The *Mathematica*® Journal

Volume 9, Issue 3

Contents

Departments

Staff and Contributors 502
Letters 503
Calendar 504
News Bulletins 505
New Products 507
New Resources 512
New Publications 513
Article Submissions to *The Mathematica Journal* 659
Classifieds 663

Columns

Tricks of the Trade *edited by Paul Abbott* 520
In and Out *edited by Paul Abbott* 528
Trott's Corner *by Michael Trott* 541

Feature Articles

WaveX: Extracting Wavelets from Seismic Data *by John M. Novak* . . 555
The Modular Group *by Paul R. McCreary, Teri Jo Murphy, and Christan Carter* 564
On the Beauty of Uniform Distribution Modulo One *by Karl Entacher* 583
Elliptic Rational Functions *by Miroslav D. Lutovac and Dejan V. Tosic* 598
Polyominoes and Related Families *by Jaime Rangel-Mondragón* 609
Algebraic Construction of Smooth Interpolants on Polygonal Domains *by Elisabeth Anna Malsch and Gautam Dasgupta* 641

The electronic version of this issue is available at www.mathematica-journal.com/issue/v9i3.

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

On the Cover

Randomly nested regular polygons in a square grid.

Staff and Contributors

Managing Editor: George Beck
Project Manager: Peggy Pope
Design Team: Jeremy Davis, Megan Gillette
Compositor: Glenn Scholebo
Archives and Web Production: Jean Buck
Web Development: Andy Hunt, Lisa Denlinger
Editorial Staff: George Beck, Marcia Krause, Larry Adelston
Customer Service: Mandy Pope

Columnists: Paul Abbott, Michael Trott
Contributors: Christan Carter, Gautam Dasgupta, Karl Entacher, Miroslav D. Lutovac, Elisabeth Anna Malsch, Paul R. McCreary, Teri Jo Murphy, John M. Novak, Jaime Rangel-Mondragón, Dejan V. Tosic

Editorial Advisory Board: Steve Christensen (MathTensor, Inc.; Christensen and Associates, Inc.); Donald Glaser (University of California, Berkeley); Donald Knuth (Stanford University); Wolfram Koepf (University of Kassel); Silvio Levy (Mathematical Sciences Research Institute); Daniel Lichtblau (Wolfram Research, Inc.); 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 Massachusetts, Amherst); Lynn Steen (St. Olaf College); Francis Sullivan (Institute for Defense Analyses); Bill Thurston (University of California, Davis); Michael Trott (Wolfram Research); Hal Varian (University of California, Berkeley); Eric Weisstein (Wolfram Research); Rob Wolff (Apple Computer, Inc.); and Stephen Wolfram (Wolfram Research).

Customer Services
www.mathematica-journal.com
+1-877-294-8184 (U.S. and Canada)
+1-217-398-6000
fax: +1-217-398-1108
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* 5 using Janson, Futura, Math CMTT and the standard *Mathematica* fonts.

Publishing
The Mathematica Journal (ISSN 1047-5974) is published by Wolfram Media, Inc. Four-issue 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.

© 2005 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. All other trademarks used herein are the property of their respective owners. *Mathematica* is not associated with Mathematica Policy Research, Inc. or MathTech, Inc. 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*.

Letters

Erratum

In the paper "Path Integral Methods for Parabolic Partial Differential Equations with Examples from Computational Finance" by Andrew Lyasoff (*The Mathematica Journal*, **9**(2) 2004 pp. 399–422), the following correction should be made.

Equation (5.2) on page 412 should be replaced by

$$\begin{aligned} Z_t = {} & e^{\sigma'[x]\,ß_t + \left(a'[x] - \frac{1}{2}\sigma'[x]^2\right)t} \\ & \times \left((a[x] - a'[x]\,x) \int_0^t e^{-\sigma'[x]\,ß_s - \left(a'[x] - \frac{1}{2}\sigma'[x]^2\right)s}\, ds \right. \\ & \left. + \; (\sigma[x] - \sigma'[x]\,x) \int_0^t e^{-\sigma'[x]\,ß_s - \left(a'[x] - \frac{1}{2}\sigma'[x]^2\right)s}\, dß_s + x \right) \qquad (5.2) \\ & - \; \sigma'[x] \times (\sigma[x] - \sigma'[x]\,x) \times e^{\sigma'[x]\,ß_t + \left(a'[x] - \frac{1}{2}\sigma'[x]^2\right)t} \\ & \times \int_0^t e^{-\sigma'[x]\,ß_s - \left(a'[x] - \frac{1}{2}\sigma'[x]^2\right)s}\, ds, \end{aligned}$$

and the equation preceding (5.3) should be replaced by

$$Z_t = e^{a'[x]\,t} \times \left((a[x] - a'[x] \times x) \times t + \sigma[x] \times \int_0^t e^{-a'[x]\,s}\, dß_s + x \right) + o(t)\,.$$

The approximation of the process Z_t, $t \geq 0$, which is used throughout Sections 5 and 6 is derived from the correct expressions given above. Therefore, the apparent errors in equation (5.2) and the equation preceding (5.3) do not affect in any way the results, the examples, or the *Mathematica* code presented in the paper.

These changes have been made to the electronic versions of this article at www.mathematica-journal.com/issue/v9i2/ComputationalFinance.html.

Andrew Lyasoff
andrew@lyasoff.com

Calendar

For the complete Wolfram Research calendar, see www.wolfram.com/news/events.

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

2005 Highlighted Events

February

Wolfram Research Tour of India
Wolfram Research
February 7–18, India
Info: info@wolfram.com

April

Wolfram Research Reseller Conference 2005
Wolfram Research
April 18–19, Blenheim Palace, Woodstock, UK
Info: info@wolfram.com

June

NKS Summer School 2005
Wolfram Research
June 20–July 8, Brown University, Providence, Rhode Island
Info: nks-summerschool@wolfram.com

August

International *Mathematica* Symposium 2005
IMS 2005 Organizing Committee
August 5–8, The University of Western Australia, Perth, Western Australia
Info: www.wolfram.com/news/ims2005.html

October

Wolfram Technology Conference 2005
Wolfram Research
October 6–8, Champaign, Illinois
Info: info@wolfram.com

News Bulletins

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

Mathematica Gallery Collection

Full-color posters of several graphics from the *Mathematica* gallery are now available to purchase online. To order your own poster from the *Mathematica* Gallery Collection, go to gallery.wolfram.com.

Specialized Mini-Courses Provide Live, Interactive *Mathematica* Training Online

Wolfram Education Group has expanded its *Mathematica* training course offerings with the launch of several new online mini-courses taught by core developers at Wolfram Research. These tightly focused, highly applied sessions are short and inexpensive, and include information on emerging technologies as they are integrated into *Mathematica*. For more information, go to www.wolfram.com/news/wegminicourses.html.

New Partnership Unites grid*Mathematica* with High-Performance Orion Cluster Workstations

The Orion Cluster Workstation with grid*Mathematica* installed provides a quick way for the individual user to set up and run large calculations by offering a high-level programming language, a vast collection of reliable algorithms, and easy-to-use parallel programming constructs. For more information, go to www.orionmulti.com/news/pr_083020042.

Air Force's Air Education and Training Command Implements *Mathematica* for Business Analytics

Air Education and Training Command directors at Randolph Air Force Base in San Antonio, Texas are using *Mathematica* as part of their business analytics toolbox to help predict resource requirements and streamline future training. For more information, go to www.fcw.com/fcw/articles/2004/0823/tec-analyze-08-23-04.asp.

Growing a Grid at Skidmore College

Using grid*Mathematica* for his most intensive applications, Flip Phillips has reduced his computation times from as much as two months to as little as four hours. Phillips wrote a series of programs that extend his grid by "scavenging" idle network machines for available resources to accelerate his system. For more information, go to www.wolfram.com/news/skidmoregrid.htm.

Mathematica in Education and Research

The *Mathematica in Education and Research* (*MiER*) journal is back in a new quarterly format, which will be available in both electronic and print versions. Volume 10, Number 1 is now online. The journal publishes articles, notes, and special features on the use of *Mathematica* technology in all aspects of education and research. Article submission guidelines are available on the website. For more information, visit www.ijournals.net.

New Products

Mathematica 5.1

Newly Released Mathematica 5.1 Delivers Unmatched Performance for Handling Data

Wolfram Research announces *Mathematica* 5.1, the data functions release—the latest version of the world-renowned science and technology software system *Mathematica*.

Building on *Mathematica* 5.0's dramatic speed, scope, and scalability improvements, Version 5.1 adds a host of new capabilities, especially for working with large-scale, diverse types of data.

"We've introduced innovative algorithms to deliver unmatched performance for all steps in the data handling process—importing, analyzing, manipulating, or plotting," says Tom Wickham-Jones, Director of Kernel Technology. "And when we say data, we encompass textual and network as well as numerical data."

Complementing these new capabilities are enhancements to *Mathematica*'s unique automatic algorithm selection—the inherent intelligence that automatically applies the best algorithm to each task. *Mathematica* 5.1 adds more algorithms and better intelligence in a number of key areas.

"When you use a function in *Mathematica*, you're telling it what you want it to do, not how to do it," says Conrad Wolfram, Director of Strategic Development. "Each function name represents a task rather than a simple algorithm—leaving users to concentrate on specifying the right task while *Mathematica* focuses on applying the best algorithms. In many areas Version 5.1 will now outperform manual selection of algorithms by the most advanced practitioners both in performance and consistency of the results."

There are more than 50 new functions, toolkits, and performance improvements in *Mathematica* 5.1. Highlights include:

- Industrial-strength string manipulation
- Built-in universal database connectivity
- Highly optimized binary data I/O
- Additional import-export formats, including XLS and AVI
- Integrated web services support
- 2D and 3D automated network visualization
- High-performance array visualization

- Numerical linear algebra performance enhancements
- Fully integrated piecewise functions
- Integration over implicitly defined regions
- Event handling in numerical differential equations
- New algorithms for symbolic differential equations
- Cluster analysis capabilities added
- Interactive exploration tool for differential equations
- *MathematicaMark* benchmarking tool
- *GUIKit* interface and application builder—now built in
- and many more innovations

"Our release numbering tends to be rather understated," says Wolfram. "There are many development years of new technology in Version 5.1—even though it's just a 0.1 release."

More info
www.wolfram.com/products/mathematica/newin51

Wolfram *Publicon*

Technical Publishing Made Easy with New Wolfram Publicon Software

Wolfram *Publicon*, a new integrated software solution for composing sophisticated technical documents, is now available from Wolfram Research. Created for the growing number of academic researchers, students, and industry professionals who need to create or publish documents with technical content, *Publicon* produces platform-independent files that can be exported to HTML, XML, LaTeX, or custom *Publicon* formats.

Up to this point, technical writers have had only cumbersome means of integrating mathematical typesetting into their documents. *Publicon* seamlessly incorporates intuitive math typesetting technology with additional templates for chemical equations, special characters, and symbols to produce publication-quality documents quickly and easily. Mathematical notation from a *Publicon* document can be copied and pasted for and from computation in *Mathematica*, the award-winning technical computing system from Wolfram Research, allowing readers meaningful and dynamic interaction with the text.

Formatting documents to the demands of publishers and technical societies like the American Mathematical Society, the American Physical Society, and BioMed Central is greatly simplified with this new standard for technical publishing. Built-in palettes, templates, and style sheets simplify the creation of documents

that conform to industry-standard formats, but also allow for complete customization to match any style.

Among its many features that streamline the publication process, *Publicon* includes:

- A guided, template-driven document creation system
- A scrolling, WYSIWYG interface ideal for online presentation
- Support for the composition of structured documents matching REVTeX, AMS-TeX, standard LaTeX, Open Access publisher BioMed Central's XML (supported by PubMed Central), and custom *Publicon* formats
- HTML, XML, and LaTeX export
- Automatic conversion to and from MathML
- Math and chemistry typesetting for word-processor-like equation composition
- Full searchability within typeset equations for any character—including all operators, bracket varieties, and radicals
- Automated reference management for instant bibliographies
- Notes and cross-reference insertion tools
- Style sheets that control all aspects of a document's functionality and appearance
- Spell checking with a built-in technical dictionary
- Platform-independent document creation
- A searchable help browser for quick-and-easy access to all of *Publicon*'s features

"You don't need to learn a new language or have any experience with publishing to create elegant technical documents with *Publicon*," says André Kuzniarek, manager of document technology at Wolfram Research. "From implementing mathematical and scientific typesetting to structuring the document for online publication, *Publicon* does it all with the click of a few buttons. With *Publicon*, technical authors have a complete system in which to write, while publishers have the guarantee of receiving documents custom-formatted to their exacting standards."

"This is a major advance," says Matthew Cockerill, Technical Director of Open Access publisher BioMed Central. "With *Publicon* it is finally possible for scientific authors to create manuscripts in a structured way and output them in an XML format suitable for web publication. It will lead to faster publication times and significantly reduce the costs involved in scientific publishing."

Publicon is available for Windows and Mac OS X and can be purchased for download at store.wolfram.com/view/app/publicon.

More info
www.wolfram.com/publicon

Mathematica Applications

Global Optimization 5.0

Global Optimization 5.0 is a collection of functions for constrained and unconstrained global nonlinear optimization. It uses *Mathematica* as an interface for defining nonlinear systems to be solved and for computing function numeric values.

More info
www.wolfram.com/products/applications/globalopt

web*Mathematica* Author

web*Mathematica* Author is a *Mathematica* add-on that aids in the creation and development of JavaServer Pages (JSP) for use with web*Mathematica*. It provides users with functions and palette options that simplify the construction and layout of commonly used HTML and webMathematica tags, thereby easing the web-*Mathematica* development process.

More info
www.wolfram.com/products/webmathematica/resources/author

machine learning framework 1.2

machine learning framework is a powerful tool for all types of data mining and machine learning applications. This complete software solution is built for all experts and professionals who want to extract sophisticated computational models from data.

More info
www.wolfram.com/products/applications/mlf

MathModelica 2.1

MathModelica 2.1 is a cost-effective solution to develop and maintain complex engineering models and simulate all kinds of physical systems to be documented and analyzed in *Mathematica*.

More info
www.wolfram.com/products/applications/mathmodelica

UnRisk PRICING ENGINE 2

UnRisk PRICING ENGINE 2 accelerates high-end derivative analytics for *Mathematica*. *UnRisk* 2 features the complete reorganization and expansion of the numerical schemes to solve and calibrate two-factor models with unparalleled accuracy and speed.

More info
www.wolfram.com/products/applications/unrisk

MathLink for AppleScript

uni software plus GmbH has developed MathLink for AppleScript, a free tool that allows users to call *Mathematica* tasks from AppleScript and use AppleScript-enabled applications like FileMaker, Adobe Photoshop, and Microsoft Excel with *Mathematica*.

More info

www.unisoftwareplus.com/products/mathlinkosax

New Resources

Documentation Center

The Wolfram Research Documentation Center has been redesigned for easier searching and retrieval of information, and is now updated to include the complete documentation for *Mathematica* 5.1. For more information, visit documents.wolfram.com.

Wolfram Technology Conference Proceedings

The 2004 Wolfram Technology Conference was an outstanding forum for the wide range of topics concerning *Mathematica* users from around the world. The intensive three-day event offered more than 80 presentations, classes, and workshops on Wolfram technologies and their applications. For a complete list of this year's conference presentations and to download any of the available presentation notebooks, visit
www.wolfram.com/news/events/techconf2004/presentations.html.

Mathematica Information Center Editor's Pick

Each week, *Mathematica* Information Center editor Ed Pegg Jr picks an exciting new submission that explores mathematics in an interesting and insightful way to feature on the website. For the latest Editor's Pick, visit
library.wolfram.com/infocenter/EditorsPick.

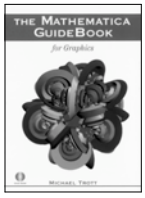
Mathematica Graphics Gallery

Several vivid, full-color images from the *Mathematica* Graphics Gallery can now be purchased as posters from the *Mathematica* Gallery Collection. These stimulating graphics by Michael Trott, author of *The Mathematica GuideBooks*, bring together art and science to produce powerful, eye-catching visualizations of interesting mathematical forms. For more information, visit gallery.wolfram.com.

Automata

The latest version of the Automata package for manipulating finite state machines and their syntactic semigroups is now online. The hybrid *Mathematica*/C++ package also includes a number of operations on one-dimensional cellular automata. For more information and to download this package, visit library.wolfram.com/infocenter/MathSource/4603.

New Publications



The *Mathematica* GuideBook for Graphics

Michael Trott, 2004, Springer-Verlag, 1340 pp., hardcover, includes DVD-ROM

This book provides a comprehensive step-by-step development of how to use *Mathematica* to visualize functions and data, manipulate graphics, and optimize their appearance. Two-dimensional graphics, contour plots, plots of surfaces, free-form three-dimensional surfaces, and animations are the core topics. Hundreds of detailed examples and programs show a large variety of visualization techniques, algorithms, methods, and tricks. These tools allow the reader to create virtually any possible graphic, from simple curves to scientific visualizations and artistic images and logos. *Mathematica* graphics functions are discussed in detail, explained in numerous examples, and put to work in programs that are all contained on the accompanying DVD.

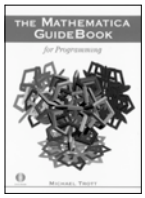
Unique features:

- Step-by-step introductions to all of *Mathematica*'s graphics capabilities
- Comprehensive presentation of two- and three-dimensional graphics primitives and directives, as well as plotting capabilities for functions and data
- Hundreds of unique and innovative scientific visualizations and artistic images
- Website for book with additional materials and updates: www.MathematicaGuideBooks.org
- Accompanying DVD containing all material as an electronic book with complete and executable *Mathematica* code and programs, rendered color graphics, and animations

Chapters

Two-Dimensional Graphics | Three-Dimensional Graphics | Contour and Density Plots | Index

Available:
store.wolfram.com/view/book/ISBN0387950109.str



The *Mathematica* GuideBook for Programming

Michael Trott, 2004, Springer-Verlag, 1028 pp., hardcover, includes DVD-ROM

This book provides a comprehensive, step-by-step development of *Mathematica* programming capabilities and contains an enormous collection of examples and worked exercises. It guides the reader to become fluent in the structure of *Mathematica* expressions, expression evaluation, pure and named functions, and in procedural, rule-based, and functional programming constructs. Each *Mathematica* function is discussed in detail, explained in numerous examples, and put to work in real programs that are all contained on the accompanying DVD.

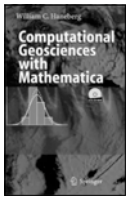
Unique features:

- Step-by-step presentation of *Mathematica* functions assuming no prior *Mathematica* programming experience
- Clear organization, complete topic coverage, and an accessible writing style for both novices and experts
- Detailed discussion of procedural, rule-based, and functional programming
- Hundreds of worked examples, illustrations, programs, and fully worked self-study exercises for understanding concepts and learning how to solve real-life problems
- Website for book with additional materials and updates: www.MathematicaGuideBooks.org
- Accompanying DVD containing all material as an electronic book with complete and executable Mathematica code and programs, rendered color graphics, and animations

Chapters

Introduction to *Mathematica* | Structure of *Mathematica* Expressions | Definitions and Properties of Functions | Meta-*Mathematica* | Restricted Patterns and Replacement Rules | Operations on Lists, and Linear Algebra | Index

Available:
store.wolfram.com/view/book/ISBN0387942823.str



Computational Geosciences with *Mathematica*

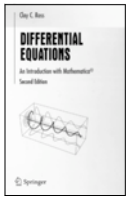
William C. Haneberg, 2004, Springer-Verlag, 381 pp., hardcover, includes CD-ROM

This is the only book written by a geologist specifically to show geologists and geoscientists how to use *Mathematica* to formulate and solve problems. It spans a broad range of geologic and mathematical topics, which are drawn from the author's extensive experience in research, consulting, and teaching. This combined reference and text leads readers step by step through geologic applications such as custom graphics programming, data input and output, linear and differential equations, linear and nonlinear regression, Monte Carlo simulation, time series and image analysis, and the visualization and analysis of geologic surfaces. It is packed with actual *Mathematica* output and includes boxed Computer Notes with tips and exploration suggestions. The accompanying CD-ROM contains notebooks of all text and graphics, plus an appendix on color graphics and specialized functions.

Chapters

Introduction to *Mathematica* | Special Plots for Geoscience Data | Manipulating and Solving Equations | Random Variables and Univariate Probability Distributions | Probabilistic Simulation | Interpolation and Regression | Visualizing and Analyzing Surfaces | Digital Signal and Image Processing | Appendix A: *Mathematica* Functions in the Computational Geoscience Package | Appendix B: Working with Color

Available:
store.wolfram.com/view/book/ISBN3540402454.str



Differential Equations: An Introduction with *Mathematica*, Second Edition

Clay C. Ross, 2004, Springer Science+Business Media Inc., 431 pp., hardcover

This undergraduate introductory differential equations textbook presents a convenient way for professors to integrate symbolic computing into the study of differential equations and linear algebra. *Mathematica* provides the necessary computational power and is employed from the very beginning of the text. Each new idea is interactively developed using it.

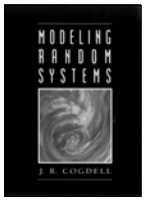
After first learning about the fundamentals of differential equations and linear algebra, the student is immediately given an opportunity to examine each new concept using *Mathematica*. All ideas are explored using *Mathematica*, and though the computer eases the computational burden, the student is encouraged to think about what the computations reveal, how they are consistent with the mathematics, what any conclusions mean, and how they may be applied.

This new edition updates the text to *Mathematica* 5 and offers a more extensive treatment of linear algebra. It has been thoroughly revised and corrected throughout.

Chapters

About Differential Equations | Linear Algebra | First-Order Differential Equations | Applications of First-Order Equations | Higher-Order Linear Differential Equations | Applications of Second-Order Equations | The Laplace Transform | Higher-Order Differential Equations with Variable Coefficients | Differential Systems: Theory | Differential Systems: Applications | References | Index

Available:
store.wolfram.com/view/book/ISBN0387212841.str



Modeling Random Systems

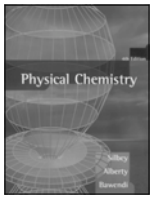
J. R. Cogdell, 2004, Pearson Education, Inc., 704 pp., hardcover

An introductory textbook for undergraduate engineering students learning the basic concepts of randomness and modeling random systems. Features include easy-to-follow lessons, end-of-chapter problems, and a quick tutorial on using *Mathematica* for real-life modeling situations. The publisher will distribute an electronic version.

Chapters

The Probability Paradigm | The Binomial Model and Random Variables | Continuous Random Variables and the Gaussian Model | Statistics | The Poisson Model | Modeling Random Signals | Tutorial on *Mathematica* | Tutorial on Convolution | Summary of Common Probability Models | Index

Available:
store.wolfram.com/view/book/ISBN0131414372.str



Physical Chemistry, Fourth Edition

Robert J. Silbey, Robert A. Alberty, and Moungi G. Bawendi, 2005, John Wiley & Sons, 944 pp., hardcover

A leading book for 80 years, *Physical Chemistry* features clear explanations of the concepts and methods of physical chemistry for students who have had a year of calculus and a year of physics. The basic theory of chemistry is presented from the viewpoint of academic physical chemists, but the many practical applications of physical chemistry are integrated throughout the text. The problems in the text also reflect a skillful blend of theory and practical applications.

This text is ideally suited for a standard undergraduate physical chemistry course taken by chemistry, chemical engineering, and biochemistry majors in their junior or senior year.

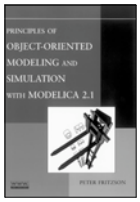
Highlights of the Fourth Edition:

- A total of 170 *Mathematica* problems
- Increased emphasis on the thermodynamics and kinetics of biochemical reactions, including the denaturation of proteins and nucleic acids
- Expanded coverage of the uses of statistical mechanics, nuclear magnetic relaxation, nanoscience, and oscillating chemical reactions
- Many new tables and figures throughout the text

Chapters

Zeroth Law of Thermodynamics and Equations of State | First Law of Thermodynamics | Second and Third Laws of Thermodynamics | Fundamental Equations of Thermodynamics | Chemical Equilibrium | Phase Equilibrium | Electrochemical Equilibrium | Thermodynamics of Biochemical Reactions | Quantum Theory | Atomic Structure | Molecular Electronic Structure | Symmetry | Rotational and Vibrational Spectroscopy | Electronic Spectroscopy of Molecules | Magnetic Resonance Spectroscopy | Statistical Mechanics | Kinetic Theory of Gases | Experimental Kinetics and Gas Reactions | Chemical Dynamics and Photochemistry | Kinetics in the Liquid Phase | Macromolecules | Electric and Magnetic Properties of Molecules | Solid-State Chemistry | Surface Dynamics | Appendix A: Physical Quantities and Units | Appendix B: Values of Physical Constants | Appendix C: Tables of Physical Chemical Data | Appendix D: Mathematical Relations | Appendix E: Greek Alphabet | Appendix F: Useful Information on the Web | Appendix G: Symbols for Physical Quantities and their SI Units | Appendix H: Answers to the First Set of Problems | Index

Available:
store.wolfram.com/view/book/ISBN044451760X.str



Principles of Object-Oriented Modeling and Simulation with Modelica 2.1

Peter Fritzson, 2004, IEEE Press, 897 pp., paperback

Designed for students, researchers, and engineers familiar with basic programming concepts, this book gives a basic introduction to the concepts of modeling and simulation, the basics of object-oriented component-based modeling for the novice, and a comprehensive overview of modeling and simulation in a number of application areas. It provides a tutorial and reference for the Modelica 2.1 (including some anticipated extensions) complete with a comprehensive overview of application model libraries from many domains. Executable examples are included from many areas—physics, mechanics, electrical systems, thermodynamics, flow systems, computer science, concurrent and real-time processes, biology, ecology, chemistry, economy, etc.

Samples of executable models, teaching material, interactive tutorials, and recent book updates are available at the book's website, www.mathcore.com/drmodelica.

Chapters

Introduction to Modeling and Simulation | A Quick Tour of Modelica | Classes, Types, and Declarations | Inheritance, Modifications, and Generics | Components, Connectors, and Connections | Literals, Operators, and Expressions | Arrays | Equations | Algorithms and Functions | Packages | Annotations, Units, and Quantities | System Modeling Methodology and Continuous Model Representation | Discrete Event, Hybrid, and Concurrency Modeling | Basic Laws of Nature | Application Examples | Modelica Library Overview | A Mathematical Representation for Modelica Models | Techniques and Research | Environments | Appendix A: Modelica Formal Syntax | Appendix B: *Mathematica*-Style Modelica Syntax | Appendix C: Solutions to Exercises | Appendix D: Modelica Standard Library | Appendix E: Modelica Scripting Commands | Appendix F: Related Object-Oriented Modeling Languages | Appendix G: A Modelica XML Representation | References | Index

Available:
store.wolfram.com/view/book/ISBN0471471631.str



The SIAM 100-Digit Challenge: A Study in High-Accuracy Numerical Computing

Folkmar Bornemann, Dirk Laurie, Stan Wagon, Jörg Waldvogel,
2004, Society for Industrial and Applied Mathematics,
306 pp., paperback

The SIAM 100-Digit Challenge was a contest posed by Nick Trefethen of Oxford University in the January/February 2002 issue of *SIAM News*. This book shows in detail how each of those problems can be solved, as described by four authors who belonged to winning teams that successfully solved all 10 problems. The book presents multiple approaches to the solution for each problem, including schemes that can be scaled to provide thousand-digit accuracy if required and can solve even larger related problems. In the process, the authors visit just about every major technique of modern numerical analysis: matrix computation, numerical quadrature, limit extrapolation, error control, interval arithmetic, contour integration, iterative linear methods, global optimization, high-precision arithmetic, evolutionary algorithms, eigenvalue methods, and many more. Code is provided for many *Mathematica* solutions, including some from the perfect-score entry submitted by the Wolfram Research team.

Chapters

Foreword | Preface | The Story | A Twisted Tail | Reliability amid Chaos | How Far Away Is Infinity? | Think Globally, Act Locally | A Complex Optimization | Biasing for a Fair Return | Too Large to Be Easy, Too Small to Be Hard | In the Moment of Heat | Gradus ad Parnassum | Hitting the Ends | Convergence Acceleration | Extreme Digit-Hunting | Code | More Problems | References | Index

Available:
store.wolfram.com/view/book/ISBN089871561X.str

Tricks of the Trade

Edited by Paul Abbott

This is a column of programming tricks and techniques, most of which, we hope, will be contributed by our readers, either directly as submissions to *The Mathematica Journal* or as an edited answer to a question posted in the *Mathematica* news group, comp.soft-sys.math.mathematica.

■ Nondecreasing Function

Consider determining real coefficients a and b such that $f_{a,b}(x) = x^4 + 2\,a\,x^3 + b\,x^2 + (1 - 3\,a)\,x + 4$ is *nondecreasing* for $-1 \le x \le 1$.

In Version 5.1, applying **Resolve** to the requirement $f'_{a,b}(x) \ge 0$ yields the answer.

```
In[1]:= f_a_,b_(x_) = x^4 + 2 a x^3 + b x^2 + (1 - 3 a) x + 4;

In[2]:= coefs = Resolve[∀_{x,-1≤x≤1} f'_{a,b}(x) ≥ 0]
```

$$\text{Out[2]=}\quad a = 0 \bigwedge b = -\frac{3}{2}$$

The derivative is clearly nonnegative for $-1 \le x \le 1$.

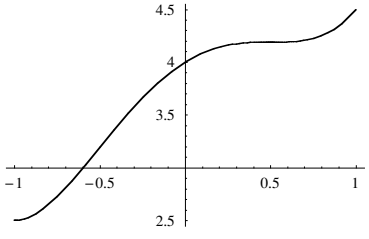
```
In[3]:= f'_{a,b}(x) /. ToRules[coefs] // Factor
```

$$\text{Out[3]=}\quad (x + 1)\,(2\,x - 1)^2$$

A plot gives visual confirmation that the function is nondecreasing.

```
In[4]:= Plot[Evaluate[f_{a,b}(x) /. ToRules[coefs]], {x, -1, 1}]
```

From In[4]:=



■ Proofs of Inequalities

Let $x_1, x_2, \ldots, x_n$ be real numbers. Prove that

$$s_n = \frac{x_1}{x_1^2+1} + \frac{x_2}{x_1^2+x_2^2+1} + \cdots + \frac{x_n}{x_1^2+x_2^2+\cdots+x_n^2+1} < \sqrt{n}$$

for all integer $n \geq 1$.

Define s_n.

```
In[1]:= s_n_ := Sum[x_i / (Sum[x_j^2, {j, 1, i}] + 1), {i, 1, n}]
```

First, we use **NMaximize** to examine the inequality for $1 \leq n \leq 20$.

