1[z]}{z^2}$ },
  {Log_1[ $\frac{1}{z}$ ] → w_1[z], Log_2[e $\frac{\text{Log}_1[\frac{1}{z}]}{z}$ ] → w_2[z], e $\frac{w_2[z]}{z}$ }}
```

```
In[52]:= makeOde[Log[Log[ $\frac{1}{\text{Log}[\text{Log}[z]]}$ ]], z]
```

```
Out[52]= {{w'_1[z] ==  $\frac{1}{z}$ , w'_2[z] ==  $\frac{1}{z w_1[z]}$ ,
  w'_3[z] ==  $-\frac{1}{z w_1[z] w_2[z]}$ , w'_4[z] ==  $-\frac{1}{z w_1[z] w_2[z] w_3[z]}$ },
  {Log_1[z] → w_1[z], Log_2[Log_1[z]] → w_2[z], Log_3[ $\frac{1}{\text{Log}_2[\text{Log}_1[z]]}$ ] → w_3[z],
  Log_4[Log_3[ $\frac{1}{\text{Log}_2[\text{Log}_1[z]]}$ ]] → w_4[z]}, w_4[z]}
```

Now we must consider the sheets we actually want to display. When near a branch point (i.e., at the corners of our sectors), we will choose the sheets that we want to continue analytically. For radicals we will use all possible sheets (if the reader is going to use high order radicals one might want to restrict this). For logarithms we will use by default three neighboring sheets (the one “below” and the one “above” the “original” one). As with radicals, we take all sheets into account when using **Root**-objects.

```
In[53]:= sheets[ω : Poweri[ξ-, r : Rational[p-, q-]], ___] :=
```

```
Table[ω → e $\frac{2\pi i j}{q}$  ξr, {j, 0, q-1}];
```

```
In[54]:= sheets[ω : Logi[ξ-], k1-: -1, k2-: 1] :=
```

```
Table[ω → Log[ξ] + 2 k i π, {k, k1, k2}]
```

```
In[55]:= sheets[ω : Root-[ξ-, d-], ___] := Table[ω → Root[ξ, j], {j, 1, d}]
```

This method results in the following sheet realizations of our numbered functions.

```
In[56]:= sr = sheets /@ plts
```

$$\begin{aligned} \text{Out[56]} = & \left\{ \left\{ \text{Power}_1 \left[\frac{i - 2z + z^3}{1+z}, \frac{1}{2} \right] \rightarrow \sqrt{\frac{i - 2z + z^3}{1+z}}, \right. \right. \\ & \left. \text{Power}_1 \left[\frac{i - 2z + z^3}{1+z}, \frac{1}{2} \right] \rightarrow -\sqrt{\frac{i - 2z + z^3}{1+z}} \right\}, \\ & \left\{ \text{Log}_2 \left[1 - i \text{Power}_1 \left[\frac{i - 2z + z^3}{1+z}, \frac{1}{2} \right] \right] \rightarrow \right. \\ & \quad \left. -2i\pi + \text{Log} \left[1 - i \text{Power}_1 \left[\frac{i - 2z + z^3}{1+z}, \frac{1}{2} \right] \right], \right. \\ & \text{Log}_2 \left[1 - i \text{Power}_1 \left[\frac{i - 2z + z^3}{1+z}, \frac{1}{2} \right] \right] \rightarrow \text{Log} \left[1 - i \text{Power}_1 \left[\frac{i - 2z + z^3}{1+z}, \frac{1}{2} \right] \right], \\ & \text{Log}_2 \left[1 - i \text{Power}_1 \left[\frac{i - 2z + z^3}{1+z}, \frac{1}{2} \right] \right] \rightarrow \\ & \quad \left. 2i\pi + \text{Log} \left[1 - i \text{Power}_1 \left[\frac{i - 2z + z^3}{1+z}, \frac{1}{2} \right] \right] \right\}, \\ & \left\{ \text{Power}_3 \left[\frac{i - 2z + z^3}{1+z}, \frac{1}{2} \right] \rightarrow \sqrt{\frac{i - 2z + z^3}{1+z}}, \right. \\ & \left. \text{Power}_3 \left[\frac{i - 2z + z^3}{1+z}, \frac{1}{2} \right] \rightarrow -\sqrt{\frac{i - 2z + z^3}{1+z}} \right\}, \\ & \left\{ \text{Log}_4 \left[1 + i \text{Power}_3 \left[\frac{i - 2z + z^3}{1+z}, \frac{1}{2} \right] \right] \rightarrow \right. \\ & \quad \left. -2i\pi + \text{Log} \left[1 + i \text{Power}_3 \left[\frac{i - 2z + z^3}{1+z}, \frac{1}{2} \right] \right], \right. \\ & \text{Log}_4 \left[1 + i \text{Power}_3 \left[\frac{i - 2z + z^3}{1+z}, \frac{1}{2} \right] \right] \rightarrow \text{Log} \left[1 + i \text{Power}_3 \left[\frac{i - 2z + z^3}{1+z}, \frac{1}{2} \right] \right], \\ & \text{Log}_4 \left[1 + i \text{Power}_3 \left[\frac{i - 2z + z^3}{1+z}, \frac{1}{2} \right] \right] \rightarrow \\ & \quad \left. 2i\pi + \text{Log} \left[1 + i \text{Power}_3 \left[\frac{i - 2z + z^3}{1+z}, \frac{1}{2} \right] \right] \right\} \end{aligned}$$

Forming the outer product of all possible sheet realizations of all multivalued functions, we get all potential sheets.

```
In[57]:= allSheets[sheetRealizations_] :=
  Flatten[Outer[List, ##] &@@sheetRealizations,
    Length[sheetRealizations] - 1];
```

For our example function there are $3 \times 3 \times 2 \times 2 = 36$ potential sheets. Usually, however, we will not really have so many different sheets! When we calculate numerical values of the sheets (see below), some of the sheets might turn out to be numerically the same. A simple example is $w(z) = \arctan(\sqrt{z})$. This expression contains one square root to be continued, but the canonicalized form $\frac{1}{2} i (\log(1 - i \sqrt{z}) - \log(1 + i \sqrt{z}))$ contains two copies of the same square root which are differently numbered. Later on we will discuss how to avoid displaying the same sheet twice.

```
In[58]:= asp = allSheets[sr];
```

```
In[59]:= Length[asp]
```

```
Out[59]= 36
```

Here is the first realization.

```
In[60]:= asp[[1]]
```

$$\begin{aligned} \text{Out[60]} = & \left\{ \text{Power}_1 \left[\frac{i - 2z + z^3}{1 + z}, \frac{1}{2} \right] \rightarrow -\sqrt{\frac{i - 2z + z^3}{1 + z}}, \right. \\ & \text{Log}_2 \left[1 - i \text{Power}_1 \left[\frac{i - 2z + z^3}{1 + z}, \frac{1}{2} \right] \right] \rightarrow 2i\pi + \text{Log} \left[1 - i \text{Power}_1 \left[\frac{i - 2z + z^3}{1 + z}, \frac{1}{2} \right] \right], \\ & \text{Power}_3 \left[\frac{i - 2z + z^3}{1 + z}, \frac{1}{2} \right] \rightarrow -\sqrt{\frac{i - 2z + z^3}{1 + z}}, \\ & \left. \text{Log}_4 \left[1 + i \text{Power}_3 \left[\frac{i - 2z + z^3}{1 + z}, \frac{1}{2} \right] \right] \rightarrow 2i\pi + \text{Log} \left[1 + i \text{Power}_3 \left[\frac{i - 2z + z^3}{1 + z}, \frac{1}{2} \right] \right] \right\} \end{aligned}$$

Now we implement two functions that transform a given differential equation of the form $w'(z) = f(z)$ into one in which r or φ (r or φ being polar coordinates) stay constant. The dependent variable remains unchanged.

```
In[61]:= odeAzimuthal[w_{i-}[z_] == rhs_, {w_{i-}, z_}, {\psi_{-}, \varphi_{-}}, r_] :=
  \psi_i'[\varphi] == rhs \partial_{\varphi} (r e^{i\varphi}) /. w_{j-}[z] \to \psi_j[\varphi] /. z \to r e^{i\varphi}
```

```
In[62]:= odeRadial[w_{i-}[z_] == rhs_, {w_{i-}, z_}, {\psi_{-}, \rho_{-}}, d_] :=
  \psi_i'[\rho] == rhs \partial_{\rho} (\rho d) /. w_{j-}[z] \to \psi_j[\rho] /. z \to d \rho
```

We get the following differential equations for our example function.

```
In[63]:= odesφ = odeAzimuthal[#, {#[1, 0, 1], z}, {ψ, φ}, r] & /@ odes2
```

$$\begin{aligned} \text{Out[63]} = \left\{ \begin{aligned} \psi'_1[\varphi] &= \frac{i e^{i\varphi} r ((-2 - i) + 3 e^{2i\varphi} r^2 + 2 e^{3i\varphi} r^3) \psi_1[\varphi]}{2 (1 + e^{i\varphi} r) (i - 2 e^{i\varphi} r + e^{3i\varphi} r^3)}, \\ \psi'_2[\varphi] &= \frac{i e^{i\varphi} r ((-2 - i) + 3 e^{2i\varphi} r^2 + 2 e^{3i\varphi} r^3) \psi_1[\varphi]}{2 (1 + e^{i\varphi} r) (i - 2 e^{i\varphi} r + e^{3i\varphi} r^3) (i + \psi_1[\varphi])}, \\ \psi'_3[\varphi] &= \frac{i e^{i\varphi} r ((-2 - i) + 3 e^{2i\varphi} r^2 + 2 e^{3i\varphi} r^3) \psi_3[\varphi]}{2 (1 + e^{i\varphi} r) (i - 2 e^{i\varphi} r + e^{3i\varphi} r^3)}, \\ \psi'_4[\varphi] &= \frac{i e^{i\varphi} r ((-2 - i) + 3 e^{2i\varphi} r^2 + 2 e^{3i\varphi} r^3) \psi_3[\varphi]}{2 (1 + e^{i\varphi} r) (i - 2 e^{i\varphi} r + e^{3i\varphi} r^3) (-i + \psi_3[\varphi])} \end{aligned} \right\} \end{aligned}$$

```
In[64]:= odesr = odeRadial[#, {#[1, 0, 1], z}, {ψ, ρ}, d] & /@ odes2
```

$$\begin{aligned} \text{Out[64]} = \left\{ \begin{aligned} \psi'_1[\rho] &= \frac{d ((-2 - i) + 3 d^2 \rho^2 + 2 d^3 \rho^3) \psi_1[\rho]}{2 (1 + d \rho) (i - 2 d \rho + d^3 \rho^3)}, \\ \psi'_2[\rho] &= \frac{d ((-2 - i) + 3 d^2 \rho^2 + 2 d^3 \rho^3) \psi_1[\rho]}{2 (1 + d \rho) (i - 2 d \rho + d^3 \rho^3) (i + \psi_1[\rho])}, \\ \psi'_3[\rho] &= \frac{d ((-2 - i) + 3 d^2 \rho^2 + 2 d^3 \rho^3) \psi_3[\rho]}{2 (1 + d \rho) (i - 2 d \rho + d^3 \rho^3)}, \\ \psi'_4[\rho] &= \frac{d ((-2 - i) + 3 d^2 \rho^2 + 2 d^3 \rho^3) \psi_3[\rho]}{2 (1 + d \rho) (i - 2 d \rho + d^3 \rho^3) (-i + \psi_3[\rho])} \end{aligned} \right\} \end{aligned}$$

Now we will finally solve the coupled set of differential equations to get the Riemann surface. The function **sheetSectorPatch** generates the polygons for one sheet realization within one sector.

We set the **MaxSteps**, **PrecisionGoal**, and **AccuracyGoal** options of **NDSolve** to values higher than the default. This makes sure that the examples treated in the next section will work fine. For more complicated functions, it may be necessary to use more steps and a higher precision. The use of high-precision arithmetic via **WorkingPrecision** $\rightarrow$ *valueLargerThan\$MachinePrecision* might also be needed.

```
In[65]:= SetOptions[NDSolve, PrecisionGoal -> 12,
  AccuracyGoal -> 12, MaxSteps -> 5000];
```

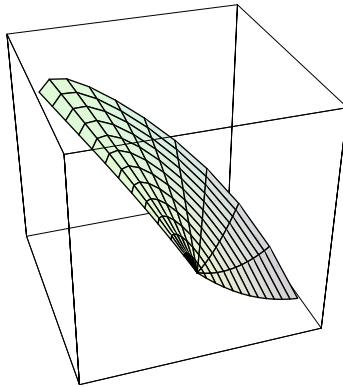
```
In[66]:= sectorSheetPatch[reIm_,
  {w_, z_}, wψ_, wSheet_, {odesφ_, {ψ_, φ_}, r_},
  {odesr_, {ψ_, ρ_}, dir_}, sector : {r1_, φ1_, r2_, φ2_},
  {ppφ_, ppr_}] :=
Module[{radialStartingValues, radialIFs,
  azimuthalIFs, points, ra, φa, rb, φb},
  (* get initial values for the differential equation *)
  radialStartingValues =
    N[({(Last/@wSheet) /. wSheet) /. z -> r1 Exp[I φ1], 25];
  (* the functions of the odes *)
  ψs = #[1, 0, 1] & /@ odesr;
  (* calculate initial conditions for radial odes *)
  radialInits = Apply[Equal,
    Transpose[{#[r1] & /@ ψs, radialStartingValues}], {1}];
```

```
(* solve the differential equation radially *)
radialIFs = NDSolve[
  Join[odesr, radialInits] /.  $d \rightarrow \text{Exp}[I \varphi 1]$ ,  $\psi s$ , { $\rho$ ,  $r1$ ,  $r2$ }]];
(* solve the differential equation azimuthally *)
azimuthalIFs = Table[ra =  $r1 + i / \text{ppr} (r2 - r1)$ ;
  NDSolve[Join[odes $\varphi$  /.  $r \rightarrow ra$ , Apply[Equal, Transpose[{ $\varphi 1$ ] & /@  $\psi s$ ,
    #[ra] & /@  $\psi s$  /. radialIFs[[1]]}], {1}]],  $\psi s$ , { $\varphi$ ,  $\varphi 1$ ,  $\varphi 2$ }],
    {i, 0, ppr}];
(* generate the function itself *)
 $\Psi = (w\psi /. w_i. [\xi\_ ] \rightarrow \psi_i [\varphi b] /. azimuthalIFs) /. z \rightarrow rb e^{i \varphi b}$ ;
(* calculate points for the patch *)
points = Table[rb =  $r1 + i / \text{ppr} (r2 - r1)$ ;  $\varphi b = \varphi 1 + k / \text{pp}\varphi (\varphi 2 - \varphi 1)$ ;
  {rb Cos[ $\varphi b$ ], rb Sin[ $\varphi b$ ], ReIm[ $\Psi[[i + 1, 1]]$ ]},
  {i, 0, ppr}, {k, 0, pp $\varphi$ }]];
(* make polygons for all sheets *)
Apply[Polygon[Join[#1, Reverse[#2]]] &,
  Transpose /@ Partition[Partition[#, 2, 1] & /@ points, 2, 1],
  {2}]]
```

The following shows one sheet inside the first sector for `wExample`.

```
In[67]:= sectorSheetPatch[Im, {w, z}, subs[[-1]], asp[[1],
  {odes $\varphi$ , { $\psi$ ,  $\varphi$ },  $r$ }, {odesr, { $\psi$ ,  $\rho$ },  $d$ },
  s1[[2]], {10, 12}] // Graphics3D // Show[#,
  BoxRatios  $\rightarrow$  {1, 1, 1}, ViewPoint  $\rightarrow$  {1, 3, 2}, PlotRange  $\rightarrow$  All] &;
```

From In[67]:=



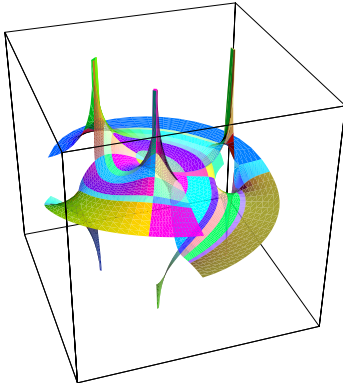
One sheet (not always the same) for all sectors is shown.

```

In[68]:= sheetPic = Show[Graphics3D[{EdgeForm[],
  {SurfaceColor[Hue[Random[]], Hue[Random[]], 3 Random[]], #} & /@
    (sectorSheetPatch[Im, {w, z}, subs[[-1]], asp[1]],
      {odes $\varphi$ , { $\psi$ ,  $\varphi$ }, r}, {odes $x$ , { $\psi$ ,  $\rho$ },  $d$ },
      #, {16, 6}] & /@ s1)}], BoxRatios -> {1, 1, 1},
  ViewPoint -> {1, 3, 2}];

```

From In[68]:=



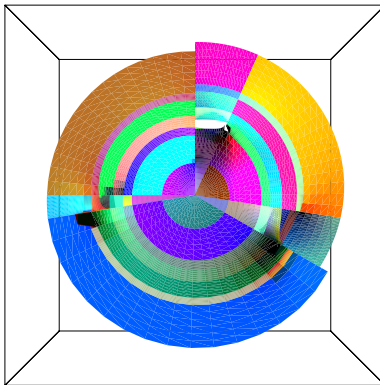
Here is a view of the surface patches from the last picture from “above.” The sector structure is clearly visible.

```

In[69]:= Show[sheetPic, ViewPoint -> {0, 0, 3}];

```

From In[69]:=



We now consider the generation of all sheets of interest inside one sector. The first step is to calculate the numerical values of the sheets at one of the corners of a sector. If two such values agree, then they represent the same sheet (both are generated by a differential equation without multiple solutions for given initial conditions). The built-in function **Union** relies on sorting and is not applicable for unioning complex numbers that do not form a well-ordered set. The following function **complexUnion** implements the needed unioning for the complex starting values. Be aware that the function **complexUnion** has complexity $O(n^2)$ in comparison to the much smaller complexity $O(n \ln(n))$ of **Union**—but for the short lists we have here, this speed/complexity difference does not matter.

```

In[70]:= complexUnion[l_, SameTest → func_] :=
  Fold[Function[{u, v}, If[Select[u, func[#, v] &, 1] === {},
    Append[u, v], u]], {First[#]}, Rest[#]] & [Union[l]]

```

The function **differentSheetRealizations** returns a list of all the different initial conditions present in *sheetRealizations* at the point $z = z_1$.

```

In[71]:= differentSheetRealizations[allSheets_, wnum_, z_, z1_] :=
  complexUnion[{#, N[(wnum /. #) /. z → z1, 25]} & /@ allSheets,
    SameTest → Function[{x, y},
      If[x[[2]] == 0, x[[2]] == y[[2]], Abs[(x[[2]] - y[[2]]) / x[[2]] < 10-20]]];

```

Now we generate all patches inside a given sector. After we apply **Re** or **Im** or the like to the solutions of the differential equations, some sheets might coincide. It is not a good thing to have a graphics expression containing nearly identical polygons because they intersect frequently and a lot of time is spent in calculating and rendering their intersections. To avoid this situation we implement a function **dropMultipleSheets** which eliminates such multiple polygons/sheets.