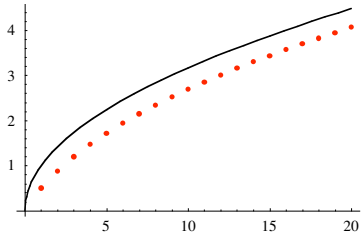
```
In[2]:= pts = Table[First[NMaximize[s_n, Table[{x_i, -5, 5}, {i, n}]]], {n, 20}]
```

```
Out[2]= {0.5, 0.880086, 1.19591, 1.47044, 1.71578, 1.93915,
  2.14528, 2.33746, 2.51805, 2.68883, 2.85119, 3.0062, 3.15475,
  3.29756, 3.43522, 3.56823, 3.69702, 3.82195, 3.94334, 4.06148}
```

```
In[3]:= << Graphics`;
```

```
In[4]:= DisplayTogether[Plot[Sqrt[n], {n, 0, 20}],
  ListPlot[pts, PlotStyle → {Hue[1], AbsolutePointSize[2]}]]
```

From In[4]:=



From the form and statement of the inequality, a proof by induction [mathworld.wolfram.com/PrincipleofMathematicalInduction.html] looks like the way to proceed. Leon Aigret (aigret@myrealbox.com) presented the following inductive proof on the alt.math.recreational newsgroup, here restated using *Mathematica*.

For $n = 1$, the proof is trivial.

```
In[5]:= Reduce[s_1 < 1]
```

```
Out[5]= x_1 ∈ ℝ
```

We need to show that $s_{n-1} < \sqrt{n-1} \Rightarrow s_n < \sqrt{n}$ for all $n \geq 2$. Consider the transformation $x_k \to \sqrt{1+x_1^2}\ y_{k-1}$ for $k \geq 2$. Here is the result with $n = 4$.

```
In[6]:= Factor /@ (s4 /. xk_/;k≥2 → Sqrt[1 + x1^2] yk-1)
```

$$\text{Out[6]=}\quad \frac{x_1}{x_1^2+1}+\frac{y_1}{\sqrt{x_1^2+1}\,(y_1^2+1)}+\frac{y_2}{\sqrt{x_1^2+1}\,(y_1^2+y_2^2+1)}+\frac{y_3}{\sqrt{x_1^2+1}\,(y_1^2+y_2^2+y_3^2+1)}$$

In this result we recognize s_3 involving y instead of x.

```
In[7]:= % == s1 + 1/Sqrt[x1^2 + 1] (s3 /. x → y) // Simplify
```

```
Out[7]= True
```

In general we have

$$s_n(x)\longrightarrow \frac{x_1}{x_1^2+1}+\frac{y_1}{\sqrt{x_1^2+1}\,(y_1^2+1)}+\frac{y_2}{\sqrt{x_1^2+1}\,(y_1^2+y_2^2+1)}+\cdots+$$
$$\frac{y_{n-1}}{\sqrt{x_1^2+1}\,(y_1^2+y_2^2+\cdots+y_{n-1}^2)+1}=\frac{x_1}{x_1^2+1}+\frac{1}{\sqrt{x_1^2+1}}\,s_{n-1}(y).$$

So, assuming that $s_{n-1}<\sqrt{n-1}$, we need to prove that

$$\frac{x_1}{x_1^2+1}+\frac{\sqrt{n-1}}{\sqrt{x_1^2+1}}<\sqrt{n}\,.$$

It is straightforward to show that this inequality holds for $n\geq 1$ and $x_1\in\mathbb{R}$ so we have completed the proof.

```
In[8]:= Reduce[x1/(x1^2 + 1) + Sqrt[n - 1]/Sqrt[x1^2 + 1] < Sqrt[n], ℝ]
```

```
Out[8]= n ≥ 1
```

Andrzej Kozlowski (www.mimuw.edu.pl/~akoz) proved the following stronger statement:

$$t_n=\sum_{i=1}^{n}\left(\frac{x_i}{\sum_{j=1}^{i}x_j^2+1}\right)^2<1$$

for every positive integer n.

It is easy to see that this statement implies the original inequality using Cauchy's inequality [mathworld.wolfram.com/CauchysInequality.html],

$$\left(\sum_{i=1}^{n}a_i\,b_i\right)^2\leq\left(\sum_{i=1}^{n}a_i^2\right)\left(\sum_{i=1}^{n}b_i^2\right).$$

Setting $b_i = 1$,

$$\left(\sum_{i=1}^{n} a_i^2\right) \geq \frac{1}{n}\left(\sum_{i=1}^{n} a_i\right)^2,$$

and hence

$$1 > \sum_{i=1}^{n}\left(\frac{x_i}{\sum_{j=1}^{i} x_j^2 + 1}\right)^2 \geq \frac{1}{n}\left(\sum_{i=1}^{n}\frac{x_i}{\sum_{j=1}^{i} x_j^2 + 1}\right)^2 \Rightarrow$$

$$\sum_{i=1}^{n}\frac{x_i}{\sum_{j=1}^{i} x_j^2 + 1} < \sqrt{n}\,.$$

Moreover, the proof is easier since the inductive step is now trivial.

$$t_n(x) \longrightarrow \frac{x_1^2}{(x_1^2+1)^2} +$$

$$\frac{1}{x_1^2+1}\left(\frac{y_1^2}{(y_1^2+1)^2} + \frac{y_2^2}{(y_1^2+y_2^2+1)^2} + \cdots + \frac{y_{n-1}^2}{(y_1^2+y_2^2+\cdots+y_{n-1}^2+1)^2}\right) =$$

$$\frac{x_1^2}{(x_1^2+1)^2} + \frac{t_{n-1}(y)}{x_1^2+1}.$$

So, assuming that $t_{n-1} < 1$, we need to prove that

$$t_n < \frac{x_1^2}{(x_1^2+1)^2} + \frac{1}{x_1^2+1} < 1.$$

This is clearly true for $x_1 \neq 0$.

```
In[9]:= Reduce[x1^2/(x1^2 + 1)^2 + 1/(x1^2 + 1) < 1, ℝ]
Out[9]= x1 < 0 ∨ x1 > 0
```

If $x_1 = 0$, then $t_n(x) = t_{n-1}(y) < 1$ so the proof is complete.

In addition, the inequality leads to some intriguing observations since it implies that the sums, considered as functions on the real line, are bounded and attain their maxima. So it is natural to consider the functions $f_n(x)$ obtained by setting all the $x_i = x_j$.

```
In[10]:= f_n_(x_) = Sum[(x/(i x^2 + 1))^2, {i, 1, n}]
Out[10]= (PolyGamma[1, 1 + 1/x^2] - PolyGamma[1, n + 1 + 1/x^2])/x^2
```

The limit as $n \to \infty$ can be computed in closed form.

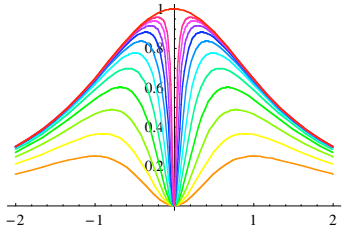
```
In[11]:= f∞(x_) = Sum[(x/(i x^2 + 1))^2, {i, 1, ∞}]
```

$$\text{Out[11]=}\quad \frac{\psi^{(1)}\left(\frac{x^2+1}{x^2}\right)}{x^2}$$

It is interesting to visualize the convergence of $f_n \to f_\infty$.

```
In[12]:= Plot[Evaluate[Join[Table[f2^n(x), {n, 0, 10}], {f∞(x)}]],
           {x, -2, 2}, PlotStyle → Table[Hue[i/12], {i, 12}]]
```

From In[12]:=



■ Explicit Roots of Transcendental Equations

Cauchy's integral theorem [mathworld.wolfram.com/CauchyIntegralTheorem.html] states that if $f(z)$ is analytic in some simply connected region $\mathcal{R}$, then

$$\oint_\gamma f(z)\,dz = 0 \tag{1}$$

for any closed contour γ completely contained in $\mathcal{R}$. The roots of a function $g(z)$ can be determined by locating the singularities of the reciprocal of the function, $h(z) = 1/g(z)$. If $h(z)$ has a simple pole at z_0, then $f(z) = (z - z_0)\,h(z)$ is analytic. Luck and Stevens [1] use this to obtain an *explicit* expression for the root z_0 of $g(z)$,

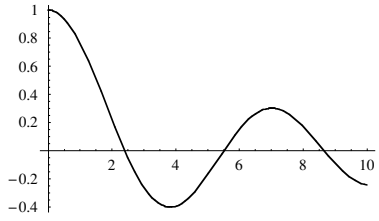
$$\oint_\gamma (z - z_0)\,h(z)\,dz = 0 \Rightarrow z_0 = \frac{\oint_\gamma z\,h(z)\,dz}{\oint_\gamma h(z)\,dz}, \tag{2}$$

where the contour γ contains the single pole at z_0.

For example, consider computing the roots of $\mathcal{J}_0(z)$. The roots lie on the real axis.

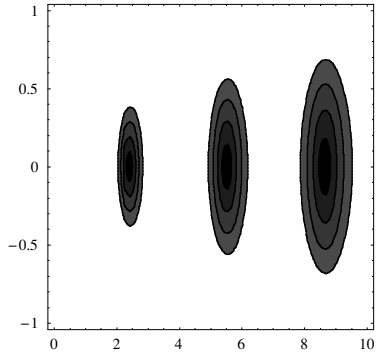
In[1]:= **Plot[$\mathcal{J}_0(z)$, {z, 0, 10}]**

From In[1]:=



In[2]:= **ContourPlot[|$\mathcal{J}_0(x + i\,y)$|, {x, 0, 10}, {y, −1, 1}, Contours → 0.05 Range[4], PlotPoints → 100]**

From In[2]:=



To determine the first root, we use equation (2), integrating around an arbitrary contour enclosing just this root.

In[3]:= **b(z_) = 1 / $\mathcal{J}_0(z)$;**

In[4]:= $$j_{0,1} = \frac{\text{NIntegrate}[z\,b(z),\ \{z,\ 1,\ 2 - i,\ 3,\ 2 + i,\ 1\}]}{\text{NIntegrate}[b(z),\ \{z,\ 1,\ 2 - i,\ 3,\ 2 + i,\ 1\}]}\ \text{// Chop}$$

Out[4]= 2.40483

We verify that this value is correct to machine precision.

In[5]:= $\mathcal{J}_0(j_{0,1})$

Out[5]= -4.89674×10^{-16}

Alternatively, evaluating both integrals around the same circular contour

$$z = a + r\,e^{i\theta} \Rightarrow dz = i\,r\,e^{i\theta}\,d\theta,$$

equation (2) becomes

$$z_0 = a + r\left(\frac{\int_0^{2\pi} f(\theta)\,e^{2\,i\theta}\,d\theta}{\int_0^{2\pi} f(\theta)\,e^{i\theta}\,d\theta}\right), \tag{3}$$

where $f(\theta) = b(a + r\,e^{i\theta})$. The values of a and r do not matter as long as the contour circumscribes the root.

To determine the second zero of $\mathcal{J}_0(z)$, we choose a circle centered at $a = 5$ with radius $r = 1$ that circumscribes this root and no other.

```
In[6]:= a = 5; r = 1; f(θ_) := b(a + r e^(i θ));
```

We use equation (3) to determine $j_{0,2}$.

```
In[7]:= j0,2 = a + r NIntegrate[f(θ) e^(2 i θ), {θ, 0, 2 π}] / NIntegrate[f(θ) e^(i θ), {θ, 0, 2 π}] // Chop
Out[7]= 5.52008
```

This is an excellent approximation to the second root.

```
In[8]:= J0(j0,2)
Out[8]= 8.36295 × 10^-16
```

The nth complex Fourier coefficient is defined by [mathworld.wolfram.com/FourierSeries.html]

$$c_n = \frac{1}{2\pi}\int_0^{2\pi} f(\theta)\, e^{i n \theta}\, d\theta,$$

so equation (3) reduces to

$$z_0 = a + r\,\frac{c_2}{c_1}, \tag{4}$$

involving a simple ratio of Fourier coefficients.

To determine the third zero of $\mathcal{J}_0(z)$, we choose a circle centered at $a = 9$ with radius $r = 1$ that circumscribes this root and no other, computing the Fourier coefficients *approximately* using **Fourier**, sampling $f(\theta)$ uniformly over $[0, 2\pi]$ just 16 times.

```
In[9]:= a = 9; cs = Rest[Fourier[Table[f(θ), {θ, 0, 2 π - π/8, π/8}]]]
Out[9]= {-14.7356, 5.10251, -1.76687, 0.611787, -0.211934,
  0.0731576, -0.0260457, 0.00728334, -0.00824435, -0.0100118,
  -0.042844, -0.0765791, -0.346537, -0.463324, -2.64574}
```

We use equation (4) to determine $j_{0,3}$.

```
In[10]:= j0,3 = a + r cs[[2]] / cs[[1]]
Out[10]= 8.65373
```

Then we check that this value is a good approximation to the third root.

```
In[11]:= J0(j0,3)
Out[11]= -7.16756 × 10^-8
```

Increasing the number of sample points improves the accuracy of roots computed via **Fourier**.

■ References

[1] R. Luck and J. W. Stevens, "Explicit Solutions for Transcendental Equations," *SIAM Review*, **44**(2), 2002 pp. 227–233.

Paul Abbott
School of Physics, M013
The University of Western Australia
35 Stirling Highway
Crawley WA 6009, Australia
tmj@physics.uwa.edu.au
physics.uwa.edu.au/~paul

In and Out

Edited by Paul Abbott

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. Answers posted to the *Mathematica* news group, comp.soft-sys.math.mathematica, that appear here are edited for clarity and length.

■ In-flight Puzzle

Q: This puzzle appeared in an in-flight magazine: In the following diagram, place the numbers 1 through 9 so that each row, column, and 3×3 subsquare contains each number exactly once.

	3		9				8	
		6	2		3	7	9	
			1					
	2		3				7	
				7			6	4
1								
	5				4	9		
	7	2						
	9			5		8	3	

How can I solve this using *Mathematica*?

A: Bobby Treat (drbob@bigfoot.com) answers: Here is a naïve solution that depends on there being, at each stage, at least one cell that is fully determined and that does not check for contradictions. It works for the sort of problem you would expect to encounter in an in-flight magazine.

Input the 9×9 table with empty cells replaced by empty lists.

In[1]:=
$$\text{puzzle} = \begin{pmatrix} \{\} & 3 & \{\} & 9 & \{\} & \{\} & \{\} & 8 & \{\} \\ \{\} & \{\} & 6 & 2 & \{\} & 3 & 7 & 9 & \{\} \\ \{\} & \{\} & \{\} & 1 & \{\} & \{\} & \{\} & \{\} & \{\} \\ \{\} & 2 & \{\} & 3 & \{\} & \{\} & \{\} & 7 & \{\} \\ \{\} & \{\} & \{\} & \{\} & 7 & \{\} & \{\} & 6 & 4 \\ 1 & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} \\ \{\} & 5 & \{\} & \{\} & \{\} & 4 & 9 & \{\} & \{\} \\ \{\} & 7 & 2 & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} \\ \{\} & 9 & \{\} & \{\} & 5 & \{\} & 8 & 3 & \{\} \end{pmatrix};$$

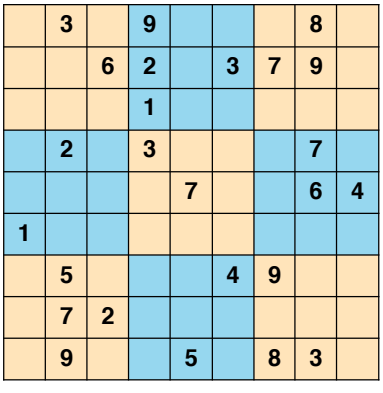
PuzzleDisplay, written by David Park (djmp@earthlink.net), is used to display the puzzle.

In[2]:=
```
Needs["Graphics`Colors`"]
```

In[3]:=
```
PuzzleDisplay[matrix_] :=
  Module[{workmatrix = Reverse[matrix], i, color, entry},
   color[red[_]] := Red; color[_] := Black;
   Show[Graphics[
     {Navajo,
      Rectangle[{0, 0}, {9, 9}],
      SkyBlue,
      MapThread[Rectangle[{#1, #2}, {#1 + 3, #2 + 3}] &,
       {{0, 3, 3, 6}, {3, 0, 6, 3}}],
      Black,
      Table[Line[{{i, 0}, {i, 9}}], {i, 0, 9}],
      Table[Line[{{0, i}, {9, i}}], {i, 0, 9}],
      Table[{color[entry = Part[workmatrix, i, j]],
        Text[entry /. {{} → " ", red → Identity}, {j - 1 / 2, i - 1 / 2}]},
       {i, 1, 9}, {j, 1, 9}]
     }],
    AspectRatio → Automatic,
    TextStyle →
     {FontFamily → "Helvetica", FontSize → 9, FontWeight → "Bold"},
    ImageSize → 150]];
```

In[4]:=
```
PuzzleDisplay[puzzle]
```

From In[4]:=



The function **SubSquare** finds the 3×3 subtable corresponding to the element $\{i, j\}$.

```
In[5]:= SubSquare[p_List][i_, j_] :=
          p[[3 Floor[(i - 1)/3] + Range[3], 3 Floor[(j - 1)/3] + Range[3]]]

In[6]:= SubSquare[puzzle][4, 5]

Out[6]= {{3, {}, {}}, {{}, 7, {}}, {{}, {}, {}}}
```

Excluded determines what values *cannot* be entered in the cell $\{i, j\}$, for cells that are not yet known.

```
In[7]:= Excluded[p_List][i_, j_] := If[ListQ@p[[i, j]],
          Union@Flatten@{SubSquare[p][i, j], p[[All, j]], p[[i, All]]}, 0]
        SetOptions[Graphics, AspectRatio → Automatic];
```

Four values are excluded for the cell {4, 5}.

```
In[9]:= Excluded[puzzle][4, 5]

Out[9]= {2, 3, 5, 7}
```

OneChoice, applied to the cell $\{i, j\}$, returns the existing value (if any), which is the only choice if eight digits have been excluded, or {} if the choice is not determined.

```
In[10]:= OneChoice[p_List][i_, j_] := Module[{e = Excluded[p][i, j]},
           Switch[Length@e,
            0, p[[i, j]],
            8, red@First@Complement[Range@9, e],
            _, {}]
         ]
```

There is only one choice for the cell {5, 2}.

```
In[11]:= OneChoice[puzzle][5, 2]

Out[11]= red[8]
```

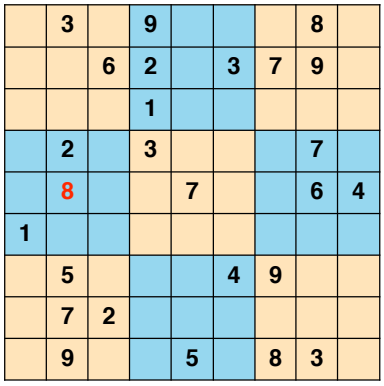
Here is the iteration step, **Iterate**.

```
In[12]:= Iterate[p_, options___Rule] :=
           Module[{a = Array[OneChoice[p], {9, 9}]},
            If[Animate /. {options} /. Options[Iterate], PuzzleDisplay[a]];
            a /. red → Identity
           ];
         Options[Iterate] = {Animate → True};
```

And here is the animated solution. You can use the Up and Down Arrow keys to step through the animation once it has started.

```
In[14]:= solution = FixedPoint[Iterate, puzzle];
         SelectionMove[EvaluationNotebook[], All, GeneratedCell]
         FrontEndTokenExecute["OpenCloseGroup"]; Pause[0.5];
         FrontEndTokenExecute["SelectionAnimate"]
```

From In[14]:=



■ MonomialOrder

Q : An option for specifying the **MonomialOrder** in **GroebnerBasis** is to provide an explicit weight matrix. But I do not understand the syntax for doing this. Are there examples of this available? In addition, I would like to see a weight matrix for some of the standard monomial orderings.

A : Daniel Lichtblau (danl@wolfram.com) answers: What is required for n variables is an $n \times n$ rational (or integer) matrix of full rank that comprises a well-founded term ordering. This in turn is fulfilled iff the first nonzero entry in each column is positive. Here is an example from a set of problems posed as a challenge for the ISSAC 1997 conference.

```
In[1]:= polys = {8 w^2 + 5 x w - 4 y w + 2 z w + 3 w + 5 x^2 + 7 y^2 + 7 z^2 - 7 x + 2 x y -
            7 y - 7 x z - 8 y z - 8 z + 8, 3 w^2 - 5 x w - 3 y w - 6 z w + 9 w +
            4 x^2 + 9 y^2 + 7 z^2 + 7 x + 2 x y + 5 y - 2 x z + 6 y z + 7 z + 5,
          -2 w^2 + 9 x w + 9 y w - 7 z w - 4 w + 8 x^2 + 6 y^2 - 6 z^2 + 8 x +
            9 x y + 4 y - 3 x z - 7 y z + 8 z + 2, 7 w^2 + 5 x w + 3 y w - 5 z w -
            5 w + 2 x^2 - 4 y^2 - 4 z^2 + 4 x + 9 x y + 6 y - 7 x z - 5 y z - 9 z + 2};

In[2]:= vars = {x, y, z, w};
```

For lexicographic ordering, you can use, for example, an identity matrix.

```
In[3]:= gbl1 = GroebnerBasis[polys, vars];

In[4]:= gbl2 = GroebnerBasis[polys, vars, MonomialOrder → IdentityMatrix[4]];

In[5]:= gbl1 === gbl2

Out[5]= True
```

For degree reverse lexicographic ordering, you can use a matrix with first row set to all ones, next row all zeros except last entry negative one, then move the negative one step to the left in successive rows.

```
In[6]:= gbdrl1 = GroebnerBasis[polys, vars,
            MonomialOrder → DegreeReverseLexicographic];

In[7]:= drlmat4 = {{1, 1, 1, 1}, {0, 0, 0, −1}, {0, 0, −1, 0}, {0, −1, 0, 0}};

In[8]:= gbdrl2 = GroebnerBasis[polys, vars, MonomialOrder → drlmat4];

In[9]:= gbdrl1 === gbdrl2

Out[9]= True
```

■ Plot Zooming

Q : I know that you can change the **PlotRange** option to zoom into a plot region, but it is awkward for every zoom to use this option. Is there a way to zoom into a plot using the mouse?

A : Ingolf Dahl (ingolf.dahl@telia.com) answers: At library.wolfram.com/infocenter/-MathSource/191 you will find a handy palette, written by Timothy Lottes, which is designed for this purpose. It works by controlling the **ImageRegion** parameter of the picture.

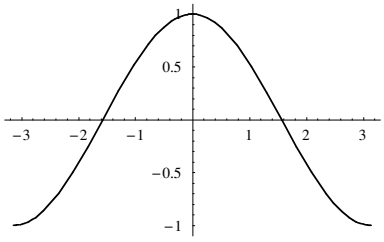
■ OpenCloseGroup

Q : I am using the following code to automatically close my graphics groups.

```
In[1]:= CloseAnimation[] :=
         (SelectionMove[EvaluationNotebook[], All, GeneratedCell];
          FrontEndTokenExecute["OpenCloseGroup"])

In[2]:= Table[Plot[Cos[x - t], {x, -π, π}, PlotRange → {-1.1, 1.1}], {t, 0, 6}];
        CloseAnimation[]
```

From In[2]:=



The problem is that when I only generate a single cell, **OpenCloseGroup** will generate a beep. How can I prevent that beep or test if **GeneratedCell** selected a single cell or several cells?

A: John Fultz (jfultz@wolfram.com) answers: You can test if multiple cells have been selected using

```
LinkWrite[$ParentLink,CellInformation[EvaluationNotebook[]]];
Length[LinkRead[$ParentLink]]
```

If the length is greater than 1, then you can use the **OpenCloseGroup** token.

In[3]:=
```
CloseAnimation[] :=
 (SelectionMove[EvaluationNotebook[], All, GeneratedCell];
  LinkWrite[$ParentLink, CellInformation[EvaluationNotebook[]]];
  If[Length[LinkRead[$ParentLink]] > 1,
   FrontEndTokenExecute["OpenCloseGroup"]])
```

CellInformation is new to version 5 and is, in general, a very good way to test various properties of the selection without having to do a **NotebookRead** (which could get very expensive if you have a selection that consumes a lot of memory). **CellInformation** takes a notebook argument and returns a list of lists, one for each cell in the selection (only one if you have only selected a single cell or the content of a cell). Each list contains rules along with information about the object that is selected (similar to the way **NotebookInformation** works).

- The first argument of **CellInformation** must be a **NotebookObject**.
- The rules currently returned are as follows.

Style	the cell's style
ContentData	the type of cell (for example, **GraphicsData**, **BoxData**, **TextData**)
Evaluating	boolean, whether the cell is in the evaluation queue
Rendering	boolean, whether the cell is in the rendering queue

NeedsRendering	boolean, whether the cell would need to be rendered if displayed
CursorPosition	the position of the cursor; could be **CellBracket** or a list of numbers representing the content selection
FirstCellInGroup	boolean, whether the cell is a head cell of a group
CellSerialNumber	unique numerical ID for the cell
Formatted	boolean, whether the cell or the cell expression is being displayed
InlineCellPosition	if selection is in an inline cell, then the cell's position relative to the topmost cell

⚠ Of course, since this is part of the undocumented packet interface, we reserve the right to make changes to it. I hope to see this or something like it promoted to a documented function similar to **NotebookInformation** in the future.

■ String Trimming

Q: How can I trim a string, such as

```
In[1]:= mystring = "  Have a great day.  ";
```

in order to remove leading and trailing spaces?

A: Bobby Treat (drbob@bigfoot.com) writes: One way is to use **Characters** and **StringJoin**.

```
In[2]:= StringJoin[Characters[mystring] //.
          {{" " .., x__} → {x}, {x__, " " ..} → {x}}] // FullForm
Out[2]//FullForm= "Have a great day."
```

Brian Higgins (bghiggins@ucdavis.edu) writes: Another way to achieve your goal is to use the **trim** method in Java via *J/Link*.

```
In[3]:= Needs["JLink`"];
        InstallJava[];
In[5]:= astring = JavaNew["java.lang.String", mystring]
Out[5]= «JavaObject[java.lang.String]»
In[6]:= astring[trim[]] // FullForm
Out[6]//FullForm= "Have a great day."
```

At www.higgins.ucdavis.edu/Jlink.php there are various examples of integrating Java with *Mathematica*, including one that makes use of the Java regex (regular expressions) package that is useful for more complicated string manipulations.

In *Mathematica* 5.1, you also have the ability to work with regular expressions or use **StringReplace**. This trims white space off ends of strings.

```
StringReplace[mystring,
  (StartOfString~~Whitespace)|(Whitespace~~EndOfString)→""] //
 InputForm
```

■ Root versus Radicals

Q: Version 5 returns the eigenvalues of the following simple matrix in terms of **Root** objects.

In[1]:=
$$\text{mat} = \begin{pmatrix} -1 & 1 & 0 \\ 1 & -2 & 1 \\ 0 & 1 & -2 \end{pmatrix};$$

In[2]:= λ = Eigenvalues[mat]

Out[2]=
$$\{\text{Root}[\#1^3 + 5\,\#1^2 + 6\,\#1 + 1\ \&,\ 1],\ \text{Root}[\#1^3 + 5\,\#1^2 + 6\,\#1 + 1\ \&,\ 2],\ \text{Root}[\#1^3 + 5\,\#1^2 + 6\,\#1 + 1\ \&,\ 3]\}$$

What does this output mean? I prefer the explicit answer that Version 4 returned. How do I coerce Version 5 to return an explicit answer?

A: Daniel Lichtblau (danl@wolfram.com) answers: The **Root** form is a concise way of expressing algebraic numbers via the minimal polynomial they satisfy, along with a canonical ordering in the complex plane (specified by the second argument of **Root**). To get a radical form, you can use **ToRadicals**.

In[3]:= rad = ToRadicals[λ]

Out[3]=
$$\left\{-\frac{5}{3} - \frac{\left(\frac{7}{2}\right)^{2/3}\left(1 + i\sqrt{3}\right)}{3\sqrt[3]{-1 + 3i\sqrt{3}}} - \frac{1}{6}\left(1 - i\sqrt{3}\right)\sqrt[3]{\frac{7}{2}\left(-1 + 3i\sqrt{3}\right)},\right.$$
$$-\frac{5}{3} - \frac{\left(\frac{7}{2}\right)^{2/3}\left(1 - i\sqrt{3}\right)}{3\sqrt[3]{-1 + 3i\sqrt{3}}} - \frac{1}{6}\left(1 + i\sqrt{3}\right)\sqrt[3]{\frac{7}{2}\left(-1 + 3i\sqrt{3}\right)},$$
$$\left.-\frac{5}{3} + \frac{7^{2/3}}{3\sqrt[3]{\frac{1}{2}\left(-1 + 3i\sqrt{3}\right)}} + \frac{1}{3}\sqrt[3]{\frac{7}{2}\left(-1 + 3i\sqrt{3}\right)}\right\}$$

Clearly, this output is more complicated than the **Root** form.

```
In[4]:= {LeafCount[rad], LeafCount[λ]}
Out[4]= {206, 61}
```

In addition to size, there are other reasons to prefer the **Root** form:

- It is typically faster to obtain.
- It is numerically more stable to evaluate. In general, radical formulations are prone to numeric problems. **Root** objects do not have this liability.
- When the roots of an irreducible cubic are all real but not rational, the so-called "casus irreducibilus" shows that they still must be expressed in terms of i (mathworld.wolfram.com/CasusIrreducibilus.html). This means that numeric evaluation will give small imaginary parts unless, by happenstance, they exactly cancel. Small numeric error from round-off makes this unlikely.

```
In[5]:= N[λ]
Out[5]= {-3.24698, -1.55496, -0.198062}
In[6]:= N[rad]
Out[6]= {-3.24698 + 0. i, -1.55496 - 2.22045 × 10^-16 i, -0.198062 + 1.11022 × 10^-16 i}
```

- For sufficiently complicated algebraics, it is often faster to evaluate the **Root** form numerically, at least at high precision.

```
In[7]:= N[λ, 10000]; // Timing // First
Out[7]= 0.08 Second
In[8]:= N[rad, 10000]; // Timing // First
Out[8]= 0.35 Second
```

- Polynomial combinations of **Root** objects simplify using **RootReduce**.

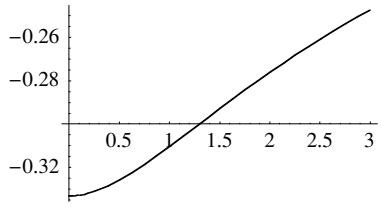
```
In[9]:= Plus @@ λ // RootReduce
Out[9]= -5
In[10]:= Times @@ λ // RootReduce
Out[10]= -1
In[11]:= λ^3 + 5 λ^2 + 6 λ + 1 // RootReduce
Out[11]= {0, 0, 0}
```

- Derivatives of **Root** objects with respect to a parameter are expressed in terms of **Root** objects. This is useful for eigenvalue sensitivity analysis.

```
In[12]:= ∂_x Root[ #^3 - x # + 1 &, 1]
                                  Root[#1^3 - x #1 + 1 &, 1]
Out[12]= ------------------------------------------------------
                                3 Root[#1^3 - x #1 + 1 &, 1]^2 - x
```

```
In[13]:= Plot[%, {x, 0, 3}]
```

From In[13]:=



So, one can avoid the **Root** form by using **ToRadicals**—but for all practical computation, you are better off working with **Root** objects.

■ Products of Polynomials

Q: I am computing large numbers of products of high-degree polynomials mod prime p, but the generic **PolynomialMod** function is too slow. How can I speed up this process?