```

In[72]:= dropMultipleSheets[sheets_] :=
  Module[{sheetData, goodOnes, actSheet, ε = 10-3},
    (* take out the function values and number the sheets *)
    sheetData = MapIndexed[{#2[[1]], #1, Plus @@ Abs[Flatten[#1]]} &,
      Last /@ (Level[#, {-2}] & /@ sheets)];
    goodOnes = {};
    (* put all different ones in the list goodOnes *)
    While[sheetData != {},
      actSheet = sheetData[[1]];
      AppendTo[goodOnes, First[actSheet]];
      sameSheets = Select[sheetData,
        (* the relative difference of the sheet values *)
        Function[s,  $\frac{\text{Plus} @@ \text{Abs}[\text{actSheet}[[2]] - s[[2]]]}{\text{actSheet}[[3]] + s[[3]]} < \epsilon$ ]];
      sheetData = DeleteCases[sheetData, Alternatives @@ sameSheets];
    (* return the different sheets *)
    sheets[[goodOnes]]

```

The function **sectorSheetPatches** finally produces all sheets as specified by *sheetRealizations*.

```

In[73]:= sectorSheetPatches[reIm_, {w_, z_}, wψ_, wnum_, sheetRealizations_,
  {odesφ_, {ψ_, φ_}, r_}, {odesr_, {ψ_, ρ_}, dir_},
  sector : {r1_, φ1_, r2_, φ2_}, {ppφ_, ppr_}] :=
Module[{sheetValues, sheetPolys},
  (* the various log and radical sheet realizations *)
  sheetValues = First /@
    differentSheetRealizations[sheetRealizations, wnum, z, r1 ei φ1];
  (* calculating the sheet patches *)
  sheetPolys = sectorSheetPatch[reIm,
    {w, z}, wψ, #, {odesφ, {ψ, φ}, r}, {odesr, {ψ, ρ}, dir},
    sector, {ppφ, ppr}] & /@ sheetValues;
  dropMultipleSheets[sheetPolys]
]

```

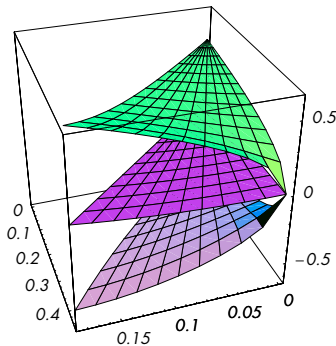
Here all the patches for our example function for the fifth sector are shown. One easily sees the branch point where the sheets touch each other.

```

In[74]:= Show[Graphics3D[{EdgeForm[Thickness[0.0001]],
  {SurfaceColor[Hue[Random[]], Hue[Random[]], 3 Random[]], #} & /@
  sectorSheetPatches[Im, {w, z}, subs[[-1]], wnum, asp,
    {odesφ, {ψ, φ}, r}, {odesr, {ψ, ρ}, d}, s1[[5], {10, 12}]]},
  BoxRatios -> {1, 1, 1}, ViewPoint -> {1, 3, 2}, Axes -> True];

```

From In[74]:=



The function **patchPlotPoints** divides the plot points in radial and azimuthal directions in a “fair” manner between the individual sectors.

```

In[75]:= patchPlotPoints[{r1_, φ1_, r2_, φ2_}, rMax_, {ppφ_, ppr_}] :=
  Max[#, 1] & /@ Round[{ppφ, ppr} {φ2 - φ1, r2 - r1} / {2 π, rMax}]

```

Now we finally have all the ingredients together to implement the function **RiemannSurface** which will generate the polygons of a few sheets of the Riemann surface of the function **w**. We add a few simple tests to keep inappropriate functions from getting passed along.

```

In[76]:= RiemannSurface::cantFindAllBranchPoints =
  "Unable to calculate all branchpoints for the function '1'.";

In[77]:= RiemannSurface::noBranchPointsPresent =
  "'1' is not a multivalued function. Simply use
  Plot3D or ParametricPlot3D to display the surface.";

```

```
In[78]:= RiemannSurface::containsFunctionCurrentlyNotTreated =
  "'1' contains the functions '2' which
  are not treated in this implementation.";
```

The function **notTreatedFunctions** returns a list of all functions that are not dealt with in this implementation of construction of Riemann surfaces. **foreignFunction** collects all functions that are not explicitly listed as “known” functions.

```
In[79]:= notTreatedFunctions[w_, z_] :=
Module[{foreignFunction, u},
  (* z is the independent variable *)
  foreignFunction[z] = {};
  (* u is the dummy variable needed for the Root pure functions *)
  foreignFunction[u] = {};
  (* numerical values *)
  foreignFunction[_?NumericQ] := {};
  (* recursively go down in Power, ArcTrigs, and Root *)
  foreignFunction[Power[b_, e_]] := foreignFunction/@{b, e};
  foreignFunction[pt_Plus | pt_Times] :=
    foreignFunction/@(List@@pt);
  (* the known numeric functions *)
  Map[(foreignFunction[#f_] = foreignFunction[f]) &,
    {Exp, Log, Sin, Cos, Tan, Cot, Sec, Csc,
     Sinh, Cosh, Tanh, Coth, Sech, Csch,
     ArcSin, ArcCos, ArcTan, ArcCot, ArcSec, ArcCsc,
     ArcSinh, ArcCosh, ArcTanh, ArcCoth, ArcSech, ArcCsch}];
  (* the Root case *)
  foreignFunction[Root[p_, _]] := foreignFunction[p[u]];
  (* return first "foreign" function *)
  First/@Union[Flatten[{foreignFunction[w]}]]]
```

The basic test **numericQ** checks to see if **w** is a numeric function of the independent variable **z** at all points (to catch inputs similar to **RiemannSurface[{Sqrt[x]+Log[x], z}, ...]**).

```
In[80]:= numericQ[w_, z_] :=
Module[{x},
  SetAttributes[x, NumericFunction]; NumericQ[w /. z -> x[1]]]

In[81]:= RiemannSurface::notANumericFunction =
  "'1' is not a numeric function of '2'.";
```

The function **RiemannSurface** integrates all of the above step-by-step generation of the sheets into one function.

```

In[82]:= RiemannSurface[{w_, z_}, reIm_, {ppφ_, ppr_}, k1_: -1, k2_: 1] :=
Module[{wcan, bps, sec, wnum, ψ, φ, rplts, subs, odes1,
odes2, sr, asp, odesφ, odesr, rMax, polys, test1,
test2, test3, oF},
(* are all functions occurring in w known? *)
test1 = (oF = notTreatedFunctions[w, z]) == {};
If[test1,
wcan = canonicalizeFunction[w];
(* can we calculate all branch points? *)
test2 = ((bps = N[branchPoints[w, z], 30]) != $Failed);
(* is there at least one finite branch point? *)
test3 = bps != {};
polys = If[test2 && test3,
(* all the steps from above *)
sec = sectors[bps];
wnum = numberMultiValuedTerms[wcan, z];
plts = powerAndLogTerms[wnum, z];
subs = substitutions[plts, wnum, wf, z];
odes1 = toODE[#, z] & /@ subs[[1]];
odes2 = derivativeFreeRhs[odes1];
sr = sheets[#, k1, k2] & /@ plts;
asp = allSheets[sr];
odesφ = odeAzimuthal[#, {#[1, 0, 1], z}, {ψ, φ}, r] & /@ odes2;
odesr = odeRadial[#, {#[1, 0, 1], z}, {ψ, ρ}, d] & /@ odes2;
rMax = Max[#[[3]] & /@ sec];
sectorSheetPatches[reIm, {wf, z}, subs[[-1]], wnum, asp,
{odesφ, {ψ, φ}, r}, {odesr, {ψ, ρ}, d},
#, patchPlotPoints[#, rMax, {ppφ, ppr}]] & /@ sec];
polys /;
(* generate appropriate messages if something fails *)
(If[Not[test1], Message[
RiemannSurface::containsFunctionCurrentlyNotTreated, w, oF]];
If[test1 && Not[test2], Message[
RiemannSurface::cantFindAllBranchPoints, w]];
If[test1 && Not[test3], Message[
RiemannSurface::noBrachPointsPresent, w]];
TrueQ[test1 && test2 && test3]) /; If[numericQ[w, z],
True, RiemannSurface::notANumericFunction; False]

```

For convenience, we now allow equations as inputs so that implicit forms can be used, too.

```

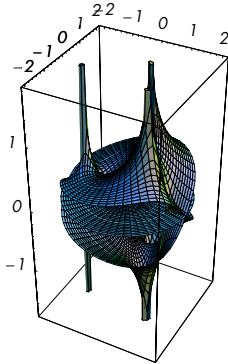
In[83]:= RiemannSurface[{eq_Equal, w_, z_},
reIm_, {ppφ_, ppr_}, k1_: -1, k2_: 1] :=
RiemannSurface[{z /. First[Solve[eq == 0, w]], z},
reIm, {ppφ, ppr}, k1, k2]

```

Now we have implemented all functions needed to visualize the Riemann surface. For our example function we arrive at the following pictures for the imaginary and real parts.

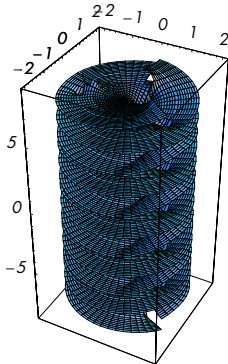
```
In[84]:= Show[Graphics3D[{EdgeForm[Thickness[0.0001]],
  SurfaceColor[Hue[0.40], Hue[0.80], 1.77],
  RiemannSurface[{wExample, z}, Im, {50, 28}, -1, 1]},
  Axes → True, BoxRatios → {1, 1, 2}];
```

From In[84]:=



```
In[85]:= Show[Graphics3D[{EdgeForm[Thickness[0.0001]],
  SurfaceColor[Hue[0.40], Hue[0.80], 1.77],
  RiemannSurface[{wExample, z}, Re, {50, 28}, -1, 1]},
  Axes → True, BoxRatios → {1, 1, 2}];
```

From In[85]:=



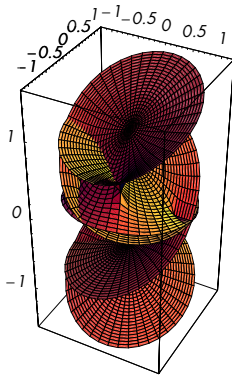
As the previous picture shows, the imaginary part is a faithful image of the Riemann surface.

Here is another example: the Riemann surface of an algebraic function.

```
In[86]:= wExample = Root[-1 + 2 z # - #^5 &, 3] +  $\frac{z}{4}$ ;
```

```
In[87]:= Show[Graphics3D[{EdgeForm[Thickness[0.0001]],
  SurfaceColor[Hue[0.95], Hue[0.19], 2.76],
  RiemannSurface[{wExample, z}, Im, {40, 24}, -1, 1],
  Axes -> True, BoxRatios -> {1, 1, 2}]];
```

From In[87]:=



Note: we can get a purely polynomial description of this function by using the function **Resultant**.

```
In[88]:= Resultant[w - root - z/4, (-1 + 2 z # - #^5) &[root], root]
```

```
Out[88]= 1 + w^5 - 2 w z - 5 w^4 z/4 + z^2/2 + 5 w^3 z^2/8 - 5 w^2 z^3/32 + 5 w z^4/256 - z^5/1024
```

Here is another example that contains the inverse of the error function.

```
In[89]:= wExample = InverseErf[z];
```

```
In[90]:= RiemannSurface[{wExample, z}, Im, {40, 20}, -1, 1]
```

\$MaxExtraPrecision::meprec :

In increasing internal precision while attempting to evaluate Im[x\$2349[1]], the limit \$MaxExtraPrecision = 50. was reached. Increasing the value of \$MaxExtraPrecision may help resolve the uncertainty.

RiemannSurface::containsFunctionCurrentlyNotTreated :

InverseErf[0, z] contains the functions {InverseErf[0, z]} which are not treated in this implementation.

```
Out[90]= RiemannSurface[{InverseErf[0, z], z}, Im, {40, 20}, -1, 1]
```

You did not really expect the last example to work, did you? As mentioned in the introduction, the Riemann surface of such functions will be constructed in Part III of this series, and this is only the third installment of Part II! According to the test implemented in **notTreatedFunctions** the **foreignFunction InverseErf** is caught and detected.

■ Outlook

In the next and final installment of “Visualization of Riemann Surfaces Part II,” I will show approximately 20 pictures which demonstrate the functions implemented so far, and implement Riemann surfaces over Riemann spheres. The latter will allow us to treat infinity like any other point.

Leibniz: Implementing a Drag-and-Drop Calculator

Joe Gregg

*Lawrence University
Appleton, WI 54911
greggj@lawrence.edu*

Leibniz is a combination word processor and front end for *Mathematica* designed for use in mathematics education. Leibniz uses a sophisticated pattern matching scheme to implement a simple and elegant interface for performing symbolic calculations. This article will describe the architecture of the Leibniz system and discuss some of the special techniques used in Leibniz to control evaluation.

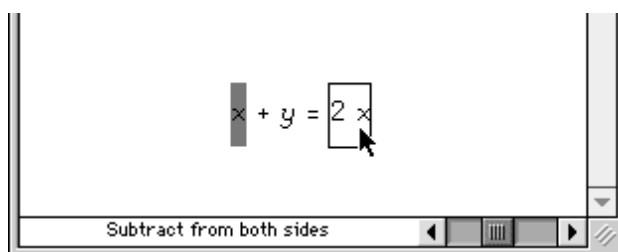
■ What is Leibniz?

Leibniz is a stand-alone text and equation editor for Windows and MacOS computers implemented in C++. Leibniz communicates with the *Mathematica* kernel via *MathLink*, sending requests for calculations to the kernel and formatting and displaying the results. In addition to the application itself, the system also includes **LZCompute.m**, a *Mathematica* package that mediates the communication between the Leibniz front end and the kernel.

Leibniz implements a powerful and easy-to-use mouse-driven calculation system. Users perform calculations by selecting a portion of an expression to evaluate and pressing enter, or rearrange the expression by dragging a selection from one part of an expression to a new location. For example, to compute an integral the user would type the integral, select it, and press enter. To move a term from one side of an equation to the other by, say, subtracting the term from both sides, the user selects the term and drags it to the other side of the equation. This system is particularly powerful and easy to use because Leibniz takes advantage of *Mathematica*'s powerful pattern matching capabilities to match the form of an expression to manipulations that could be performed on it.

When the user selects a portion of an equation, the application parses the equation and translates it into an internal data structure that reflects the structure of the equation. At the same time, the application calls a function contained in the **LZCompute** package. The purpose of this function is to examine the structure of the expression the user has selected and to return a list of possible mathematical manipulations that could be performed on the expression. These possible manipulations come back ranked in order of likelihood, with the most likely option displayed to the user in a message panel located in the lower left corner of the document window. The user can then press enter to have that calculation carried out, or hold down the command key and click on the selection to pop up a menu containing the other calculation options contained in the list.

Dragging operations are implemented in a similar fashion. As soon as the user clicks in the selected portion of an expression and starts to move the mouse, Leibniz queries the kernel to see if any manipulations are possible. As the user drags the mouse into each new part of an expression, the Leibniz front end calls a function in the **LZCompute** package to determine whether or not that portion of the expression is a valid target for a drag-and-drop manipulation. As soon as Leibniz has found a valid target, the application displays a box around the target portion of the expression and displays a message in the message panel indicating what manipulation the system thinks the user is trying to accomplish.



As soon as the user releases the mouse button, Leibniz sends the expression to the kernel along with a request to perform the desired manipulation. When the result comes back, Leibniz formats and displays the result. If multiple manipulations are possible, the most likely manipulation gets displayed in the message panel followed by the ellipsis character (...) to indicate that more options are possible. To access alternative options during a drag-and-drop operation, the user holds the mouse pointer stationary for a brief period to pop-up a menu of alternative options.

■ The Architecture of the Leibniz System

The behavior described above is implemented through a combination of functions performed by the application and the kernel. Roughly speaking, the application contains the code necessary to translate expressions to and from a form that the kernel can understand, while code in the **LZCompute** package implements all of the necessary pattern matching and calculation tasks.

The application is responsible for parsing mathematical expressions and building an internal tree structure to reflect the structure of the expression. In addition, the tree structure also contains information about the location of each subexpression. As the user drags the mouse, the application uses this information to map the current mouse location to the portion of the expression the mouse is in. When results come back from the kernel, the application translates them from *Mathematica* form to a form suitable for display.

Most of the interesting behavior in this system is implemented in the **LZCompute** package. This is due to a conscious design decision. There were many manipulations and calculations that could have been performed either in the application itself or by code contained in the package. Whenever the option came up to either hard code a capability in C++ in the application or as *Mathematica* code in the package, I chose to implement the operation in the package. This provides two obvious benefits: the code in the package is easier to modify, and end users with a knowledge of *Mathematica* can modify the behavior of the system to suit their needs.

□ Implementing Drag-and-Drop Calculations

When the user clicks and drags inside an expression, Leibniz sends a request to the kernel to evaluate a potential drag-and-drop operation. It does this by calling the function **LZEvaluateShuffle** defined in the **LZCompute** package. For example, to evaluate the dragging operation shown in the illustration above, Leibniz would call the following.

```
LZEvaluateShuffle[LZMover[x] + y == LZTarget[2 x]]
```

Note the **LZMover** and **LZTarget** calls wrapped around parts of the expression. These function calls are used to indicate which part of the expression is being dragged (the mover) and which part of the expression is the target. **LZEvaluateShuffle** returns a list of code numbers corresponding to actions that are possible.

If **LZEvaluateShuffle** had to just return a single response code, life would be pretty simple. We could make a series of definitions like this.

```
LZEvaluateShuffle[Plus[LZMover[a_], b_] == LZTarget[c_]] := 110;
```

To indicate how **LZEvaluateShuffle** should respond to a particular dragging request. In order to return a list of matches, we have to be a little more clever. This is how **LZEvaluateShuffle** is actually implemented.

```
LZEvaluateShuffle[expr_] :=  
  FixedPoint[(EvalShuffle[expr, #]) &, {}];  
ES[expr_, n_] :=  
  (EvalShuffle[expr, list_] := Append[list, n] /; FreeQ[list, n]);
```

LZEvaluateShuffle is implemented as a fixed point calculation starting with an empty list and repeatedly applying the **EvalShuffle** function. Rather than return a single code number, **EvalShuffle** appends its code number to the list only if the list does not already contain that particular code number. **Eval**

Shuffle is defined via a series of calls to the **ES** function: each of those calls defines **EvalShuffle** for a particular pattern. For example, the **ES** rule that matches the drag request shown above is the following.