A: Daniel Lichtblau (danl@wolfram.com) answers: There is fast dedicated technology built into the **Algebra`** context for manipulating dense univariate polynomials, represented by coefficient lists, modulo p.

```
In[1]:= ?Algebra`*
```

Algebra`

FactorSquareFreeModList
MatrixPowerMod
PartialSquareFreeDecompositionModList
PolynomialDerivativeModList
PolynomialDivisionModList
PolynomialExtendedGCDModList
PolynomialFactorModList
PolynomialGCDModList
PolynomialMakeMonicModList
PolynomialMinusModList
PolynomialPlusModList
PolynomialPowerMod
PolynomialPowerModList
PolynomialPthRootModList
PolynomialQuotientModList
PolynomialRemainderModList
PolynomialSubtractModList
PolynomialTimesModList
ReduciblePolynomialModListQ
SquareFreeDecompositionModList
SquareFreeModQ
SquareFreePartModList

The one you want is **PolynomialTimesModList**.

Start with code to generate random polynomials modulo some given p (which need not be prime).

```
In[2]:= randomPoly[deg_, mod_, x_] :=
         Table[Random[Integer, mod - 1], {deg + 1}].x^Range[0,deg]
```

Construct a pair of degree 1200 random polynomials.

```
In[3]:= deg = 1200;
        SeedRandom[1234];
        p = Prime[1234];
        poly1 = randomPoly[deg, p, x];
        poly2 = randomPoly[deg, p, x];
```

Multiply them using **PolynomialMod**.

```
In[8]:= Timing[prod1 = PolynomialMod[poly1 poly2, p];] // First
Out[8]= 10.77 Second
```

Converting to coefficient lists, multiplying, and converting back is much faster.

```
In[9]:= Timing[
          listpoly1 = CoefficientList[poly1, x];
          listpoly2 = CoefficientList[poly2, x];
          prod2list =
           Algebra`PolynomialTimesModList[listpoly1, listpoly2, p];
          prod2 = prod2list.x^Range[0,Length[prod2list]-1] ;] // First
Out[9]= 0.09 Second
```

Check the results for consistency.

```
In[10]:= prod1 === prod2
Out[10]= True
```

Generally speaking, you would want to do as much of the computation as possible using the list representation, and only convert from and to explicit polynomials, if at all, at the beginning and end of the process.

■ Closed Form Integral

Q : How can I compute a closed expression for

$$a_{k,l} = \int_0^1 P_k(x)\, P_l(2x - 1)\, dx,$$

where $P_n(z)$ is a Legendre polynomial?

A : Orthogonality of the Legendre polynomials requires that

$$P_k(x) = \sum_{l=0}^{k} (2\,l + 1)\, a_{k,l}\, P_l(2\,x - 1).$$

One can write $P_k(x)$ as a Gaussian hypergeometric function (functions.wolfram.com/Polynomials/LegendreP/26/01/01/0001),

$$P_k(x) = {}_2F_1\left(-k,\, k + 1;\, 1;\, \frac{1 - x}{2}\right).$$

Specializing equation (2.5) of Fields and Wimp [1], to the case where $p = 2$ and $q = 1$, one has

$${}_2F_1(a_1,\, a_2;\, b;\, z\,w) =$$

$${}_3F_2(\beta + 1,\, a_1,\, a_2;\, \alpha + \beta + 2,\, b;\, z) + \sum_{n=1}^{\infty} \frac{(a_1)_n\,(a_2)_n\,z^n}{(b)_n\,(n + \alpha + \beta + 1)_n}$$

$${}_3F_2(n + \beta + 1,\, n + a_1,\, n + a_2;\, 2\,n + \alpha + \beta + 2,\, b + n;\, z)\, P_n^{(\alpha,\beta)}(2\,w - 1).$$

Put $a_1 = -k$, $a_2 = k + 1$, $b = 1$, $z = 1/2$, $w = 1 - x$, $\alpha = \beta = 0$ into this formula.

```
In[1]:= 2F1[a1, a2; b; z w] == Sum[(Pochhammer[a1, n] Pochhammer[a2, n] z^n)/
          (Pochhammer[b, n] Pochhammer[n + α + β + 1, n]), {n, 0, ∞}]
          3F2[n + β + 1, n + a1, n + a2; 2 n + α + β + 2, b + n; z] JacobiP[n, α, β, 2 w - 1] /.
        {a1 → -k, a2 → k + 1, b → 1, α → 0, β → 0, z → 1/2, w → 1 - x, n → l}
```

$$\text{Out[1]=}\quad {}_2F_1\left(-k,\, k + 1;\, 1;\, \frac{1 - x}{2}\right) == \sum_{l=0}^{\infty} \frac{2^{-k}\,{}_2F_1(l - k,\, -k + l + 1;\, 2\,l + 2;\, -1)\, P_l(2\,(1 - x) - 1)\,(-k)_l\,(k + 1)_l}{(1)_l\,(l + 1)_l}$$

The term $(-k)_l$ causes the summand to vanish if $l > k$, making the sum finite. We simplify the summand as follows.

```
In[2]:= % /. HoldPattern[Sum[s_, r_]] :> Sum[Simplify[s], r]
```

$$\text{Out[2]=}\quad {}_2F_1\left(-k,\, k + 1;\, 1;\, \frac{1 - x}{2}\right) == \sum_{l=0}^{\infty} \frac{2^{-k}\,{}_2F_1(l - k,\, -k + l + 1;\, 2\,l + 2;\, -1)\, P_l(1 - 2\,x)\,(-k)_l\,(k + 1)_l}{(1)_l\,(l + 1)_l}$$

It is a good idea to verify that this expression is correct, at least for small k.

```
In[3]:= Table[%, {k, 0, 3}] // Simplify
```

```
Out[3]= {True, True, True, True}
```

Finally, using the parity property, $P_l(2x-1) = (-1)^l P_l(1-2x)$, we obtain the expansion coefficients $a_{k,l}$.

```
In[4]:= a_k_,l_ =
          (Last[%%] /. HoldPattern[Sum[s_, _]] :> Simplify[s]) / ((2 l + 1) P_l(1 - 2 x))
            (-1)^l
```

$$\text{Out[4]=}\quad \frac{(-1)^l\, 2^{-k}\, {}_2F_1(l-k, -k+l+1; 2l+2; -1)\,(-k)_l\,(k+1)_l}{(2l+1)\,(1)_l\,(l+1)_l}$$

We check this result for $0 \le k, l \le 3$.

```
In[5]:= Table[{l, k, ∫_0^1 P_k(x) P_l(2 x - 1) dx == a_k,l}, {l, 0, 3}, {k, 0, 3}]
```

$$\text{Out[5]=}\quad \begin{pmatrix} \{0,0,\text{True}\} & \{0,1,\text{True}\} & \{0,2,\text{True}\} & \{0,3,\text{True}\} \\ \{1,0,\text{True}\} & \{1,1,\text{True}\} & \{1,2,\text{True}\} & \{1,3,\text{True}\} \\ \{2,0,\text{True}\} & \{2,1,\text{True}\} & \{2,2,\text{True}\} & \{2,3,\text{True}\} \\ \{3,0,\text{True}\} & \{3,1,\text{True}\} & \{3,2,\text{True}\} & \{3,3,\text{True}\} \end{pmatrix}$$

■ References

[1] J. L. Fields and J. Wimp, "Expansions of Hypergeometric Functions in Hypergeometric Functions," *Mathematics of Computation*, **15**(76), 1961 pp. 390–395.

Paul Abbott
School of Physics, M013
The University of Western Australia
35 Stirling Highway
Crawley WA 6009, Australia
tmj@physics.uwa.edu.au
physics.uwa.edu.au/~paul

TROTT'S CORNER

The Sound of an Unusual Drum

Michael Trott

The eigenvalue spectrum of the Laplace operator (and its negative) is of fundamental importance in mathematics and physics. In this column, the lowest eigenvalues of a fractal 2D drum are calculated using a finite element discretization. The corresponding eigenfunctions and their gradients are visualized. The cumulative number of the first 1000 eigenvalues is calculated and compared with Weyl's asymptotic formula.

■ Introduction

For fans and friends of partial differential equations (PDEs), *Mathematica* 5 is a great step forward. The solution capabilities for parabolic and hyperbolic equations have been greatly expanded compared with Version 4. Now it is possible to choose various derivative approximation schemes and solve $1 + d$ problems. This means that now we can calculate, for example, Schrödinger equation solutions, Wigner functions, nonlinear reaction diffusion equations, and others in a variety of situations. Elliptic PDEs, on the other hand, cannot (yet) be solved using a direct call to `NDSolve`. But Version 5 has another excellent new toolset—sparse arrays and various linear algebra functions that deal with them. For simple linear elliptic PDEs, this is enough to program a solution "manually." In this Corner, I will discuss a vibrating drum with an unusual shape—one with many corners.

For physicists, the solutions of the Helmholtz eigenvalue problem, $-\Delta \psi_\lambda + V \psi_\lambda = \lambda^2 \psi_\lambda$, have a magic attraction; seeing this equation makes their eyes start to glow and unavoidably thoughts about "Sturm–Liouville," "Weyl deficiency indices," and "spectral density" come to mind. This equation describes many physical phenomena ranging from vibrating drums to quantum mechanics. Depending on the dimension, the boundary shape and boundary conditions, and the form of V, an astonishing number of interesting problems and phenomena arise. Typical investigations deal with the distribution of the eigenvalues, as well as quantitative and qualitative features of the eigenfunctions. One recent area of interest is nontrivial shapes. The eigenvalue problem for fractal Koch squares and Koch triangles was investigated in great detail in [1]. In this Corner, we will use a two-dimensional domain with another unusual boundary, with V set to

zero and simple homogeneous Dirichlet conditions. We will calculate some of the eigenvalues and the eigenfunctions of a highly irregular drum.

■ Construction of the Drum

In[1]:=
```
Off[General::spell]; Off[General::spell1];
```

We start with a graphic of the drum, the union of a set of more than 300 squares in the plane.

In[2]:=
```
squareCenters =
  Union[{{1, 1}, {1, 3}, {1, 7}, {1, 11}, {1, 13}, {1, 17}, {1, 21}, {1, 23}, {1, 29},
     {1, 33}, {1, 39}, {1, 45}, {1, 47}, {3, 3}, {3, 7}, {3, 11}, {3, 17}, {3, 21},
     {3, 27}, {3, 29}, {3, 33}, {3, 35}, {3, 37}, {3, 39}, {3, 41}, {3, 47}, {5, 3},
     {5, 5}, {5, 7}, {5, 9}, {5, 11}, {5, 13}, {5, 17}, {5, 19}, {5, 21}, {5, 23}, {5, 25},
     {5, 27}, {5, 35}, {5, 41}, {5, 43}, {5, 45}, {5, 47}, {7, 1}, {7, 3}, {7, 9}, {7, 19},
     {7, 23}, {7, 27}, {7, 31}, {7, 35}, {7, 37}, {7, 41}, {7, 45}, {9, 7}, {9, 9},
     {9, 11}, {9, 13}, {9, 15}, {9, 19}, {9, 27}, {9, 31}, {9, 33}, {9, 35}, {11, 1},
     {11, 3}, {11, 11}, {11, 15}, {11, 17}, {11, 19}, {11, 23}, {11, 31}, {11, 35},
     {11, 37}, {11, 39}, {11, 43}, {11, 45}, {11, 47}, {13, 3}, {13, 7}, {13, 11},
     {13, 15}, {13, 19}, {13, 23}, {13, 27}, {13, 31}, {13, 35}, {13, 39}, {13, 41},
     {13, 43}, {13, 47}, {15, 3}, {15, 5}, {15, 7}, {15, 9}, {15, 11}, {15, 23}, {15, 25},
     {15, 27}, {15, 29}, {15, 31}, {17, 1}, {17, 3}, {17, 9}, {17, 15}, {17, 17}, {17, 21},
     {17, 23}, {17, 31}, {17, 35}, {17, 37}, {17, 41}, {17, 43}, {17, 47}, {19, 7},
     {19, 9}, {19, 11}, {19, 13}, {19, 15}, {19, 27}, {19, 29}, {19, 31}, {19, 33},
     {19, 35}, {19, 43}, {19, 45}, {19, 47}, {21, 1}, {21, 3}, {21, 15}, {21, 17},
     {21, 19}, {21, 21}, {21, 23}, {21, 31}, {21, 35}, {21, 37}, {21, 39}, {21, 41},
     {21, 43}, {21, 47}, {23, 3}, {23, 5}, {23, 7}, {23, 9}, {23, 13}, {23, 15}, {23, 21},
     {23, 27}, {23, 31}, {23, 41}, {25, 5}, {25, 19}, {25, 21}, {25, 23}, {25, 27},
     {25, 29}, {25, 31}, {25, 33}, {25, 37}, {25, 39}, {25, 41}, {25, 43}, {25, 45},
     {25, 47}, {27, 1}, {27, 5}, {27, 7}, {27, 11}, {27, 15}, {27, 19}, {27, 29},
     {27, 33}, {27, 39}, {29, 1}, {29, 3}, {29, 5}, {29, 11}, {29, 13}, {29, 15},
     {29, 19}, {29, 21}, {29, 23}, {29, 25}, {29, 29}, {29, 33}, {29, 37}, {29, 39},
     {29, 41}, {29, 43}, {29, 45}, {31, 5}, {31, 7}, {31, 9}, {31, 11}, {31, 15},
     {31, 17}, {31, 19}, {31, 25}, {31, 27}, {31, 29}, {31, 45}, {31, 47}, {33, 1},
     {33, 3}, {33, 5}, {33, 11}, {33, 15}, {33, 19}, {33, 23}, {33, 25}, {33, 29},
     {33, 33}, {33, 35}, {33, 37}, {33, 41}, {33, 43}, {33, 45}, {35, 9}, {35, 11},
     {35, 25}, {35, 35}, {35, 45}, {35, 47}, {37, 1}, {37, 5}, {37, 9}, {37, 15},
     {37, 17}, {37, 21}, {37, 23}, {37, 25}, {37, 27}, {37, 31}, {37, 33}, {37, 35},
     {37, 37}, {37, 39}, {37, 47}, {39, 1}, {39, 3}, {39, 5}, {39, 9}, {39, 11}, {39, 13},
     {39, 15}, {39, 27}, {39, 39}, {39, 43}, {41, 5}, {41, 7}, {41, 9}, {41, 15},
     {41, 17}, {41, 19}, {41, 23}, {41, 27}, {41, 29}, {41, 31}, {41, 33}, {41, 37},
     {41, 39}, {41, 41}, {41, 43}, {41, 45}, {41, 47}, {43, 1}, {43, 3}, {43, 5},
     {43, 13}, {43, 15}, {43, 19}, {43, 21}, {43, 23}, {43, 31}, {43, 39}, {43, 43},
     {45, 5}, {45, 9}, {45, 15}, {45, 23}, {45, 25}, {45, 31}, {45, 33}, {45, 35},
     {45, 37}, {45, 39}, {45, 47}, {47, 1}, {47, 3}, {47, 5}, {47, 7}, {47, 9}, {47, 13},
     {47, 15}, {47, 17}, {47, 19}, {47, 25}, {47, 27}, {47, 35}, {47, 43}, {47, 47},
     {49, 1}, {49, 7}, {49, 13}, {49, 19}, {49, 21}, {49, 27}, {49, 29}, {49, 33},
     {49, 35}, {49, 37}, {49, 39}, {49, 41}, {49, 43}, {49, 45}, {49, 47}}] / 2;

Length[squareCenters]
```

Out[3]= 317

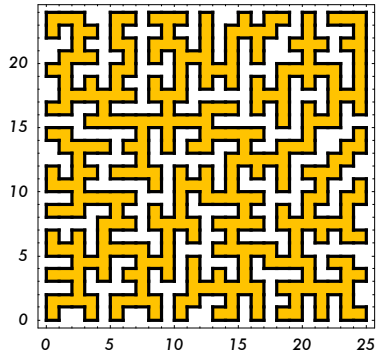
```
In[4]:= drumSquares = Polygon[{# + {-1, -1} / 2, # + {1, -1} / 2,
          # + {1, 1} / 2, # + {-1, 1} / 2}] & /@ squareCenters;
```

The edges of the drum are the edges of unit squares that appear exactly once.

```
In[5]:= drumEdges = Line[First[#]] & /@ Select[Split[Sort[Sort /@
          Flatten[Partition[Append[#[[1]], #[[1, 1]]], 2, 1] & /@ drumSquares, 1]]],
        Length[#] == 1 &];
```

```
In[6]:= Show[Graphics[
        {{Hue[0.12], #} & /@ drumSquares, {Thickness[0.01], drumEdges}},
        AspectRatio → Automatic, Frame → True]]
```

From In[6]:=



■ Solving the Eigenvalue Problem

To solve the Helmholtz equation for the drum, we will use a finite difference approximation of the Laplacian. To obtain a sufficient discretization, we subdivide the individual drum squares into `rL` points in each direction.

```
In[7]:= rL = 10; L = 1 / rL;
```

The two functions `refinePolygon` and `refineEdge` subdivide the squares that form the drum and the outer edges of the drum.

```
In[8]:= refinePolygon[Polygon[{p1_, p2_, p3_, p4_}], rL_] :=
          Module[{d1 = p2 - p1, d2 = p4 - p1},
           Table[Polygon[{p1 + j / rL d1 + k / rL d2, p1 + (j + 1) / rL d1 + k / rL d2,
               p1 + (j + 1) / rL d1 + (k + 1) / rL d2, p1 + j / rL d1 + (k + 1) / rL d2}],
             {j, 0, rL - 1}, {k, 0, rL - 1}] // Flatten];
```

```
In[9]:= refineEdge[Line[{p1_, p2_}], rL_] :=
          Module[{d1 = p2 - p1}, Table[Line[{p1 + j / rL d1, p1 + (j + 1) / rL d1}],
            {j, 0, rL - 1}] // Flatten];
```

Here are the refined squares and the interior and boundary points of the discretized problem.

```
In[10]:= refinedSquares = refinePolygon[#, rL] & /@ drumSquares;
```

```
In[11]:= allVertices = Union[Level[refinedSquares, {-2}]];

In[12]:= boundaryVertices =
           Union[Level[refineEdge[#, rL] & /@ drumEdges, {-2}]];

In[13]:= innerVertices = Complement[allVertices, boundaryVertices];
```

We will have to determine the function values for each eigenfunction at 28521 points.

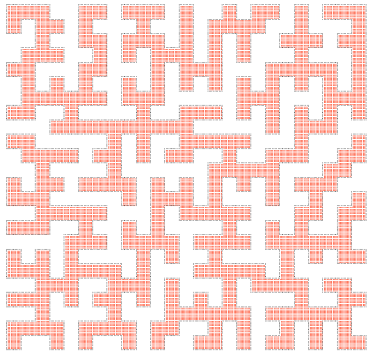
```
In[14]:= {dimD = Length[innerVertices], Length[boundaryVertices]}

Out[14]= {28521, 6360}
```

Here is a graphic of the resulting discretized drum.

```
In[15]:= Show[Graphics[{{PointSize[0.001], Hue[0], Point /@ innerVertices},
           {PointSize[0.002], GrayLevel[0], Point /@ boundaryVertices}}],
          PlotRange → All, AspectRatio → Automatic]
```

From In[15]:=



The following rules for testing and retrieving the boundary and interior points allow us to build the global finite differences easily and quickly. These rules also make it possible to quickly access the node number of a discretization point in the visualization functions to be defined later.

```
In[16]:= Do[vertexDNumber[innerVertices[[k]]] = k;
           vertexDCoordinates[k] = innerVertices[[k]];
           isInnerDVertex[innerVertices[[k]]] = True, {k, dimD}];

         isInnerDVertexQ[vertex_] :=
          isInnerDVertexQ[vertex] = TrueQ[isInnerDVertex[vertex]]

In[18]:= Do[boundaryVertexNumber[boundaryVertices[[k]]] = k;
           boundaryVertexCoordinates[k] = boundaryVertices[[k]];
           isBoundaryVertex[boundaryVertices[[k]]] = True,
           {k, Length[boundaryVertices]}];

         isBoundaryVertexQ[vertex_] :=
          isBoundaryVertexQ[vertex] = TrueQ[isBoundaryVertex[vertex]]
```

Next, we must approximate the second derivative at the grid points. The function `stencilWeightList[`ν`][`*leftPoints*`, `*rightPoints*`]` gives the list of the weights of the νth derivative using the function values at the left and right neighbor points. The function uses the built-in command `NDSolve`FiniteDifferenceDerivative`. See Advanced Documentation: `NDSolve` in the Help Browser for more details.

```
In[20]:= stencilWeightList[ν_][leftPoints_, rightPoints_] :=
          stencilWeightList[ν][leftPoints, rightPoints] =
           Module[{o = leftPoints + 1 + rightPoints, grid},
             grid = Range[o]; Coefficient[#, f /@ grid] & /@
              NDSolve`FiniteDifferenceDerivative[Derivative[ν], grid,
               f /@ grid, DifferenceOrder → o - ν]][[leftPoints + 1]]
```

For instance, these are the weights for a stencil of length seven for the second derivative.

```
In[21]:= stencilWeightList[2][3, 3]
```

$$\text{Out[21]}= \left\{\frac{1}{90}, -\frac{3}{20}, \frac{3}{2}, -\frac{49}{18}, \frac{3}{2}, -\frac{3}{20}, \frac{1}{90}\right\}$$

Now we assemble the system of coupled linear equations representing the approximations of the derivatives. The function `addVertexWeights` adds the contributions of the list of finite difference weights *vws* for each vertex.

```
In[22]:= addVertexWeights[vws_] :=
          If[Length[#] === 1, First[#], #[[1, 1]] → (Plus @@ (Last /@ #))] & /@
           Split[Sort[vws], #1[[1]] === #2[[1]] &]
```

For a given stencil of odd length $2\alpha + 1$, the function `makeNonVanishingDMatrixElements` returns a list of the nonvanishing elements of the discretization of the negative of the 2D Laplace operator. We incorporate the Dirichlet boundary conditions through an antisymmetric continuation of the function across the boundary, using an odd stencil length, to make sure that is always easily possible.

```
In[23]:= makeNonVanishingDMatrixElements[α_Integer] := addVertexWeights[
           Module[{centerVertex, jB, swl = -stencilWeightList[2][α, α]},
            Flatten[Table[centerVertex = innerVertices[[k]];
              {{k, k} → 2 swl[[α + 1]], Function[dir, jB = .; Table[nextVertex =
                    centerVertex + j L dir; Which[isInnerDVertexQ[nextVertex],
                    {k, vertexDNumber[nextVertex]} → swl[[α + 1 + j]],
                    isBoundaryVertexQ[nextVertex], jB = j; Sequence @@ {}, True,
                    reflectedVertex = centerVertex + jB L dir - (j - jB) L dir;
                    {k, vertexDNumber[reflectedVertex]} → -swl[[α + 1 + j]],
                  {j, 1, α}]] /@ {{+1, 0}, {-1, 0}, {0, 1}, {0, -1}}},
              {k, dimD}]]]];
```

The resulting matrix approximates the negative Laplacian. It has 332637 nonvanishing entries.

```
In[24]:= MD7 = SparseArray[makeNonVanishingDMatrixElements[3]]

Out[24]= SparseArray[<332637>, {28521, 28521}]
```

The matrix is symmetric by construction.

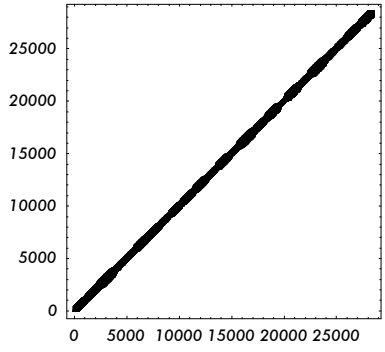
```
In[25]:= Max[MD7 - Transpose[MD7]]

Out[25]= 0
```

And it is nicely banded.

```
In[26]:= Show[Graphics[
           {GrayLevel[0], Point[#1]} & @@@ makeNonVanishingDMatrixElements[3]],
          PlotRange → All, Frame → True, AspectRatio → Automatic]
```

From In[26]:=



Now we are in a position to calculate the eigenvalues and eigenfunctions. Here are the first few eigenvalues, which we reorder and scale.

```
In[27]:= drumEigensystem[n_, opts___] :=
          MapAt[1 / L^2 # &, Reverse /@ Eigensystem[N[MD7], n,
             Method → {"Arnoldi", opts, Tolerance → 10^-13,
               BasisSize → Abs[25 n], MaxIterations → 10^4}], 1]

In[28]:= {evals1, evecs1} = drumEigensystem[24, Shift → 0];
         evals1

Out[29]= {6.64705, 6.734, 7.02307, 7.13222, 7.14203, 7.14338,
          7.14347, 7.16452, 7.17458, 7.28647, 7.29062, 7.29112,
          7.29209, 7.31277, 7.32897, 7.32952, 7.42389, 7.47142,
          7.51433, 7.52146, 7.53928, 7.70142, 7.71503, 7.74563}
```

■ Visualization of the Eigenfunctions

For the visualization of the eigenfunctions, we define a function `eigenVibration3DPlot`.

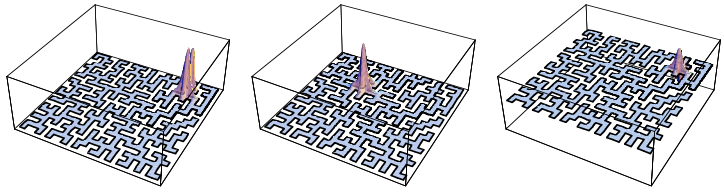
```
In[30]:= drum3DPlot[vertexValueFunction_, opts___] := Graphics3D[{EdgeForm[],
           Map[vertexValueFunction, refinedSquares, {-2}], {GrayLevel[0],
            Thickness[0.006], Map[Append[#, 0] &, drumEdges, {-2}]}},
          opts, PlotRange → All, BoxRatios → {1, 1, 0.4}];
```

```
In[31]:= eigenVibration3DPlot[evec_, opts___] :=
          With[{ev = evec / Sign[Plus @@ evec]},
           drum3DPlot[Function[v, Append[v,
              If[isInnerDVertexQ[v], ev[[vertexDNumber[v]]], 0]]], opts]]
```

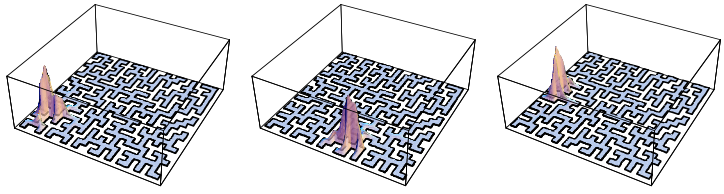
Here we show the first nine eigenfunctions. The two lowest states appear in the two crossings. Such crossings can hold eigenstates by themselves [2].

```
In[32]:= Show[GraphicsArray[#]] & /@
          Partition[eigenVibration3DPlot /@ Take[evecs1, 9], 3]
```

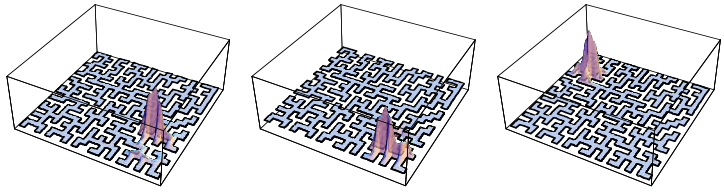
From In[32]:=



From In[32]:=



From In[32]:=



The higher eigenfunctions oscillate more and are better visualized using a contour plot, implemented as the function `eigenVibrationContourPlot`. Here we use the contour values that give homogeneous contour spacing.

```
In[33]:= homogeneousContours[data_, cl_] := With[{λ = Length[data]},
           #[[Round[λ / cl / 2]]] & /@ Partition[Sort[data], Round[λ / cl]]]
```

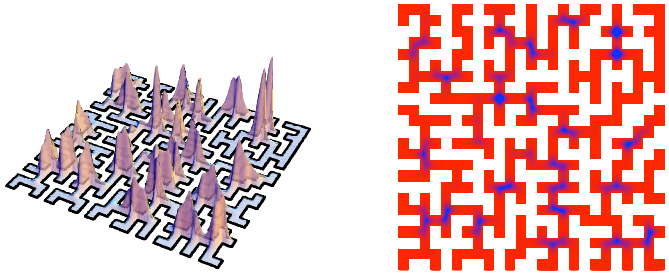
```
In[34]:= {maxX, maxY} = Max /@ Transpose[squareCenters];
         squareCentersC = Complement[Flatten[
             Table[{x, y}, {x, 1 / 2, maxX}, {y, 1 / 2, maxY}], 1], squareCenters];
         coverSquares = With[{a = 0.50001},
            Polygon[{# + {-a, -a}, # + {a, -a}, # + {a, a}, # + {-a, a}}] & /@
             squareCentersC];
```

```
In[37]:= eigenVibrationContourPlot[evec_, opts___] :=
           Module[{ev = Abs[evec], data}, data = Table[
              If[isInnerDVertexQ[{x, y}], ev[[vertexDNumber[{x, y}]]], 0],
              {y, 0, maxY + 1 / 2, L}, {x, 0, maxX + 1 / 2, L}];
            ListContourPlot[data,
             opts, Contours → homogeneousContours[ev, 80],
             ColorFunction → (RGBColor[1 - #, 0, #] &), ContourLines → False,
             PlotRange → All, Frame → False, MeshRange → {{0, 25}, {0, 24}},
             Epilog → {GrayLevel[1], coverSquares}]];
```

The next graphic shows the sum of the squares of the first 24 states. All of the lowest states are localized in the "larger crossing."

```
In[38]:= With[{evecSum = Plus @@ (evecs1^2)},
          Show[GraphicsArray[Block[{$DisplayFunction = Identity},
             {eigenVibration3DPlot[evecSum, Boxed → False],
              eigenVibrationContourPlot[evecSum]}]]]]
```

From In[38]:=



Next, we calculate some of the higher eigenfunctions near the eigenvalue $2\pi^2 \approx 19.7392$. This will be a good test of the method because the function $\psi(x, y) \sim \sin(\pi x)\sin(\pi y)$ is obviously an eigenfunction for the drum and fulfills the boundary conditions trivially.