```
ES[Plus[LZMover[a_], b_] == LZTarget[c_]] = 110;
```

How does Leibniz know which subexpression to mark as the target? After all, a given element in a mathematical expression is a member of multiple possible subexpressions. When the user moves the mouse to a new location in an expression, the application uses the information stored in the parse tree to map the location to a subexpression, typically a leaf in the parse tree. Leibniz then calls **LZEvaluateShuffle** with that subexpression identified as the target. If the result comes back empty, Leibniz moves up to the next largest subexpression containing the mouse location and calls **LZEvaluateShuffle** again with that subexpression identified as the target. This process continues until an action is found or there are no more possibilities for the target expression. Despite the fact that this process often requires multiple calls to **LZEvaluateShuffle** via *MathLink*, the process is fast enough to work comfortably in real time as the user drags the mouse over the expression.

Once Leibniz gets a list of code numbers back from the kernel, it calls the **LZMessage** function to retrieve descriptive strings for each of the potential actions. The descriptive string for the most likely option gets displayed immediately in the message panel. For example, the **LZMessage** entry that corresponds to the **ES** entry shown above is this.

```
LZMessage[110] = "Subtract from both sides"
```

As soon as the user releases the mouse button, Leibniz sends a request to evaluate the dragging operation. The **LZShuffle** function handles this request. The first parameter to **LZShuffle** is the code number for the desired action, and the second parameter is the expression being manipulated with mover and target identified. For example, the **LZShuffle** definition that matches the **ES** rule shown above is this.

```
LZShuffle[110, Plus[LZMover[a_], b_] == LZTarget[c_]] :=  
  Plus[b] == c - a;
```

■ Controlling Evaluation

One of the characteristics of *Mathematica* and other computer algebra systems that educators sometimes find frustrating is the fact that these systems generate results without showing intermediate steps. Leibniz addresses this problem by giving the user very fine control over just what part of an expression gets modified and how drastically the expression gets modified in a single step.

By selecting only a portion of an expression to work on, or by dragging and dropping within a larger expression, the user gets to focus the calculation. Leibniz supports this focusing behavior by sending just the relevant part of an expression to the kernel. When the result comes back, its gets swapped into place

in the larger expression it came from, and that expression is then formatted and displayed.

Even with this focusing behavior, special care is needed to avoid unwanted evaluation. Consider our seemingly innocent example of a rule for moving a term from one side of an equation to the other.

```
LZShuffle[110, Plus[LZMover[a_], b_] == LZTarget[c_]] :=  
Plus[b] == c - a;
```

This seems straightforward enough, but what happens when, say, **b** is an integral? The user's intention will be to subtract that integral from both sides without evaluating it, but *Mathematica*'s default evaluation mechanisms will cause **b** to be evaluated, thus making the integral disappear after subtracting it from both sides. One obvious solution would be to give **LZShuffle** the attribute **HoldAll** and replace the result above as follows.

```
Hold[Plus[b]] == Hold[c] - Hold[b]
```

This solves the problem of the unwanted evaluation, but then causes the expression to fail to evaluate correctly when the user wants it to. For example, if **c** were 3 and **b** were 2, the user would expect to be able to subtract 2 from both sides of the original equation giving a new right-hand side of 1. However, with both **c** and **b** held as shown here, the user would see a new right-hand side of the unevaluated expression 3 - 2.

The lack of control offered by the standard evaluation mechanisms and the **Hold** mechanism make it necessary to find a more refined strategy for controlling evaluation.

The first element of this strategy is to force every expression that the system works with to be a purely symbolic expression. This is done by replacing all conventional mathematical functions such as **Plus**, **Integrate**, **D**, and so forth with purely symbolic analogs called **LZPlus**, **LZIntegrate**, **LZD**, and so on. Thus, rather than rendering the expression **a+b** as **Plus[a,b]**, the Leibniz parser passes the expression to the kernel as **LZPlus[a,b]**.

With a little bit of care, we can do pattern matching with these new expressions just as easily as we could with more conventional expressions. For example, by giving the function **LZPlus** the attribute **OrderLess**, we can write pattern matching rules for **LZPlus** without regard to the ordering of terms involved.

Because none of these new functions come with evaluation rules, everything gets held by default. To release the implicit hold, **LZCompute** uses a couple of mechanisms. The first mechanism releases the hold partially so that the usual evaluation rules can perform obvious algebraic simplifications.

```
LZPEval[a_] := a /. {LZPlus → Plus, LZTimes → Times, LZDivide → Divide,  
  (LZPower[x_, n_] /; FreeQ[x, LZMatrix]) → Power[x, n],  
  LZRadical[x_, n_] → Power[x, 1/n]};
```

Thus, the **LZShuffle** rule shown above becomes the following.

```
LZShuffle[110, LZPlus[LZMover[a_], b_] == LZTarget[c_]] :=
  LZPlus[b] == LZPEval[c - b];
```

This solves both of the problems described earlier. If **c** is 3 and **b** is 2, the right-hand side will correctly evaluate to 1. If **b** is an integral, **LZPEval** will leave it unevaluated in the **LZIntegrate** form.

The second evaluation mechanism used is a complete evaluation mechanism, which converts every function in an expression from its LZ form to its conventional form. This mechanism is implemented via a function called **LZEval**.

```
LZEval[a_] := MapAll[LZRelease, a];
```

The **LZRelease** function maps each individual LZ function to its conventional equivalent. For example, here is part of the definition of **LZRelease**.

```
LZRelease[LZIntegrate[a_, {b_, c_, d_}]] := Integrate[a, {b, c, d}];
```

This hold and release mechanism provides other benefits, as well. Because **LZRelease** has to map the held forms to their conventional equivalents, we can take advantage of the need to do that mapping to provide more sophisticated handling for certain special cases. The best example of this is matrix multiplication. Experienced *Mathematica* programmers know that if we try to compute the product of a two by two matrix and a column vector by naively writing

```
Times[{a, b}, {c, d}, {e, f}]
```

we get the incorrect result

```
{a e, b e}, {c f, d f}
```

If we remember to use **Dot** instead of **Times**, everything works out just fine. Leibniz solves this problem for the user by using **LZTimes** for everything that looks like a product and then relying on the release mechanism to sort out the details. **LZRelease** uses some simple and straightforward pattern matching to ensure that **LZTimes** maps to the appropriate equivalent.

```
LZRelease[LZTimes[a_]] := LZResolveProduct[a];
LZResolveProduct[a_] = a;
LZResolveProduct[a_, b_] := a.b /; And[MatrixQ[a], MatrixQ[b]];
LZResolveProduct[a_, b_] := a b /; Not[And[MatrixQ[a], MatrixQ[b]]];
LZResolveProduct[a_, b_, c_] :=
  LZResolveProduct[LZResolveProduct[a, b], c];
```

■ The Future of Leibniz

Leibniz is an evolving system. The rule base is continually being refined as Leibniz users discover the need for new kinds of calculations. The open architecture of the Leibniz systems makes it relatively easy to add rules for new kinds of calculations.

Another recent addition to the system is a dialog facility. *Mathematica* code in the **LZCompute.m** package can now call a number of functions that bring up dialogs in the Leibniz front end. One function brings up a dialog that prompts the user to enter an expression. Another function brings up a dialog that will present the user with a list of choices for them to select from. By allowing the code in **LZCompute.m** to interact directly with the user Leibniz can begin to move beyond the simple imperative style of “put that there” or “compute this”.

Over the long term, Leibniz will function as a test bed for ideas in interface design. By pointing out the limitations of the pattern-matching, drag-and-drop interface, Leibniz can point the way toward better and more powerful interface mechanisms for symbolic calculation.

About the Author

Joe Gregg is an associate professor of mathematics and computer science at Lawrence University in Appleton, Wisconsin. He maintains a Leibniz web site at www.leibnizsoftware.com.

Quantum Turing Machine Simulator

Joachim Hertel

jhertel@h-star.com

We present a toolkit to construct, run, and study Quantum Turing Machines.

■ Introduction

The computational process has long been an interesting and challenging subject of basic research in mathematics, physics, and computer sciences (see [1] for introduction and overview). One of the fundamental insights is that *information* is indeed *physical* [2, 3, 4, 5] and therefore subject to basic laws of physics.

In recent years it has become clear [6], that information theory, like many other branches in physics, expands into the quantum regime, and it is now customary to call this subject matter Quantum Information Theory (QIT). Recent branches of QIT include Quantum Computing [7, 8, 9], Quantum Encryption [10], and Quantum Teleportation [11].

In QIT the classical bit is replaced by a quantum-bit (a.k.a. *qubit*), which is a two-level quantum system, and can physically be implemented (e.g., as a spin-1/2). For further background information on fundamentals of QIT, see [12] and references therein. The actual availability of a (large scale) quantum computer might still be in the (far) future, but numerous projects are on their way to develop and deliver QIT simulation programs to study various aspects of quantum algorithms and other QIT topics. For an overview and review of several known QIT simulation programs, see [13].

Here, we present and discuss a Quantum Turing Machine Simulator (QTS), which enables one to construct and run Quantum Turing Machines, and to study some of their fundamental properties. Additional documentation and source code is available [14].

Similar to its classical counterpart, a Quantum Turing Machine (QTM) typically consists of a QTM head, living in an n -dimensional Hilbert space, and an infinite lattice of qubits, each qubit living in (at least) a two-dimensional Hilbert space. The QTM head scans the lattice and acts *locally* with only the qubit at the head's

location. The computational details of a specific QTM are given by the local transition function (a.k.a. step operator) T and its adjoint T^* .

In Deutsch's [15] architecture of a QTM, the step operator T is unitary and represents the finite time translation operator. The machine state ψ then advances in time according to $\psi(t) = U(t)\psi(0)$. Given U , it is generally *not* possible to construct a *local* Hamilton operator H that generates U , that is, $U(t) = e^{iHt}$. Benioff's [16] approach is based on a step operator T which is not associated with a time step but represents an individual step in the computation itself. In order to keep the quantum computation reversible, T must be distinct path generating [16, 17]. By construction, Benioff's T is local and need *not* be unitary nor self-adjoint. Taking T and its adjoint T^* , Benioff then follows Feynman's [18] prescription for constructing a local self-adjoint Hamilton operator according to $H \approx T + T^*$. With this H one can formally compute $U(t)$ by using a power series expansion from which it is clear that this U is *no longer* local.

Our approach is based on the Quantum Turing Model and Benioff's local and distinct path-generating step operators. We are aware of the fact that any implementation of any quantum simulator eventually face the fact of unmanageable large resource consumption (both CPU time and main storage) due to the exponential size of the Hilbert space of composite quantum systems [19]. The QTS toolkit is coded and optimized for *Mathematica* [20] 4.0 but runs on Version 3 as well.

Besides building QTMs and gaining some educational quantum experience by running them, one can use the simulator to investigate the flow of coherence in quantum networks. This exciting new type of application for QTMs has recently been investigated by Mahler and Kim [21, 22] and is based on simulating a quantum gate circuit as a QTM where the Turing head rotates on a closed quantum tape. One can then compute the generalized Bloch vector [23] to obtain information about coherence and entanglement. Using our QTS one can perform such experiments. However, here we present an introductory calculation of the coherence vector of a Turing head only.

■ Quantum Turing Machines

□ Notations and Basic Properties

The laws of quantum physics are reversible in time [24]. One of the earliest proposals of a quantum Hamiltonian model of reversible computation is due to Benioff [25, 26]. The physical model of a QTM, as considered by Benioff, corresponds to a one-tape machine. The finite state head moves along an (infinite) lattice of finite dimensional qubits. The computational basis $\mathcal{B}$, which spans a Hilbert space $\mathcal{H}$ is given by the set $\{|i, j, S\rangle\}$ of states. Here

j denotes the lattice site currently scanned by the head

l denotes the internal state of the head

S denotes the tensor product over all qubit states $|s\rangle_j$, $-\infty \leq j \leq +\infty$

To keep $\mathcal{H}$ separable, we choose $|s\rangle_j \neq 0$ for a *finite* number of lattice sites only (zero tail condition).

Following Benioff [16, 17, 27], a QTM can be described by a step operator T and its adjoint T^* , which in turn are defined as the sum of elementary step operators $T_{l,s}$:

$$T = \sum_{l,s} T_{l,s}$$

$$T_{l,s} = \gamma_{l,s} \sum_{j=-\infty}^{+\infty} w_{l,s} Q_l v_{l,s} P_{s,j} u_{l,s} P_j$$

with

$Q_l :=$ projector for the head in state l

$P_j :=$ projector for the head being at site j

$P_{s,j} :=$ projector for the qubit at j being in state s

The operators w , v , and u must be unitary and are defined as follows.

$w_{l,s}$ describes head state changes and operates in the Hilbert space of the QTM head.

$v_{l,s}$ describes qubit state changes and operates in the Hilbert space of a qubit.

$u_{l,s}$ describes the motion of the head depending on head state l and qubit state s .

To ensure reversibility of the QTM computational process, the step operator T and its adjoint T^* must be distinct path generating. That means iterations of T and T^* on any state $|1, j, S\rangle$ either annihilate that state or generate a finite or infinite set of states which do *not* join, branch, or intersect. Each such set of states is called a *computational path* $\mathcal{P}$. For a given distinct path generating step operator T with adjoint T^* , we define the associated Hamiltonian operator following the prescription of Feynman [18] as

$$H = K(2 - T - T^*)$$

with $K = \text{const.}$

■ QTS Toolkit

□ The Toolkit Package: VectorStructures

This package contains basic functions to generate a Hilbert space, tensor products of Hilbert spaces, symbolic or numeric vectors and calculate inner products, density matrices, and to apply operators to vectors in a Hilbert space.

MakeHilbertSpace [<i>n, label</i>]	creates a list $b=\{s,ip\}$. Here s is a list of n labeled symbols, for example, $s=\{s[0],...,s[n-1]\}$ and ip represents a symbol which is used as the Head for the inner product. The inner product is defined as a complex-valued sesquilinear form $ip(.,.)$ and ensures that s represents a complete orthonormal (ON) system.
IP [<i>x, y, b</i>]	is the inner product of the Hilbert space b . IP is defined as a complex-valued sesquilinear form $ip(x,y)$.
MakeTensorProduct : HilbertSpace [<i>b, n</i>]	creates the n -fold tensor product of the base Hilbert space b .
TPIP [<i>x, y, t</i>]	defines the inner product on the tensor product space t in terms of the IP of the base space.
DensityMatrix [<i>basisvector, basis</i>]	computes the density matrix for a given basis Hilbert space vector.
ApplyOperator [<i>op, basis, vector</i>]	applies the operator op to the Hilbert space vector in the given basis.

□ The Toolkit Package: QTM

This package contains basic functions to configure and build up a QTM. It also defines the data type **Qtm** which is used to describe the total QTM state at any given point during the computation.

Qtm	is the header for the fundamental data type describing an overall state of a QTM.
Qtm [<i>s-dim, s-label, ex_interval, exstate_list, bdim, b-label, b-pos</i>]	is used to define a state for the QTM. $s\text{-dim}$ specifies the Hilbert space dimension of the lattice spins, $s\text{-label}$ is the label used to enumerate the ON basis for that Hilbert space. $ex_interval$ specifies the lattice positions of excited spins, $exstate_list$ specifies the states of the excited spins, $b\text{-dim}$ and $b\text{-label}$ define the dimension and labels for the Hilbert space of the Turing header, and $b\text{-pos}$ is the header lattice position.

InitQTM [<i>s-dim</i> , <i>s-label</i> , <i>ex_interval</i> , <i>exstate_list</i> , <i>bdim</i> , <i>b-label</i> , <i>b-pos</i>]	is used to define the initial state for the QTM. <i>s-dim</i> specifies the Hilbert space dimension of the lattice spins, <i>s-label</i> is the label used to enumerate the ON basis for that Hilbert space. <i>ex_interval</i> specifies the lattice positions of excited spins, <i>exstate_list</i> specifies the states of the excited spins, <i>b-dim</i> and <i>b-label</i> define the dimension and labels for the Hilbert space of the Turing header, and <i>b-pos</i> is the initial header lattice position.
HeadPos [<i>qtm</i>]	retrieves the current Turing head position for a given QTM state <i>qtm</i> .
HeadState [<i>qtm</i>]	retrieves the current Turing head state for a given QTM state <i>qtm</i> .
SiteState [<i>j</i> , <i>qtm</i>]	retrieves the current spin state at lattice site <i>j</i> for a given QTM state <i>qtm</i> .
GetHeadBasisElement [<i>n</i> , <i>qtm</i>]	retrieves the <i>n</i> th component of the current Turing head state given the QTM state <i>qtm</i> .
GetQubitBasisElement [<i>j</i> , <i>n</i> , <i>qtm</i>]	retrieves the <i>n</i> th component of the current spin state at lattice site <i>j</i> given the QTM state <i>qtm</i> .
ProjHeadAtSite [<i>j</i> , <i>qtm</i>]	is the projection operator for the Turing head at site <i>j</i> given the QTM state <i>qtm</i> .
ProjHeadInState [<i>l</i> , <i>qtm</i>]	is the projection operator for the Turing head being in state <i>l</i> given the QTM state <i>qtm</i> .
ProjQubitAtSiteInState [<i>j</i> , <i>k</i> , <i>qtm</i>]	is the projection operator for the spin (qubit) at lattice site <i>j</i> in state <i>k</i> given the QTM state <i>qtm</i> .
ModShiftHeadPos [<i>n</i> , <i>m</i> , <i>qtm</i>]	shifts the current Turing head position mod [<i>n</i> , <i>m</i>] lattice sites.
UpdHeadState [<i>op</i> , <i>qtm</i>]	applies the operator <i>op</i> to the current Turing head state given the QTM state <i>qtm</i> . Typically, <i>op</i> is an unitary operator in the Hilbert space of the Turing head.

UpdSiteState [<i>op</i> , <i>j</i> , <i>qtm</i>]	applies the operator <i>op</i> to the current lattice spin state at lattice site <i>k</i> given the QTM state <i>qtm</i> . Typically, <i>op</i> is an unitary operator in the Hilbert space of the lattice spin.
QInner [<i>p</i> , <i>q</i>]	is the inner product of the QTM states <i>p</i> , <i>q</i> .
FmtState [<i>qtm</i>]	generates a textbook format of the QTM state <i>qtm</i> .
QTMEval [Plus [<i>qtm1</i> , ..., <i>qtmn</i>]]	takes a sum of QTM states <i>q1</i> , <i>q2</i> , ..., <i>qn</i> as input. States which differ in their head state but are identical otherwise (so called similar states) are added with respect to their head states. Let <i>qi</i> and <i>qj</i> be two similar states. Then HeadState [QTMEval [Plus [<i>qi</i> , <i>qj</i>]]] = HeadState [<i>qi</i>] + HeadState [<i>qj</i>] where the head states are added in the joined head Hilbert space of <i>qi</i> and <i>qj</i> , respectively.

■ Toolkit Application: Interferometer QTM

The Interferometer QTM has been defined and discussed by Benioff [17]. It illustrates that there are dpq step operators, which generate computational graphs having an interferometric structure with different actions on the lattice site qubits in the two interferometric arms.

The step operator is defined as:

$$T := \sum_{j=-\infty}^{+\infty} [Q_0 P_{0,j} u + w_{12} Q_0 P_{1,j} u + w^2 Q_1 v P_{0,j} u^* + w^2 Q_2 P_{0,j} u^* + w^2 Q_3 P_{1,j} u + w^2 Q_4 P_{1,j} u + w^2 Q_5 P_{0,j} v^* u + w^2 Q_6 P_{0,j} u + Q_0 w_{78}^* P_{1,j} u] P_j$$

The Turing head lives in a nine-dimensional Hilbert space $\mathcal{H}_9$ with basis $\mathcal{B}_{\text{Head}} = \{b[0], \dots, b[8]\}$.

The lattice spins are living in a two-dimensional Hilbert space $\mathcal{H}_2$ with basis $\mathcal{B}_{\text{Spin}} = \{b[0], b[1]\}$. The basis state $b[1]$ serves as a marker state to trigger certain actions while the Turing head is scanning the lattice sites.

The unitary operators are defined as follows.

- u*: shift operator, shifts the head position one lattice site to the right, that is, $u P_j = P_{j+1} u$.
- u^* : shift operator, shifts the head position one lattice site to the left, that is, $u^* P_j = P_{j-1} u^*$.

w : head - state - shift - mod[9] operator and is implemented in matrixform as:

$$w = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

w_{12} : shift operator, takes head $b[0]$ into the linear superposition

$\frac{1}{\sqrt{2}} (b[1] + b[2])$ and is implemented in matrixform as:

$$w_{12} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \frac{1}{\sqrt{2}} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \frac{1}{\sqrt{2}} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

w_{78} : shift operator, takes head $b[0]$ into the linear superposition

$\frac{1}{\sqrt{2}} (b[7] + b[8])$ and is implemented in matrixform as:

$$w_{78} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

v : the two-dimensional unitary lattice site operator is arbitrary. To fix an example, we take the Hadamard transformation implemented in matrixform as:

$$v = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix}$$

The *initial* QTM state is defined to be:

head in state $b[0]$ at lattice site position $j = -3$

lattice state $S \doteq \otimes_{j=-\infty}^0 b[0]_j \otimes b[1]_1 \otimes b[0]_2 \otimes b[1]_3 \otimes \otimes_{j=4}^{+\infty} b[0]_j$

hence: $\psi_0 \doteq |b[0], -3, S\rangle$. See Figure 1.

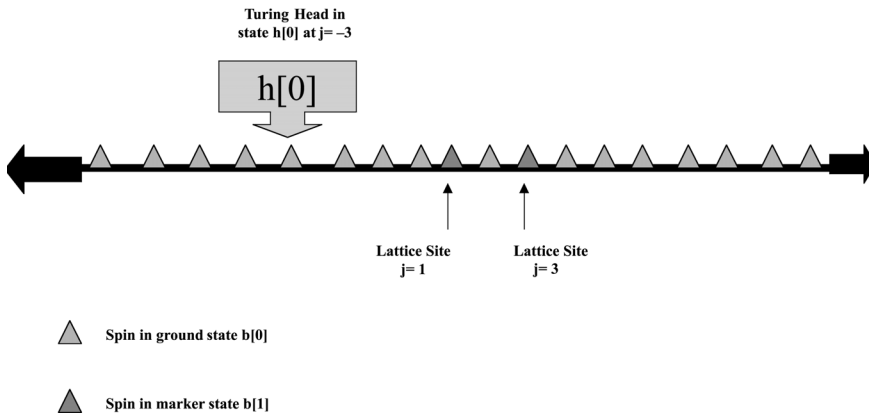


Figure 1.

Next we show the main steps to build this QTM using the QTS toolkit.

Set up the step operator T as a sum of partial step operators.