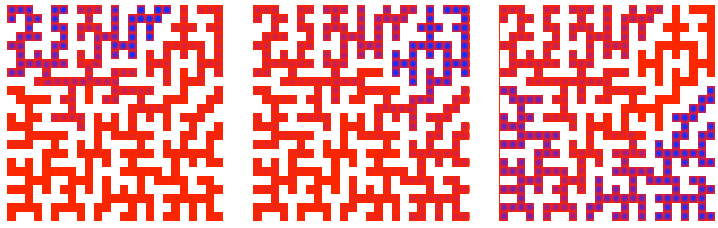
```
In[39]:= {evals2, evecs2} = drumEigensystem[12, Shift → L^2 (2 Pi^2 + 0.8)];
         evals2

Out[40]= {19.6675, 19.7075, 19.722, 19.7263, 19.7392, 20.57,
          21.1371, 21.1863, 21.2331, 21.2556, 21.2921, 21.4037}
```

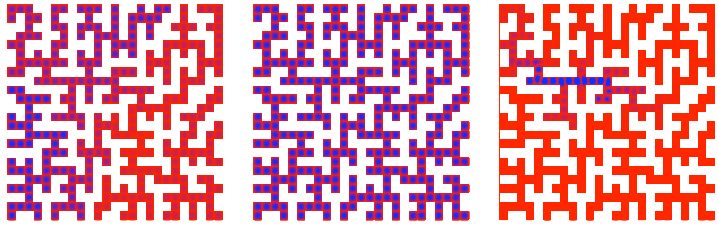
Here we show the corresponding eigenfunctions. The fifth of the selected eigenfunctions is the global product state. Overall we see that the eigenfunctions at this eigenvalue are structured along the drum spine with approximately one maxima per square and only a little structure in the local perpendicular directions.

```
In[41]:= Show[GraphicsArray[#]] & /@ Block[{$DisplayFunction = Identity},
           Partition[eigenVibrationContourPlot /@ Take[evecs2, 9], 3]]
```

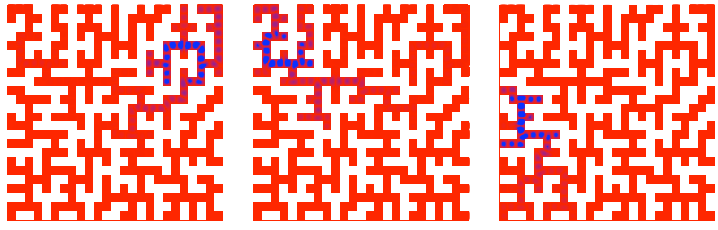
From In[41]:=



From In[41]:=



From In[41]:=



Next, we calculate some eigenvalues and eigenfunctions at still higher eigenvalues.

In[42]:=
```
{evals3, evecs3} = drumEigensystem[12, Shift → L^2 (8. Pi^2)];
evals3
```

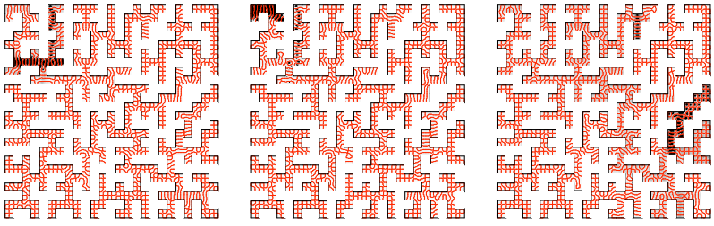
Out[43]=
```
{78.7033, 78.7251, 78.7605, 78.7798, 78.8091, 78.892,
 78.9135, 78.9487, 78.9614, 79.0539, 79.117, 79.1624}
```

This time we show the nodal curves of the eigenfunctions in addition to the absolute values of the eigenfunctions. The typical state is localized in a small part of the drum. Only the global product state extends over the whole drum.

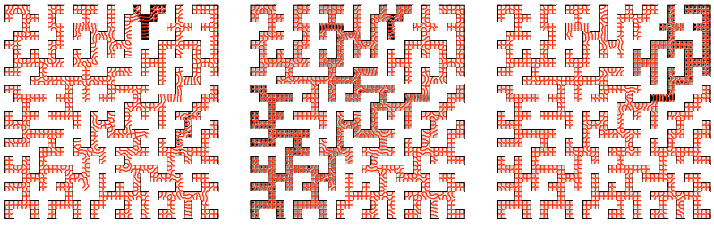
In[44]:=
```
eigenVibrationNodalCurvePlot[evec_, opts___] :=
  Module[{ev = Chop[evec, 10^-8], data}, data = Table[If[
      isInnerDVertexQ[{x, y}], ev[[vertexDNumber[{x, y}]]], 10^-20],
     {y, 0, maxY + 1 / 2, L}, {x, 0, maxX + 1 / 2, L}];
   Show[
    Block[{$DisplayFunction = Identity}, {eigenVibrationContourPlot[
      evec, ColorFunction → (GrayLevel[1 - #] &)],
     ListContourPlot[data, Contours → {0},
      ContourStyle → {{Thickness[0.002], Hue[0]}},
      ContourShading → False, ContourLines → True, PlotRange → All,
      Frame → False, MeshRange → {{0, 25}, {0, 24}}],
     Graphics[{GrayLevel[1], coverSquares}],
     Graphics[{GrayLevel[0], Thickness[0.005], drumEdges}]}]]];
```

In[45]:=
```
Show[GraphicsArray[#]] & /@ Block[{$DisplayFunction = Identity},
  Partition[eigenVibrationNodalCurvePlot /@ Take[evecs3, 12], 3]]
```

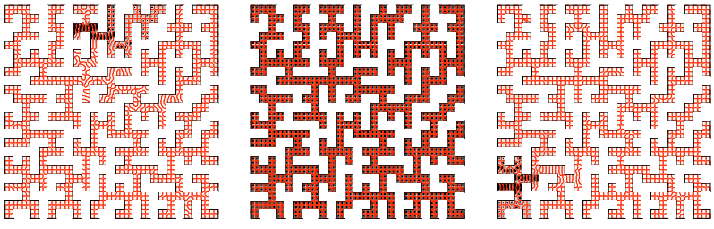
From In[45]:=



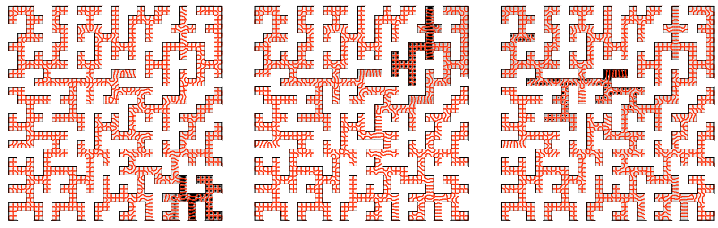
From In[45]:=



From In[45]:=



From In[45]:=



It is interesting to look at the functions $|\nabla \psi_\lambda(x, y)|$. The function `grad1DValue` again uses a finite difference approximation to calculate an approximate value of $\nabla \psi_\lambda(x, y)$.

In[46]:=
```
grad1DValue[vertex_, dir_, maxStencilWidth_, evec_] :=
 Module[{jL, jR, j, α = (maxStencilWidth - 1) / 2, sData, swl, vcs},
  If[isBoundaryVertexQ[vertex],
   Which[isBoundaryVertexQ[vertex + L dir] ||
     isBoundaryVertexQ[vertex - L dir], {jL, jR} = {0, 0},
    isInnerDVertexQ[vertex + L dir], {jL, jR} = {0, maxStencilWidth},
    isInnerDVertexQ[vertex - L dir], {jL, jR} = {maxStencilWidth, 0}],
   {jL, jR} = (j = 1; While[nextVertex = vertex + # j L dir;
        j ≤ maxStencilWidth + 1 && isInnerDVertexQ[nextVertex], j++];
       j - 1) & /@ {-1, +1};
   Which[jL > α && jR > α, {jL, jR} = {α, α}, jL + jR > maxStencilWidth,
    If[jR ≥ jL, jL = Min[α, jL]; jR = maxStencilWidth - jL,
     jR = Min[α, jR]; jL = maxStencilWidth - jR]];
  If[{jL, jR} =!= {0, 0}, swl = stencilWeightList[1][jL, jR];
   vcs = Table[vertexDNumber[vertex + j L dir], {j, -jL, jR}] /.
     _vertexDNumber :> 0];
  Which[{jL, jR} === {0, 0}, 0, vcs[[+1]] === 0,
   Rest[swl].evec[[Rest[vcs]]], vcs[[-1]] === 0,
   Most[swl].evec[[Most[vcs]]], True, swl.evec[[vcs]]]]
```

In[47]:=
```
gradValue[vertex_, maxStencilWidth_, evec_] :=
 Sqrt[#.#] &@{grad1DValue[vertex, {1, 0}, maxStencilWidth, evec],
   grad1DValue[vertex, {0, 1}, maxStencilWidth, evec]}
```

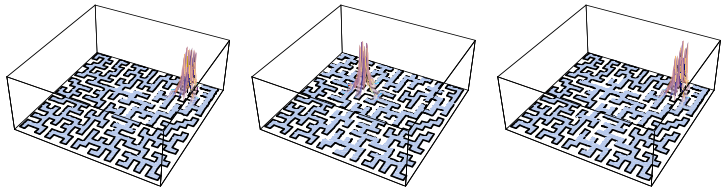
We now define a function `eigenVibrationGrad3DPlot` to visualize the gradients as 3D plots similar to the function `eigenDVibration3DPlot`.

In[48]:=
```
eigenVibrationGrad3DPlot[gradInnerVector_,
  gradBoundaryVector_, opts___] := drum3DPlot[Function[v, Append[v,
    If[isInnerDVertexQ[v], gradInnerVector[[vertexDNumber[v]]],
     gradBoundaryVector[[boundaryVertexNumber[v]]]]]]]
```

Here are the norms values of the gradients of the lowest eigenfunctions. As is well known, the gradients of harmonic functions diverge as interior corners [3].

```
In[49]:= Show[GraphicsArray[eigenVibrationGrad3DPlot[
           Table[gradValue[vertexDCoordinates[k], 7, #], {k, dimD}],
           Table[gradValue[boundaryVertexCoordinates[k], 7, #],
            {k, Length[boundaryVertices]}]] & /@ Take[evecs1, 3]]]
```

From In[49]:=



■ The Weyl Law

Finally, we calculate some more eigenvalues. Because the Arnoldi method calculates only some of the eigenvalues, we will call the function `drumEigensystem` repeatedly to calculate the first 1000 eigenvalues. To keep the memory usage low, we will calculate 24 eigenvalues each time. By aligning the newly calculated eigenvalues with the last set, we are sure not to miss an eigenvalue.

```
In[50]:= nextEvalCenter[l_] := (Plus @@ Take[l, -4] / 4) + 1 / 2 (l[[5]] - l[[4]])
```

```
In[51]:= evalNumberMax = 1000;

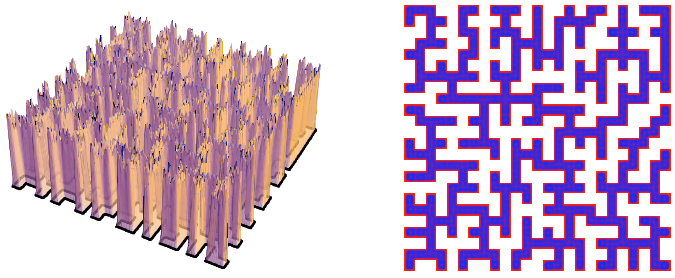
         {currentEvals, currentEvecs} = drumEigensystem[24, Shift → 0];
         evalList = currentEvals;
         evecSum = Plus @@ (currentEvecs^2);
         nodalLengthList = nodalLength /@ currentEvecs;
```

```
In[56]:= While[
          evDim = 24;
          While[{nextEvals, nextEvecs} =
            drumEigensystem[evDim, Shift → L^2 nextEvalCenter[evalList]];
           δs = #.# & /@ ((Take[evalList, {-4, -1}] - #) & /@
               Partition[nextEvals, 4, 1]); Print[{evDim, δs}];
          minPos = Position[δs, _?((# < 10^-6) &)];
          newPosis = If[minPos =!= {{Length[δs]}} && minPos =!= {},
            {minPos[[1, 1]] + 4, Length[nextEvals]}, {}];
          newPosis === {}, evDim = Round[3 / 2 evDim]];
         evalList = Join[evalList, Take[nextEvals, newPosis]];
         evecSum = evecSum + Plus @@ (Take[nextEvecs, newPosis]^2);
         nodalLengthList =
          Join[nodalLengthList, nodalLength /@ Take[nextEvecs, newPosis]];
         Length[evalList] < evalNumberMax, Null]
```

The sum of the first 1000 eigenvectors squared yields the following castle-like graphic. We see a smooth constant background density with relatively small oscillations superimposed.

```
In[57]:= Show[GraphicsArray[Block[{$DisplayFunction = Identity},
           {eigenVibration3DPlot[evecSum, Boxed → False],
            eigenVibrationContourPlot[evecSum]}]]]
```

From In[57]:=



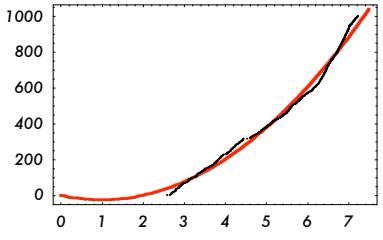
One of the holy grails of $-\Delta\,\psi_\lambda + V\,\psi_\lambda = \lambda^2\,\psi_\lambda$ is counting the number of eigenvalues λ that are less than Λ. According to the classical Weyl law, the number of eigenvalues $n(\lambda)$ up to the eigenvalue λ is given by the asymptotic formula $n(\Lambda) = A/(4\pi)\,\Lambda^2 - l/(4\pi)\,\Lambda$ [4] for smooth boundaries. Here A is the area of the drum and l the length of its boundary. We see a good overall agreement between the theoretical curve and the calculated cumulative eigenvalue count. At $\Lambda = 2\,\pi^2$—where the eigenfunctions start to develop nontrivial lateral structure—we see a clear spectral gap and a change in the slope of the calculated $n(\Lambda)$. For small Λ, the linear contribution from the length of the circumference dominates the quadratic area contribution.

```
In[58]:= {drumBoundaryLength, drumArea} =
          {Length@drumEdges, Length@drumSquares}

Out[58]= {636, 317}
```

```
In[59]:= Show[Graphics[{{Hue[0], Thickness[0.01],
           Line[Table[{Λ, drumArea / (4 Pi) Λ^2 - drumBoundaryLength / (4 Pi) Λ},
             {Λ, 0, 7.5, 1 / 10}]]}, {GrayLevel[0], PointSize[0.006],
           MapIndexed[Point[{#1, #2[[1]]}] &, Sqrt[evalList]]}},
         Frame → True, PlotRange → All]]
```

From In[59]:=



While for $\Lambda \lesssim 4.5$ the calculated $n(\Lambda)$ seems to be linear overall, analyzing the detailed behavior of the terms that contribute to $n(\Lambda)$ would lead us outside of the realm of this column. The formula provided earlier was for smooth boundaries. For fractal-like boundaries, we have a correction to the volume term of the

form $c\,\Lambda^d$ instead [5], where d is the fractal dimension of the boundary. And, indeed, a quadratic function fits the corrections slightly better than a linear one.

```
In[60]:= δIDOS = MapIndexed[{#1, #2[[1]] - drumArea / (4 Pi) #1^2} &,
            Select[Sqrt[evalList], # < 4.5 &]];
         fitLinear[x_] = Fit[δIDOS, {1, x}, x];
         fitQuadratic[x_] = Fit[δIDOS, {1, x^2}, x];
         Function[fit, (Plus @@ ((#2 - fit[#1])^2 & @@@ δIDOS))] /@
          {fitLinear, fitQuadratic}

Out[63]= {7827.27, 7069.3}
```

In conclusion, we leave it to the interested reader to calculate the distribution of the eigenvalue spacings, the autocorrelation functions of the eigenfunctions, and the localization lengths, and to compare them with the properties of a quantum graph corresponding to the drum spine, to add a magnetic field perpendicular to the drum, and so on; or, to go in a more artistic instead of scientific direction and "play" the drum using `Play` on the eigenvalues.

■ References

[1] S. Homolya, C. F. Osborne, and I. D. Svalbe, "Density of States for Vibrations of Fractal Drums," *Physical Review* E, **67**(2), 2003 pp. 026211-1–026211-13.

[2] R. L. Schult, D. G. Ravenhall, and H. W. Wyld, "Quantum Bound States in a Classically Unbound System of Crossed Wires," *Physical Review* B, **39**(8), 1989 pp. 5476–5479.

[3] S. Russ and B. Sapoval, "Increased Damping of Irregular Resonators," *Physical Review* E, **65**(3), 2002 pp. 036614-1–036614-10.

[4] H. P. Baltes and E. R. Hilf, *Spectra of Finite Systems*, Mannheim: BI Wissenschaftsverlag, 1976.

[5] M. V. Berry, "Distribution of Modes in Fractal Resonators," *Structural Stability in Physics* (W. Guttinger and H. Elkheimer, eds.), Berlin: Springer-Verlag, 1979 pp. 51–53.

Michael Trott
Special Functions Developer
Wolfram Research, Inc.
mtrott@wolfram.com

WaveX: Extracting Wavelets from Seismic Data

John M. Novak

***Mathematica* is a powerful tool for the analysis and manipulation of data. It can be used to provide a competitive edge in many commercial endeavors, including the field of petroleum exploration. One of the many tools created in *Mathematica* by GMG/AXIS for internal use is WaveX, a tool for extracting relative wavelets from seismic data based on well log information. Here we demonstrate some of the uses of the tool and discuss how *Mathematica* allowed us to create a superior tool.**

■ The Problem

Petroleum exploration employs a variety of methods for characterizing the earth's subsurface while seeking potential hydrocarbon reservoirs. One of the most common techniques used on land is to create vibrations on the surface, perhaps by a dynamite explosion, then to listen via an array of geophones for echoes from beneath the surface. This and associated technologies are collectively called *seismic exploration*.

The seismic data recorded by the sensors around the explosion are processed and massaged to extract meaningful signals, which may be visualized as an image of the subsurface. These images can then be analyzed with the goal of understanding something about the structure of the earth, such as the types and nature of the rock and whether fluids, such as oil or gas, exist within pores in the rock.

One way to calibrate the seismic data gathered at the surface with the properties of the earth is to drill a hole, or well, into the earth. We can more directly measure the characteristics of the earth at that point. This information can then be compared to the seismic data, a process called *a well tie*. Tying the well involves matching correlative events observed in the well (or the *well log*, typically acoustic velocity and bulk density measurements) with the seismic data. This process generates a waveform whose features can be used to assess and adjust the seismic data. This waveform is called a *wavelet*. (The wavelet is not the same as the mathematical structure of the same name.) The character of the extracted wavelet can also provide feedback about the quality of the well tie.

Commercial software used for tying wells and extracting wavelets imposes certain algorithms and methodologies upon the user. GMG/AXIS Geophysics desired a more flexible tool that could be modified to suit the peculiarities of individual data sets and could be adapted to different algorithms. *Mathematica* appeared to be an effective platform on which to create this tool.

The Solution

With the creation of a number of functions for seismic file import, graphics display, and data analysis, *Mathematica* was converted into a powerful environment for well ties and wavelet extraction. The following sequence of inputs demonstrates an example workflow using this functionality in *Mathematica* 5.

The flow begins by loading the packages with the required utility. On production machines, this is typically autoloaded.

```
<< Seismic`
```

Next, we load in the data, consisting of the well log to be tied and a window of seismic data from receivers near the well location. In this case, we specify one set of data, or *trace*, from each side of the well location, forming a 3×3 grid of traces. The data set can then be manipulated by a handle, which we return and here store in the variable `dset`.

```
dset = LoadWaveXData["welllog.las", "seismic.segy", 1]
```

```
Columns:
 {DEPTH, CALI, DT, EGRCN, ERHOB, ILD, COND, TDPAIR_DT, DT_CLEAN,
  GR_MEDIAN, GR_DT, GR, USEDPOR, QL_VSH, DT_NODEPTH, SONIC_ROTATION,
  DT_ND_EDIT, DT_EDIT, DT_ND_CORRECTED, COAL, DT_EDIT_BACKUP,
  DT_EDIT_DRIFT, DT_EDIT_COPY, DT_EDIT_CORRECTED, DT_SEISMIC}
```

```
run1
```

Separating out the correlation stage from the data loading allows us to iterate correlations without the expense of reloading the data.

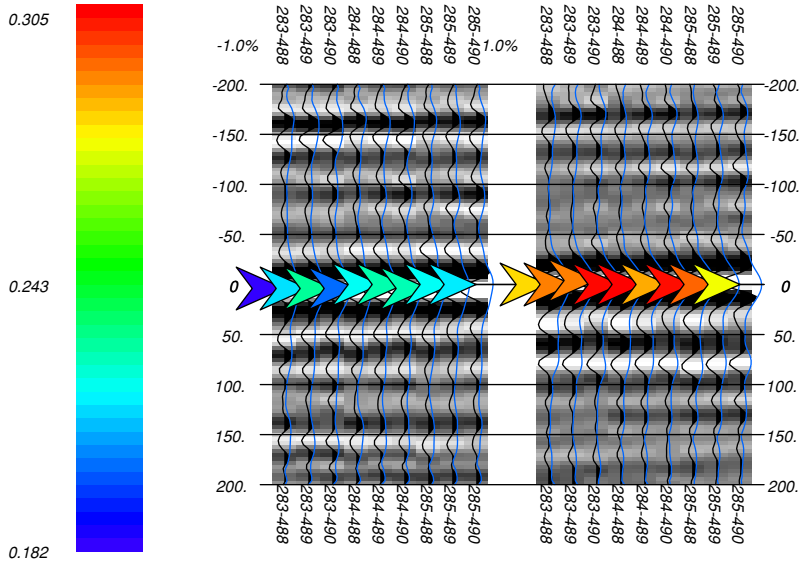
The well log data typically consists of a series of measurements of rock velocity and density at half-foot increments. Seismic data is measured in time, while the log data is in depth; these must be put in the same domain for further comparison. By finding the rock velocities within the well, we can estimate the travel time for a shock wave going from the surface, down to a particular depth, and back up to the receiver. This data is usually not complete, however, which may cause us to incorrectly estimate a travel time corresponding to depth. Well log data can also suffer from systematic errors in measurement, which can be accounted for when tying the log by applying a small time shift between the well log (*sonic*) and the seismic data. The user does this by specifying the correspondence between particular times in the two data sets. The sonic can also be stretched out in time to compensate for other systematic effects; this is given as a range of percentage stretches. A portion of the seismic and the sonic are then

correlated over some range of lags, given below as 900 milliseconds of data correlated over ±200 milliseconds.

```
WaveXCorrelate[dset, StretchRange → {-1, 1, 2},
  SonicTie → 950, TieLength → 900, CorrelationLength → 200];
```

So, each extracted seismic trace is cross-correlated with each stretched sonic log. For the parameters used previously, this results in 18 time series of ±200 milliseconds in length. We are interested in knowing the value and location of the peak correlation coefficients. The function `ShowCorrelations` displays all the correlations, instantaneous amplitude envelopes, pointers to the peak locations, and so on. The bar on the left gives the color coding for the correlation pointers; this example underlays the data with a density plot of the amplitudes, which in some cases may make some features more obvious.

```
ShowCorrelations[dset, ColorFunction → GrayScale]
```



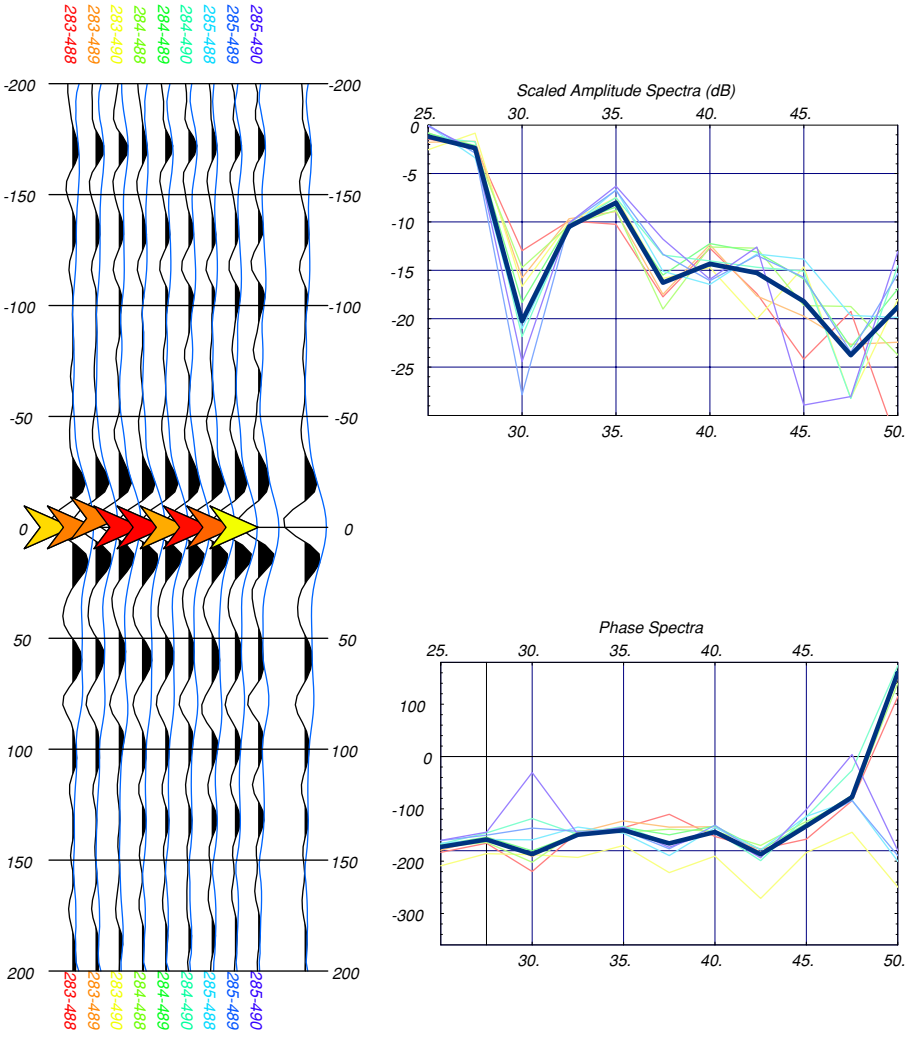
Examining the previous graph, we may decide that 1% stretch provides the best correlations. We select it for further operations.

```
PickStretch[dset, 1.0]

run2
```

The correlations are combined to give us a prototype for the wavelet. They may be combined in different ways, the simplest being a *stack*, or average. The ideal seismic wavelet in this context is zero-phase; for our purposes this means we want the phase spectra over the dominant frequency range to be near zero. `ShowStretch` can be used to display the characteristics of the correlations and the extracted wavelet (where the wavelet is the rightmost vertical trace in the following graphic).

```
ShowStretch[dset,
  ShowUnwrapped → True, WaveletType → StackedWavelet,
 AlignPhasePicks → False, FrequencyRange → {25, 50}]
```



Frequently, the integration processor will choose an appropriate linear phase transformation by hand to make the wavelet approximately zero-phase. However, automated techniques are also available. For example, here we perform a linear fit on the phase data, which can then be used to transform the phases via a linear phase shift.

```
reg = RegressMedianPhase[dset, {15, 40}]
```

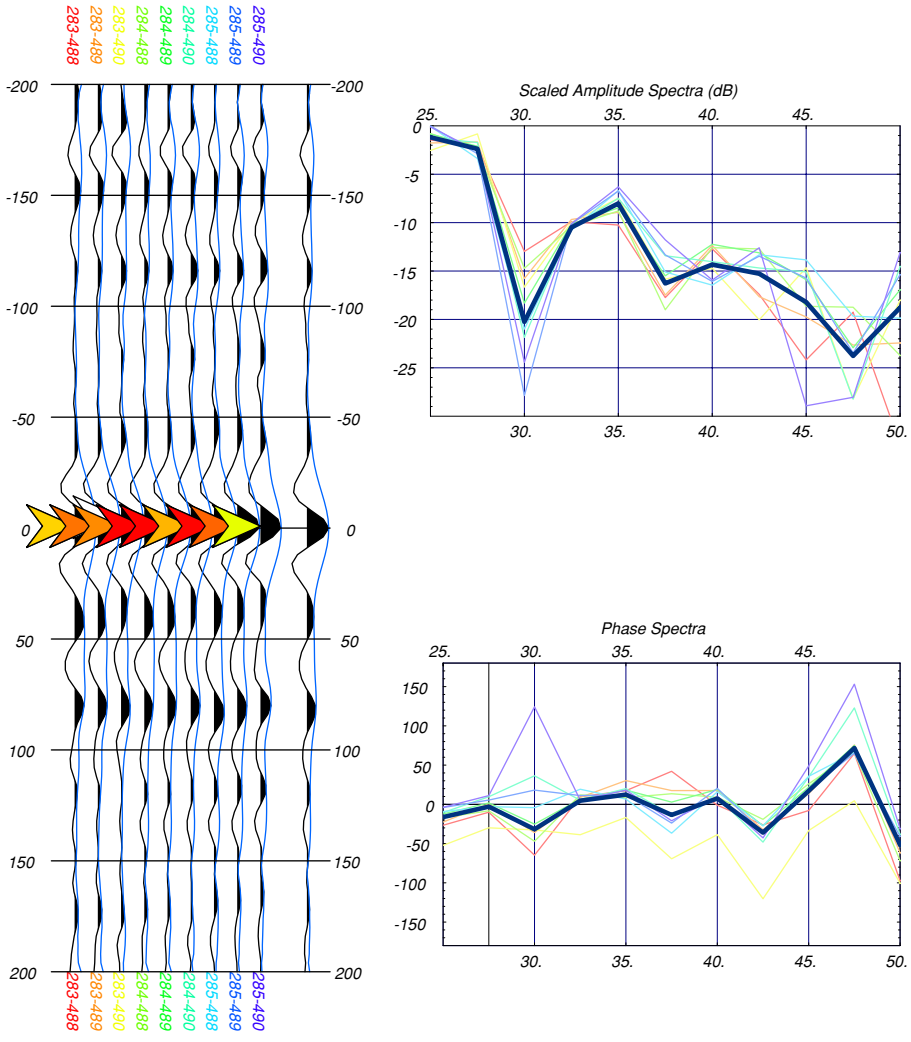
```
{0.00561417, -2.87241}
```

```
LinearPhaseShift[dset, reg]

run2
```

We can view the transformed wavelet, which now looks much closer to the desired shape with a strong central peak and balanced side lobes.

```
ShowStretch[dset, WaveletType → StackedWavelet,
 AlignPhasePicks → False, FrequencyRange → {25, 50}]
```

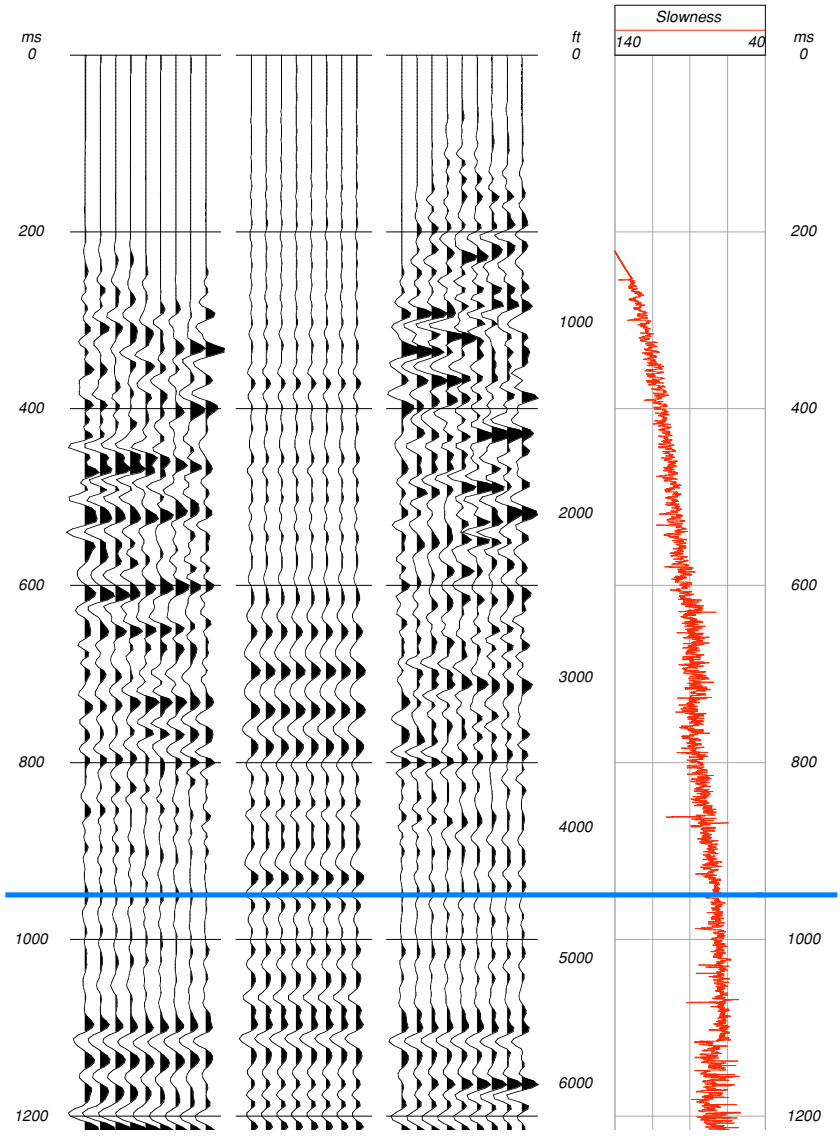


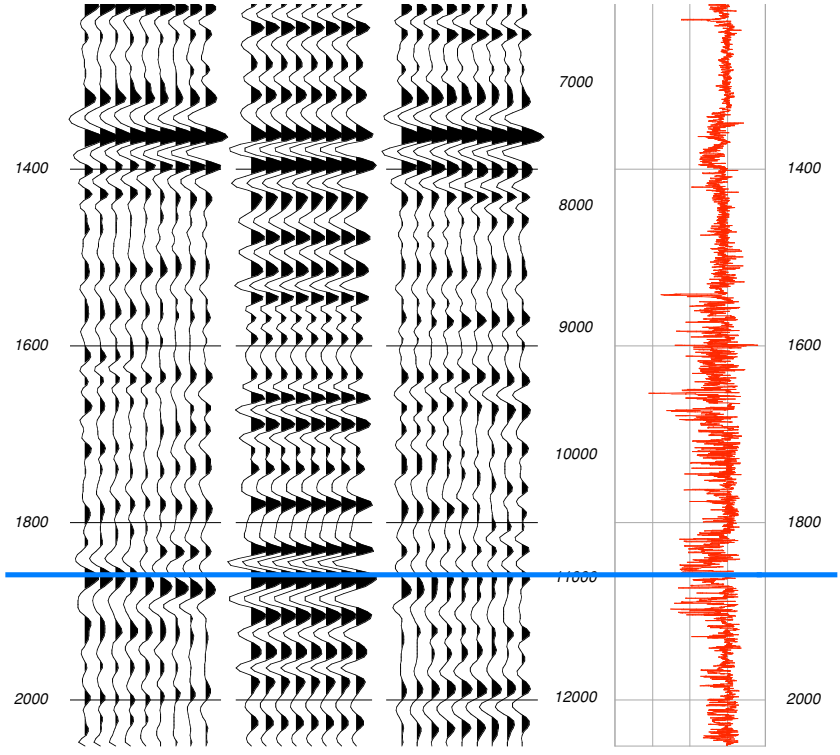
This extracted wavelet and the well logs can then be used to generate a synthetic seismic trace, which can be compared to the actual seismic data to verify the quality of the tie. If the tie is good, we would expect events (significant peaks and troughs in the time series) to be aligned between the synthetic and the real seismic. The synthetic data is generated by assuming that a sufficiently large velocity or density contrast in the rock will correspond to a reflection; thus, from

the well logs, we can generate a time series of *reflection coefficients* that, when convolved with a wavelet, should look much like the seismic data.