```
T[q_Qtm,w12_List,w_List,w78_List,v_List]:=Sum[TS[j,q,Abs[HeadPos[q]],
w12,w,w78,v],{j,1,9}];
```

Do the same for the adjoint step operator.

Now define the basic case structure for calling the appropriate partial step operators with the correct choice.

```

TS[i_Integer, q_Qtm, n_Integer, w12_, w_, w78_, v_] :=
Which[
i==1, TS1[q, n],
i==2, TS2[q, n, w12],
i==3, TS3[q, n, w, v],
i==4, TS4[q, n, w],
i==5, TS5[q, n, w],
i==6, TS6[q, n, w],
i==7, TS7[q, n, w, v],
i==8, TS8[q, n, w],
i==9, TS9[q, n, w78],
i>9, Print["Error"]
];

```

Do the same for the adjoint step operator.

Next define the individual partial step operators.

```

TS1[q_Qtm, n_Integer] :=
Sum[ProjHeadInState[hd[0, q], ProjQubitAtSiteInState[k, qu[0, q],
ShiftHeadPos[1, ProjHeadAtSite[k, q]]]], {k, -n, n}];

TS2[q_Qtm, n_Integer, w_List] := Sum[UpdHeadState[w,
ProjHeadInState[hd[0, q], ProjQubitAtSiteInState[k, qu[1, q],
ShiftHeadPos[1, ProjHeadAtSite[k, q]]]]], {k, -n, n}];

TS3[q_Qtm, n_Integer, w_List, v_List] :=
Sum[UpdHeadState[w.w, ProjHeadInState[hd[1, q],
UpdSiteState[v, k, ProjQubitAtSiteInState[k, qu[0, q],
ShiftHeadPos[-1, ProjHeadAtSite[k, q]]]]]], {k, -n, n}];

TS4[q_Qtm, n_Integer, w_List] := Sum[UpdHeadState[w.w,
ProjHeadInState[hd[2, q], ProjQubitAtSiteInState[k, qu[0, q],
ShiftHeadPos[-1, ProjHeadAtSite[k, q]]]]], {k, -n, n}];

TS5[q_Qtm, n_Integer, w_List] := Sum[UpdHeadState[w.w,
ProjHeadInState[hd[3, q], ProjQubitAtSiteInState[k, qu[1, q],
ShiftHeadPos[1, ProjHeadAtSite[k, q]]]]], {k, -n, n}];

TS6[q_Qtm, n_Integer, w_List] := Sum[UpdHeadState[w.w,
ProjHeadInState[hd[4, q], ProjQubitAtSiteInState[k, qu[1, q],
ShiftHeadPos[1, ProjHeadAtSite[k, q]]]]], {k, -n, n}];

TS7[q_Qtm, n_Integer, w_List, v_List] :=
Sum[UpdHeadState[w.w, ProjHeadInState[hd[5, q],
ProjQubitAtSiteInState[k, qu[0, q], UpdSiteState[hc[v], k,
ShiftHeadPos[1, ProjHeadAtSite[k, q]]]]]], {k, -n, n}];

TS8[q_Qtm, n_Integer, w_List] := Sum[UpdHeadState[w.w,
ProjHeadInState[hd[6, q], ProjQubitAtSiteInState[k, qu[0, q],
ShiftHeadPos[1, ProjHeadAtSite[k, q]]]]], {k, -n, n}];

```

```
TS9[q_Qtm, n_Integer, w_List] := Sum[ProjHeadInState[hd[0, q],
  UpdHeadState[hc[w], ProjQubitAtSiteInState[k, qu[1, q],
    ShiftHeadPos[1, ProjHeadAtSite[k, q]]]], {k, -n, n}];
```

Do the same for the adjoint partial step operators.

□ Prepare the Initial State

```
qc = InitQTM[2, b, Interval[{0, 4}], {{1, b[1]}, {3, b[1]}}, 9, h, -3]
```

Use the formatting function **FmtState** to produce a textbook format.

```
FmtState[qc]
```

$$(h[0])_{-3} \otimes (b[0])_{-\infty} \otimes \dots \otimes (b[0])_0 \otimes \\ (b[1])_1 \otimes (b[0])_2 \otimes (b[1])_3 \otimes (b[0])_4 \otimes \dots \otimes (b[0])_{\infty}$$

□ Run the QTM

Define a shortcut step operator **TC** as

```
TC[q_Qtm] := TQTM[q, w12, w, w78, v]
```

and produce some output for a given iteration count.

```
Do[Print["Iteration: ", n];
  Print["State: ", FmtState[QTMEval[Nest[TC, qc, n]]], {n, 1, 10}]
```

Iteration: 1

State: $(h[0])_{-2} \otimes (b[0])_{-\infty} \otimes \dots \otimes (b[0])_0 \otimes \\ (b[1])_1 \otimes (b[0])_2 \otimes (b[1])_3 \otimes (b[0])_4 \otimes \dots \otimes (b[0])_{\infty}$

Iteration: 2

State: $(h[0])_{-1} \otimes (b[0])_{-\infty} \otimes \dots \otimes (b[0])_0 \otimes \\ (b[1])_1 \otimes (b[0])_2 \otimes (b[1])_3 \otimes (b[0])_4 \otimes \dots \otimes (b[0])_{\infty}$

Iteration: 3

State: $(h[0])_0 \otimes (b[0])_{-\infty} \otimes \dots \otimes (b[0])_0 \otimes \\ (b[1])_1 \otimes (b[0])_2 \otimes (b[1])_3 \otimes (b[0])_4 \otimes \dots \otimes (b[0])_{\infty}$

Iteration: 4

State: $(h[0])_1 \otimes (b[0])_{-\infty} \otimes \dots \otimes (b[0])_0 \otimes \\ (b[1])_1 \otimes (b[0])_2 \otimes (b[1])_3 \otimes (b[0])_4 \otimes \dots \otimes (b[0])_{\infty}$

Iteration: 5

State: $\left(\frac{h[1]}{\sqrt{2}} + \frac{h[2]}{\sqrt{2}} \right)_2 \otimes (b[0])_{-\infty} \otimes \dots \otimes \\ (b[0])_0 \otimes (b[1])_1 \otimes (b[0])_2 \otimes (b[1])_3 \otimes (b[0])_4 \otimes \dots \otimes (b[0])_{\infty}$

Iteration: 6

$$\begin{aligned} \text{State: } & \left(\frac{h[3]}{\sqrt{2}} \right)_1 \otimes (b[0])_{-\infty} \otimes \dots \otimes (b[0])_0 \otimes (b[1])_1 \otimes \left(-\frac{b[0]}{\sqrt{2}} + \frac{b[1]}{\sqrt{2}} \right)_2 \otimes \\ & (b[1])_3 \otimes (b[0])_4 \otimes \dots \otimes (b[0])_{\infty} + \left(\frac{h[4]}{\sqrt{2}} \right)_1 \otimes (b[0])_{-\infty} \otimes \dots \otimes \\ & (b[0])_0 \otimes (b[1])_1 \otimes (b[0])_2 \otimes (b[1])_3 \otimes (b[0])_4 \otimes \dots \otimes (b[0])_{\infty} \end{aligned}$$

Iteration: 7

$$\begin{aligned} \text{State: } & \left(\frac{h[5]}{\sqrt{2}} \right)_2 \otimes (b[0])_{-\infty} \otimes \dots \otimes (b[0])_0 \otimes (b[1])_1 \otimes \left(-\frac{b[0]}{\sqrt{2}} + \frac{b[1]}{\sqrt{2}} \right)_2 \otimes \\ & (b[1])_3 \otimes (b[0])_4 \otimes \dots \otimes (b[0])_{\infty} + \left(\frac{h[6]}{\sqrt{2}} \right)_2 \otimes (b[0])_{-\infty} \otimes \dots \otimes \\ & (b[0])_0 \otimes (b[1])_1 \otimes (b[0])_2 \otimes (b[1])_3 \otimes (b[0])_4 \otimes \dots \otimes (b[0])_{\infty} \end{aligned}$$

Iteration: 8

$$\begin{aligned} \text{State: } & \left(\frac{h[7] + h[8]}{\sqrt{2}} \right)_3 \otimes (b[0])_{-\infty} \otimes \dots \otimes \\ & (b[0])_0 \otimes (b[1])_1 \otimes (b[0])_2 \otimes (b[1])_3 \otimes (b[0])_4 \otimes \dots \otimes (b[0])_{\infty} \end{aligned}$$

Iteration: 9

$$\begin{aligned} \text{State: } & (h[0])_4 \otimes (b[0])_{-\infty} \otimes \dots \otimes (b[0])_0 \otimes \\ & (b[1])_1 \otimes (b[0])_2 \otimes (b[1])_3 \otimes (b[0])_4 \otimes \dots \otimes (b[0])_{\infty} \end{aligned}$$

Iteration: 10

$$\begin{aligned} \text{State: } & (h[0])_5 \otimes (b[0])_{-\infty} \otimes \dots \otimes (b[0])_0 \otimes \\ & (b[1])_1 \otimes (b[0])_2 \otimes (b[1])_3 \otimes (b[0])_4 \otimes \dots \otimes (b[0])_{\infty} \end{aligned}$$

■ Researching QTM Properties with the QTS Toolkit

□ Make the Interferometer Oscillating

Introduce more $b[1]$ marker states to produce oscillating interferometer graphs. For example, define a new QTM state **q1** as

```
q1 =
InitQTM[2, b, Interval[{-4, 8}], {{-3, b[1]}, {-1, b[1]}, {1, b[1]}, {3, b[1]},
{5, b[1]}, {7, b[1]}}, 9, h, -3]
```

in formatted form:

$$\begin{aligned} & (h[0])_{-3} \otimes (b[0])_{-\infty} \otimes \dots \otimes (b[0])_{-4} \otimes (b[1])_{-3} \otimes \\ & (b[0])_{-2} \otimes (b[1])_{-1} \otimes (b[0])_0 \otimes (b[1])_1 \otimes (b[0])_2 \otimes (b[1])_3 \otimes \\ & (b[0])_4 \otimes (b[1])_5 \otimes (b[0])_6 \otimes (b[1])_7 \otimes (b[0])_8 \otimes \dots \otimes (b[0])_{\infty} \end{aligned}$$

Rerun the step operator and compare with the previous results.

Check if T is Orthogonality Preserving

Provide numerical evidence, that T is orthogonality preserving (i.e., $[TT^*, T^*T] = 0$). For a strict mathematical proof, see [17].

Setup the adjoint step operator as **TCADJ** and perform

SameQ[FmtState[TC[TCADJ[u]], FmtState[TCADJ[TC[u]]]] for $u = \text{Nest}[TC, qc, n]$ and large n values.

Check if T is Distinct Path Generating

Provide numerical evidence that T is distinct path generating. For a strict mathematical proof, see [17].

Solution:

Show that **QInner** $[u, \text{TC}[u]] = 0$.

Use $u = \text{Nest}[TC, qc, n]$ for large n .

Do the same for **TADJ**.

■ Locality of the Feynman Hamiltonian

The step operator T is local by construction, that is, a single computational step changes at most the qubit at the lattice position of the head. In order to provide a quantitative measure of locality of powers of H , we prepare a QTM state

q[i] := InitQTM[2, b, Interval[{0, 4}], {{1, b[0]}, {3, b[0]}}, 9, h, i]

and calculate the set of functions l_i defined as

$$l_i[n] := \frac{\text{Abs}[\text{QInner}[q[0], H^n[q[i]]]]}{\sqrt{\text{Abs}[\text{QInner}[H^n[q[i]], H^n[q[i]]]']}}$$

$l_i[n]$ measure the overlap of $q[0]$ and $H^n[q[i]]$ for given i, n . As a consequence of the locality of H we expect that the overlap for fixed n decreases fast with increasing i because the head positions of $q[0]$ and $q[i]$ become more spatially separated and we need more and more iterations of H to produce any significant overlap between $q[0]$ and $q[i]$. As an example, we plot $l_i[5]$ (see Figure 2).

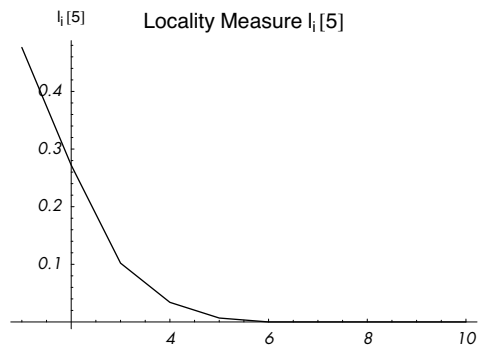


Figure 2.

If we take $l_i[n]$ for fixed i and increase n , we expect to have a significant overlap when $n \geq i$. As an example, we plot $l_5[n]$ (see Figure 3).

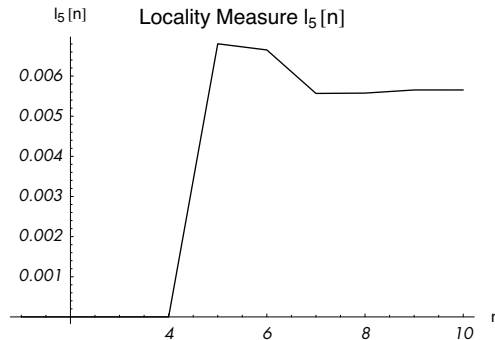


Figure 3.

From this we can see that the time translation operator $U(t) := e^{iHt}$ is completely delocalized because infinitely many powers of H contribute, that is, $\text{Abs}[\text{QInner}[\mathbf{q}[0], \mathbf{U}(t)[\mathbf{q}[i]]] > 0$ for all $i \geq 0$.

■ Turing Head Coherence

Any operator in an n -dimensional Hilbert space $\mathcal{H}_n$ can be represented by the generating operators of the special group $\text{SU}(n)$, [23]. Specifically, this can be done for the density matrix ρ . Consider the set $\{1, \lambda_i\}$, $1 \leq i \leq n^2 - 1$, λ_i being the $\text{SU}(n)$ generators.

Given a density matrix ρ , the associated coherence vector $\vec{\lambda}$ (a.k.a. generalized Bloch vector) is given as the $n^2 - 1$ -dimensional vector of expectation values $\text{Tr}(\rho \lambda_i)$. In general one has (see Chapter 2 of [23] for full details):

$$\vec{\lambda} = \{\lambda_i, 1 \leq i \leq n^2 - 1\} = \{u_{j,k}, v_{j,k}, w_l\}$$

with

$$\begin{aligned} u_{j,k} &= P_{j,k} + P_{k,j} \\ v_{j,k} &= i(P_{j,k} - P_{k,j}) \\ w_l &= -\sqrt{\frac{2}{l(l+1)}} \left(\sum_{j=1}^l P_{j,j} - l P_{l+1,l+1} \right) \end{aligned}$$

where $1 \leq j < k \leq n$, $1 \leq l \leq n-1$, and $P_{j,k} \doteq |j\rangle \langle k|$ is the transition operator between the basis states $|j\rangle$ and $|k\rangle$ in $\mathcal{H}_n$.

The length of $\vec{\lambda}$ is constrained by $|\vec{\lambda}|^2 \leq |\vec{\lambda}_{\max}|^2 = 2 \left(1 - \frac{1}{n}\right)$.

One can classify a density matrix ρ in terms of $\vec{\lambda}$ as follows.

pure: $\vec{\lambda}$ is in the w_l subspace, that is, all $u_{j,k}$ and all $v_{j,k}$ are zero
and $|\vec{\lambda}|^2 = |\vec{\lambda}_{\max}|^2$.

mixed: $\vec{\lambda}$ is in the w_l subspace, that is, all $u_{j,k}$ and all $v_{j,k}$ are zero
and $|\vec{\lambda}|^2 < |\vec{\lambda}_{\max}|^2$.

coherent: $\vec{\lambda}$ is not in the w_l subspace, that is, not all $u_{j,k}$ and /or not all $v_{j,k}$ are zero.

We calculate the coherence vector $\vec{\lambda}$ for the Turing head of our sample machine. (Note: calculating the multi-node coherence vector for the total machine state is beyond the present paper. See [21, 22] for results in that direction.)

Since the Turing head of our sample machine lives in a nine-dimensional Hilbert space $\mathcal{H}_9$, we have to consider 80-dimensional coherence vectors:

$$\vec{\lambda} = \{u_{1,2}, \dots, u_{8,9}, v_{1,2}, \dots, v_{8,9}, w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8\}$$

Since the $u_{j,k}$ and $v_{j,k}$ are straightforward, we list the eight w_l only:

$$w_1 \doteq -P_{1,1} + P_{2,2}$$

$$w_2 \doteq -\frac{P_{1,1} + P_{2,2} - 2P_{3,3}}{\sqrt{3}}$$

$$w_3 \doteq -\frac{P_{1,1} + P_{2,2} + P_{3,3} - 3P_{4,4}}{\sqrt{6}}$$

$$w_4 \doteq -\frac{P_{1,1} + P_{2,2} + P_{3,3} + P_{4,4} - 4P_{5,5}}{\sqrt{10}}$$

$$w_5 \doteq -\frac{P_{1,1} + P_{2,2} + P_{3,3} + P_{4,4} + P_{5,5} - 5P_{6,6}}{\sqrt{15}}$$

$$w_6 \doteq -\frac{P_{1,1} + P_{2,2} + P_{3,3} + P_{4,4} + P_{5,5} + P_{6,6} - 6P_{7,7}}{\sqrt{21}}$$

$$w_7 \doteq -\frac{P_{1,1} + P_{2,2} + P_{3,3} + P_{4,4} + P_{5,5} + P_{6,6} + P_{7,7} - 7P_{8,8}}{2\sqrt{7}}$$

$$w_8 \doteq \frac{1}{6} (-P_{1,1} - P_{2,2} - P_{3,3} - P_{4,4} - P_{5,5} - P_{6,6} - P_{7,7} - P_{8,8} + 8P_{9,9})$$

The following table summarizes the results for the Turing head coherence vector computation of the above-mentioned QTM for several iterations.

state into a coherent superposition at the first $b[1]$ marker and the adjoint of w_{78} collapses the coherent superposition back to a pure basis state.

■ Further Suggestions

Modify the unitary operators acting in the Turing head and see their effect on coherence. Try to compute the Turing head coherence vector for iterations of the Feynman Hamiltonian constructed from the step operator and its adjoint.

■ Performance

Feynman [19] pointed out that simulating quantum physics on a classical computer will eventually need unmanageable resources. To simulate a typical quantum scenario, a classical computer must handle matrices of exponentially large size. Recall that to simulate n qubits a Hilbert space of dimension 2^n is needed. Running the QTS will inevitably meet these basic constraints. To provide some guidelines for expected runtimes when operating a QTM with the QTS package, we have compiled some timing results. For the step operator, as defined in package QTMC, we define the Feynman Hamiltonian as $H \approx T + T^*$ and the unitary time translation operator $U(t) := e^{iHt}$.

Let $U_N(t)$ be the N th order approximation of $U(t)$, that is, $U_N(t) := \sum_{j=0}^N \frac{(itH)^j}{j!}$.

Let