Seismic data is frequently collected from a single line of sensors, in which case the data is called 2D, or from a grid of sensors, where it is known as 3D. In this case, the data is 3D, but for conciseness we will only compare the synthetic to a single line of receivers via the `Show3D→False` option. Furthermore, as indicated previously, the start of the well data is often questionable, so we frequently create an artificial well log for this starting section, which can be adjusted for the alignment we derived. Here we suggest to the routines that the first 800 feet of data can be replaced by an artificial curve.

```
ShowSynthetic[dset, VelocitiesCutoff → 800, Show3D → False]
```





Once we are satisfied with the results, the extracted wavelet, synthetic seismic, and modified logs can be saved for handling elsewhere.

Naturally, the previous flow demonstrates only a few of the many options available to a user; many of these functions include numerous options to control the nature of the display or the format of the data. A large selection of transformations to perform on the data are also present, and *Mathematica*'s programming environment allows arbitrary additional manipulations as required.

■ Detailed Issues

Here are a few of the many particular design issues encountered in creating the tool suite, which may be of special interest to programmers.

□ Data Structures

One feature that an experienced *Mathematica* user will immediately notice is the use of a handle to the data (a string name), instead of the more familiar functional manipulation of lists of data. There are several reasons for this unusual design choice, the most important of which is that seismic data sets can be very large. If the data lists were returned directly by load routines and the user forgot to suppress the output, it is likely that the machine would hang for significant amounts of time as the front end attempts to format the output data. For WaveX

manipulations, this might be tens of thousands of data points; for other manipulations of seismic data, this could easily be hundreds of thousands of data points on the low end. (Data sets of hundreds of gigabytes are not uncommon, though these are not read all at once into *Mathematica*.) The use of handles in this toolset is an attempt to shield the user from this effect.

The handles also provide a convenient means of grouping associated data, forming a sort of ad-hoc database. For example, a number of functions will place intermediate results in a symbol of the form `name[`*handle*`]=`*data*, which can then be retrieved for later analysis. Over time, utilities have been evolved for this tool suite to streamline the process and allow the user to determine what data is available after operations. The data can then be easily extracted for input into more traditional *Mathematica* functions, though of course the user must then take special care to suppress large output.

For a parallel to other functions, consider the use of handles to streams (e.g., `InputStream`) or to notebooks (e.g., `NotebookObject`). While in these cases, the handle is to data stored in an external program, the basic principle is similar.

□ File Formats

There are a number of file formats for handling both seismic data and well logs. However, two particular formats are most common. The SEG-Y format for seismic data is a binary format, which typically has floating-point values stored in a rarely used IBM floating-point format, a header section using EBCDIC character encoding, and record headers in a binary integer format. The LAS format for log data, on the other hand, is an ASCII text format storing data in a simple columnar format, but with a series of headers that detail various useful sorts of log information. Each format provides its own challenges for the implementation of import/export routines.

One immediate design point that had to be considered was whether this functionality should tie into the standard functions `Import` and `Export`. In the end, this decision was driven primarily by implementation time constraints, and the simpler choice of creating separate functions for the import and export of these data types was used.

An issue regarding the reading of binary data had to do with efficiency. Seismic data sets are quite large, so care was first taken to provide mechanisms for extracting only the specified portions of the data. The use of stream manipulation capabilities via `StreamPosition` was quite valuable. Unfortunately, we are still constrained by 2 gigabyte limits in *Mathematica*'s file-handling capabilities, but there is optimism that with the increasing availability of 64-bit computing, we will soon be able to deal with larger files.

Because the trace data was stored in a nonstandard format, we could not draw directly on the `Experimental`BinaryImport` function, or on the `Utilities`BinaryFiles` standard package. Instead, a list of bytes was read in via `Read[`*stream*`, Byte]`, which could then be transformed by internal conversion routines. The functions initially partitioned the list into groups of four bytes,

which then were transformed into the corresponding floating-point numbers. A small compiled function (generated with `Compile`) was used for the transformation, which provided some speed improvement. However, it was discovered that if the flat list of input bytes were provided to a compiled function that processed a whole trace at once, a substantial performance improvement could be seen. The lesson is that compiled functions become more useful if we can move the entire processing loop inside, though unfortunately some sacrifice in modularity must be made with the current capabilities of `Compile`.

The SEG-Y and LAS formats were developed by industry organizations, but not all implementors follow the data format guidelines precisely. This meant that the routines had to be written to account for variations found in practice. This was particularly common for the LAS data format, where we encountered a number of files that were contrary to the design specification. The code had to be sufficiently modular to be easily adapted to these variations. Rule-based programming proved to be quite valuable in this context.

■ Conclusions

Mathematica's powerful programming capabilities combined with the notebook interface proved to be an effective system for performing well ties and extracting the relative wavelet, among many other geophysically-related tasks. A significant amount of development was required to start operating in the data environment, but once targeted data import, export, and visualization functionality was available, the possibilities were limited only by the researcher's imagination.

The use of *Mathematica* has expanded into many areas in our company, including well log editing, residual velocity analysis, amplitude-versus-offset analysis, and other specialized uses, as well as report preparation for clients (via HTML export) and internal research. *Mathematica* has once again proven itself to be a powerful environment for technical computation.

About the Author

John M. Novak developed a number of *Mathematica*'s standard add-on packages and applications for Wolfram Research, Inc. He now works for the AXIS Imaging Division of the seismic data processing company GX Technology (a subsidiary of Input/Output, Inc.), helping to spread the use of *Mathematica* to new arenas.

John M. Novak
GX Technology Corporation
225 E. 16th Ave. Suite 1200
Denver, CO 80203
jnovak@gxt.com

The Modular Group

A Finitely Generated Group with Interesting Subgroups

Paul R. McCreary
Teri Jo Murphy
Christan Carter

The action of Möbius transformations with real coefficients preserves the hyperbolic metric in the upper half-plane model of the hyperbolic plane. The modular group is an interesting group of hyperbolic isometries generated by two Möbius transformations, namely, an order-two element, $g_2(z) = -1/z$, and an element of infinite order, $g_\infty(z) = z + 1$. Viewing the action of the group elements on a model of the hyperbolic plane provides insight into the structure of hyperbolic 2-space. Animations provide dynamic illustrations of this action.

■ Introduction

Transformations of spaces have long been objects of study as evidenced by many of the early examples of groups. Among the most important transformations are the *isometries*, those transformations that preserve lengths. Euclidean isometries are translations, rotations, and reflections. The groups and subgroups of Euclidean isometries of the plane are so familiar to us that we may not think of them as revealing much about the space they transform. In hyperbolic space, however, light traveling or even a person traveling on a hyperbolic shortest-distance path will tend to veer away from the boundary. Thus, the geometry is unusual enough that viewing the actions of isometries of hyperbolic 2-space reveals some of the shape of that space. Two-dimensional hyperbolic space is referred to as the *hyperbolic plane*.

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

In[2]:= Show[hypPlaneUHPPict, DisplayFunction → $DisplayFunction]
```

From In[2]:=

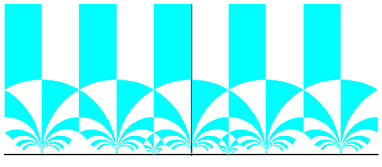


Figure 1. The upper half-plane model of the hyperbolic plane.

We will be examining how elements of the modular group rearrange the triangular-shaped regions shown in the previous diagram. The curved paths are arcs of circles orthogonal to the x-axis. In Euclidean spaces, the shortest-distance paths are straight lines. In hyperbolic space, shortest paths lie on circles that intersect the boundary of the space at right angles. Hyperbolic distances are computed in a way so that there is a "penalty" to be paid for traveling near to the plane's boundary. Thus, the shortest-distance paths (or hyperbolic geodesics) between two points must "bend" away from the boundary.

In the animations that follow, it will be instructive to focus on the action that a transformation takes on the family of circles that meet the x-axis at right angles. The transformations that we consider, namely, members of the modular group, preserve this family of circles. The circles in the family are shuffled onto different members in the family, but no new circles are created and none taken away. One could say that, in the context of hyperbolic geometry, the transformations preserve the family of all shortest-distance paths. Indeed, this is an excellent thing for an isometry transformation to do.

The following animation is an example of how a single element of the modular group acts on a cluster of regions, moving them about a common fixed point. This action will be discussed in detail in a later section. Note that increasing the integer input for the corresponding animation function will produce more frames and a smoother animation, but will also result in a longer waiting time while the frames are being created.

```
In[3]:= g3Animation[3]
```

From In[3]:=

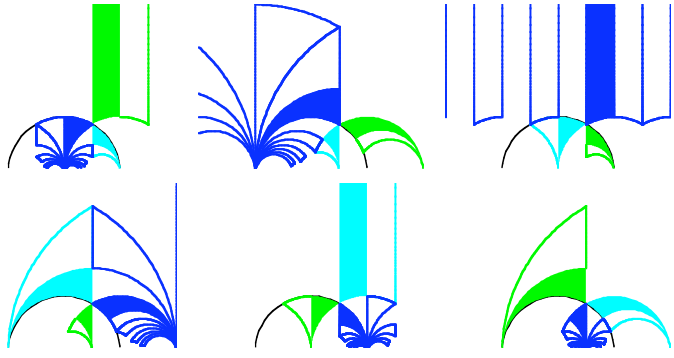


Figure 2. A single element of the modular group acting repeatedly on a cluster of regions.

Fuzzy, pixilated screen images reveal, in their own way, more than the crisp, still shots that we can obtain from the modern laser printer. The dynamic representation of the relationships yields an intuitive understanding that can easily exceed that from any number of crisp, still shots. If a picture is worth a thousand words,

then, even with limited screen resolution, an animation is worth a thousand pictures.

The context of this paper is described in Chapter 2 of Ford's *Automorphic Functions* [1]. In this small text one can find illustrations that inspired our animations. The formulas, which made coding the animations much simpler than one might expect, are given and justified in detail.

■ Möbius Transformations

We first consider a class of functions known as Möbius transformations. These transformations, named after the same mathematician with whom we associate the one-sided, half-twisted Möbius band, are one-to-one transformations on the extended complex plane $\mathbb{C}^* = \mathbb{C} \bigcup \{\infty\}$. Möbius transformations are of the following type:

$$f(z) = \frac{a\,z + b}{c\,z + d}, \qquad a, b, c, d \in \mathbb{C}, \qquad a\,d - b\,c \neq 0.$$

Over the reals, a Möbius transformation $f(z) = \frac{a\,z+b}{c\,z+d}$ with real coefficients falls into one of two categories: either $c = 0$ and the graph is a line, or $c \neq 0$ and the graph is a hyperbola. Here is a representation of this latter type of function.

```
In[4]:= Show[realHyperbolaPic, DisplayFunction → $DisplayFunction]
```

From In[4]:=

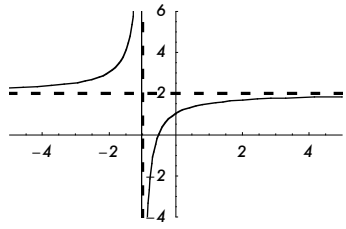


Figure 3. Graph of $f(x) = \frac{2\,x+1}{x+1}$ shown with asymptotes.

Our purpose will be to investigate how Möbius transformations stretch and twist regions in the *extended complex plane*. The *complex plane* is the usual Euclidean plane with each point identified as a complex number, namely, $(x, y) = x + i\,y$. The *extended complex plane* is formed from the complex plane by adding the point at infinity.

When a Möbius transformation, F, acts on a complex number, $F(x + i\,y) = u + i\,v$, we may view the action as "moving" the point (x, y) onto the point (u, v). We will be helped in what follows by one important fact: Möbius transformations map circles and lines in the extended plane onto circles and lines in the extended plane. A comprehensive proof of this fact may be found in most elementary texts on complex variables (e.g., *Complex Variables* by Levinson and Redheffer [2, 158]).

The extended complex plane is the domain in which the figures of our animations "live." Each point of the figure is acted on by the Möbius transformations as though the point were a complex number. These transformations spin hyperbolic 2-space about a fixed point or shift the space in one direction or another.

■ The Modular Group and Hyperbolic Space

The *modular group M* is a special class of Möbius transformations:

$$M := \left\{ f(z) = \frac{az + b}{cz + d} \;\middle|\; a, b, c, d \in \mathbb{Z},\ a\,d - b\,c = 1 \right\}.$$

That is, the modular group is the set of Möbius transformations that have integer coefficients and coefficient matrices with determinants equal to one.

What is a group? Recall that a *group* is a set G together with a binary operation $\odot$ satisfying certain properties: the set G must be closed under the operation and the operation must be associative. There must also be an identity element for the operation, and all inverses of elements in G must themselves be elements of G. The proof that the modular group is, in fact, a *group* is a standard exercise in a course on complex analysis [2, 277–8].

We will assume that the elements of the modular group do indeed form a group and investigate some of its interesting subgroups.

□ Fundamental Regions

One of our main goals is to investigate how the elements of the modular group act on fundamental regions, that is, how the regions are stretched and bent when we view them as Euclidean objects. Indeed, as hyperbolic objects, the regions are all carbon copies of each other, in much the same way that the squares on a checkerboard are all identical.

In general, a group of one-to-one transformations acting on a topological space partitions that space into *fundamental regions*. For a collection of sets $\{U_i\}$ to be a collection of fundamental regions, certain properties must hold. First and foremost, the U_i must be pairwise disjoint. Second, given any transformation f in the group, $f(U_i) \cap U_i = \varnothing$, unless f is the identity map. Finally, given any two regions U_i and U_j, there exists some transformation f, such that $f(U_i) = U_j$.

Generally, in order to cover the entire space disjointly, each fundamental region must contain some but not all of its boundary points. This is a technical matter with which we will not concern ourselves.

In fact, we will relax the above definition so that we may include *all* of the boundary points when we consider any particular fundamental region. Thus, we will allow the intersection of adjacent fundamental regions to overlap, but only on their boundaries. The essential feature remains that there is no *area* in the overlap of adjacent regions.

Note also that a group of transformations does not necessarily yield a *unique* partition of the space into fundamental regions. Thus, the following fundamental regions are merely two *representative* fundamental regions.

```
In[5]:= Show[twoFundamentalRegionsPic, DisplayFunction → $DisplayFunction]
```

From In[5]:=

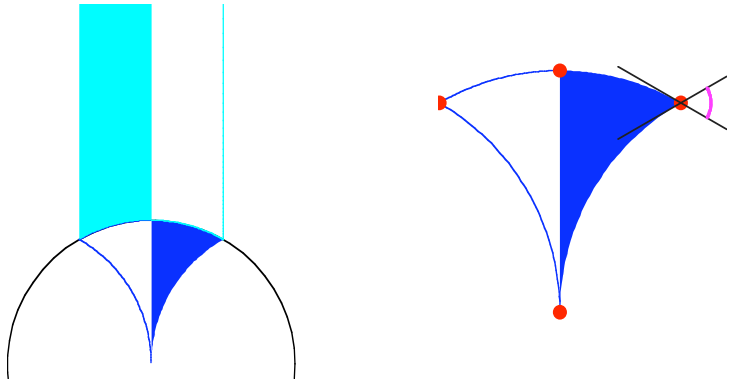


Figure 4. Two fundamental regions and a single region with vertices marked.

Each fundamental region is made up of a shaded region with an unshaded region outlined with the same color. One of the regions is enlarged on the right to identify vertices and indicate a method to measure angles. Each fundamental region contains four vertices that can be fixed (or left in place) by elements of the modular group. The illustration on the right has its four vertices identified with red dots. Tangents are drawn at one vertex to indicate how the angle measures can be determined. The left and right vertices of the region have 60-degree angles. The vertex at the top has a straight, 180-degree angle. The vertex at the bottom has a zero-degree angle because the tangents to the intersecting arcs coincide at that point. Any hyperbolic polygon with a vertex on the boundary of the space, the x-axis in this case, will have a zero angle at that vertex. The corresponding four angles in each fundamental region have the same measures as those indicated here. Each vertex can be fixed by some element in the modular group. Further, each fundamental region can be mapped onto any other fundamental region by an element of the modular group.

A classic view of the matter is to see the upper half-plane as tessellated or tiled by triangular-shaped regions, as in the first illustration of this article. A checkerboard tessellation of the Euclidean plane can be constructed by sliding copies of a square to the left, right, up, and down. Eventually the plane is covered with square tiles. The modular group tessellates the hyperbolic plane in an analogous way. The elements of the group move copies of a fundamental region until triangular-shaped tiles cover the upper half-plane model of the hyperbolic plane. Of course these tiles do not appear to be identical to our eyes that are trained to match shapes and lengths in Euclidean geometry. However, the triangular-shaped tiles are all identical if measured using the hyperbolic metric. In the tiling

process, all areas in the upper half-plane are covered by tiles and no tiles have any overlapping areas. In fact, this procedure is precisely how the hyperbolic plane illustration was constructed. The boundary points for a single fundamental region were acted on by function elements of the modular group and the resulting points were drawn as a polygon or boundary line in the illustration.

It helps to note that each transformation in M has at least one *fixed point*. Some transformations in M have two fixed points. Only the identity map has more than two. In the illustrations that follow, we will observe the placement of fixed points and the way transformations map fundamental regions near those fixed points.

□ Hyperbolic Lengths

The hyperbolic metric is a rather curious metric that challenges our notion of "distance." Under the hyperbolic metric, the shortest distance between two points is seldom along a straight line, but rather along an arc of a circle. For example, in the upper half-plane, the shortest (hyperbolic) path between points $z = i$ and $z = 1 + i$ is the top arc of the circle $\left\{z : |\, z - \frac{1}{2} \,| = \frac{\sqrt{5}}{2}\right\}$ that passes through both points and is perpendicular to the real axis.

```
In[6]:= Show[hyperbolicPathPic, DisplayFunction → $DisplayFunction]
```

From In[6]:=

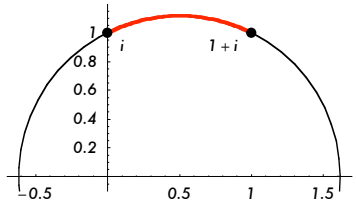


Figure 5. The shortest hyperbolic path between the points $z = i$ and $z = 1 + i$.

Without discussing precisely how hyperbolic lengths and areas are measured, we state that every image under a transformation in the modular group is *congruent* to every other image under the hyperbolic metric. Thus, all of our fundamental regions shown in the animations are actually the same "size" in the hyperbolic metric.

■ Structure of the Modular Group

We structure our investigation of the modular group by considering four cyclic subgroups. Recall that a *cyclic subgroup* can be generated by computing all powers of a single group element. The four cyclic subgroups we present are representative of the four possible types of subgroups found in the modular group.

□ Examples of Cyclic Subgroups

For our first subgroup, consider the function $g_2(z) = -1/z$, that is, a Möbius transformation with coefficients $a = 0$, $b = -1$, $c = 1$, $d = 0$. We subscript the function because g_2 is an element of order two. Thus, g_2 is its own inverse, so that $(g_2 \circ g_2)(z) = z$. In this case g_2 generates a subgroup with only two elements, namely, $M_2 = (g_2) = \{z, -1/z\}$. Note that in this article we adopt the standard notation that parentheses ( ) indicate the set of elements generated by taking all powers of the elements enclosed by the parentheses. Curly braces { }, on the other hand, enclose the delineated list of elements in the set.

Figure 6 depicts the way in which g_2 maps the two fundamental regions, shown in Figure 4, onto one another. In fact, the action of g_2 on the fundamental regions is to "rotate" them 180° onto each other about the central fixed point $z = i$. The actual mapping is performed in a single step. In particular, only the first and middle frames contain illustrations of fundamental regions. However, we have defined a sequence of intermediate mappings in order to illustrate through animation the mapping properties of g_2. In a later section we discuss how the function g_2 was broken into parts in such a way that the hyperbolic nature of the motion was preserved.

```
In[7]:= Do[Show[g2Action[nn], DisplayFunction → $DisplayFunction],
         {nn, 0, numG2 / 2}];
        Do[Show[g2Action[nn], DisplayFunction → $DisplayFunction],
         {nn, numG2 / 2, numG2}]
```

From In[7]:=

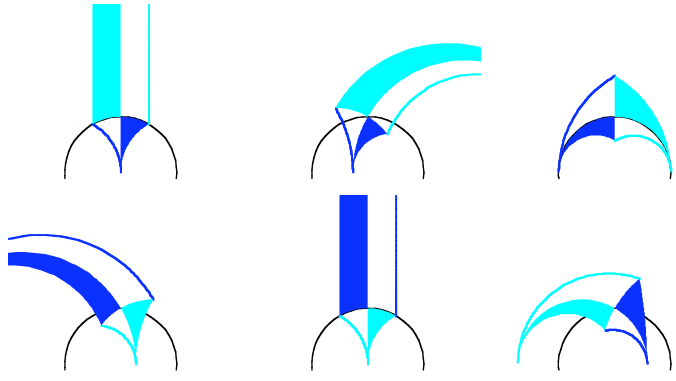


Figure 6. Action of an order-two element, $g_2(z) = \frac{-1}{z}$.

This example highlights the fact that vertical lines are paths of least distance in the upper half-plane model of the hyperbolic plane. Indeed, it is usual to view straight lines as circles that have radii with infinite length and that pass through the point at infinity. With this "bending" of the definition of circles, vertical lines have all the characteristics required of geodesics in the hyperbolic plane. They are perpendicular to the x-axis that is the boundary of the upper half-plane

model of the hyperbolic plane, and they are circles that pass through the point at infinity.

Our second example is a subgroup of infinite order generated by the linear shift $g_\infty(z) = z + 1$, that is, a Möbius transformation with $a = 1$, $b = 1$, $c = 0$, $d = 1$. The function g_∞ has infinite order, for $g_\infty^n(z) = z + n$ and no point is ever returned to its original position no matter how many times g_∞ is applied. The subgroup produced by taking all powers of g_∞ and its inverse, $g_\infty{}^{-1}(z) = z - 1$, is denoted as $M_\infty = (z + 1) = \{\ldots z - 2, z - 1, z, z + 1, z + 2 \ldots\}$. The hyperbolic isometry g_∞ is notable among the elements of the modular group because it is also a Euclidean isometry.

Every point in the plane shifts one unit to the right under the action of g_∞. The infinite half strips in the following animation are images of each other under powers of g_∞. For contrast, we also provide images of these infinite half-strip regions under the map $g_2(z) = -1/z$. Under the hyperbolic metric, the "smaller" regions in red are each congruent to the half-strip regions in blue. These images are bunched in a flower-like arrangement attached to the real axis at the origin. As the blue infinite regions are pushed from left to right, their red images echo their motion in a counterclockwise direction. Note that these two actions are not produced by a single transformation. As discussed in a later section, the two transformations that cause these actions are closely related to each other as algebraic conjugates.

```
In[8]:= Do[Show[infAction2D[nn], DisplayFunction → $DisplayFunction],
         {nn, 0, numInf}]
```

From In[8]:=

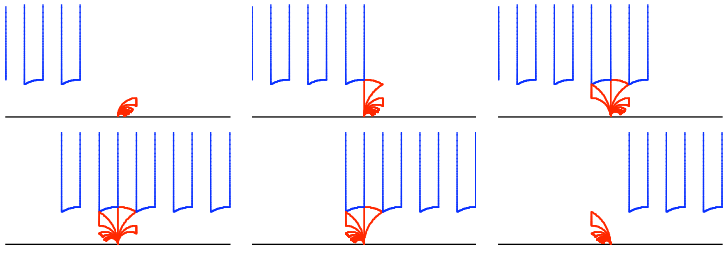


Figure 7. Copies of fundamental regions moving back and forth with corresponding regions anchored at the origin.

The third cyclic subgroup of M that we consider is generated by a composition of the first two functions. We define $g_3(x) = g_\infty \circ g_2(z) = \frac{z-1}{z}$. The subgroup generated by this element is denoted $M_3 = (g_3) = \{z, \frac{z-1}{z}, \frac{-1}{z-1}\}$, a subgroup of order three. We viewed the action of this subgroup in Figure 2. A number of fundamental regions are shown associated with the blue fundamental region attached to the origin in the animation's first frame. We include these in order to provide a better orientation for the scene. A brief pause is inserted at the end of each complete action of g_3, a rotation one-third the way around the fixed point, $\frac{1}{2} + i\,\frac{\sqrt{3}}{2}$. Of special interest is how the common point of the blue cluster moves

as the rotation takes place. The point begins at the origin and slides towards the left along the negative x-axis. When the blue lines of the cluster become vertical, this is precisely when that common point arrives at or passes through the point at infinity! All of the attached blue curves then become parallel lines that "meet at infinity." The point then slides along the positive x-axis to arrive once again and pass through the origin. It is fair to say that the motion of the point, as it passes through the origin, is a "mirror image" of the motion of the point as it passes through the point at infinity.

```
In[9]:= g3Animation[3]
```

From In[9]:=

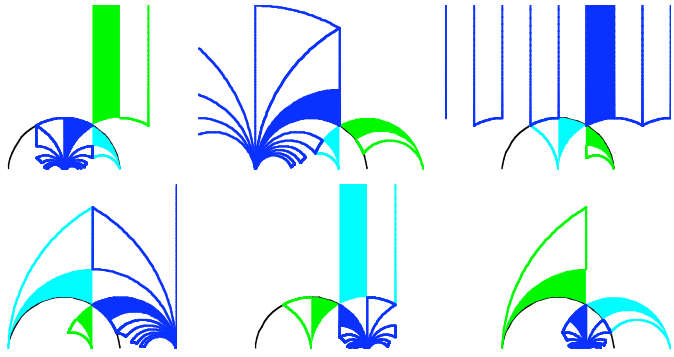


Figure 8. Action of an order-three element, $g_3(z) = \frac{z-1}{z}$.

The rotations we saw in the action of g_2 and g_3 are of order two and order three, that is, after the rotation is repeated a number of times, all points are back to their original positions. In contrast, the function g_∞ generates an infinite subgroup. When we iterate g_∞, the right shifts accumulate on the point at infinity. Points in the left half-plane get repelled by infinity, while points in the right half-plane get attracted to infinity. Of course, since all points in the left half-plane eventually map to points in the right half-plane, all points are, in some sense, simultaneously attracted to and repelled by infinity under the action of g_∞. Indeed, the point at infinity is the single fixed point for the action of g_∞. In the final section of this article we will produce a fixed point associated with the action of g_∞ that we can actually see.

There is another type of transformation in the modular group that generates an infinite subgroup M_b. This subgroup is different from M_∞ in the sense that there are two distinct fixed points—one an attractor, the other a repeller. The generator for the subgroup of our example is $g_b(z) = \frac{2z+1}{z+1}$. The following animation depicts the action of g_b on fundamental regions in the plane. Note that all points exterior to the red circle on the left are mapped to the interior of the green circle on the right. The variety of colors outlining fundamental regions makes it easier to follow individual regions as the transformation progresses. The animation begins with regions exterior to the red and green circles. These regions are all mapped to the area between the green circle and smaller yellow circle. If the

action of g_h were to be repeated, the regions would be mapped into the interior of increasingly small circles inside the smallest (yellow) circle shown. The attracting fixed point for g_h lies within these shrinking, nested circles.