```
q:=InitQTM[2,b,Interval[{0,4}],{{1,b[1]},{3,b[1]}},9,h,-3];
U[h_,q_,n_]:=Expand[Sum[ $\frac{I^j \text{QTMEval}[\text{Nest}[h,q,j]]}{j!}$ , {j,0,n}]];
```

We measure the CPU time needed to compute $f(N) := U_N(1)[q]$.

All measurements are taken on a standalone PC with no other application running. The technical details of the PC configuration are as follows:

Intel Pentium II, MMX, 128 MB RAM, 450 MHz, Windows 98, *Mathematica* 4

The timings provided by *Mathematica* are shown in Figure 4 below.

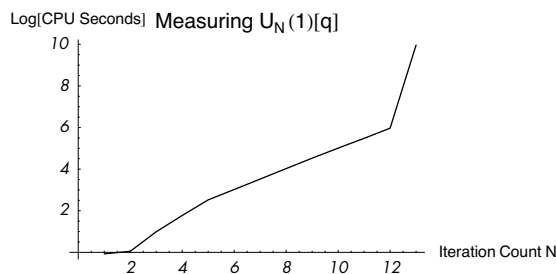


Figure 4.

■ References

- [1] R. P. Feynman, *Feynman Lectures on Computation*, Reading, MA: Perseus Books, 1996.
- [2] C. H. Bennett, "The Thermodynamics of Computation: A Review," *International Journal of Theoretical Physics*, **21**, 1982 pp. 905–940.
- [3] R. Landauer, "Irreversibility and Heat Generation in the Computing Process," *IBM Journal of Research and Development*, **5**, 1961 pp. 183–191.
- [4] R. W. Keyes and R. Landauer, "Minimal Energy Dissipation in Logic," *IBM Journal of Research and Development*, **14**, 1970 pp. 152–157.
- [5] C. H. Bennett, "Logical Reversibility of Computation," *IBM Journal of Research and Development*, **17**, 1973 pp. 525–532.
- [6] C. H. Bennett and P. W. Shor, "Quantum Information Theory," *IEEE Transactions on Information Theory*, **44**, 1998 pp. 2724–2742.
- [7] P. W. Shor, "Algorithms for Quantum Computation: Discrete Logarithms and Factoring," *Proceedings of 35th Annual Symposium on Foundations of Computer Science*, IEEE Computer Society Press, Los Alamitos, CA, 1994 pp. 124–134.
- [8] C. H. Bennett, E. Bernstein, G. Brassard, and U. Vazirani, "Strengths and Weaknesses of Quantum Computation," *SIAM Journal on Computing*, **26**, 1997 pp. 1510–1523.
- [9] P. W. Shor, "Polynomial Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer," *SIAM Review*, **41**, 1999 pp. 303–332.
- [10] C. H. Bennett and G. Brassard, "Quantum Cryptography: Public Key Distribution and Coin Tossing," *Proceedings of IEEE International Conference on Computers, Systems, and Signal Processing*, Bangalore, India, 1984 pp. 175–179.
- [11] C. H. Bennett, G. Brassard, C. Crepeau, R. Josza, A. Peres, and W. K. Wootters, "Teleporting an Unknown Quantum State via Dual Classical and Einstein–Podolsky–Rosen Channels," *Physical Review Letters*, **70**, 1993 pp. 1895–1898.
- [12] P. Rodgers, "Quantum Information," *Physics World*, **3**, 1998 pp. 33–40.
- [13] J. Wallace, "Quantum Computer Simulators—A Review." (June 3, 1999), www.dcs.ex.ac.uk/~jwallace/simrevab.htm.
- [14] www.h-star.com/conresearch.html.
- [15] D. Deutsch, "Quantum Theory, the Church–Turing Principle and the Universal Quantum Computer," *Proceedings of the Royal Society. London. Series A*, **400**, 1985 pp. 96–117.
- [16] P. Benioff, "Quantum Ballistic Evolution in Quantum Mechanics: Application to Quantum Computers," *Physical Review A*, **54**, 1996 pp. 1106–1123.
- [17] P. Benioff, "Models of Quantum Turing Machines," *Fortschritte der Physik*, **46**, 1998 pp. 423–441.
- [18] R. P. Feynman, "Quantum Mechanical Computers," *Optic News*, **11**, 1985 pp. 11–20.
- [19] R. P. Feynman, "Simulating Physics with Computers," *International Journal of Theoretical Physics*, **21**, 1982 pp. 467–488.
- [20] S. Wolfram, *The Mathematica Book*, 4th ed., Wolfram Media, Cambridge: Cambridge University Press, 1999.
- [21] G. Mahler and I. Kim, "Moving Quantum Agents in a Finite Environment," P. Blanchard, D. Giulini, E. Joos, C. Kiefer, and I.-O. Stamatescu (Eds.) *Decoherence: Theoretical,*

Experimental, and Conceptual Problems Proceedings of a Workshop Held at Bielefeld, Germany, 10–14 November 1998, *Lecture Notes in Physics*, **538**, Springer Verlag.

- [22] G. Mahler and I. Kim, "Pattern Formation in Quantum Turing Machines," *Physics Review A*, **60**, 1999 pp. 692–695.
- [23] See Section: Additional Material.
- [24] R. P. Feynman, R. B. Leighton, and M. Sands, *Lectures on Physics*, Vol. 3, 6th ed., Reading, MA: Addison-Wesley Publishing Company, 1977.
- [25] P. Benioff, "The Computer as a Physical System: A Microscopic Quantum Mechanical Hamiltonian Model of Computers as Represented by Turing Machines," *Journal of Statistical Physics*, **22**, 1980 pp. 563–591.
- [26] P. Benioff, "Quantum Mechanical Hamiltonian Models of Turing Machines that Dissipate No Energy," *Physical Review Letters*, **48**, 1982 pp. 1581–1585.
- [27] P. Benioff, "Quantum Mechanical Hamiltonian Models of Computers," *Annals of the New York Academy of Sciences*, **480**, 1986 pp. 475–486.

■ Additional Material

qts.zip

Available at www.mathematica-journal.com.

About the Author

The author has done research in quantum field theory. His current research interest includes quantum information theory and computability theory.

Real Quaternions and Rotations

Robert Piziak

*Department of Mathematics
Baylor University
Waco, Texas 76798-7328
piziakr@baylor.edu*

Danny W. Turner

*Department of Mathematics
Winthrop University
Rock Hill, SC 29733
turnerd@winthrop.edu*

Hamilton's algebra of quaternions is introduced and implemented using *Mathematica* Version 4. The polar form of a quaternion is developed and is applied to the problem of computing the effect of a sequence of rotations on an object in three-dimensional space. Quaternion equivalents for rotations via Euler angles are illustrated.

■ Introduction

It can be said that the real quaternions were discovered (invented?) on October 16, 1843 by Sir William Rowan Hamilton (1805–1865), called Ireland's foremost mathematician. The story is well known and recounted in [1] and [2]. Indeed, a conference celebrating this historic event was held August 17–20, 1993 in Dublin, Ireland.

The real numbers $\mathbb{R}$ have an addition and a multiplication that satisfy the axioms of a field. In 1833, Hamilton showed that the two-dimensional space $\mathbb{R}^2$ of ordered pairs of reals also has operations extending those of $\mathbb{R}$ to yield the field $\mathbb{C}$ of complex numbers. Could these operations be extended further into the space of real three-tuples $\mathbb{R}^3$ or even to $\mathbb{R}^4$? This was the problem troubling Hamilton when a brilliant spark of insight came upon him while walking along a canal in Dublin. The story goes that he was so excited, he took out his pocket knife and carved the defining relations $\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1$ in the stone of the

Bourne Bridge then under construction. While we cannot condone this creative act of vandalism, we certainly empathize with the exuberance that a mathematician feels when the solution to a long standing problem is finally recognized.

■ Real Quaternions

Hamilton's great insight of abandoning the commutative law of multiplication led him to the four-dimensional space $\mathbb{R}^4$ of ordered four-tuples of real numbers, [3], and it is in this setting that we will use the computer algebra system *Mathematica* to deal with quaternions whose name comes from the Latin word *quaternio* meaning a *set of four*. Let $\mathbb{H}$ denote the algebraic system $(\mathbb{R}^4, +, \cdot, *)$ where addition is just vector (i.e., slotwise) addition and multiplication is given as follows.

$$(a, b, c, d) \cdot (x, y, z, w) = (ax - by - cz - dw, ay + bx + cw - dz, az + cx + dy - bw, aw + dx + bz - cy)$$

This definition of multiplication makes it plain why *Mathematica* will make the exploration of this system much less tedious. The idea of complex conjugation extends to $\mathbb{H}$ by defining quaternion *conjugation* $*$ by

$$(a, b, c, d)^* = (a, -b, -c, -d)$$

and this gives a notion of *norm* N defined as follows.

$$N((a, b, c, d)) = (a, b, c, d) \cdot (a, b, c, d)^* = a^2 + b^2 + c^2 + d^2.$$

We define the *magnitude* of a quaternion q to be $\sqrt{N(q)}$ and symbolize this by $|q|$.

Every nonzero quaternion q has a multiplicative *inverse* q^{-1} given by

$q^{-1} = \frac{q^*}{N(q)}$. We make contact with Hamilton's relations by defining $\mathbf{h} = (1, 0, 0, 0)$, $\mathbf{i} = (0, 1, 0, 0)$, $\mathbf{j} = (0, 0, 1, 0)$ and $\mathbf{k} = (0, 0, 0, 1)$. His relations follow from the multiplication defined above. If we identify a real number a with the quaternion $(a, 0, 0, 0)$ an *arbitrary quaternion* $q = (a, b, c, d)$ can be written $a\mathbf{h} + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}$, or commonly $(a + b\mathbf{i}) + c\mathbf{j} + d\mathbf{k}$, as in *Mathematica*. A complex number $a + b\mathbf{i}$ (or $a + bi$) can be identified with the quaternion $(a, b, 0, 0)$. Note that a real scalar a is identified with the quaternion $a\mathbf{h}$, which we call a *real quaternion*. Although quaternion operations are easily coded in *Mathematica*, there is currently a standard add-on package that provides many useful quaternion operations. Below we load the package and then provide some examples that demonstrate how some of the commands work.

```
In[1]:= << Algebra`Quaternions`
        Off[General::spell]; Off[General::spell1];
```

The following illustrate some calculations with the quaternions $A = (29, 34, 0, \sqrt{2}) = 29\mathbf{h} + 34\mathbf{i} + \sqrt{2}\mathbf{k}$ and $B = (80, 28, 1, 0) = 80\mathbf{h} + 28\mathbf{i} + \mathbf{j}$. First we define some quaternions and illustrate simple operations. The assignment for $\mathbf{Q}$ will be convenient.

```
In[3]:= Q = Quaternion;
A = Q[29, 34, 0,  $\sqrt{2}$ ];
B = Q[80, 28, 1, 0];
A - 2 B
```

```
Out[6]= Quaternion[-131, -22, -2,  $\sqrt{2}$ ]
```

Here is how to produce a “classic” form of a quaternion.

```
In[7]:= FromQuaternion[A + B]
```

```
Out[7]= (109 + 62 i) + J +  $\sqrt{2}$  K
```

Computing the norm, magnitude and conjugate are straightforward.

```
In[8]:= q = Q[1, 2, 3, 4];
Norm[q]
Abs[q]
Conjugate[q] // FromQuaternion
```

```
Out[9]= 30
```

```
Out[10]=  $\sqrt{30}$ 
```

```
Out[11]= (1 - 2 i) - 3 J - 4 K
```

Now let us look at the very important noncommutative quaternion multiplication operator. The next cell shows the long name and a shorter version, using the special operator ******. Of course, the indicated multiplication is correct symbolically.

```
In[12]:= q1 = Q[a, b, c, d];
q2 = Q[x, y, z, w];
NonCommutativeMultiply[q1, q2]
q1 ** q2 == %

Out[14]= Quaternion[-d w + a x - b y - c z,
  c w + b x + a y - d z, -b w + c x + d y + a z, a w + d x - c y + b z]

Out[15]= True
```

The inverse of a nonzero quaternion is simple to compute.

```
In[16]:= Clear[qinv];
Print["q = ", FromQuaternion[q]]
qinv =  $\frac{\text{Conjugate}[q]}{\text{Norm}[q]}$ ;
Print["q inverse = ", qinv // FromQuaternion]
```

$$q = (1 + 2i) + 3J + 4K$$

$$q \text{ inverse} = \left(\frac{1}{30} - \frac{i}{15} \right) - \frac{J}{10} - \frac{2K}{15}$$

Also, the inverse of a quaternion q is computable via the notation q^{-1} or $\frac{1}{q}$, yielding the easiest way to carry out this operation.

In[20]:= q^{-1}

Out[20]= Quaternion $\left[\frac{1}{30}, -\frac{1}{15}, -\frac{1}{10}, -\frac{2}{15}\right]$

It is simple to look at powers of quaternions. For nonsymbolic expressions, the process is exactly what one expects.

In[21]:= $q ** q ** q ** q // \text{Simplify}$
 q^4

Out[21]= Quaternion[668, -224, -336, -448]

Out[22]= Quaternion[668., -224., -336., -448.]

For symbolic expressions, input format makes a difference. However, a way to make our own power operator `quatpower[quaternion, power]` for symbolic quaternions is shown.

In[23]:= $\text{temp} = (q1 ** q1 ** q1) // \text{Simplify}$
 $q1^3$
`quatpower[q_, n_] :=`
`Nest[NonCommutativeMultiply[q, #] &, 1, n] // Simplify`
`quatpower[q1, 3] == temp`

Out[23]= Quaternion[a (a² - 3 (b² + c² + d²)), -b (-3 a² + b² + c² + d²),
 -c (-3 a² + b² + c² + d²), -d (-3 a² + b² + c² + d²)]

Out[24]= Quaternion[a, b, c, d]³

Out[26]= True

The quaternions $\mathbb{H}$ form a noncommutative division ring (skew field) with an involution (i.e., conjugation). It was the noncommutativity of multiplication that held Hamilton back for some time. It is probably difficult for us today to understand the courage it took to abandon the commutative law of multiplication in Hamilton's time.

Without getting into the philosophy of what constitutes a proof, we could easily "verify" various axioms for a division ring and create a series of exploratory examples for classroom use, for example. As an illustration, we look at multiplicative *commutators* $qrq^{-1}r^{-1}$ in $\mathbb{H}$. This is not the kind of computation the weak of heart would like to do by hand. But with *Mathematica*, it is a snap. Let us find the general symbolic form of any commutator and check that the norm of the result is one. Note the conversion to `List` forms to expedite the symbolic manipulations.

In[27]:= $q = Q[a, b, c, d];$
 $r = Q[x, y, u, v];$
 $\text{normq} = \text{Plus} @@ (\text{List} @@ q^2);$
 $\text{normr} = \text{Plus} @@ (\text{List} @@ r^2);$
 $\text{commutator} = \text{List} @@ (q ** r ** \text{Conjugate}[q] ** \text{Conjugate}[r]) / (\text{normq} \text{normr}) //$
`FullSimplify;`
`Print["The ", first, " component of the commutator is: ", commutator[[1]]]`
`Print["The norm of the commutator is: ", Plus @@ (commutator^2) // Simplify]`

The first component of the commutator is:

$$\frac{a^2 + b^2 - c^2 - d^2 + \frac{2((c u + d v)^2 - b^2 (u^2 + v^2) + (c^2 + d^2) x^2 + 2 b (c u + d v) y)}{u^2 + v^2 + x^2 + y^2}}{a^2 + b^2 + c^2 + d^2}$$

The norm of the commutator is: 1

Obviously, the pedagogical possibilities are endless. For another idea, there is a beautiful connection between quaternion multiplication of *pure quaternions* (projection on to **IJK**-space) and the standard $\mathbb{R}^3$ vector operations of inner product (dot product) and cross product. For now, however, we have adequate practice using the quaternion package and will move on to concepts related to our main theme of rotations.

■ The Polar Form of a Quaternion

In order to deal with rotations in $\mathbb{R}^3$, it will be convenient to represent quaternions in polar form. The reader is no doubt familiar with the polar form of a complex number $a + bI = r(\cos(t) + I \sin(t))$ where $r = \sqrt{a^2 + b^2}$, $\cos(t) = \frac{a}{r}$, $\sin(t) = \frac{b}{r}$, and $I^2 = -1$. There is an analogous representation of a quaternion. Let $q = (a, b, c, d)$. Then $q = a\mathbf{h} + s\mathbf{n}$ where $s = \sqrt{b^2 + c^2 + d^2}$ and $\mathbf{n} = (0, \frac{b}{s}, \frac{c}{s}, \frac{d}{s})$. Let $r = |q| = \sqrt{a^2 + b^2 + c^2 + d^2} = \sqrt{a^2 + s^2}$. Then $q = r(\frac{a}{r}\mathbf{h} + \frac{s}{r}\mathbf{n})$. But $(\frac{a}{r})^2 + (\frac{s}{r})^2 = 1$, so $(\frac{a}{r}, \frac{s}{r})$ lies on the unit circle of $\mathbb{R}^2$. Hence there is an angle t , $0 \leq t < 2\pi$ with $\cos(t) = \frac{a}{r}$ and $\sin(t) = \frac{s}{r}$. Thus $q = r(\cos(t)\mathbf{h} + \sin(t)\mathbf{n})$. We can easily compute the polar form of a nonzero quaternion. Three functions to accomplish this, **PolarQ**, **angle**, and **PolarQtrig**, are defined below. Each function takes a quaternion as its argument.

```
In[34]:= PolarQ[q_] := Abs[q] ((Q[ $\frac{q[[1]]}{Abs[q]}$ , 0, 0, 0] // FromQuaternion) +
 $\frac{AbsIJK[q]}{Abs[q]} (AdjustedSignIJK[q] // FromQuaternion)$ );
angle[q_] := N[ $\frac{ArcCos[\frac{q[[1]]}{Abs[q]}}{Degree}$ ] deg;
PolarQtrig[q_] :=
Abs[q] (Cos[angle[q]] (Q[1, 0, 0, 0] // FromQuaternion) +
Sin[angle[q]] (AdjustedSignIJK[q] // FromQuaternion));
```

Let us compute the polar form of $(-13.46, 1.89, -3, -6)$. We see the axis $\mathbf{n}$ and a numerical as well as symbolic polar form of the quaternion showing the associated angle.

```

In[37]:= tempq = Q[-13.46, 1.89, -3, -6];
          tempq // FromQuaternion
          n = AdjustedSignIJK[tempq] // FromQuaternion
          PolarQ[tempq]
          PolarQtrig[tempq]

Out[38]= (-13.46 + 1.89 i) - 3 J - 6 K

Out[39]= 0.271187 i - 0.430455 J - 0.86091 K

Out[40]= 15.1573 (-0.888021 + 0.459803 (0.271187 i - 0.430455 J - 0.86091 K))

Out[41]= 15.1573
          (Cos[152.626 deg] + (0.271187 i - 0.430455 J - 0.86091 K) Sin[152.626 deg])

```

One nice application of the polar form is extracting roots of a quaternion. Since $n^2 = -\mathbf{h}$ and $n \cdot \mathbf{h} = \mathbf{h} \cdot n$, De Moivre's Theorem works for quaternions and gives that $q^m = r^m (\cos(mt)\mathbf{h} + \sin(mt)n)$. If $q^m = y = \rho(\cos(\tau)\mathbf{h} + \sin(\tau)n)$ where $0 \leq \tau \leq \pi$, then $r^m = \rho$, $\cos(mt) = \cos(\tau)$, $\sin(mt) = \sin(\tau)$. Hence the m th roots of y are given by $r = \rho^{1/m}$, $t = \frac{\tau}{m} + k(\frac{2\pi}{m})$ for $k = 0, 1, 2, \dots, m-1$. If y is real, $\tau = mt$ is an integer multiple of π so that n is arbitrary (with the exception that $|n| = s = 0$ for $y > 0$ and $m = 2$). Thus we have the interesting result that (a) a nonreal quaternion has exactly m quaternionic m th roots; (b) a positive real quaternion has two square roots, but infinitely many m th roots when $m > 2$; and (c) a negative real quaternion has infinitely many m th roots for all $m \geq 2$. Let us find all cube roots of the quaternion $(1, 1, 1, 1) = 2(\cos(60^\circ)\mathbf{h} + \sin(60^\circ)n)$.

```

In[42]:= m = 3;
          q = Q[1, 1, 1, 1];
          w = PolarQtrig[q]
          w /. deg -> Degree // Simplify
          phi = angle[q]

Out[44]= 2 (Cos[60. deg] + (i/sqrt(3) + J/sqrt(3) + K/sqrt(3)) Sin[60. deg])

Out[45]= (1. + 1. i) + 1. J + 1. K

Out[46]= 60. deg

In[47]:= r = Abs[q]^(1/m)

Out[47]= 2^(1/3)

In[48]:= theta = (phi/m + d 360. deg/m);
          n = AdjustedSignIJK[q];
          cuberootsofq =
            Table[Flatten[r (Cos[theta] (Q[1, 0, 0, 0] // FromQuaternion) +
              Sin[theta] (n // FromQuaternion))], {d, 0, m-1}]
          numroots = cuberootsofq /. deg -> Degree // N // Simplify

```

```

Out[50]= { 2^{1/3} (Cos[20. deg] + ( \frac{i}{\sqrt{3}} + \frac{J}{\sqrt{3}} + \frac{K}{\sqrt{3}} ) Sin[20. deg] ),
          2^{1/3} (Cos[140. deg] + ( \frac{i}{\sqrt{3}} + \frac{J}{\sqrt{3}} + \frac{K}{\sqrt{3}} ) Sin[140. deg] ),
          2^{1/3} (Cos[260. deg] + ( \frac{i}{\sqrt{3}} + \frac{J}{\sqrt{3}} + \frac{K}{\sqrt{3}} ) Sin[260. deg] ) }

Out[51]= { (1.18394 + 0.248791 i) + 0.248791 J + 0.248791 K,
          (-0.965156 + 0.467574 i) + 0.467574 J + 0.467574 K,
          (-0.218783 - 0.716365 i) - 0.716365 J - 0.716365 K}

```

We can check our answers by cubing each of these quaternions.

```

In[52]:= (numroots // ToQuaternion) ^ 3 == {q, q, q}

Out[52]= True

```

■ Rotations

A *rotation* R in three-dimensional space is an orthogonal (i.e., length-preserving) linear transformation with determinant plus one. It is a fact that 1 is always an eigenvalue of such a map R , [4], so there has to be a nonzero vector $\mathbf{n}$ with $R(\mathbf{n}) = \mathbf{n}$. There is no loss in taking $\mathbf{n}$ to be a unit vector (normalize if you have to!).

In other words, every rotation is a rotation about a unit axis, the axis being the “line” through $\mathbf{n}$. Now select unit vectors $\mathbf{u}$ and $\mathbf{v}$ with $\text{crossproduct}(\mathbf{u}, \mathbf{v}) = \mathbf{n}$ and $R(\mathbf{u}) = \cos(t)\mathbf{u} + \sin(t)\mathbf{v}$ and $R(\mathbf{v}) = -\sin(t)\mathbf{u} + \cos(t)\mathbf{v}$. We see a rotation is determined by the data $(t, \mathbf{n})$ and we write $R = R_{t, \mathbf{n}}$ to so designate. Note that only vectors orthogonal to $\mathbf{n}$ are actually rotated by $R_{t, \mathbf{n}}$.

For example, consider an airplane with an inertial coordinate system put at the center of gravity of the plane. Say the x -axis goes from tail to nose, the y -axis goes across the wings, and the z -axis penetrates the airplane vertically. Then a rotation about the x -axis is called a “roll,” about the y -axis is called a “pitch,” and about the z -axis is called a “yaw.” If you yaw through α degrees and then roll through β degrees, is that the same as rolling through β degrees and then yawing through α degrees?

More generally, if $R_{t_1, \mathbf{n}_1}$ and $R_{t_2, \mathbf{n}_2}$ are rotations and they are performed sequentially, does the order matter and is there a single rotation that yields the same maneuver? Even more generally, what if we have a finite sequence of rotations $R_{t_1, \mathbf{n}_1}, \dots, R_{t_p, \mathbf{n}_p}$? There is a quick answer to the second question.

Since the rotations form the subgroup $\text{SO}(3)$ of the orthogonal group $\text{O}(3)$, yes, the composition (i.e., sequential application) of rotations is a rotation. But what is the angle and axis of the composite? The matrix representation of the rotation $R_{t, \mathbf{n}}$, say $\text{mat}(R_{t, \mathbf{n}})$, where $\mathbf{n} = (x, y, z)$, has been computed relative to the standard basis of $\mathbb{R}^3$ (and is of some interest in robotics, [5]). Here is the matrix.

$\text{mat}(R_{t,\mathbf{n}}) =$

$$\begin{pmatrix} x^2(1 - \cos(t)) + \cos(t) & x y(1 - \cos(t)) - z \sin(t) & x z(1 - \cos(t)) + y \sin(t) \\ x y(1 - \cos(t)) + z \sin(t) & y^2(1 - \cos(t)) + \cos(t) & y z(1 - \cos(t)) - x \sin(t) \\ x z(1 - \cos(t)) - y \sin(t) & y z(1 - \cos(t)) + x \sin(t) & z^2(1 - \cos(t)) + \cos(t) \end{pmatrix}$$

But, given this matrix (say numerically) how do you see the angle t and the axis $\mathbf{n}$? The authors in [6] have been quoted as saying “The greatest strength of quaternions is their ability to represent rotations.” It is this theme we develop to complete our paper. We could use matrix multiplication to answer our questions about rotations, but it is much easier and much more efficient (as well as elegant) to use quaternion multiplication.

■ Multiplication Is Rotation

Solomon [7] introduces the multiplication map $M_q(p) = q \cdot p \cdot q^*$ on $\mathbb{H}$ showing it to be a linear map taking pure quaternions to pure quaternions with the neat composition property $M_{q_1} \cdot M_{q_2} = M_{q_1 \cdot q_2}$. Now if $\mathbf{n} = (a, b, c)$ is a unit vector in $\mathbb{R}^3$ and $\mathbf{n}q = (0, a, b, c)$, then $q_{t,\mathbf{n}} = \cos(\frac{t}{2})\mathbf{h} + \sin(\frac{t}{2})\mathbf{n}q$ is a quaternion of unit magnitude for any angle t . Then $M_{q_{t,\mathbf{n}}}$ becomes a rotation through angle t on $\mathbb{R}^3$ with axis $\mathbf{n}$. Julstrom [8] shows the equivalence of the matrix approach to the multiplication map based on the function M . Note that for a unit quaternion, the conjugate is the inverse. The operations will be incorporated in the function **Rotate**[*vec*, *angle*, *axis*] that takes a vector *vec* about an axis *axis* a given number of degrees *angle*.

```
In[53]:= RotateQ[theta_, v_] :=
  Q @@ {Cos[theta/2], 0, 0, 0} + Sin[theta/2] Prepend[v, 0] /
    Sqrt[Plus @@ (v^2)];
Rotate[vector_, angle_, axis_] :=
  N[RotateQ[angle, axis] ** (Q @@ (Prepend[vector, 0])) **
    (RotateQ[angle, axis])^-1] // Rest // Chop
```

Let us rotate the vector (1, 1, 1) 45 degrees about the x -axis represented by (1, 0, 0).

```
In[55]:= Rotate[{1, 1, 1}, 45 Degree, {1, 0, 0}]
```

```
Out[55]:= Quaternion[1., 0, 1.41421]
```

Thus the vector (1, 1, 1) when rotated about the x -axis has new position (1, 0, 1.41421). Now look at another example. Rotate (3, 4, 5) 50 degrees about the axis determined by the nonunit vector (2, -1, 1).

```
In[56]:= Rotate[{3, 4, 5}, 50 Degree, {2, -1, 1}]
```

```
Out[56]:= Quaternion[-0.0527686, -0.0347517, 7.07079]
```

So, (3, 4, 5) rotates to (−0.0527686, −0.0347517, 7.07079).

Next we illustrate the very important fact that any sequence of rotations about various axes through the origin is equivalent to a single rotation about a single axis intersecting the origin. In fact, if $r_1 = \cos(\frac{t}{2})\mathbf{h} + \sin(\frac{t}{2})\mathbf{n}q$ and $r_2 = \cos(\frac{s}{2})\mathbf{h} + \sin(\frac{s}{2})\mathbf{m}q$ represent two rotations then $r_2 r_1 = M_{q_{s,m}} \cdot M_{q_{t,n}} = M_{q_{s,m} \cdot q_{t,n}}$ so we can recover the composite angle and axis from the product of two quaternions. To illustrate take the vector (5, 7, 9) first rotated by 25 degrees about axis (1, .5, 1) and then 15 degrees about (.5, 0, 1)

```
In[57]:= Rotate[Rotate[{5, 7, 9}, 25 Degree, {1, .5, 1}], 15 Degree, {.5, 0, 1}]
```

```
Out[57]:= Quaternion[3.27799, 5.27079, 10.7923]
```

So the vector (5, 7, 9) terminates at (3.27799, 5.27079, 10.7923). Next do the rotations in the reverse order and see what difference that makes.

```
In[58]:= Rotate[Rotate[{5, 7, 9}, 15 Degree, {.5, 0, 1}], 25 Degree, {1, .5, 1}]
```

```
Out[58]:= Quaternion[3.18801, 4.76137, 11.0529]
```

Now (5, 7, 9) terminates at (3.18801, 4.76137, 11.0529). So the order matters!

We can illustrate the multiplicative property of the mapping M by looking directly at the quaternions that represent the two rotations above.

```
In[59]:= r = rotateq[N[25 Degree], {1, .5, 1}]
s = rotateq[N[15 Degree], {.5, 0, 1}]
v = {5, 7, 9};
```

```
Out[59]:= Quaternion[0.976296, 0.144293, 0.0721465, 0.144293]
```

```
Out[60]:= Quaternion[0.991445, 0.0583731, 0, 0.116746]
```

```
In[62]:= t = s ** r
t ** Q @@ Prepend[v, 0] ** t^-1 // Rest
```

```
Out[62]:= Quaternion[0.942675, 0.191625, 0.0799521, 0.261249]
```

```
Out[63]:= Quaternion[3.27799, 5.27079, 10.7923]
```

Now the amazing fact is that we can easily extract the axis and the angle from a quaternion that represents a rotation. The first component of the rotation is the cosine of half of the angle of rotation while the remaining three components give the axis though not necessarily as a unit vector. The latter is easily normalized. So using the example above, we can determine what the angle of t is as well as the unit axis.

```
In[64]:= 2 angle[t]
AdjustedSignIJK[t] // Rest
```

```
Out[64]:= 38.9884 deg
```

```
Out[65]:= Quaternion[0.574224, 0.239584, 0.782858]
```

Thus the single angle that accomplishes $\mathbf{r}$ and $\mathbf{s}$ above is 38.9884 degrees about the axis (.574224, .239584, .782858). We verify this next.

```
In[66]:= Rotate[{5, 7, 9}, 38.9884 Degree, {.574224, .239584, .782858}]
```

```
Out[66]:= Quaternion[3.27799, 5.27079, 10.7923]
```

Next let us do a 90 degree rotation about the x -axis followed by a 90 degree rotation about the z -axis and ask what single rotation accomplishes this sequence.

```
In[67]:= r = rotateq[N[90 Degree], {1, 0, 0}]
s = rotateq[N[90 Degree], {0, 0, 1}]
t = s ** r
AdjustedSignIJK[t] // Rest
2 angle[t]
```

```
Out[67]:= Quaternion[0.707107, 0.707107, 0, 0]
```

```
Out[68]:= Quaternion[0.707107, 0, 0, 0.707107]
```

```
Out[69]:= Quaternion[0.5, 0.5, 0.5, 0.5]
```

```
Out[70]:= Quaternion[0.57735, 0.57735, 0.57735]
```

```
Out[71]:= 120. deg
```

So we see the sequence of these 90 degree rotations is equivalent to a 120 degree rotation about $(1, 1, 1)$, which has been normalized to $(.57735, .57735, .57735)$. As a last illustration here, let us find the result of a 180 degree rotation about the x -axis followed by a 120 degree rotation about $(1, 1, -1)$ followed by a 90 degree rotation about $(0, 1, 0)$.

```
In[72]:= r = rotateq[N[180 Degree], {1, 0, 0}] // Chop
s = rotateq[N[120 Degree], {1, 1, -1}]
t = rotateq[N[90 Degree], {0, 1, 0}]
u = t ** s ** r // Chop
2 angle[u]
AdjustedSignIJK[u] // Rest
```

```
Out[72]:= Quaternion[0, 1., 0, 0]
```

```
Out[73]:= Quaternion[0.5, 0.5, 0.5, -0.5]
```

```
Out[74]:= Quaternion[0.707107, 0, 0.707107, 0]
```

```
Out[75]:= Quaternion[0, 0, -0.707107, -0.707107]
```

```
Out[76]:= 180. deg
```

```
Out[77]:= Quaternion[0, 0.707107, 0.707107]
```

The result is a 180 degree rotation about $(0, 1, 1)$.

Finally, what if we want to rotate a vector about an axis that does not pass through the origin? The geometry of the situation is clear. Translate the axis to the origin, do the rotation, and then translate back. Let p denote an axis and v a vector that is to be rotated through an angle t about p . The v rotates to $r(v-p)r^{-1} + p$ where r is the rotation through angle t . Let us rotate $v = (1, 1, 1)$ through 45 degrees about an axis parallel to the x -axis but passing through $p = (0, 2, 2)$.

```

In[78]:= Rotatevarb[v_, r_, p_] :=
  r ** Q @@ Prepend[v - p, 0] ** r-1 + Q @@ Prepend[p, 0] // Rest

In[79]:= v = {1, 1, 1};
p = {0, 2, 2};
r = N[rotateq[N[45 Degree], {1, 0, 0}]]
Rotatevarb[v, r, p]

Out[81]:= Quaternion[0.92388, 0.382683, 0., 0.]