```
In[10]:= Do[Show[gHAction[nn], DisplayFunction → $DisplayFunction],
          {nn, 0, numHH}];
         Show[gHAction[numHH], DisplayFunction → $DisplayFunction];
         Do[Show[gHAction[nn], DisplayFunction → $DisplayFunction],
          {nn, numHH, 0, -1}]
```

From In[10]:=

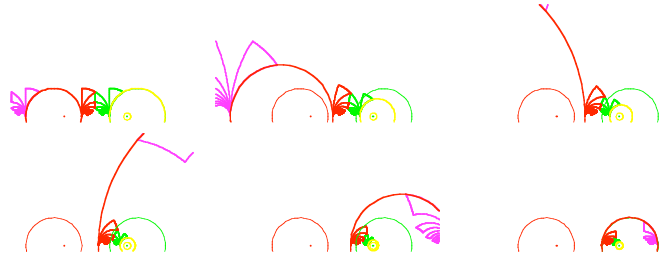


Figure 9. Action of a hyperbolic element, $g_h(z) = \frac{2z+1}{z+1}$.

The rotations and translations we have seen as examples are intimately related to Euclidean rotations and translations (see Coding Considerations). The transformation g_h is related in a similar way to a Euclidean dilation. Recall that Euclidean dilations produce figures that are similar, but not congruent. A curious characteristic of hyperbolic space is that such distinctions disappear in the hyperbolic plane. It is sufficient for figures to have the same angles to guarantee congruence. In marked contrast to Euclidean space, equal angles guarantee that corresponding side lengths are equal in the hyperbolic metric.

□ Generators of *M*

The entire group M can be generated by the two functions g_2 and g_∞, that is, $M = (g_2, g_\infty)$. Establishing this fact requires tools from linear algebra about which we will make only a few brief comments. The group GL[2, ℝ] of 2×2 matrices with real number entries and with nonzero determinants has been studied extensively and much is known about it and its subgroups. While the modular group cannot be represented as a subgroup of GL[2, ℝ], it "almost" can be. For instance, the elements $\begin{pmatrix} 1 & 2 \\ 1 & 3 \end{pmatrix}$ and $\begin{pmatrix} -1 & -2 \\ -1 & -3 \end{pmatrix}$ are considered two distinct elements in GL[2, ℝ]; however, the actions of the two associated Möbius transformation are identical because $\frac{z+2}{z+3} = \frac{-z-2}{-z-3}$. In general, for every element in the modular group there will be two elements in GL[2, ℝ] associated with it. A remarkable feature of Möbius transformations is that the group operation of compositions produces coefficients that are identical to the results of 2×2 matrix

multiplication. To see this, consider the two Möbius transformations $\frac{a z+b}{c z+d}$ and $\frac{e z+f}{g z+h}$. We first multiply the associated 2×2 matrices:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \times \begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} a e + b g & a f + b h \\ c e + d g & c f + d h \end{pmatrix}.$$

Next we carry out the composition of the two functions:

$$\frac{a \frac{e z+f}{g z+h} + b}{c e z + \frac{f}{g z+h} + d} = \frac{a (e z + f) + b (g z + h)}{c (e z + f) + d (g z + h)} = \frac{(a e + b g) z + (a f + b h)}{(c e + d g) z + (c f + d h)}.$$

The coefficients and the four matrix entries are the same!

In this way the group operation of composition of functions in the modular group can be "replaced" with the group operation of matrix multiplication in GL[2, ℝ]. It is down this path we would travel if we were to present a complete proof of the claim that the modular group is generated by the two elements g_2 and g_∞.

A major part of this claim is that any element of M can be written as a composition of g_2 and g_∞. Consider the following examples of compositions.

- $g_\infty \circ g_\infty \circ g_2(z) = \frac{2 z-1}{z}$.
- $g_\infty \circ g_\infty \circ g_2 \circ g_\infty \circ g_\infty \circ g_2(z) = \frac{3 z-2}{2 z-1}$.

Note that both of these functions have a coefficient matrix with determinant equal to one. A worthy exercise for undergraduate mathematics students is to verify by direct computations that each of the equalities hold for the indicated compositions.

■ Conjugate Subgroups

The four examples of cyclic subgroups outlined in the previous subsection give a complete description of the four types of subgroups possible in the modular group: any subgroup of the modular group is *conjugate* to a subgroup generated by g_2, g_3, an iterate of g_∞, or an iterate of an element of the same type as g_b.

Recall that in a group G, a subgroup H is *conjugate* to a subgroup K if there exists an element $g \in G$ such that the entire subgroup H can be generated by computing $g\,k\,g^{-1}$ for every $k \in K$. More compactly, we write $H = \{g\,k\,g^{-1} \mid k \in K\}$.

Consider a function f in the modular group that generates an order-three subgroup. A segment of the unit circle is included to help orient our view in the following animation.

```
In[11]:= Do[Show[unitCirclePict,
           gHActionConj2[nn], PlotRange → {{0, 2.5}, {-.1, 1.5}},
           DisplayFunction → $DisplayFunction], {nn, 0, numF5}]
```

From In[11]:=

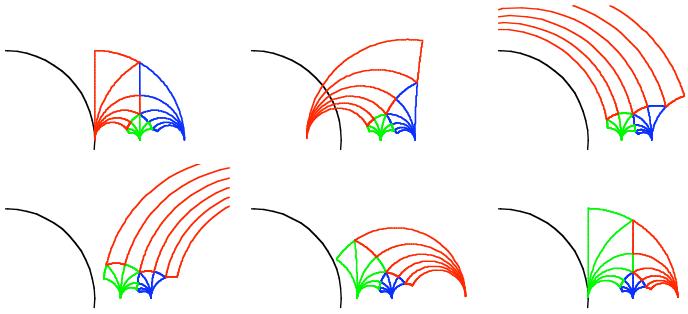


Figure 10. The action of order-three f on a selection of fundamental regions.

The subgroup generator g is conjugate equivalent to g_3, that is, there is an element $h \in M$ such that $h\, f\, h^{-1}(z) = g_3(z)$. The function h will map the fixed point of f onto $\frac{1}{2} + i\,\frac{\sqrt{3}}{2}$, the fixed point of g_3. Moreover, h will map the cluster of fundamental regions attached to the fixed point of f onto the cluster about $\frac{1}{2} + i\,\frac{\sqrt{3}}{2}$.

```
In[12]:= Do[Show[unitCirclePict, gHActionH1[numHH6 - nn],
           PlotRange → {{-1, 2.5}, {-.1, 1.5}},
           DisplayFunction → $DisplayFunction], {nn, 0, numHH6}]
```

From In[12]:=

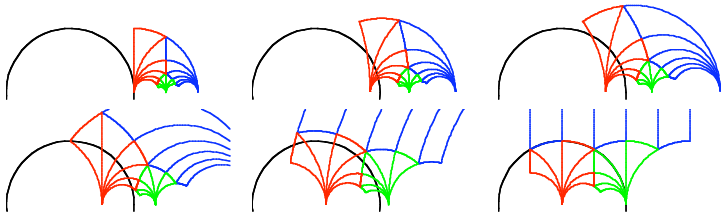


Figure 11. The action of conjugator H.

Here is a side-by-side view of the actions of f on the right and $h\, f\, h^{-1}$ on the left.

```
In[13]:= Do[Show[g3ActionChh[nn], DisplayFunction → $DisplayFunction],
          {nn, 0, numHH6 + 2 + numF5 + numHH6}]
         Show[g3ActionChh[numHH6 + 2 + numF5 + numHH6],
          DisplayFunction → $DisplayFunction]
```

From In[13]:=

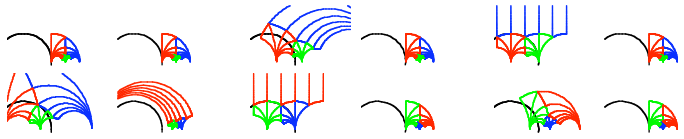
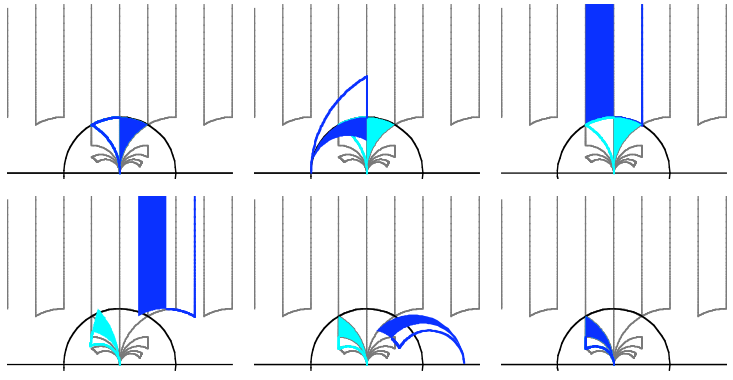


Figure 12. The action of $h\,f\,h^{-1}$ and f.

We mentioned earlier that a counterclockwise winding about the origin was associated with the left-to-right shifting of $g_\infty(z) = z + 1$. The function that carries out the winding action is $g(x) = {g_2}^{-1} \circ g_\infty \circ g_2(z) = \frac{-z}{-1+z}$. The following animation shows how this composition of g_2 and g_∞ produces the winding action. A skeleton view of the fundamental region cluster connected at the origin is shown in each frame to provide a better orientation. Once again, intermediate frames have been added to give the impression of continuous motion between the actual fundamental regions of the modular group.

```
In[15]:= Do[Show[backgroundFlower, gInfConj1[nn],
           DisplayFunction → $DisplayFunction], {nn, 0, 20}]
```

From In[15]:=



■ Coding Considerations

The code that constructs the illustrations in this notebook lies in the package file ModularGroup.m.

Construction of the animations. The action of modular group elements map the tesselation onto itself, that is, before and after pictures of the action will appear identical. The intermediate frames of the animations, which imply continuous motion rather than the abrupt jump from one configuration to another, are

produced as "fractional powers" of the group elements. They accomplish a portion of the same action on fundamental regions. The facility to produce these transformations is due to the fact that in order to produce frames for the animations, it is sufficient to compute compositions of extremely simple functions such as

Rotation: $\mathcal{R}_\theta(z) = e^{i\theta} z,\ \theta \in [0, 2\pi)$.
Scaling: $\mathcal{S}_k(z) = kz,\ k \in \mathbb{R}$.
Translation: $\mathcal{T}_a(z) = z + a,\ a \in \mathbb{C}$.

If, for example, an animation for the action of a function $g(z)$ were to have six frames from beginning to end, the points for each frame could be computed as follows.

$g_n(z) = \mathcal{A} \circ \mathcal{G}_n \circ \mathcal{A}^{-1}(z)$ where $\mathcal{A}^{-1}$ transforms the first fixed point of g to the point at ∞ and the second fixed point, if there is one, to 0. $\mathcal{A}$ transforms the points at ∞ and 0 to the fixed points of g, and $\mathcal{G}_n$ is one of the following: $\mathcal{R}_{(n/5)\,\theta}(z) = e^{i(n/5)\,\theta} z$, $\mathcal{S}_{(n/5)\,k}(z) = (n/5)\,kz$, or $\mathcal{T}_{(n/5)\,a}(z) = z + (n/5)\,a$; $n \in \{0, 1, 2, 3, 4, 5\}$. A specific example is given in the following section.

□ Conjugation and Canonical Forms

We may view the conjugation we speak of in this section as *algebraic* only if we broaden the group within which we see the conjugation taking place. Here we conjugate by elements outside the modular group, but still within the group of all Möbius transformations.

It turns out that every Möbius transformation, including every element of the modular group, is conjugate equivalent to a Euclidean rotation, translation, or dilation. Thus, for example, g_3 is conjugate equivalent to a Euclidean rotation of 120° about the origin. The conjugating transformations used in our computations are those that move fixed points to the point at infinity and to the origin. If there is a single fixed point for a group element, that fixed point is moved to the point at infinity. For a methodical discussion about Möbius transformations moving the fixed points of other transformations, see [1, 98].

```
In[16]:= Show[fourInARow, DisplayFunction → $DisplayFunction]
```

From In[16]:=

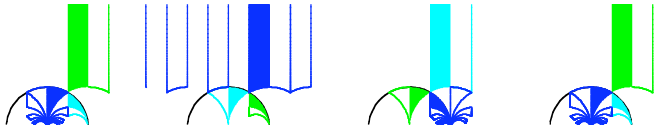


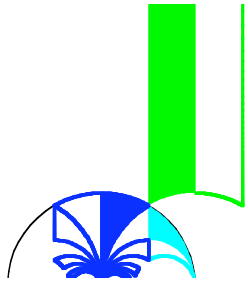
Figure 13. A set of fundamental regions (far left illustration) and the set's three images under repeated action by g_3.

Figure 13 depicts the images of three fundamental regions under the iterated action by g_3. An animation can be produced from these figures alone by using the integer 1 as input for the function `g3Animation`. However, a more revealing illustration is produced by using an integer larger than 1 that inserts intermediate

steps for the action so that the animation can suggest a continuous motion. The key to this process is the conjugate equivalence of each modular group element to a Euclidean rotation, translation, or dilation.

```
In[17]:= g3Animation[1]
```

From In[17]:=



If we permit conjugation by Möbius transformations not necessarily in M, we can relate the 2-fixed point transformations in M to "canonical" transformations —those that have fixed points at the origin and at infinity.

```
In[18]:= For[nn = 0, nn <= 25, nn++,
          Show[g3ActionC[nn], DisplayFunction → $DisplayFunction]
         ]
```

From In[18]:=

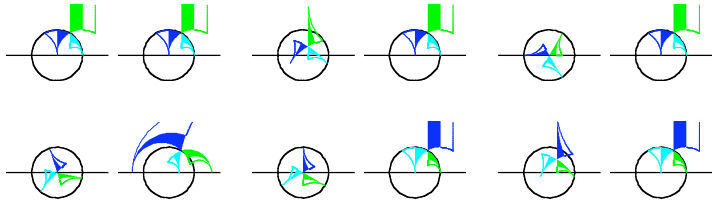


Figure 14. Conjugation with rotation of 120°.

The animation on the left shows the fixed point of g_3, that we have already seen, being translated to the origin. The second fixed point of g_3, which has remained out of our view because it lies outside of the hyperbolic plane, is simultaneously moved off to the point at infinity. Following this transformation, all circles that passed through the original fixed point become straight lines passing through the origin. A Euclidean rotation about the origin accomplishes the desired rearrangement of the regions. Finally, translating the fixed points back to their original positions maps the fundamental regions to their proper, final positions. The animation on the right shows a hyperbolic rotation about the fixed point in the upper half-plane model of the hyperbolic plane. Each frame in the right-hand animation was computed by composing the functions that are explicitly portrayed in the left-hand animation.

If we know that $f(z)$ is an element of the modular group, then there is a Möbius transformation g, such that $f = g^{-1} \circ R \circ g$, where R is a rotation of 180°, a rotation of 120°, a right shift by n units, or a dilation about the origin.

Because g_∞, the right shift by one unit, is itself an element of the modular group, the conjugation for these types of modular group elements can be accomplished within the modular group. We have seen this winding action in the previous section.

■ Applications and Extensions

In this final section we wish to reveal connections between what has already been discussed and a significant tool that is interesting in its own right, namely, stereographic projection. Further, this application in a new context may provide a deeper understanding of the modular group's action.

□ Stereographic Projection

Stereographic projection maps the extended complex plane to a sphere, called the Riemann sphere, that provides a means to see the entire complex plane all at once and supplies a point to represent the mysterious "point at infinity."

We can think of the Riemann sphere as a sphere of radius one resting on the complex plane with its "south pole" at the origin. Any nonhorizontal line through the "north pole" (or (0,0,2) in three-space) will intersect the complex plane in a single point. Conveniently, any such line will also intersect the sphere in one other point besides the north pole. We have a one-to-one correspondence between points in the plane and points on the sphere, minus the north pole. The north pole is the point at infinity! In the following animations, we color arcs in the complex plane the same as in previous animations, while arcs on the sphere are colored magenta or a dark cyan. Corresponding points on the sphere and plane are marked with large orange and cyan dots. In the animation that follows, a single track in the plane is shown in yellow and its projection on the sphere is colored blue. Two views of the same scene, one from directly above the sphere's north pole, reveal the shapes in the plane and on the sphere.

```
In[19]:= Do[Show[stereoProjAnim[nn], DisplayFunction → $DisplayFunction,
           ViewPoint → {2.357, -1.525, 1.889}], {nn, 0, numSt}]
```

From In[19]:=

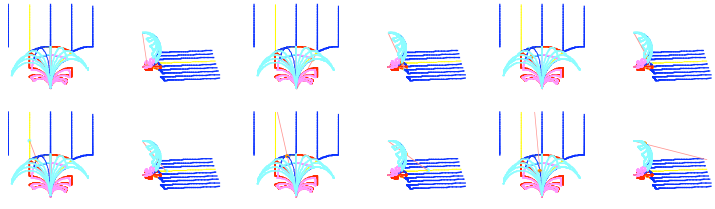


Figure 15. Stereographic projection associates points in the plane with points on the sphere.

The Riemann sphere model provides a convenient method for understanding the geometry of complex space near the point at infinity. A circle passing through the point at infinity on the Riemann sphere is merely a line in the complex plane. In the following animation, this is most clearly seen by observing the action on the stereographic projections of the blue regions that are colored in cyan on the sphere.

```
In[20]:= Do[Show[infAction3D[nn], DisplayFunction → $DisplayFunction,
           ViewPoint → {2.357, -1.525, 1.889}], {nn, 0, numInf}]
```

From In[20]:=

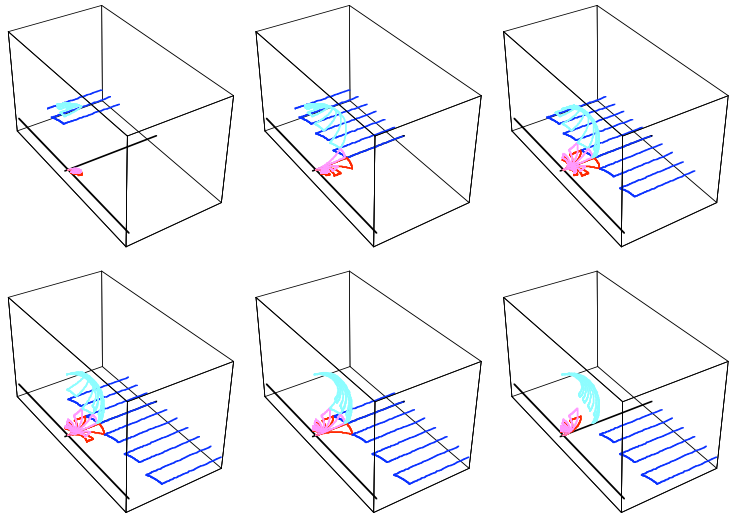


Figure 16. The action of $g_\infty(z) = z + 1$ in the half-plane and on the sphere, connected through stereographic projection.

□ Hyperbolic Geometry: Past and Present

In 1882 Poincaré helped establish the reputation of Mittag-Leffler's new journal, *Acta Mathematica*. In its inaugural issue, Poincaré announced his discovery of a model of the so-called Lobachevskian geometry, now commonly referred to as hyperbolic geometry or, popularly, as non-Euclidean geometry. More recently Bill Thurston, in work that led to his Fields Medal award, described 3-manifolds as being Euclidean, spherical, hyperbolic, or a hybrid of these three for almost all possible 3-manifolds. Jeffrey Weeks, a student of Thurston, has fascinating information about how data is being gathered over the next two years that will provide empirical evidence for the first time regarding the global shape of our universe. News updates for this project can be found at the following web address: www.geometrygames.org/weeks/ESoS/CosmologyNews.html.

□ Riemann Surfaces and the Modular Group's Name

Where did the name "modular group" come from? In general, the parameters that determine any specific Riemann surface are complex numbers and are referred to as the moduli of the surface. A Riemann surface of genus n, a donut-shaped object with n holes instead of the usual single hole, is determined by $2n - 1$ moduli (complex numbers).

Each Riemann surface of genus one (a torus or donut with one hole) is known to be equivalent to a parallelogram in the plane with opposite sides identified. A widely known version of this torus model is the flat screen for a Pac-Man video game. In the video game, when a figure departs from the screen, it simultaneously re-enters the screen from a point on the opposite side. It is as though the opposite sides were "glued together" in some magical way.

We may also assume that similar parallelograms represent equivalent surfaces. Thus, we can normalize a parallelogram so that one of its sides has unit length and lies on the x-axis between the origin and the point (1, 0). A parallelogram placed in this normalized position is completely determined by the position of its left-hand vertex lying off the x-axis. In this manner, each point in the complex plane determines a single parallelogram. Each parallelogram represents a genus-one Riemann surface (torus), once we identify opposite sides. Many different points represent parallelograms that can be cut and resewn together to be equivalent to each other, either congruent or similar. The remarkable fact here is that the points in a single fundamental region of the modular group are precisely those needed to account for every possible genus-one Riemann surface and to account for each one just once. Stated another way, each point in the fundamental region is the modulus (determining parameter) for a single genus-one Riemann surface that is distinct from all others determined by other points in the same fundamental region. This is why certain boundary points are excluded from each fundamental region in the strict definition. In other words, each boundary point determines a surface equivalent to the surface determined by a boundary point on an opposite side.

■ References

[1] L. Ford, *Automorphic Functions*, New York: McGraw-Hill, 1929.

[2] N. Levinson and R. M. Redheffer, *Complex Variables*, San Francisco: Holden-Day, 1970.

■ Additional Material

ModularGroup.m

Available at www.mathematica-journal.com/issue/v9i3/download.

Paul R. McCreary
Department of Mathematics
Xavier University of Louisiana
New Orleans, LA
pmccrear@xula.edu

Teri Jo Murphy
Department of Mathematics
University of Oklahoma
Norman, OK

Christan Carter
Department of Mathematics
Xavier University of Louisiana
New Orleans, LA

On the Beauty of Uniform Distribution Modulo One

Karl Entacher

This article provides insights into the theory of uniform distribution of sequences modulo one. Basic examples illustrate elementary concepts and special graphical presentations of (local) discrepancy, the classical measure of uniform distribution, exhibit our understanding of the beauty of this theory.

■ Introduction

The *theory of uniform distribution modulo one* was developed extensively within and among several mathematical disciplines and numerous applications, mostly in the fields of *Monte Carlo* and *quasi-Monte Carlo methods*, which include areas like numerical integration, random number generation, stochastic simulation, and approximation theory.

The central goals of this theory are the assessment of equidistribution and the construction of well-distributed point sets and sequences in various mathematical spaces.

The following sections contain several supporting ideas for introducing the theory of uniform distribution in education, offer additional information for researchers, and supplement the theory with impressive images. We start with some elementary examples. Section 2 treats *discrepancy*, which is the classical measure of uniform distribution. In Section 3 we use a special graphical presentation of local discrepancy, showing the beauty of uniform distribution. Section 4 considers further examples of point sets and the graphical visualization of the quality of their distribution.

Classical and recent concepts of the theory and further references are discussed in [1, 2]. For further information on quasi-Monte Carlo methods and their applications, see [3, 4, 5]. For efficient *Mathematica* implementations of "quasi-random numbers," see QR Stream [6]. Note that only small point sets are used for our illustrations; the number of points used in practice is substantially larger.

■ Equidistributed Point Sets

The classical goal of the theory of uniform distribution is the construction of well-distributed point sets and sequences; a classical example is the *Hammersley point set* in dimension two.

Consider integers $n,\ 0 \le n < 2^m$, with $m \in \mathbb{N}$, and let $n = n_0 + n_1\, 2 + \cdots + n_{m-1}\, 2^{m-1}$ be the binary expansion of n. The Hammersley point set in dimension $s = 2$ is defined by

$$P_0 := \{(x_n,\, y_n) = (0.\, n_{m-1}\, n_{m-2} \ldots n_0,\ 0.\, n_0\, n_1 \ldots n_{m-1}),\ 0 \le n < 2^m\}. \qquad (1)$$

The first coordinates x_n of P_0 yield the *van der Corput sequence* for $0 \le n < 2^m$, and the second coordinates y_n consecutively pass through the numbers $n/2^m$. Figure 1 exhibits the behavior of both coordinates x_n and y_n. Note that the sequence x_n orders the numbers in a balanced way in the upper and lower half of the unit interval [0, 1).

```
In[1]:= l = MyNet[4, 0];
        x = Map[First, l];
        y = Map[Last, l];
        ll = {Table[{i, ToString[i - 1]}, {i, 1, 16, 2}], Automatic};
        g1 = ListPlot[x, PlotRange → {0, 1}, Ticks → ll,
           AxesLabel → {"n", "xn"}, PlotStyle → AbsolutePointSize[2],
           DisplayFunction → Identity, AspectRatio → 1];
        g2 = ListPlot[y, PlotRange → {0, 1}, Ticks → ll,
           AxesLabel → {"n", "yn"}, PlotStyle → AbsolutePointSize[2],
           DisplayFunction → Identity, AspectRatio → 1];
        Show[GraphicsArray[{g1, g2}], DisplayFunction → $DisplayFunction]
```

From In[1]:=

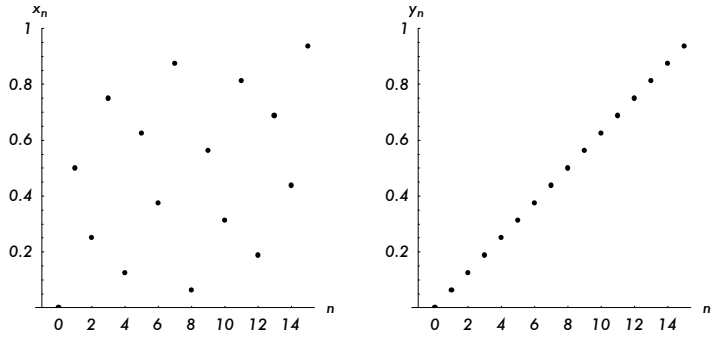


Figure 1. Behavior of the coordinates x_n and y_n of the Hammersley point set for $m = 4$.

If we truncate t bits from each coordinate of P_0, we obtain a modified point set P_t:

$$P_t := \{(0.\, n_{m-1}\, n_{m-2} \ldots n_t,\ 0.\, n_0\, n_1 \ldots n_{m-t-1}),\ 0 \le n < 2^m\}. \qquad (2)$$

Using the function `MyNet[m,t]`, the following graphics illustrate the structural behavior of P_t. Figure 2 shows plots for $m = 6$ and $t = 0, 1, 2, 3$ (from left to right).

```
In[8]:= MyNet[m_, t_] :=
         Map[FromDigits[{Drop[#, -t], 0}, 2] &, {Reverse[#], #} & /@
           Flatten[Outer[List, Sequence @@ Table[{0, 1}, {m}]], m - 1], {2}]

In[9]:= Show[GraphicsArray[ Partition[Table[Graphics[
          {AbsolutePointSize[2], Map[Point, MyNet[6, i]]},
          Frame → True, AspectRatio → 1, Axes → None,
          PlotRange → {{-0.02, 1.01}, {-0.02, 1.01}},
          FrameTicks → None, DefaultColor → Hue[0.65, 1, 0.75],
          FrameStyle → GrayLevel[0]], {i, 0, 3}], 2]]]
```

From In[9]:=

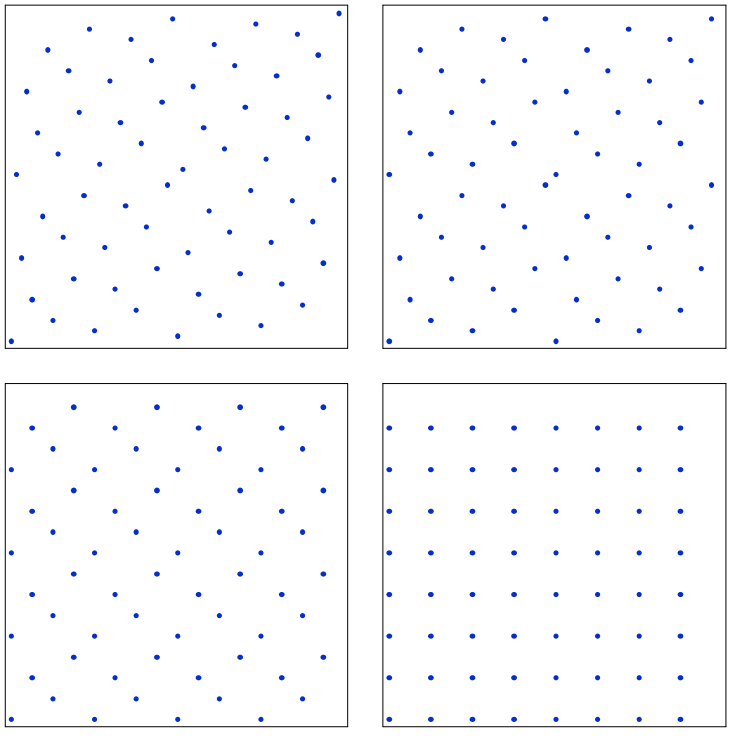


Figure 2. Hammersley point set with bit truncation for $m = 6$ and $t = 0, 1, 2, 3$.

For increasing t, the resolution of the points of P_t decreases and therefore the distribution quality decreases as well. With m even and $0 \le t \le \frac{m}{2}$, the point sets change from the well-distributed set P_0 (each grid line with resolution $1/2^m$ contains exactly one point) to the classical uniform lattice with 2^m points. In general, for the set P_t, each grid line with resolution $1/2^{m-t}$ contains exactly 2^t points (see also Figure 3).

In more modern language, the point set P_t is called a (t, m, s)-net in base 2 and dimension $s = 2$ (m is the parameter for the number of points $N = 2^m$ and t the parameter for the quality of distribution). The definition of an arbitrary (t, m, s)-net requires a slightly modified form of the distribution property described earlier: every half-open elementary s-dimensional subinterval $\mathcal{J}$ of $I = [0, 1)^s$ with volume 2^{m-t} has to contain exactly 2^t points. Figure 3 visualizes the latter requirement for P_0 and P_1, $m = 4$. Some representative elementary two-dimensional intervals $\mathcal{J}$ are indicated by colored rectangles (the lighter rectangles cover the darker ones). Note that the constant number of points within such intervals, by definition, must also be valid for all properly shifted elementary rectangles within I.

```
In[10]:= vec1 = Table[i / 2^4, {i, 1, 2^4 - 1}];
         vec2 = Table[i / 2^3, {i, 1, 2^3 - 1}];
         g1 = Graphics[{{Hue[0.65, 1, 0.75], Rectangle[{0.0, 0.0}, {1, 1 / 16}]},
                {Hue[0.65, 0.9, 0.75], Rectangle[{0.0, 0.0}, {1 / 16, 1}]},
                {Hue[0.65, 0.6, 0.75], Rectangle[{0.0, 0.0}, {1 / 2, 1 / 8}]},
                {Hue[0.65, 0.4, 0.75], Rectangle[{0.0, 0.0}, {1 / 8, 1 / 2}]},
                {Hue[0.65, 0.2, 0.75], Rectangle[{0.0, 0.0}, {1 / 4, 1 / 4}]},
            {AbsolutePointSize[3], Map[Point, MyNet[4, 0]]}},
           {AspectRatio → 1,  Frame → True, Axes → None, FrameTicks → None,
            PlotRange → {{0.0, 1.0}, {0.0, 1.0}}, GridLines → {vec1, vec1}}];
         g2 = Graphics[{
                {Hue[0.65, 1, 0.75], Rectangle[{0.0, 0.0}, {1, 1 / 8}]},
                {Hue[0.65, 0.9, 0.75], Rectangle[{0.0, 0.0}, {1 / 8, 1}]},
                {Hue[0.65, 0.6, 0.75], Rectangle[{0.0, 0.0}, {1 / 2, 1 / 4}]},
                {Hue[0.65, 0.4, 0.75], Rectangle[{0.0, 0.0}, {1 / 4, 1 / 2}]},
            {AbsolutePointSize[3], Map[Point, MyNet[4, 1]]}},
           {AspectRatio → 1, Frame → True, Axes → None, FrameTicks → None,
            PlotRange → {{0.0, 1.0}, {0.0, 1.0}}, GridLines → {vec2, vec2}}];
         Show[GraphicsArray[{g1, g2}]]
```

From In[10]:=

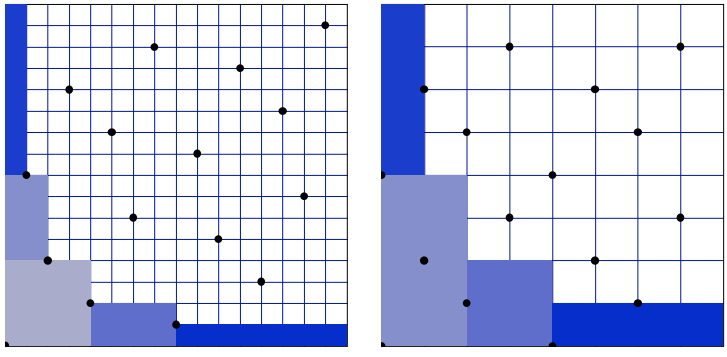


Figure 3. Visualization of the (t, m, s)-net property for P_0 and P_1, $m = 4$.

A modern task in the theory of uniform distribution is the construction of arbitrary (t, m, s)-nets with small parameters t in large dimensions s.

■ Discrepancy

In the previous section, the term equidistributed point set was used without defining equidistribution. An assessment of distribution quality needs a suitable measure. The classical measure in the theory of uniform distribution is *discrepancy*, which in a certain sense measures the deviation of a point set from an ideal distribution.

We will introduce a special form of discrepancy, the so-called *star-discrepancy*. Consider a point set $P = (x_n)_{n=0}^{N-1}$, in the s-dimensional unit cube $I = [0, 1)^s$. The star-discrepancy $D_N^*(P)$ of P is defined as

$$D_N^*(P) := \sup_{\mathcal{J} \in Y^*} D(\mathcal{J}, P), \text{ where } D(\mathcal{J}, P) := \left| \frac{\text{number of } x_n \in \mathcal{J}}{N} - \text{Vol}(\mathcal{J}) \right|, \quad (3)$$

and Y^* denotes the class of all multidimensional subintervals $\mathcal{J}$ of I of the form $\mathcal{J} := \prod_{i=1}^{s} \mathcal{J}_i$, where $\mathcal{J}_i = [0, u_i)$ and $0 \le u_i \le 1$. Hence, the star-discrepancy is attained for a (half-open) subinterval of I with the left corner in the origin (see Figure 3 for some *discrete* examples $\mathcal{J} = [0, u_1) \times [0, u_2)$) providing maximal deviation of the relative number of points within the interval from its volume $\text{Vol}(\mathcal{J}) = \prod_{i=1}^{s} u_i$. The number $D(\mathcal{J}, P)$ is called *local discrepancy*.

The measure $D_N^*(P)$ may also be seen as a number-theoretical adaptation of a well-known goodness-of-fit test in statistics, the two-sided Kolmogorov–Smirnov test that is used to test the hypothesis that a sample P stems from a particular continuous probability distribution. In our case, the target distribution is the uniform distribution on I.

The computation of discrepancy is very time consuming with a complexity of $O(N^s)$ for N points in the s-dimensional unit cube. Hence, an assessment of uniform distribution is usually achieved by theoretical discrepancy estimates or by an application of several other measures of equidistribution, which, although they are easier to compute, are sometimes not that intuitive.

Furthermore, every numerical application of uniformly distributed point sets works with finite precision numbers. Therefore, it is important to apply discrepancy for finite precision subintervals (see Figure 3 for some examples). This leads to the concept of *discrete discrepancy* [3], which is defined in the same way as in equation (3), but now Y^* denotes the class of all (discrete) subintervals $\mathcal{J}$ of the form $\mathcal{J} = \prod_{i=1}^{s} \mathcal{J}_i$, where $\mathcal{J}_i = [0, a_i / M)$, with integers $0 \le a_i \le M$.

The `DiscreteDiscrepancy[p,m]` function gives a list of local discrepancies $D(\mathcal{J}, P)$ for all discrete intervals $\mathcal{J}$ with resolution $M = 2^m$, $m \ge 2$ and a given point set P. To calculate this function, we represent the intervals $\mathcal{J}$ by the list of endpoints $\left(\frac{i}{2^m}, \frac{j}{2^m}\right)$, $1 \le i, j \le 2^m$, and use simple list operations.

```
In[15]:= DiscreteDiscrepancy[p_List, m_Integer] :=
           Module[{intervals, func},
            intervals = Table[{i, j}/2^m, {i, 1, 2^m}, {j, 1, 2^m}];
            intervals = Flatten[intervals, 1];
            func[l_] := Length[Select[p,
               (#[[1]] < l[[1]] && #[[2]] < l[[2]]) &]];
            Abs[Map[func, intervals]/Length[p] -
              Map[Apply[Times, #] &, intervals]]];
```

The following matrix shows the output of `DiscreteDiscrepancy[`P_1`,3]`. The value at position (i, j) equals local discrepancy $D(\mathcal{J}, P)$ for the interval $\mathcal{J}$ with endpoints $\left(\frac{i}{2^m}, \frac{j}{2^m}\right)$, $1 \le i, j \le 2^m$.

```
In[16]:= Example = DiscreteDiscrepancy[MyNet[3, 1], 3];
         MatrixForm[Partition[Example, 8]]
```

Out[17]//MatrixForm=

$$\begin{pmatrix}
\frac{7}{64} & \frac{3}{32} & \frac{5}{64} & \frac{1}{16} & \frac{11}{64} & \frac{5}{32} & \frac{9}{64} & \frac{1}{8} \\
\frac{3}{32} & \frac{1}{16} & \frac{1}{32} & 0 & \frac{3}{32} & \frac{1}{16} & \frac{1}{32} & 0 \\
\frac{5}{64} & \frac{1}{32} & \frac{7}{64} & \frac{1}{16} & \frac{9}{64} & \frac{3}{32} & \frac{11}{64} & \frac{1}{8} \\
\frac{1}{16} & 0 & \frac{1}{16} & 0 & \frac{1}{16} & 0 & \frac{1}{16} & 0 \\
\frac{11}{64} & \frac{3}{32} & \frac{9}{64} & \frac{1}{16} & \frac{15}{64} & \frac{5}{32} & \frac{13}{64} & \frac{1}{8} \\
\frac{5}{32} & \frac{1}{16} & \frac{3}{32} & 0 & \frac{5}{32} & \frac{1}{16} & \frac{3}{32} & 0 \\
\frac{9}{64} & \frac{1}{32} & \frac{11}{64} & \frac{1}{16} & \frac{13}{64} & \frac{3}{32} & \frac{15}{64} & \frac{1}{8} \\
\frac{1}{8} & 0 & \frac{1}{8} & 0 & \frac{1}{8} & 0 & \frac{1}{8} & 0
\end{pmatrix}$$

Discrete discrepancy is the maximum of the matrix, $\frac{15}{64}$. In our example, the maximum is attained for two intervals $\mathcal{J}$ with endpoints $(\frac{5}{8}, \frac{5}{8})$ and $(\frac{7}{8}, \frac{7}{8})$. As one may guess, calculating `DiscreteDiscrepancy[`p`,`m`]` in this simple way becomes very difficult for increasing m. However, we can use this function to analyze the structure of local discrepancy for small point sets in order to gain elementary information for a subsequent theoretical discrepancy estimation of larger point sets. This very useful strategy allowed the author to construct a method to estimate the star-discrepancy of arbitrary nets P_t [7].

■ Visualizations

In this section, we visualize the structure of local discrepancy $D(\mathcal{J}, P)$ for our point sets P_t, $t \ge 0$, for all discrete intervals $\mathcal{J}$ with resolution $M = 2^m$. We start with a visualization in dimension two. Therefore, we use `DiscreteDiscrepancy[ ]` and calculate the list of values $D(\mathcal{J}, P)$ for all such two-dimensional intervals, which are represented by their endpoints as described earlier. The following function simply uses `RasterArray[ ]` to color the list of local discrepancies. Small values of $D(\mathcal{J}, P)$ are colored in dark blue. The larger the values become, the brighter the coloring. The brightest squares show where the maximum (and therefore the discrete discrepancy) is attained. The explicit value is shown above each plot.

```
In[18]:= Visualize2D[p_List, m_Integer] := Module[{mi, ma, result},
           result = DiscreteDiscrepancy[p, m];
           mi = Min[result];
           ma = Max[result];
           Graphics[{RasterArray[Partition[
              Map[RGBColor[#, 0.1 + 0.9 #, 0.6] &,
                   result / (ma - mi)], 2^m]]},
           AspectRatio → 1, PlotRange → All,
           PlotLabel → StyleForm[TraditionalForm[ma]]] ];
```

Figure 4 shows `Visualize2D[`p`,`m`]` for P_t, $t = 0, 1, 2, 3$ and $m = 6$. Compared to Figure 2, these graphics reflect the change of distribution in a more impressive way.

```
In[19]:= Show[GraphicsArray[
          {{Visualize2D[MyNet[6, 0], 6], Visualize2D[MyNet[6, 1], 6]},
           {Visualize2D[MyNet[6, 2], 6], Visualize2D[MyNet[6, 3], 6]}}]]
```

From In[19]:=

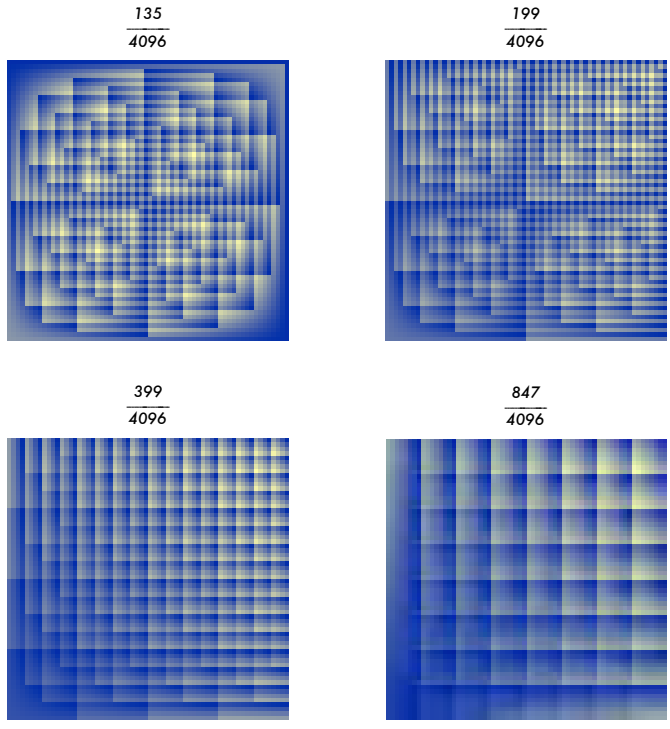


Figure 4. Visualization of local discrepancy for P_t, $t = 0, 1, 2, 3$ and $m = 6$.

Figure 5 also includes the points within the graphic for the larger case $m = 8$ and $t = 0$. A comparison of the first graphic in Figures 4 and 5 shows certain self-similarities. The reader is also asked to visualize the cases $m = 9, 10$.

```
In[20]:= g = Graphics[ {AbsolutePointSize[2],
            Map[Point, 2^8 MyNet[8, 0]]},
          Frame → None, AspectRatio → 1, Axes → None, FrameTicks → None];
         Show[Visualize2D[ MyNet[8, 0], 8], g]
```

From In[20]:=

$$\frac{711}{65536}$$

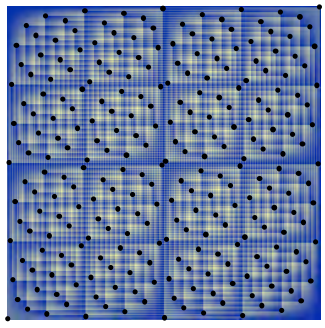


Figure 5. Visualization of local discrepancy of P_0 for $m = 8$.

In Figure 6, the distribution behavior of the point sets is visualized in three dimensions as well. The function `Visualize3D[p,m]` generates a `Graphics3D` object containing cuboids with heights corresponding to $D(\mathcal{J}, P)$. Figures 6 and 7 show the cases $m = 6$, $t = 0$, and $t = 3$. Two of the plots in Figure 4 are projections into the x-y plane of Figures 6 and 7.

```
In[22]:= Visualize3D[p_List, m_Integer] :=
          Module[ {mi, ma, result, tmp1, tmp2, pos, l, ll},
           l = Table[{i, j} / 2^m, {i, 1, 2^m}, {j, 1, 2^m}];
           l = Flatten[l, 1];
           result = DiscreteDiscrepancy[p, m];
           mi = Min[result]; ma = Max[result];
           tmp1 = (ma - mi) / 2^m; tmp2 = Union[result];
           pos = Table[Flatten[ Position[result, tmp2[[i]]]],
             {i, 1, Length[tmp2]} ];
           ll =
            Table[ {Flatten[{l[[pos[[i]][[j]]]], tmp2[[i]]}],
               Flatten[{l[[pos[[i]][[j]]]] + 1 / 2^m, tmp2[[i]] + tmp1}]},
             {i, 1, Length[tmp2]}, {j, 1, Length[pos[[i]]]} ];
           Table[ Graphics3D[{RGBColor[
               (i - 1) / Length[ll], (i - 1) / Length[ll], 0.7],
              PointSize[0.012], EdgeForm[Thickness[0.001]],
              Map[Cuboid[#[[1]], #[[2]]] &, ll[[i]]]},
             Ticks → {{1}, {0, 1}, {mi, ma}},
             PlotRange → {mi - 2 tmp1, ma + 2 tmp1},
             Boxed → True, BoxRatios → {1, 1, 1},
             ViewPoint → {-1.3, -1.4, 1.9}, Lighting → False,
             Axes → True],
            {i, 1, Length[ll]}] ];
```

```
In[23]:= Show[Visualize3D[MyNet[6, 0], 6]]
```

From In[23]:=

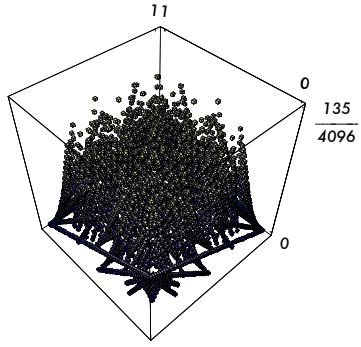


Figure 6. Three-dimensional visualization of local discrepancy of P_0 for $m = 6$.

```
In[24]:= Show[Visualize3D[MyNet[6, 3], 6]]
```

From In[24]:=

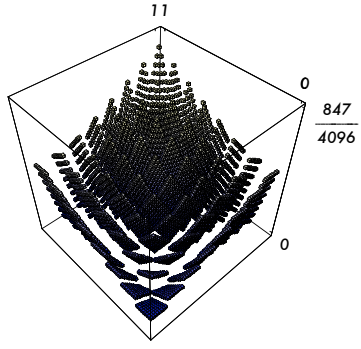


Figure 7. Three-dimensional visualization of local discrepancy of P_3 for $m = 6$.

■ More Point Sets

Based on the theory of uniform distribution, many different point sets and sequences have been developed or studied in detail. In this section, we show further examples and visualize the behavior of their local discrepancy with our graphics tools.

The *Halton sequence* in dimension two differs from the Hammersley sequence only in that the second coordinate in equation (1) uses the ternary expansion for n instead of the binary expansion. The following graphics show a Halton sequence $x_n \in [0, 1)^2$ for $1 \le n \le 2^7$ and its visualization of local discrepancy.

```
In[25]:= m = 7;
         vec1 = Reverse[Table[1/2^i, {i, 1, m}]];
         vec2 = Reverse[Table[1/3^i, {i, 1, m}]];
         l = Range[2^m];
         l1 = Map[IntegerDigits[#, 2, m] &, l].vec1;
         l2 = Map[IntegerDigits[#, 3, m] &, l].vec2;
         hal = Transpose[{l1, l2}];
         g = Graphics[{PointSize[0.02], Map[Point, hal]},
            Frame → True, AspectRatio → 1, Axes → None,
            FrameTicks → None, PlotRegion → {{0, 0.85}, {0, 0.85}}];
         Show[GraphicsArray[{g, Visualize2D[hal, m]}, GraphicsSpacing → 0]]
```

From In[25]:=

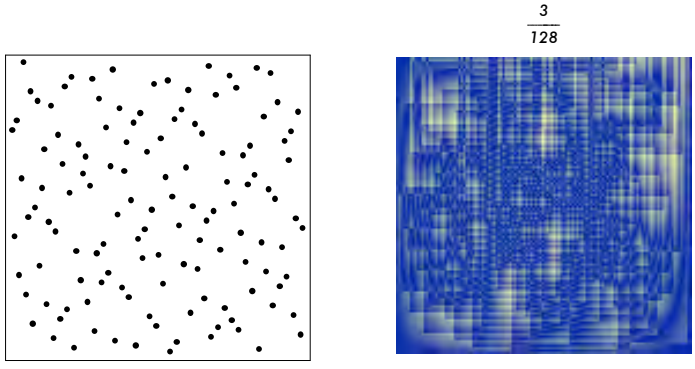


Figure 8. Local discrepancy of a Halton sequence ($m = 7$).

Lattices are essentially different from the nets shown in earlier sections. Special lattices are obtained if we produce all "overlapping" vectors (x_n, x_{n+1}) from a point set $P = (x_n)_{n=0}^{M-1}$, where the numbers x_n are generated with $x_n := y_n / M$ and y_n via the recurrence $y_{n+1} \equiv a\, y_n + b \pmod{M}$. The parameters a and b, $0 \le a, b < M$, and the starting value y_0 have to fulfill certain conditions in order to get the full period M for the recurrence (see [3, 8]). The parameter a is responsible for the distribution quality of the lattice. If a is chosen well, the vector $g = (1, a)$ is called a good lattice point, because the corresponding lattice is generated from multiples of g. In Figures 9 and 10 the modulus $M = 2^7$ and the additive constant $b = 1$. In Figure 9 we use the bad parameter $a = 125$ and in Figure 10 the good parameter $a = 117$.

```
In[34]:= m = 7;
         f[x_] := Mod[125 x + 1, 2^m];
         gen = NestList[f, 0, 2^m];
         gen = Partition[gen / 2^m, 2, 1];
         g = Graphics[ {PointSize[0.02], Map[Point, gen]},
            Frame → True, AspectRatio → 1, Axes → None,
            FrameTicks → None, PlotRegion → {{0, 0.85}, {0, 0.85}}];
         Show[GraphicsArray[{g, Visualize2D[gen, m]}, GraphicsSpacing → 0 ]]
```

From In[34]:=

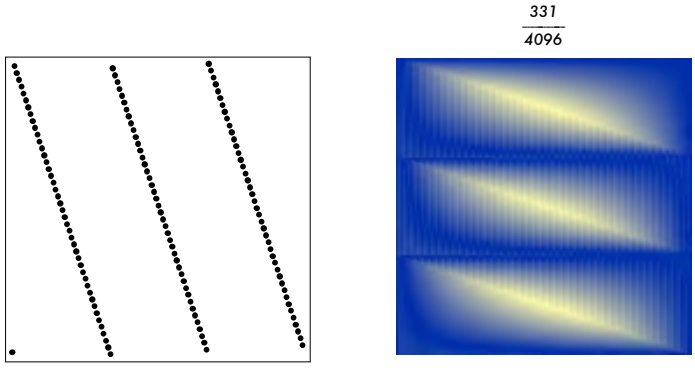


Figure 9. Local discrepancy of bad lattice points with $M = 2^7$ and $a = 125$.

```
In[40]:= m = 7;
         f[x_] := Mod[117 x + 1, 2^m];
         gen = NestList[f, 0, 2^m];
         gen = Partition[gen / 2^m, 2, 1];
         g = Graphics[ {PointSize[0.02], Map[Point, gen]},
            Frame → True, AspectRatio → 1, Axes → None, FrameTicks → None,
            PlotRegion → {{0, 0.85}, {0, 0.85}}];
         Show[GraphicsArray[{g, Visualize2D[gen, m]},
           GraphicsSpacing → 0]]
```

From In[40]:=

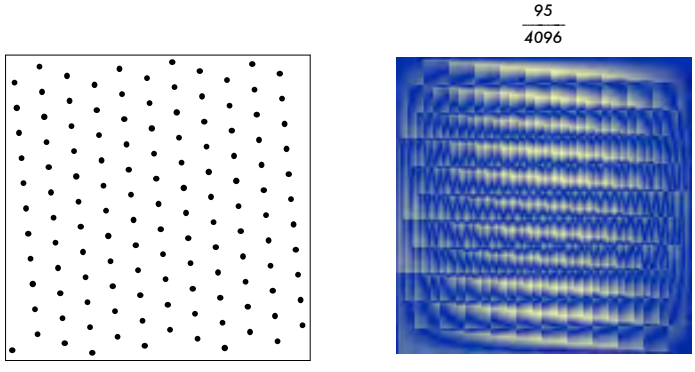


Figure 10. Local discrepancy of good lattice points with $M = 2^7$ and $a = 117$.

The weak distribution for $a = 125$ is clearly seen in the visualization of local discrepancy. The good lattice is more dispersed, like Figure 5. The reader is asked to try to compute the three-dimensional visualization of the bad lattice.

There is another important application of such lattices. The modular generation method yields a classical method for the generation of (*pseudo-*) *random numbers*. If one uses only a small sample of numbers from the previous recurrence with large modulus M (sample size lower than the square root of modulus M recommended), then the numbers behave like "real" random numbers in many applications. The generation method is called the *linear congruential generator* (LCG). Figure 11 visualizes a sample of $N = 2^6$ numbers from a widely used LCG called the "Minimal Standard" generator ($M = 2^{31} - 1$, $a = 16807$, $b = 0$) [9] (for further references see [10]).

```
In[46]:= m = 6;
         f[x_] := Mod[16807 x, 2^31 - 1];
         gen = NestList[f, 1, 2^m];
         gen = Partition[gen / (2^31 - 1), 2, 1];
         g = Graphics[{PointSize[0.02], Map[Point, gen]},
            Frame → True, AspectRatio → 1, Axes → None,
            FrameTicks → None, PlotRegion → {{0, 0.85}, {0, 0.85}}];
         Show[GraphicsArray[{g, Visualize2D[gen, m]},
           GraphicsSpacing → 0]]
```

From In[46]:=

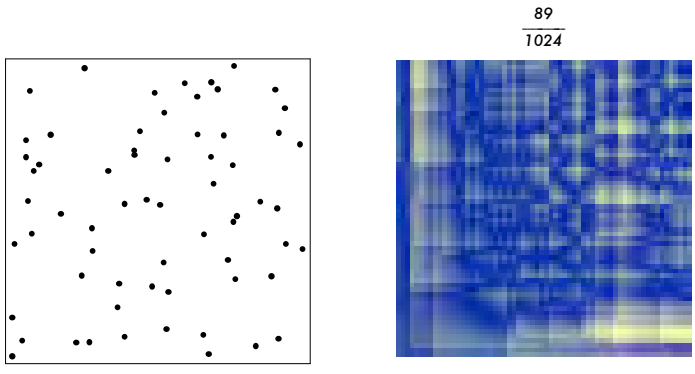


Figure 11. Local discrepancy of a sample from the Minimal Standard random number generator.

Linear congruential pseudorandom number generators have often been criticized for the underlying lattice structures they produce if all overlapping vectors are considered. Therefore, several other (nonlinear) generators have been proposed where overlapping vectors generated from such random numbers are not contained in lattices (or small unions of lattices). Thus, we consider a "baby" version and a more recent nonlinear generator, called the *explicit inversive congruential generator* (EICG), defined by Eichenauer-Herrmann [11]. The reader is asked to recover the modular generation method by means of the following *Mathematica*

implementation. From the baby generator with modulus $M = 2^7 - 1$, we produce all possible overlapping vectors (Figure 12), and from a large EICG with modulus $M = 2^{31} - 1$ and the same multiplier 117, we generate a sample of 2^7 numbers (Figure 13).

```
In[52]:= m = 7;
         p = 2^7 - 1;
         f[n_] := PowerMod[117 n, p - 2, p];
         gen = Table[f[n], {n, 1, 2^m + 1}];
         gen = Partition[gen / (2^7 - 1), 2, 1];
         g = Graphics[ {PointSize[0.02], Map[Point, gen]},
            Frame → True, AspectRatio → 1, Axes → None,
            FrameTicks → None, PlotRegion → {{0, 0.85}, {0, 0.85}}];
         Show[GraphicsArray[{g, Visualize2D[gen, m]}, GraphicsSpacing → 0]]
```

From In[52]:=

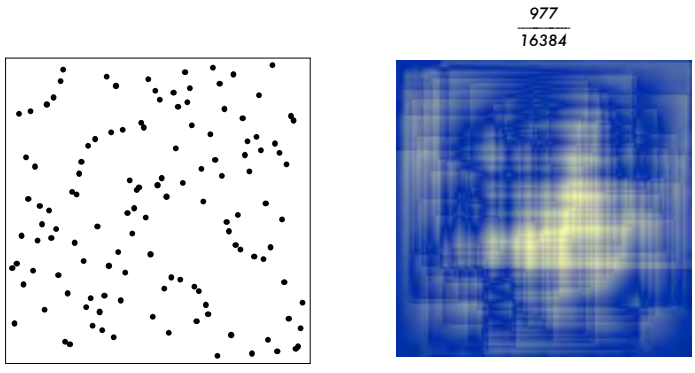


Figure 12. Local discrepancy of a baby EICG with $M = 2^7 - 1$ and $a = 117$.

```
In[59]:= m = 7;
         p = 2^31 - 1;
         f[n_] := PowerMod[117 n, p - 2, p];
         gen = Table[f[n], {n, 1, 2^m + 1}];
         gen = Partition[gen / (2^31 - 1), 2, 1];
         g = Graphics[ {PointSize[0.02], Map[Point, gen]},
            Frame → True, AspectRatio → 1, Axes → None,
            FrameTicks → None, PlotRegion → {{0, 0.85}, {0, 0.85}}];
         Show[GraphicsArray[{g, Visualize2D[gen, m]}, GraphicsSpacing → 0]]
```

From In[59]:=

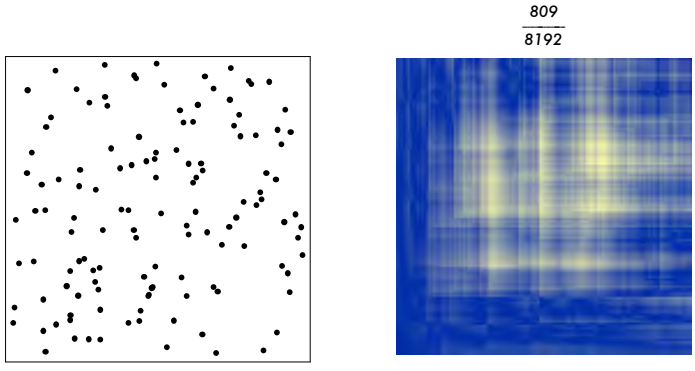


Figure 13. Local discrepancy of a small sample of an EICG with $M = 2^{31} - 1$ and $a = 117$.

■ Summary

In this article we showed some basic principles from the theory of uniform distribution of sequences modulo one using *Mathematica* functions and graphics. The functions can easily be used or extended for introductory lessons in this field. The graphical representation of the structural behavior of local discrepancy provides additional information on the behavior of discrepancy for different point sets and impressive images. These images may also be used to motivate further distribution measures such as the L^2-discrepancy.

■ Acknowledgments

This research was supported by the Austrian Science Fund (FWF) Grants S8303-MAT and S8311-MAT. The author would like to thank the editors and the anonymous referees for their comments and suggestions to improve the quality of the article.

■ References

[1] L. Kuipers and H. Niederreiter, *Uniform Distribution of Sequences*, New York: John Wiley, 1974.

[2] M. Drmota and R. F. Tichy, *Sequences, Discrepancies and Applications*, Vol. 1651 of *Lecture Notes in Mathematics*, Berlin: Springer, 1997.

[3] H. Niederreiter, *Random Number Generation and Quasi-Monte Carlo Methods*, Philadelphia: SIAM, 1992.

[4] H. Niederreiter et al. (eds.), *Monte Carlo and Quasi-Monte Carlo Methods 1996, 1998, 2000, 2002*, The series of proceedings for the conferences MC&QMC 1996–2002, Springer-Verlag.

[5] P. Hellekalek and G. Larcher, *Random and Quasi-Random Point Sets*, Vol. 138 of *Lecture Notes in Statistics*, Berlin: Springer, 1998.

[6] MathConsult, "*MathDirect* Mathematical Software." (Jan 22, 2000) www.mathdirect.com/products.

[7] K. Entacher, "Discrepancy Estimates Based on Haar Functions," *Mathematics and Computers in Simulation*, **55**, 2001 pp. 49–57.

[8] D. E. Knuth, *The Art of Computer Programming,* Vol. 2, *Seminumerical Algorithms*, 3rd ed., Boston: Addison-Wesley, 1997.

[9] S. K. Park and K. W. Miller, "Random Number Generators: Good Ones Are Hard to Find," *Communications of the Association for Computing Machinery*, **31**, 1988 pp. 1192–1201.

[10] B. Hechenleitner and K. Entacher, "*The Spectral Test Server*," (Sep 1, 2004) spectral.fh-sbg.ac.at.

[11] J. Eichenauer-Herrmann, "Statistical Independence of a New Class of Inversive Congruential Pseudorandom Numbers," *Mathematics of Computation*, **60**, 1993 pp. 375–384.

About the Author

Karl Entacher received his B. S., M. S., and Ph.D. degrees in mathematics from the University of Salzburg and completed his Habilitation on Scientific Computing at the same university. He currently holds a faculty position in the School of Telecommunications Engineering at Salzburg University of Applied Sciences and Technologies. His research interests include simulation, Monte Carlo and quasi-Monte Carlo methods, random number generation, parallel random numbers, and low-discrepancy point sets.

Karl Entacher
School of Telecommunications Engineering
Salzburg University of Applied Sciences & Technologies
Schillerstrasse 30
A-5020 Salzburg
Austria
karl.entacher@fh-sbg.ac.at

Elliptic Rational Functions

Miroslav D. Lutovac
Dejan V. Tosic

This article introduces a new algorithm for computing the elliptic rational function $R_n(\xi, x)$. The key feature of the algorithm is a symbolic computation of some of the $R_n(\xi, x)$ using only elementary functions. Our algorithm is more efficient than traditional methods based on the Jacobi elliptic functions and the complete elliptic integral, which can be numerically intensive. $R_n(\xi, x)$ is used extensively in signal processing and system design as the best minimax approximation of a unit square pulse.

■ Introduction

Designers of many practical systems searched for a rational function $R_n(\xi, x)$ in the variable x that has

- the equiripple property and $|R_n(\xi, x)| \le 1$ over the interval $|x| \le 1$
- the largest value of $\min(|R_n(\xi, x)|)$ for $|x| \ge \xi > 1$
- the minimal order n

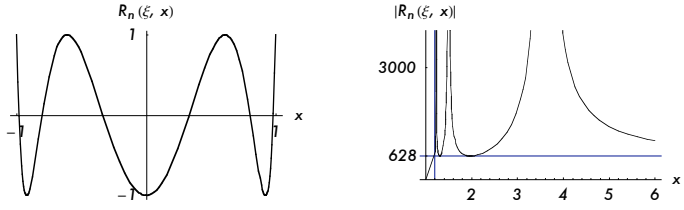
The rational function with those properties was found [1, 2] by using the Jacobi elliptic functions [3] and it is referred to as the *elliptic rational function* [4].

A function has the *equiripple* property if it oscillates between maximums and minimums of equal amplitude [1]. A quotient of two polynomials is called a *rational function* in the variable x, and the highest power in the polynomials is called the *order* of the rational function. The minimal value of $|R_n(\xi, x)|$ for $|x| \ge \xi$ is called the *discrimination factor* and is designated by $L_n(\xi)$. In signal processing theory ξ is known as the *selectivity factor* and can be any real number greater than 1, $\xi > 1$. Note that $R_n(\xi, x)$ is not a rational function in ξ. A typical plot of $R_n(\xi, x)$ is shown for $n = 6$ and $\xi = 1.2$.

We have defined the function `EllipticRationalFunction` that implements $R_n(\xi, x)$. The algorithm is detailed in the subsequent sections.