Out[82]:= Quaternion[1., 2., 0.585786]

```

Thus (1, 1, 1) has final resting position (1, 2, .585786).

For more on this fascinating topic confer Kuipers [9]

■ A Graphics Animation via Quaternions

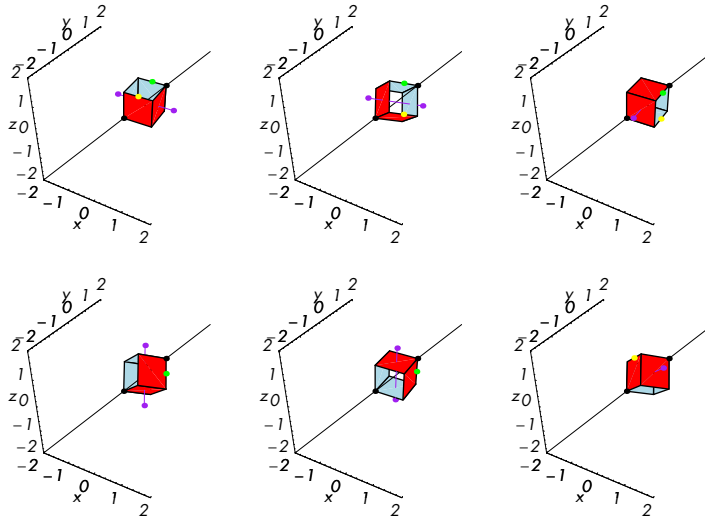
We construct an example of an animation that shows how easy it is to rotate a three-dimensional graphics object using a quaternion approach. Basically a “fancy box” is rotated about the axis (1,1,1). Note that most of the code below is used to define the graphics object. Our quaternion based **Rotate** function, defined earlier, then easily implements the desired rotations.

```

In[83]:= Needs["Graphics`Colors`"];
faces = {{{{0, 0, 0}, {1, 0, 0}, {1, 0, 1}, {0, 0, 1}},
  {{{0, 0, 0}, {0, 0, 1}, {0, 1, 1}, {0, 1, 0}},
  {{{1, 0, 0}, {1, 1, 0}, {1, 1, 1}, {1, 0, 1}},
  {{{1, 1, 0}, {0, 1, 0}, {0, 1, 1}, {1, 1, 1}}};
axis = {1, 1, 1};
dots = {{.5, 0, 1}, {.5, 1, 1}, {1.5, .5, .5}, {-.5, .5, .5}};
Do[{t = angle Degree,
  points = Table[List @@ Rotate[dots[[m]], t, axis], {m, 1, 4}],
  rotatedfaces = Table[
    Table[List @@ Rotate[faces[[c, m]], t, axis], {m, 1, 4}], {c, 1, 4}],
  box = Table[Polygon[rotatedfaces[[m]]], {m, 1, 4}],
  Show[Graphics3D[{{FaceForm[Red, LightBlue], box}, PointSize[.02],
    Thickness[.003], Line[{{-2, -2, -2}, {2, 2, 2}}],
    PointSize[.03], Yellow, Point[points[[1]]], Green,
    Point[points[[2]]], Purple, Point[points[[3]]], Point[points[[4]]],
    Line[{points[[3]], points[[4]]}, Black, Point[{0, 0, 0}],
    Point[{1, 1, 1}]}, Axes → True, Lighting → False,
  Boxed → False, AxesLabel → {x, y, z},
  PlotRange → {{-2, 2}, {-2, 2}, {-2, 2}}]], {angle, 0., 360., 30.}]

```

From In[83]:=



■ Euler Angles

Although our emphasis in this article is on quaternions, there are of course other ways to approach the subject of rotations and we would like to briefly mention rotations via Euler angles. The standard add-on package **Geometry`Rotations`** provides us with the function **Rotate3D**. More specifically, **Rotate3D[vec, phi, theta, psi, (pt:{a,b,c})]** rotates the vector *vec* by specified Euler angles about point *pt*. The rotation given by the Euler angles *phi*, *theta*, and *psi* can be decomposed into a sequence of three successive rotations:

phi about the *z*-axis,

theta about the *x*-axis, range 0 to π (typical, but not required), and

psi about the *z*-axis (again).

This version of Euler angles is not unique and definitions vary.

For an interesting in-depth article on rotation techniques in the context of commercial computer game software confer [10]. Bobick's paper provides numerous technical reasons why quaternions are a must in many three-dimensional applications as he compares and contrasts various techniques.

In our article we shall show a simple way to make an Euler-to-quaternion conversion and produce some examples. First load some useful standard add-on packages.

```
In[88]:= << Geometry`Rotations`
         << Graphics`Animation`
```

Noting the definition of Euler angles used in the **Rotations** package, and using some of the theory of quaternions previously discussed, we can construct a single

quaternion that is equivalent to a given Euler angles rotation. The negative signs make the rotation choices (CW, CC) about the axes consistent with the choices made in the add-on package. The function **etoq**[*phi*, *theta*, *psi*] creates the desired quaternion.

```
In[90]:= etoq[phi_, theta_, psi_] :=
  rotateq[-psi, {0, 0, 1}] ** rotateq[-theta, {1, 0, 0}] **
  rotateq[-phi, {0, 0, 1}];
```

The first three basic equivalences are obviously correct, while the fourth is more complicated.

```
In[91]:= PolarQtrig[etoq[Pi/2, 0, 0]]
PolarQtrig[etoq[0, Pi/4, 0]] // Simplify
PolarQtrig[etoq[0, 0, 30 Degree]] // Simplify
PolarQtrig[etoq[Pi/6, 29.0 Degree, -12.0 Degree]]

Out[91]= Cos[45. deg] + K Sin[45. deg]
Out[92]= Cos[22.5 deg] + i Sin[22.5 deg]
Out[93]= Cos[15. deg] + K Sin[15. deg]
Out[94]= 1.
  (Cos[17.015 deg] + (0.798811 i + 0.306635 J + 0.517567 K) Sin[17.015 deg])
```

Next we can use the quaternion created by **etoq** to carry out a rotation of vector *vec* about point *pt* that is equivalent to the Euler rotation. The function is **etoqrot**[*vec*, *phi*, *theta*, *psi*, *pt*].

```
In[95]:= etoqrot[vec_, phi_, theta_, psi_, pt_] :=
  etoq[phi, theta, psi] ** Q @@ Prepend[vec - pt, 0] **
  etoq[phi, theta, psi]-1 + Q @@ Prepend[pt, 0] // Rest //
  Simplify;
```

Now compare **etoqrot** and **Rotate3D** by rotating the vector (1, 1, 5) about the point (2, -1, -3) with Euler triple ($\frac{\pi}{4}$, -10° , $\frac{\pi}{12}$). We see that the rotation, as applied to *vec-pt*, involves an angle of about 60.7° around the axis $0.166486 i + 0.0446097 J - 0.985034 K$ and the final position (after translating) is (3.13382, 0.499433, 5.33798).

```
In[96]:= vec = {1, 1, 5};
pt = {2, -1, 3};
phi = Pi/4;
theta = -10.0 Degree;
psi = Pi/12;
PolarQtrig[etoq[phi, theta, psi]]
Rotate3D[vec, phi, theta, psi, pt]
etoqrot[vec, phi, theta, psi, pt]

Out[101]= 1. (Cos[30.3755 deg] +
  (0.166486 i + 0.0446097 J - 0.985034 K) Sin[30.3755 deg])

Out[102]= {3.13382, 0.499433, 5.33798}

Out[103]= Quaternion[3.13382, 0.499433, 5.33798]
```

■ A Graphics Animation via Euler Angles

Our final example demonstrates a simple “flight simulator” controlled by rotations using Euler angles. The flight starts at the origin and movements of the orange craft occur about the blue planet located at point (1, 1, -1).

```
In[104]:= a = {0, 0, 0}; pt = {1, 1, -1};
```

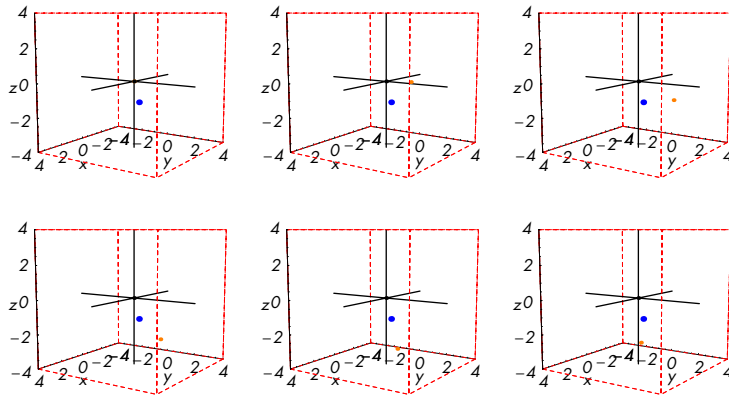
```
standardaxes = Graphics3D[
  {Line[{{-4, 0, 0}, {4, 0, 0}}], Line[{{0, -4, 0}, {0, 4, 0}}],
  Line[{{0, 0, -4}, {0, 0, 4}}]};

flight = Table[{standardaxes,
  Graphics3D[{{PointSize[.02], Point[a]}, {PointSize[.02], Orange,
    Point[Rotate3D[a, ang, ang, ang/2, pt]]},
    {PointSize[.03], Blue, Point[pt]}}]},
  {ang, 0, 360 Degree, 10 Degree} ];

myoptions = {Axes → True, AxesEdge → {{-1, -1}, {-1, -1}, Automatic},
  AxesLabel → {x, y, z}, BoxStyle → {Dashing[{0.02, 0.02}], Red},
  AspectRatio → Automatic, ViewPoint → {3, 2, .5},
  PlotRange → {{-4, 4}, {-4, 4}, {-4, 4}}};

ShowAnimation[flight, myoptions];
```

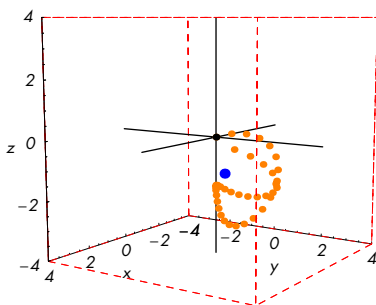
From In[104]:=



Finally, we show a trace of the entire flight path.

```
In[109]:= Show[flight, myoptions];
```

From In[109]:=



■ References

- [1] B. L. van der Waerden, "Hamilton's Discovery of Quaternions," *Mathematics Magazine*, **9**(5), 1976.
- [2] H. Eves, *An Introduction to the History of Mathematics*, 4th ed., New York: Holt, Rinehart, and Winston, 1976.
- [3] W. R. Hamilton, *Elements of Quaternions*, W. E. Hamilton, ed. London: Longmans, Green, and Company, 1866.
- [4] C. W. Curtis, *Linear Algebra: An Introductory Approach*, 3rd ed., Boston: Allyn and Bacon, Inc., 1974.
- [5] R. P. Paul, *Robot Manipulators: Mathematics, Programming and Control*, Cambridge, MA: MIT Press, 1981.
- [6] E. Pervin and J. A. Webb, "Quaternions in Computer Vision and Robotics," *Technical Report CMU-CS-82-150*, Pittsburgh, PA: Department of Computer Science, Carnegie Mellon University, 1982.
- [7] F. Solomon, "Using Quaternions to Compose Rotations," *UMAP Modules in Undergraduate Mathematics and Its Applications*, Module 313, Lexington, MA: COMAP, Inc., 1979.
- [8] B. A. Julstrom, "Using Real Quaternions to Represent Rotations in Three Dimensions," *The UMAP Journal*, Lexington, MA: COMAP, Inc., Summer 1992.
- [9] J. B. Kuipers, *Quaternions and Rotation Sequences*, Princeton, NJ: Princeton University Press, 1998.
- [10] N. Bobick, "Rotating Objects Using Quaternions," *Game Developer*, February, 1998.

Orthogonal Series Representation of Probability Density and Distribution Functions

J. Todd Reinking

*Mission Research Corporation
102 S. Tejon St.; Suite 200
Colorado Springs, CO 80909
treinking@mrccos.com*

When justified, moment methods may be used to evaluate probability density and distribution functions. These methods are particularly useful when moments are the only available information on the random variable of interest. A classic example of the technique considered here is the Gram–Charlier series representation of a probability density function. This paper provides a package to obtain three different orthogonal series expansions of a probability density function given a user-supplied function for moments. The three orthogonal series representations are based on Hermite (which gives the Gram–Charlier series), Laguerre, and Jacobi orthogonal polynomials and their corresponding weight functions.

■ Introduction

An expansion of continuous probability density functions or cumulative distribution functions into a series of orthogonal functions constructed from knowledge of moments of the probability density function can serve as a valuable tool in approximating either the density or distribution function when knowledge of the probability density function is limited to its moments. A classic example of a series of this type is the Gram–Charlier series [1]. Unfortunately, casual use of such a series is thwarted by the nontrivial relation between the series coefficients and the moments. Evaluation of more than a few coefficients of such a series is tedious at best.

Fortunately for the *Mathematica* user, the tedium can be eradicated. This paper describes a package, **ProbabilitySeries**, which defines functions for generating the formal expansion of a density or distribution function into a series of orthogonal polynomials. Specifically, functions are provided for finding the Gram–Charlier series and what are referred to here as the Gamma-series and Beta-series. The three expansions are based on the Hermite, Laguerre, and Jacobi orthogonal polynomials. Each of these polynomials is orthogonal over an interval with respect to its corresponding weight functions. These weight functions may be written in terms of the standard Gaussian, Gamma, and Beta density functions, respectively [2]. For each density function expansion considered below, the zero-order term is one of these common density functions.

Although application of these series can only briefly be discussed in the limited space here, it must be mentioned that in general there is no guarantee of convergence for the series discussed above, nor may a probability density function be uniquely determined by its moments alone [3]. Conditions under which convergence occurs are discussed in [1]. The moment methods considered here are most safely applied when there is prior reason to expect that the unknown density may be closely approximated by the weight function underlying the series expansion. A typical example would be a sum, Σ , of N random variables to which the central limit theorem is applicable. The probability density of such a sum may be approximated by a Gaussian density function when N is large. Truncations of the Gram–Charlier series may provide significant improvement over use of the Gaussian density alone, as will be evident from examples. In cases where $\Sigma \in (a, \infty)$ or $\Sigma \in (a, b)$, the Gamma-series or Beta-series, respectively, may provide an improved approximation over the Gram–Charlier series, as noted in [4].

■ General Orthogonal Series Representation

The package discussed here finds a formal series for a probability density function of the following general form. The specific parameters of this representation are given for the Gram–Charlier series, Gamma-series and Beta-series in the following section. For a density function, $f(x)$, the formal series representation is

$$f(x) = c \phi(y(x)) \left[1 + \sum_{k=3}^{\infty} D_k P_k(y(x)) \right]; \quad x_0 \leq x \leq x_1 \quad (1)$$

where $P_k(t)$ is an orthogonal polynomial of degree k with respect to the weight function $\phi(t)$ on an interval (t_0, t_1) , so

$$\int_{t_0}^{t_1} \phi(t) P_n(t) P_m(t) dt = \begin{cases} \zeta_m & m = n \\ 0 & m \neq n \end{cases} \quad (2)$$

In addition,

$$y(x) = \frac{x-a}{b} \quad (3)$$

$$D_k = \frac{1}{b c \zeta_k} \int_{x_0}^{x_1} f(x) P_k(y(x)) dx \quad (4)$$

Parameters of the orthogonal polynomial sequence, P_k , along with a and b are chosen to assure that $D_1 = D_2 = 0$ and to assure y maps (x_0, x_1) into (t_0, t_1) . (The latter is not possible for all combinations of $f(x)$ and $\phi(x)$.) Note that, since P_k is a polynomial, the integral in the definition of D_k is simply a linear combination of the moments of f . Therefore,

$$\int_{x_0}^{x_1} f(x) P_k(y(x)) dx = \mathbf{P}_k^T \cdot \mathbf{u} \quad (5)$$

where

$$\mathbf{P}_k^T = [p_{0,k} \ p_{1,k} \ \dots \ p_{k,k}] \quad \text{and} \quad P_k(x) = \sum_{n=0}^k p_{n,k} x^n \quad (6)$$

and

$$\mathbf{u}^T = [\mu_0 \ \mu_1 \ \dots \ \mu_k] \quad \text{and} \quad \mu_k = \int_{x_0}^{x_1} f(x) y^k(x) dx \quad (7)$$

A series representation for the cumulative distribution function $F(x)$

$$F(x) = \int_{-\infty}^x f(x) dx \quad (8)$$

is also possible. After substituting equation (1) into equation (8), F takes the following form.

$$F(x) = c \left[\int_{x_0}^x \phi(y(x)) dx + \rho(y(x)) \sum_{k=3}^{\infty} D_k T_{k-1}(y(x)) \right]; \quad x_0 \leq x \leq x_1 \quad (9)$$

In the previous expression, T_k is a sequence of orthogonal polynomials of the same type (Hermite, Generalized Laguerre, ...) as P_k . The D_k are identical to those in equation (1). The integral on the right-hand side of equation (9) is a standard “special function” (and *Mathematica* function) for the series considered here.

■ Three Specific Orthogonal Series Representations

In this section, the parameters of equation (1) are specified that result in the Gram–Charlier, Gamma, and Beta series. Let

$$\begin{aligned} \bar{x} &= E\{x\} \sigma_x^2 = E\{(x - \bar{x})^2\} \\ \bar{y} &= E\{y(x)\} \sigma_y^2 = E\{(y(x) - \bar{y})^2\} \end{aligned}$$

where $E\{\cdot\}$ is the expectation operator, in this case, with respect to the density $f(x)$.

Table 1 lists the probability density and distribution series parameters for the specific cases of interest here. Note that $H_k(x)$, $L_k^{(a)}(x)$, and $G_k(p, q, x)$ are the

	Gram–Charlier	Gamma	Beta
(a, b, c)	$\left(\bar{x}, \sigma_x, \frac{1}{\sqrt{2\pi\sigma_x^2}}\right)$	$\left(x_0, \beta, \frac{1}{\beta\Gamma(\alpha+1)}\right)$	$\left(x_0, x_1 - x_0, \frac{\Gamma(\alpha+\beta+2)}{\Gamma(\alpha+1)\Gamma(\beta+1)(x_1 - x_0)}\right)$
α	—	$\frac{(\bar{x}-x_0)^2}{\sigma_x^2} - 1$	$\frac{\bar{y}(\bar{y}-\bar{y}^2-\sigma_y^2)}{\sigma_y^2} - 1$
β	—	$\frac{\sigma_x^2}{\bar{x}-x_0}$	$\frac{(1-\bar{y})(\alpha+1)}{\bar{y}} - 1$
ϕ	$\exp(-x^2/2)$	$x^\alpha \exp(-x); x \geq 0$	$x^\alpha (1-x)^\beta; 0 \leq x \leq 1$
ρ	$\sigma_x^2 \exp(-x^2/2)$	$\beta x^{\alpha+1} \exp(-x)$	$x^{\alpha+1} (1-x)^{\beta+1} (x_1 - x_0)$
ζ_k	$k!$	$\frac{\Gamma(\alpha+k+1)}{\Gamma(\alpha+1)k!}$	$\frac{k! \Gamma(k+\alpha+1) \Gamma(k+\alpha+\beta+1) \Gamma(k+\beta+1)}{(2k+\alpha+\beta+1) \Gamma^2(2k+\alpha+\beta+1)}$
P_k	$(-1)^k H_k(x)$	$L_k^{(\alpha)}(x)$	$G_k(\alpha+\beta+1, \alpha+1, x)$
T_k	$(-1)^{k+1} H_k(x)$	$\frac{L_k^{(\alpha+1)}(x)}{k+1}$	$\frac{G_k(\alpha+\beta+3, \alpha+2, x)}{k+\alpha+\beta+1}$
Base PDF	$\frac{1}{\sqrt{2\pi\sigma_x^2}} \exp\left(-\frac{(x-\mu_x)^2}{2\sigma_x^2}\right)$	$\frac{x^\alpha \exp(-\frac{x}{\beta})}{\beta^{\alpha+1} \Gamma(\alpha+1)}; x \geq 0$	$\frac{\Gamma(\alpha+\beta+2)}{\Gamma(\alpha+1)\Gamma(\beta+1)} x^\alpha (1-x)^\beta; 0 \leq x \leq 1$

Table 1. PDF and CDF series parameters for the Gram–Charlier, Gamma, and Beta series.

Hermite, Generalized Laguerre, and Jacobi orthogonal polynomials of degree k as defined in [2] with respect to the corresponding weights, ϕ , above. (The Hermite and Jacobi polynomials used here are defined with respect to different weight functions than the built-in *Mathematica* functions for these polynomial types.) The weight functions used here assure that the first term of the Gram–Charlier, Gamma, or Beta series is the Gaussian, Gamma, or Beta density function, respectively, as shown in the final row of the table. This first term is referred to as the base probability density function in the remainder.

■ Code Samples

Since the actual package is too lengthy to present here, some code segments similar to those in the package will be presented. The discussion here should help all those interested in modifying the contents of the package for their own needs. The specific problem considered here is the generation of the Gram–Charlier series for a probability density function given its characteristic function.

As noted previously, the resident function **HermiteH** is defined with respect to a different weight than is natural for the problem at hand. An alternate function, **HermiteE**, with the correct weight ϕ in Table 1 is defined for use here.

```
In[1]:= Off[General::spell, General::spell1]

In[2]:= $Version

Out[2]= 4.1 for Microsoft Windows (November 2, 2000)
```

```
In[3]:= HermiteE[n_, x_] :=
      HermiteE[n, x] = Expand[x HermiteE[n - 1, x] - (n - 1) HermiteE[n - 2, x]]
In[4]:= HermiteE[0, x_] := 1
In[5]:= HermiteE[1, x_] := x
```

Use of **Expand** is strictly for economy. It keeps the resultant polynomial in simple, standard form during the recursion, allowing polynomial coefficients to be quickly extracted later.

The first argument to **HermiteE** is the polynomial degree. A quick check of **HermiteE** gives the expected result.

```
In[6]:= HermiteE[5, y]
Out[6]= 15 y - 10 y^3 + y^5
```

Two functions will be defined to carry out the linear combination of moments in equation (5). The first function, **BuildMoments**, will build up a list of moments from the characteristic function. The second function, **aGramCharlier**, will simply perform the vector multiplication indicated on the right-hand side of equation (5).

Before writing out the code segment for **BuildMoments**, recall from elementary probability theory that the characteristic function of a density function, f , is

$$\Phi(\omega) = \int_{-\infty}^{\infty} f(x) \exp(i\omega x) dx \quad (10)$$

The moments of f may be found from Φ using

$$\mu_k = (-i)^k \left. \frac{d^k \Phi(\omega)}{d\omega^k} \right|_{\omega=0} \quad (11)$$

BuildMoments appends the n th moment, evaluated using the input characteristic function as in equation (11), to a list of the first $n - 1$ moments which have been evaluated on previous calls to **BuildMoments**.

```
In[7]:= BuildMoments[n_, f_] := Module[{x, mom}, If[n == 0, Rmoments = f[t];
      MomentList = {1}, Rmoments = -I D[Rmoments, t];
      AppendTo[MomentList, Rmoments /. t -> 0];];
      MomentList]
```

Evaluation of equation (5) is finalized using the function **aGramCharlier**, whose magnitude is equivalent to $|D_n|$ (see equation (4)) for the Gram–Charlier series. (Note that this function gives the series coefficient commonly defined in textbooks for the Gram–Charlier series.)

```
In[8]:= aGramCharlier[n_, f_] :=
      (-1)^n/n! * BuildMoments[n, f].CoefficientList[HermiteE[n, x], x]
```

As a test problem, the Gram–Charlier series of the following Gaussian density function will be found.

$$f(x) = \frac{1}{\sqrt{2\pi\sigma_x^2}} \exp\left(-(x - \mu_x)^2 / 2\sigma_x^2\right) \quad (12)$$

Let the random variable x have the above density: then $y(x)$ as given in equation (3) has the following characteristic function.

```
In[9]:= py[t_] := Exp[-t^2/2]
```

In final preparation for evaluation of the Gram–Charlier series, the list of moments is initialized to the first three moments by evaluating **aGramCharlier** three times.

```
In[10]:= Table[aGramCharlier[n, py], {n, 0, 2}]
```

```
Out[10]= {1, 0, 0}
```

Evaluation of **aGramCharlier** should return the above result for $y(x)$, given any random variable x , assuming the moments of x are bounded through order 2. With **aGramCharlier** defined as above, evaluation of the first 20 terms of equation (1) is readily carried out for the Gram–Charlier series via

```
In[11]:= Exp[-y^2/(2)]/(Sqrt[2 Pi] sx)*Collect[
  1+Sum[(-1)^k HermiteE[k, y]*aGramCharlier[k, py], {k, 3, 20}],
  y]/.y -> (x-ux)/sx
```

```
Out[11]= 
$$\frac{e^{-\frac{(-ux+x)^2}{2sx^2}}}{\sqrt{2\pi} sx}$$

```

Not a very exciting result at first glance. The base PDF was returned with the same mean and variance as $f(x)$ as expected. (Note $ux = \mu_x$ and $sx = \sigma_x$ above.) As a test case, however, it is important that all 19 higher-order terms were returned as zero.

■ Package Usage

Code similar to the input of the previous section is at the core of the package **ProbabilitySeries**. The package **ProbabilitySeries** defines six functions. The first three—**GramCharlierSeries**, **GammaSeries**, and **BetaSeries**—generate a series for a density function. The remaining three functions generate a series for the complementary-cumulative distribution function, or, Q -function for short. The Q -function is related to the distribution function, $F(x)$, as follows.

$$Q(x) = 1 - F(x) \quad (13)$$

The Q -functions are: **GramCharlierQ**, **GammaQ**, and **BetaQ**. Use of several of these functions is demonstrated in the remainder of this paper. Those omitted from this discussion should be applied in a similar fashion.

As was done in the previous section, the simplest demonstration of these functions is to show that they return only the zero-order term when applied to the moments for the base density function. This exercise is carried out for the functions **GramCharlierSeries**, **GammaSeries**, and **BetaSeries**.

```
In[12]:= << ProbabilitySeries.m
```

As in the case of **GramCharlierSeries** above, each of these functions requires that either a characteristic function or a function to directly evaluate moments be supplied. In the first example, **GramCharlierSeries** is called on to generate the first 20 series terms using the characteristic function of a Gaussian density function.

```
In[13]:= charGauss[w_] := Exp[I w ux - (w sx)^2/2]
```

```
In[14]:= GramCharlierSeries[x, 20, charGauss, Moment → CharFun]
```

$$\text{Out[14]} = \frac{e^{-\frac{(-ux+x)^2}{2sx^2}}}{\sqrt{2\pi} \sqrt{sx^2}}$$

The series is exactly equal to the Gaussian density function with mean and variance specified by the characteristic function. Higher-order terms are zero as expected.

In the next two examples, the functions **momGamma** and **momBeta** generate the moments for a Gamma and Beta density, respectively.

GammaSeries[x,n,f,xorigin] returns the first n terms of the Gamma series for the PDF with the moments supplied by f. The function f either gives the nth moment directly (Moment→MomFun) or, for the default case, is a characteristic function which is used to find the moments (Moment→CharFun). PDF[x]==0 for all x < xorigin. GammaSeries returns unevaluated if x is a numerical argument < xorigin.

```
In[15]:= a = 3; b = 2;
```

```
In[16]:= momGamma[k_] := b^k Gamma[a + 1 + k] / Gamma[a + 1]
```

```
In[17]:= GammaSeries[x, 20, momGamma, 0, Moment → MomFun]
```

$$\text{Out[17]} = \frac{1}{96} e^{-x/2} x^3$$

```
In[18]:= momBeta[k_] :=
  Gamma[a + k + 1] Gamma[a + b + 2] / (Gamma[a + 1] * Gamma[a + k + b + 2])
```

```
In[19]:= BetaSeries[x, 20, momBeta, 0, 1, Moment → MomFun]
```

$$\text{Out[19]} = 60 (1 - x)^2 x^3$$

The results are exactly equal to the Gamma and Beta density function corresponding to the supplied moments. Again, higher-order terms are zero as expected.

The next two examples are concerned with the following sum

$$z = \sum_{n=1}^N u_n \quad (14)$$

where $\{u_n\}$ is a sequence of independent random variables, each uniformly distributed on $[-1/2, 1/2)$. As N approaches infinity, the density function of z will tend to a Gaussian density as predicted by the central limit theorem, so z appears a suitable target for application of one of our density function series representations. The most common application of the methods described here

are to problems of the form in equation (14). Moments of z are easily found in this case provided the characteristic function of the u_n is known.

The specific case considered is $N = 10$. This case is examined partly because z with $N = 10$ is still occasionally suggested as a computational algorithm to generate Gaussian variates given a uniform random number generator. The following efforts will show as a side result how well z approximates a Gaussian random variate. (Evaluation of equation (14) for $N = 10$ is a poor Gaussian random number generator, yielding mediocre accuracy for the required number of uniform variates. See [5] for excellent alternates.)

For this example, **fz[x]** gives an exact expression for the density function of z . In order to generate the Gram–Charlier series, a function, **fmom**, is defined to generate the moments of z directly. Note $N \equiv p$ below.

```
In[20]:= p = 10;

In[21]:= up[x_] := x^(p - 1) / (p - 1)! UnitStep[x]

In[22]:= fz[x_] := Sum[Binoimial[p, m] (-1)^m * up[x - m + p/2], {m, 0, p}]

In[23]:= fmom[n_] := If[n ≠ 0,
    Abs[D[(Series[(2 Sin[x/2] / x), {x, 0, n}]]^p, {x, n}]] / x → 0], 1]
```

The Gram–Charlier series will be evaluated for degrees 0, 5, 10, 20, and 50.

```
In[24]:= degreeList = {0, 5, 10, 20, 50}

Out[24]:= {0, 5, 10, 20, 50}

In[25]:= fGCz[x_] =
    Table[GramCharlierSeries[x, degreeList[[n]], fmom, Moment → MomFun],
        {n, 1, Length[degreeList]}];
```

Here **fGCz[x]** is a list of truncated Gram–Charlier series with degree 0, 5, 10, 20, and 50. (Note that generation of the list of series takes approximately 5 seconds on a 500 MHz Pentium III.) The exact density function evaluated with argument 2 is as follows.

```
In[26]:= fz[2] // N

Out[26]:= 0.0402557
```

The following table compares the relative error made when approximating this value using a truncated Gram–Charlier series of the indicated degree.

```
In[27]:= TableForm[Transpose[{degreeList, (fGCz[2] - fz[2]) / fz[2] // N}],
    TableHeadings → {None, {"Degree", "Relative Error"}}]
```

```
Out[27]//TableForm=
Degree      Relative Error
0           -0.0151588
5           -0.001568
10          -0.0000209021
20          -4.32387 × 10-6
50          -2.40039 × 10-6
```

Note that the series appears to converge with increasing degree as it should for this case. Normally, accuracy of the truncated series of the type considered here tends to improve as x approaches the mean value, in this case zero, of the true density function.

Figure 1 contains a plot of $\mathbf{fz}[\mathbf{x}]$ and the same truncated Gram–Charlier series used above, evaluated for x in the interval (2, 5). Note that higher-degree series are required for useful approximations in the density function *tails*. Asymptotic methods such as those described in [6] are more efficient for use in the tails of the density function than the series representations described here.

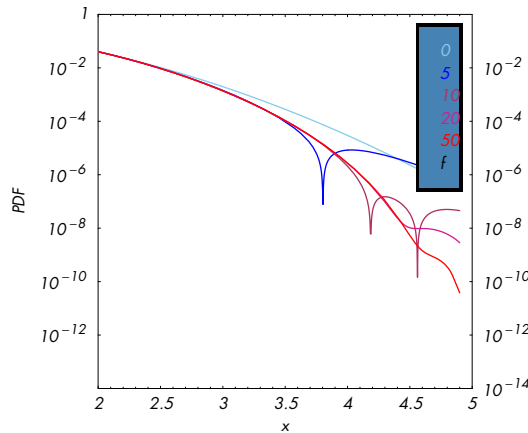


Figure 1. Comparison of exact density for z with the absolute value of five truncated Gram–Charlier series of degree (0, 5, 10, 20, 50).

Figure 1 compares the exact density of z with the absolute value of the truncated Gram–Charlier series. Note the *sharp points* on the curves, indicating where the truncated series crosses zero. If the true density were not available in this case, the first zero crossing could be used as an upper bound on the useful application domain of a given truncated series. The degree-0 curve confirms that random number generation based on z would serve as a mediocre approximation to a Gaussian random variate as noted earlier.

Since z is clearly distributed over a finite interval, $[-5, 5]$, the Beta-series, which is based on a weight function with support on a finite interval, might better approximate the density of z . This is demonstrated in Figure 2 which contains a plot of $\mathbf{fz}[\mathbf{x}]$ and the truncated Beta-series. To create this figure, series of degree 0, 5, 10, 20, and 50 were evaluated for x in the interval (2, 5). The relative approximation error of the truncated series is evaluated below for $x = 2$. (Note that generation of the list of series takes approximately 30 seconds on a dedicated 500 MHz Pentium III.)

```
In[28]:= fBetaz[x_] = Table[BetaSeries[x, degreeList[[n]], fmom,
    -p/2, p/2, Moment -> MomFun], {n, 1, Length[degreeList]}];
```

```
In[29]:= TableForm[Transpose[{degreeList, (fBeta[2] - fz[2]) / fz[2] // N}],
  TableHeadings -> {None, {"Degree", "Relative Error"}}]
```

```
Out[29]//TableForm=
```

Degree	Relative Error
0	0.00539916
5	0.00594452
10	0.000706726
20	0.0000599292
50	-7.74131×10^{-7}

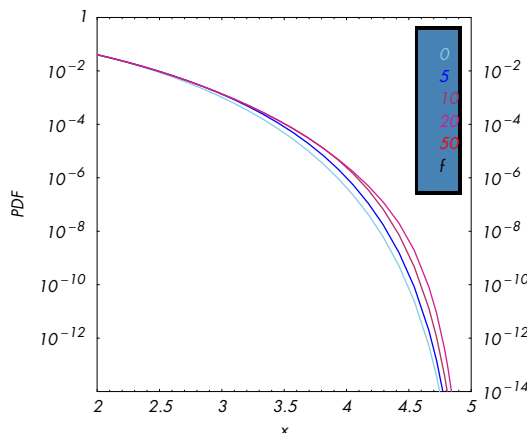


Figure 2. Comparison of the exact density for z with five truncated Beta-series of degree (0, 5, 10, 20, 50). Note that the curve for degree 50 is indistinguishable from the exact result, except just above the x -axis.

■ Conclusion

The package **ProbabilitySeries** provides six functions for generating orthogonal series expansions of probability density and distribution functions. After defining the details of the series expansion handled by the package **ProbabilitySeries**, a brief explanation of the package details was presented. Several examples of usage were included for functions defined in the package **ProbabilitySeries**. Usage of the package functions for generating a Q -series was not discussed directly; however, these functions are used in exactly the same manner as the corresponding functions for probability density function generation.

■ References

- [1] H. Cramér, *Mathematical Methods of Statistics*, Princeton: Princeton University Press, 1946.
- [2] M. Abramowitz and I. A. Stegun, *Handbook of Mathematical Functions*, New York: Dover Publications, 1965.

- [3] J. Shohat and J. Tamarkin, *The Problem of Moments*, Providence: American Mathematical Society, 1963.
- [4] A. Papoulis, *The Fourier Integral and Its Applications*, New York: McGraw-Hill, 1962.
- [5] D. E. Knuth, *The Art of Computer Programming: Seminumerical Algorithms*, Vol. 2, 2nd ed., Reading, MA: Addison-Wesley, 1981.
- [6] H. E. Daniels, "Saddlepoint Approximations in Statistics," *Annals of Mathematical Statistics*, **25**(1), 1954 pp. 631–650.

■ Additional Material

figure1.m
figure2.m
simp_leg.m
ProbabilitySeries.m

Available at www.mathematica-journal.com.

About the Author

J. Todd Reinking has worked for Mission Research Corporation since 1993. Dr. Reinking received a Ph.D. in Electrical Engineering from the University of Kansas in 1996; his current work involves radar, signal processing, and transionospheric propagation effects on radar.

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 noncommercial 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. Use filenames that suggest their contents.

Web

Articles, graphics, and other files may be uploaded on the Internet at www.mathematica-journal.com/upload.

Email

Articles and other material may be sent by email to editor@mathematica-journal.com. Do not email articles or attachments that exceed 1MB.

Hard Copy and/or Physical Media 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 sent on cdrom, zip, or other disk. Submissions will not be returned. Do not send proprietary or confidential information.

■ 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 be well structured, making good use of sections, text, and inline and displayed equations (if required), and should use the *Mathematica* style sheet “TMJv4Style.nb”. This style sheet along with an authoring palette can be downloaded from www.mathematica-journal.com/submissions.

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 the following.

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. references (see below);
3. a list of additional electronic material, including, if appropriate, information on the necessary environment (machine, operating system, and so on);
4. 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, with the footnotes as a section at the end of the article.

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 nonspecialists 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 using complete sentences 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.

Classifieds

Classified advertising is provided for *Mathematica* users. Rates are available at www.mathematica-journal.com/advertising.

Global Optimization 4.1 Released—Loehle Enterprises has released version 4.1 of *Global Optimization for Mathematica*. The new release brings to eight the number of functions for global optimization problem solving. The new function, *GlobalSearch*, combines the generalized derivative-based steepest descent method with a path-following algorithm for constraint satisfaction. This algorithm adds a new level of reliability to the package. In addition, several performance enhancements increase speed dramatically. For example, a 100-variable unconstrained quadratic minimization problem now takes only 4.5 seconds on a 600 Mhz machine. The functionality of the nonlinear regression function has been improved with the addition of ChiSquare, L1, and L2 norms as options for fit criteria.

More info: www.wolfram.com/products/applications/globalopt

Mathematica Training and Consulting—Whether you are an individual or a company seeking *Mathematica* training, Wolfram Education Group provides a learning solution with **Certified Training**. Register for classes and review the Education Benefits Program for corporate account training incentives and discounts. If you are looking for **Independent Training** for additional courses, workshops, or tutorials, the Wolfram Education Group can provide a variety of learning opportunities from independent trainers and companies with years of *Mathematica* experience. Wolfram Research also offers direct **Consulting Services**, such as reviewing and debugging your *Mathematica* code, isolating problems in large *Mathematica* programs, implementing custom features, and writing complete programs based on users' specifications.

More info: www.wolfram.com/services/training

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