```
In[1]:= With[{n = 6, ξ = 1.2, R = EllipticRationalFunction[n, ξ, x]},
         L = DiscriminationFactor[n, ξ];
         a = Plot[EllipticRationalFunction[n, ξ, x], {x, -1, 1},
           PlotStyle → {Thickness[0.005]}, AxesLabel → {x, R},
           Ticks → {{-1, 1}, {-1, 1}}, DisplayFunction → Identity];
         b = Plot[Abs[EllipticRationalFunction[n, ξ, x]],
           {x, 1, 5 ξ}, PlotStyle → {Thickness[0.005]},
           AxesLabel → {x, Abs[R]}, PlotRange → {Automatic, {0, 4000}},
           Ticks → {Automatic, {Round[L], 3000}},
           GridLines → {{ξ}, {L}}, DisplayFunction → Identity];
         Show[GraphicsArray[{a, b}]]]
```

From In[1]:=



We used our symbolic algorithm for the elliptic rational function to optimize the symbolic performance of analog and digital systems. This optimization is not possible using traditional numeric algorithms. We derived closed-form formulas for designing high-speed low-consumption systems known as quadrature mirror filter banks [5]. $R_n(\xi, x)$ is extensively used in analog signal processing as the best approximation function [6].

We found a new function, known as *Minimum-Q Elliptic* [4, 7], by symbolically optimizing the elliptic rational function. Minimum-Q Elliptic became a standard function in manufacturing integrated filters [7]. In addition, again using symbolic optimization, we implemented a very efficient digital signal processing (DSP) system using programmable logic devices and very large-scale integrated circuits [5, 8]. By an efficient DSP system, we mean processing by multiplierless systems that consist of a small number of adders and binary shifters.

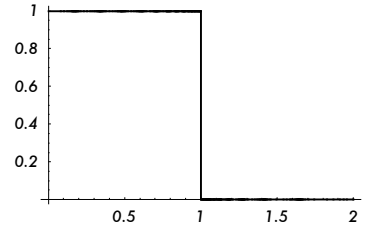
■ Application

The quintessence of the importance of $R_n(\xi, x)$ can be illustrated by the fact that $R_n(\xi, x)$ can be used for generating the best minimax approximation of a unit square pulse [1, 4]:

$$\frac{1}{1 + \epsilon^2 R_n^2(\xi, x)}. \tag{1}$$

```
In[2]:= With[{ϵ = 0.01, n = 32, ξ = 1.0001},
  Plot[1/(1 + ϵ^2 EllipticRationalFunction[n, ξ, x]^2),
   {x, 0, 2}, PlotStyle → {Thickness[0.007]}]]
```

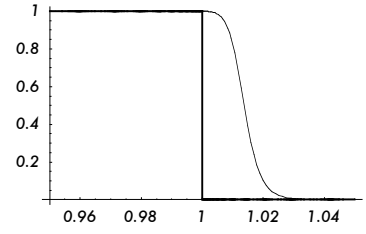
From In[2]:=



Chebyshev polynomials, for the same order, give a far inferior approximation (thin line) of the square pulse as shown in the following figure.

```
In[3]:= With[{ϵ = 0.01, n = 32, ξ = 1.0001}, Plot[
  {1/(1 + ϵ^2 EllipticRationalFunction[n, ξ, x]^2), 1/(1 + ϵ^2 ChebyshevT[n, x]^2)},
  {x, 0.95, 1.05}, PlotStyle → {Thickness[0.007], Thickness[0.002]},
  AxesOrigin → {0.95, 0}]]
```

From In[3]:=



Elliptic rational functions contain the one free parameter ξ that is used to adjust the slope of the pulse approximation. Chebyshev polynomials have no such parameter.

The pulse approximation with elliptic rational functions has many important applications in analog and digital signal processing and system design. Symbolic computation of elliptic rational functions and powerful symbolic algebra environments such as *Mathematica*, made many successful industrial designs possible [4, 5, 6, 7, 8]. These designs produced robust systems (such as analog and digital filters), shortened time to market, and helped designers make cost-effective solutions.

■ Definition

We define the elliptic rational function in terms of the Jacobi elliptic functions as

$$R_n(\xi, x) = \mathrm{cd}\left(n \frac{K\left(\frac{1}{L_n(\xi)}\right)}{K\left(\frac{1}{\xi}\right)} \mathrm{cd}^{-1}\left(x, \frac{1}{\xi}\right), \frac{1}{L_n(\xi)}\right), \tag{2}$$

where cd is one of the 12 Jacobi elliptic functions (see `JacobiCD` in [9]), cd^{-1} is the inverse cd Jacobi elliptic function (see `InverseJacobiCD` in [9]), K is the complete elliptic integral of the first kind (see `EllipticK` in [9]), n is the order (a positive integer), ξ is the *selectivity factor* ($\xi > 1$), and $L_n(\xi)$ is the discrimination factor defined in the Introduction. It can also be defined as the value of the elliptic rational function for $x = \xi$, that is, $L_n(\xi) = R_n(\xi, \xi)$ [4].

$R_n(\xi, x)$ can be represented by using the parametric equations

$$\begin{aligned} R_n(\xi, x) &= \mathrm{cd}\left(n\, w\, K\left(\frac{1}{L_n(\xi)}\right), \frac{1}{L_n(\xi)}\right) \\ x &= \mathrm{cd}\left(w\, K\left(\frac{1}{\xi}\right), \frac{1}{\xi}\right) \end{aligned} \tag{3}$$

where w is an intermediate variable [4].

Traditionally, $R_n(\xi, x)$ for known n, ξ, and x, can be computed as follows.

1. Find $K(1/\xi)$ using `EllipticK`.
2. Find w from the inverse cd Jacobi elliptic function.
3. Determine $K(1/L_n(\xi))$ and $L_n(\xi)$ from the *degree equation*

$$n \frac{K\left(\frac{1}{L_n(\xi)}\right)}{K\left(\sqrt{1 - \frac{1}{L_n^2(\xi)}}\right)} = \frac{K\left(\frac{1}{\xi}\right)}{K\left(\sqrt{1 - \frac{1}{\xi^2}}\right)}. \tag{4}$$

4. Find $R_n(\xi, x)$ using the cd Jacobi elliptic function.

The Chebyshev polynomial $T_n(x)$ can be derived from $R_n(\xi, x)$ for $\xi \to \infty$ because $\mathrm{cd}(\omega, 0) = \cos(\omega)$, that is,

$$\lim_{\xi \to \infty} (R_n(\xi, x)) = \cos(n \cos^{-1}(x)) = T_n(x). \tag{5}$$

The elliptic rational function in terms of the Jacobi elliptic functions can be expanded as a rational function in terms of x [4]. The explicit formulas of the first three functions are

$$R_1(\xi, x) = x \tag{6}$$

$$R_2(\xi, x) = \frac{\left(1 + \sqrt{1 - \frac{1}{\xi^2}}\right) x^2 - 1}{\left(-1 + \sqrt{1 - \frac{1}{\xi^2}}\right) x^2 + 1} \tag{7}$$

$$R_3(\xi, x) = x\,\frac{\left(1+\mathrm{dn}\left(\frac{2}{3}K\left(\frac{1}{\xi}\right),\frac{1}{\xi}\right)\right)^2 x^2-\left(1+2\,\mathrm{dn}\left(\frac{2}{3}K\left(\frac{1}{\xi}\right),\frac{1}{\xi}\right)\right)}{\left(-1+\mathrm{dn}^2\left(\frac{2}{3}K\left(\frac{1}{\xi}\right),\frac{1}{\xi}\right)\right)x^2+1}. \tag{8}$$

In *Mathematica*, $R_3(\xi, x)$ can be represented as follows.

```
In[4]:= (x ((1 + JacobiDN[2/3 EllipticK[1/ξ^2], 1/ξ^2])^2 x^2 -
          (1 + 2 JacobiDN[2/3 EllipticK[1/ξ^2], 1/ξ^2])))/
        (1 + (-1 + JacobiDN[2/3 EllipticK[1/ξ^2], 1/ξ^2]^2) x^2) // TraditionalForm
```

Out[4]//TraditionalForm=

$$\frac{x\left(x^2\left(\mathrm{dn}\left(\frac{2}{3}K\left(\frac{1}{\xi^2}\right)\middle|\frac{1}{\xi^2}\right)+1\right)^2-2\,\mathrm{dn}\left(\frac{2}{3}K\left(\frac{1}{\xi^2}\right)\middle|\frac{1}{\xi^2}\right)-1\right)}{\left(\mathrm{dn}\left(\frac{2}{3}K\left(\frac{1}{\xi^2}\right)\middle|\frac{1}{\xi^2}\right)^2-1\right)x^2+1}$$

Note that `JacobiDN` requires $\frac{1}{\xi^2}$ instead of $\frac{1}{\xi}$.

The meaning of the Jacobi elliptic function notations follows: cd(u, k) means k is the modulus, cd($u\,|\,m$) means $m = k^2$ is the parameter. The most common notation uses the form with k, but the elliptic functions are implemented in *Mathematica* using the form with m instead.

■ Properties

We implemented $R_n(\xi, x)$ in *Mathematica* as `EllipticRationalFunction[n,ξ,x]`.

The main properties of $R_n(\xi, x)$ follow.

- The squared function is even, $R_n^2(\xi, -x) = R_n^2(\xi, x)$.
- Over the unit interval $-1 \le x \le 1$, $R_n(\xi, x)$ is equiripple and $R_n^2(\xi, x) \le 1$.
- For $x = 1$, $R_n(\xi, 1) = 1$.
- $R_n(\xi, x)$ is monotonic increasing: $1 < R_n(\xi, x) < R_n(\xi, \xi)$, in the interval $1 < x < \xi$.
- For $x \ge \xi > 1$, $|\,R_n(\xi, x)\,| \ge L_n(\xi) = R_n(\xi, \xi)$.
- The key feature of $R_n(\xi, x)$ is

$$R_n(\xi, x) = \frac{R_n(\xi, \xi)}{R_n\left(\xi, \frac{\xi}{x}\right)}. \tag{9}$$

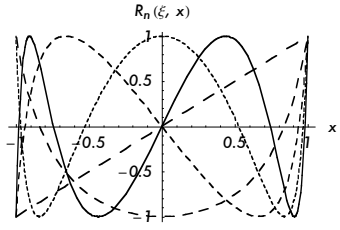
- The poles of $R_n(\xi, x)$ can be simply expressed in terms of the zeros of $R_n(\xi, x)$:

$$x_{\text{pole}} = \frac{\xi}{x_{\text{zero}}}. \tag{10}$$

The next figure illustrates the equiripple property of $R_n(\xi, x)$.

```
In[5]:= With[{ξ = 1.1, R = EllipticRationalFunction[n, ξ, x]},
         Plot[Evaluate[Table[EllipticRationalFunction[n, ξ, x], {n, 1, 5}]],
          {x, -1, 1}, AxesLabel → {x, R}, PlotStyle → {Dashing[{0.04}],
            Dashing[{0.03}], Dashing[{0.02}], Dashing[{0.01}], Dashing[{}]}]]
```

From In[5]:=



■ Nesting Property

Higher-order elliptic rational functions can be generated from lower-order functions by using the nesting property [4]

$$R_{mp}(\xi, x) = R_m(R_p(\xi, \xi), R_p(\xi, x)). \tag{11}$$

For example, $R_4(\xi, x) = R_2(R_2(\xi, \xi), R_2(\xi, x))$.

```
In[6]:= With[{ξ = 2},
          R4 = EllipticRationalFunction[2, EllipticRationalFunction[2, ξ, ξ],
            EllipticRationalFunction[2, ξ, x]]];
        R4 // FullSimplify // TraditionalForm
        N[R4 // Together] // TraditionalForm
```

Out[7]//TraditionalForm=

$$\frac{\frac{\left(1+2\sqrt{-24+14\sqrt{3}}\right)\left(\left(2+\sqrt{3}\right)x^2-2\right)^2}{\left(\left(-2+\sqrt{3}\right)x^2+2\right)^2}-1}{\frac{\left(-1+2\sqrt{-24+14\sqrt{3}}\right)\left(\left(2+\sqrt{3}\right)x^2-2\right)^2}{\left(\left(-2+\sqrt{3}\right)x^2+2\right)^2}+1}$$

Out[8]//TraditionalForm=

$$\frac{13.8743\,x^4 - 14.373\,x^2 + 1.99484}{0.017926\,x^4 - 0.516636\,x^2 + 1.99484}$$

The corresponding nesting formula can be derived for the zeros and poles of $R_n(\xi, x)$ as shown in [4]. For $n = 2^i 3^j$ the zeros and poles of $R_n(\xi, x)$ can be expressed symbolically in terms of ξ without using the Jacobi elliptic functions. Here is an example.

```
In[9]:= R2 = EllipticRationalFunction[2, ξ, x];
        zeros2 = FullSimplify[Solve[Numerator[R2] == 0, x], ξ > 1 && ξ ∈ Reals]
```

$$\text{Out[10]}= \left\{\left\{x \to -\frac{1}{\sqrt{1+\sqrt{1-\frac{1}{\xi^2}}}}\right\}, \left\{x \to \frac{1}{\sqrt{1+\sqrt{1-\frac{1}{\xi^2}}}}\right\}\right\}$$

```
In[11]:= poles2 = FullSimplify[Solve[Denominator[R2] == 0, x], ξ > 1 && ξ ∈ Reals]
```

$$\text{Out[11]}= \left\{\left\{x \to -\frac{1}{\sqrt{1-\sqrt{1-\frac{1}{\xi^2}}}}\right\}, \left\{x \to \frac{1}{\sqrt{1-\sqrt{1-\frac{1}{\xi^2}}}}\right\}\right\}$$

For orders $n \neq 2^i 3^j$, $R_n(\xi, x)$ cannot be expressed symbolically without the Jacobi elliptic functions or the like.

■ Algorithm

In this section we implement the elliptic rational functions in *Mathematica*.

□ Elliptic Rational Functions

First, we define the first-order function $R_1(\xi, x) = x$.

```
In[12]:= EllipticRationalFunction[1, ξ_, x_] := x;
```

We use the nesting property and the closed-form expressions [4] for orders $n = 2^i 3^j$.

```
In[13]:= EllipticRationalFunction[n_?EvenQ, ξ_, x_] :=
           Module[{t}, t = Sqrt[1 - 1/DiscriminationFactor[n/2, ξ]^2];
           ((t + 1) EllipticRationalFunction[n/2, ξ, x]^2 - 1)/
           ((t - 1) EllipticRationalFunction[n/2, ξ, x]^2 + 1)];

In[14]:= EllipticRationalFunction[n_?(Mod[#1, 3] == 0 &), ξ_, x_] :=
           Module[{b, c, r, y}, y = 1 - 2/DiscriminationFactor[n/3, ξ]^2;
           b = (1 - y^2)^(1/3); c = Sqrt[1 + b + b^2]; r = 1/2 (y/c + Sqrt[2 + b + 2 c] - 1);
           (EllipticRationalFunction[n/3, ξ, x]
              ((r + 1)^2 EllipticRationalFunction[n/3, ξ, x]^2 - (1 + 2 r)))/
           ((r^2 - 1) EllipticRationalFunction[n/3, ξ, x]^2 + 1)];
```

We use the Jacobi elliptic function `JacobiSN` and the complete elliptic integral of the first kind `EllipticK` for orders $n \neq 2^i 3^j$.

```
In[15]:= EllipticRationalFunction[n_Integer, ξ_, x_] :=
           EllipticRationalFunctionJacobi[n, ξ, x];

In[16]:= EllipticRationalFunctionJacobi[n_Integer, ξ_, x_] :=
           Module[{p, R, t, z}, z = EllipticRationalFunctionZeroJacobi[n, 1/ξ];
            p = EllipticRationalFunctionPoleJacobi[n, 1/ξ];
            R = Times @@ (t - z)/Times @@ (t - p); R /. t → x/R /. t → 1];
```

□ Discrimination Factor

The first-order discrimination factor equals the selectivity factor.

```
In[17]:= DiscriminationFactor[1, ξ_] := ξ;
```

We use the nesting property and the closed-form expressions [4] for orders $n = 2^i 3^j$.

```
In[18]:= DiscriminationFactor[n_?EvenQ, ξ_] :=
           Module[{b}, b = DiscriminationFactor[n/2, ξ]; (b + Sqrt[b^2 - 1])^2];

In[19]:= DiscriminationFactor[n_?(Mod[#1, 3] == 0 &), ξ_] := Module[
           {b, c, r, y}, y = 1 - 2/DiscriminationFactor[n/3, ξ]^2; b = (1 - y^2)^(1/3);
            c = Sqrt[1 + b + b^2]; r = 1/2 (y/c + Sqrt[2 + b + 2 c] - 1); Sqrt[(1 + 2 r)^3/((1 - r)^3 (1 + r))]];
```

We use the Jacobi elliptic function `JacobiSN` and the complete elliptic integral of the first kind `EllipticK` for orders $n \neq 2^i 3^j$.

```
In[20]:= DiscriminationFactor[n_Integer, ξ_] :=
           DiscriminationFactorJacobi[n, ξ];

In[21]:= DiscriminationFactorJacobi[n_Integer, ξ_] :=
           EllipticRationalFunctionJacobi[n, ξ, ξ];
```

□ Zeros and Poles

The zeros and poles of $R_n(\xi, x)$ for orders $n \neq 2^i 3^j$ are computed in terms of the Jacobi elliptic function `JacobiSN` and the complete elliptic integral of the first kind `EllipticK`.

```
In[22]:= EllipticRationalFunctionZeroJacobi[n_Integer, k_] := Module[{t, z},
           t = (JacobiSN[EllipticK[k^2] + EllipticK[k^2] (2 #1 - 1)/n, k^2] &) /@
             Range[Floor[n/2]]; If[EvenQ[n],
            z = Join[-t, Reverse[t]], z = Join[-t, {0}, Reverse[t]]]; z];

In[23]:= EllipticRationalFunctionPoleJacobi[n_Integer, k_] := Module[{t}, t =
           (1 / (k JacobiSN[EllipticK[k^2] + EllipticK[k^2] (2 #1 - 1)/n, k^2]) &) /@
             Range[Floor[n/2]]; Join[-Reverse[t], t]];
```

Closed-form expressions for the zeros and poles of $R_n(\xi, x)$ exist [4] for orders $n = 2^i 3^j$ and these formulas do not require Jacobi functions.

□ Traditional Forms

```
In[24]:= EllipticRationalFunction /:
         MakeBoxes[EllipticRationalFunction[n_, a_, x_], TraditionalForm] :=
         RowBox[{SubscriptBox["R", MakeBoxes[n, TraditionalForm]],
           "(", MakeBoxes[a, TraditionalForm], ",",
           MakeBoxes[x, TraditionalForm], ")"}]

In[25]:= DiscriminationFactor /:
         MakeBoxes[DiscriminationFactor[n_, a_], TraditionalForm] :=
         RowBox[{SubscriptBox["L", MakeBoxes[n, TraditionalForm]],
           "(", MakeBoxes[a, TraditionalForm], ")"}]
```

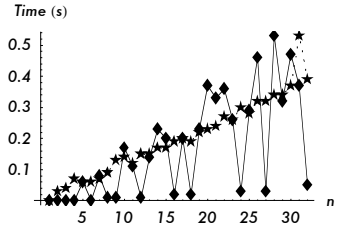
■ Timing

We time how long it takes to calculate $R_n(\xi, x)$ using our symbolic algorithm, `EllipticRationalFunction[n,ξ,x]`, and the numeric algorithm [1] that we implemented in *Mathematica* as `EllipticRationalFunctionJacobi[n,ξ,x]`. The following figure shows that the proposed algorithm (solid line) runs much faster than the traditional algorithm for orders $n = 2^i 3^j$.

```
In[26]:= With[{ξ = N[11/10, 64]}, t1List = {}; t2List = {};
          Do[t1 = Timing[N[EllipticRationalFunction[n, ξ, x]]];
           t2 = Timing[N[EllipticRationalFunctionJacobi[n, ξ, x]]];
           AppendTo[t1List, t1[[1]]]; AppendTo[t2List, t2[[1]]];, {n, 1, 32}]]
```

```
In[27]:= << "Graphics`MultipleListPlot`"
         MultipleListPlot[{t1List/Second, t2List/Second},
          PlotJoined → True, AxesLabel → {n, "Time (s)"}]
```

From In[27]:=



■ Acknowledgment

We thank the Serbian Ministry for Science, Technologies, and Development for partial support of our research on this topic.

■ References

[1] R. W. Daniels, *Approximation Methods for Electronic Filter Design*, New York: McGraw-Hill, 1974.

[2] A. Antoniou, *Digital Filters: Analysis and Design*, New York: McGraw-Hill, 1979.

[3] M. Abramowitz and I. A. Stegun, eds., *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*, New York: Dover Publications, 1972.

[4] M. D. Lutovac, D. V. Tosic, and B. L. Evans, *Filter Design for Signal Processing Using MATLAB® and Mathematica*, Upper Saddle River, NJ: Prentice Hall, 2001.

[5] G. Jovanovic-Dolecek, ed., *Multirate Systems: Design and Applications*, Hershey, PA: Idea Group Publishing, 2002.

[6] P. Karantzalis, "Free FilterCAD 3.0 Software Designs Filters Quickly and Easily," Design Note 245, *Linear Technology Design Notes*, 2000.

[7] FilterCAD™ (FCAD) 3.0, Milpitas, CA: Linear Technology Corporation, (Sep. 2000) www.linear.com/designtools/filtercad.jsp.

[8] Lj. D. Milic and M. D. Lutovac, "Design of Multiplierless Elliptic IIR Filters with a Small Quantization Error," *IEEE Transactions on Signal Processing*, **47**, 1999 pp. 469–479.

[9] S. Wolfram, *The Mathematica Book*, 4th ed., Champaign/Cambridge: Wolfram Media, Inc./Cambridge University Press, 1999.

About the Authors

Miroslav D. Lutovac is a chief scientist at the Institute for Telecommunications and Electronics and a professor at the University of Belgrade, Serbia. His research interests include the theory and implementation of analog and digital signal processing, and the symbolic analysis and synthesis of multiplierless and multirate digital systems.

Dejan V. Tosic is a professor in the School of Electrical Engineering at the University of Belgrade, Serbia. He has focused his research on creating a framework for the symbolic analysis of circuits and systems that is suitable for research as well as industrial and educational applications. He is developing automation tools for optimizing the design and synthesis of analog and digital systems.

Lutovac and Tosic are coauthors (with Brian Evans) of the book *Filter Design for Signal Processing Using MATLAB and Mathematica* [4]. They have developed *SchematicSolver*, a *Mathematica* application for mouse-driven interactive drawing of systems and for solving and implementing said systems.

Dejan V. Tosic
Miroslav D. Lutovac
School of Electrical Engineering
University of Belgrade
Bul. Kralja Aleksandra 73
11000 Belgrade, Serbia, Europe
tosic@kondor.etf.bg.ac.yu
kondor.etf.bg.ac.yu/~tosic
lutovac@kondor.etf.bg.ac.yu
kondor.etf.bg.ac.yu/~lutovac

Polyominoes and Related Families

Jaime Rangel-Mondragón

Polyominoes are generalizations of dominoes constructed by joining congruent squares side by side. By describing a polyomino as a list of Gaussian integers, we generate all different polyominoes of a given size. This method is extended to the generation of the families of polyiamonds, polyhexes, and polykites. We also give a method to tile rectangles using polyominoes and explore the fractal family of rep-tiles.

■ Introduction

In 1953, when delivering a talk at the Harvard Mathematics Club, mathematician Solomon Golomb [1] defined a new class of geometric figures he named polyominoes. Polyominoes are generalizations of dominoes [2] and have been extremely popular since their use in the game of Tetris [3]. They have also enjoyed a prominent place in the recreational mathematics literature since Martin Gardner further popularized them in 1957.

A polyomino is any connected figure that can be constructed by joining congruent squares side by side. A polyomino formed by n squares is referred to as an n-omino. In the first part of this work, we generate all n-ominoes and extend our method to the corresponding generation of polyiamonds, polyhexes, and polykites [4]. In the second part, we will tessellate rectangles using polyominoes and introduce the family of *rep-tiles*.

Some sections of this work are ordered so that we can compare performance issues among different computer configurations and also calculate timings for a 2003-vintage model 2.4 GHz personal computer using *Mathematica* 5.

```
In[1]:= Off[General::"spell", General::"spell1"]
        SetOptions[Graphics, AspectRatio → Automatic];
        << "Graphics`Colors`";
```

■ The Naive Approach

In this section we generate n-ominoes using a straightforward approach. We consider all possible 0-1 $n \times n$ matrices and select those that are orthogonally connected, that is, that have no isolated blocks of ones. To check this property

we read the first one and change its sign. We then mark those adjacent to it and repeatedly spread this changing of signs to those adjacent to them until no further changes occur. At the end, the new matrix cannot contain a one if the original matrix represented a connected shape.

```
In[4]:= connectedQ[nOmino_] := Module[{n, m, i, j, h, k, s = nOmino},
          {n, m} = Dimensions[s];
          {i, j} = First[Position[s, 1]];
          s[[i,j]] = -1;
          FixedPoint[
           (Do[If[s[[h,k]] == 1,
               If[((k > 1) && (s[[h,k-1]] < 0)) || ((k < m) && (s[[h,k+1]] < 0)) ||
                 ((h > 1) && (s[[h-1,k]] < 0)) || ((h < n) && (s[[h+1,k]] < 0)),
                s[[h,k]] = -1]], {k, m}, {h, n}];
             s) &, s];
          Position[s, 1] == {}]
```

```
In[5]:= connectedQ[{{0, 0, 1}, {0, 1, 0}, {1, 1, 1}}]
```

```
Out[5]= False
```

The following function provides a canonical form for our matrices. It pushes the entire configuration up and to the left so that neither the first row nor the first column are all zero.

```
In[6]:= standard[m_] := Module[{p = Position[m, 1], h, ans, n = Length[m]},
          ans = Table[0, {n}, {n}];
          h = {Min[First /@ p], Min[Last /@ p]};
          p = (# - h + {1, 1}) & /@ p;
          Map[(ans[[First[#],Last[#]]] = 1) &, p];
          ans]
```

```
In[7]:= standard[{{0, 0, 0, 0, 0}, {0, 0, 1, 0, 0}, {0, 1, 1, 1, 0},
          {0, 0, 0, 1, 1}, {0, 0, 0, 0, 0}}] // MatrixForm
```

```
Out[7]//MatrixForm=
0 1 0 0 0
1 1 1 0 0
0 0 1 1 0
0 0 0 0 0
0 0 0 0 0
```

Our strategy is to generate all 0-1 $n \times n$ matrices (there are 2^{n^2} of them), select those having n ones that are connected, convert them to canonical form, and, finally, remove repetitions from this list. For example, for $n = 4$, out of 65536 possible 4×4 matrices we get only 1820 that have four ones, and, of those, 113 are connected. Only 19 are left when we select those with a different canonical form.

```
In[8]:= Timing[
         n = 4;
         u = Select[Range[2^n^2], (Plus @@ IntegerDigits[#, 2] == n) &];
         v = Select[
           Map[Partition[IntegerDigits[#, 2, n^2], n] &, u], connectedQ[#] &];
         w = Union[standard /@ v];
         {MatrixForm /@ w, 2^n^2, Length[u], Length[v], Length[w]}]
```

Out[8]=

$$\left\{1.156\,\text{Second},\ \left\{\left\{\begin{pmatrix}0&0&1&0\\1&1&1&0\\0&0&0&0\\0&0&0&0\end{pmatrix},\begin{pmatrix}0&1&0&0\\0&1&0&0\\1&1&0&0\\0&0&0&0\end{pmatrix},\begin{pmatrix}0&1&0&0\\1&1&0&0\\0&1&0&0\\0&0&0&0\end{pmatrix},\begin{pmatrix}0&1&0&0\\1&1&0&0\\1&0&0&0\\0&0&0&0\end{pmatrix},\begin{pmatrix}0&1&0&0\\1&1&1&0\\0&0&0&0\\0&0&0&0\end{pmatrix},\right.\right.\right.$$

$$\left.\begin{pmatrix}0&1&1&0\\1&1&0&0\\0&0&0&0\\0&0&0&0\end{pmatrix},\begin{pmatrix}1&0&0&0\\1&0&0&0\\1&0&0&0\\1&0&0&0\end{pmatrix},\begin{pmatrix}1&0&0&0\\1&0&0&0\\1&1&0&0\\0&0&0&0\end{pmatrix},\begin{pmatrix}1&0&0&0\\1&1&0&0\\0&1&0&0\\0&0&0&0\end{pmatrix},\right.$$

$$\left.\begin{pmatrix}1&0&0&0\\1&1&0&0\\1&0&0&0\\0&0&0&0\end{pmatrix},\begin{pmatrix}1&0&0&0\\1&1&1&0\\0&0&0&0\\0&0&0&0\end{pmatrix},\begin{pmatrix}1&1&0&0\\0&1&0&0\\0&1&0&0\\0&0&0&0\end{pmatrix},\begin{pmatrix}1&1&0&0\\0&1&1&0\\0&0&0&0\\0&0&0&0\end{pmatrix},\right.$$

$$\left.\begin{pmatrix}1&1&0&0\\1&0&0&0\\1&0&0&0\\0&0&0&0\end{pmatrix},\begin{pmatrix}1&1&0&0\\1&1&0&0\\0&0&0&0\\0&0&0&0\end{pmatrix},\begin{pmatrix}1&1&1&0\\0&0&1&0\\0&0&0&0\\0&0&0&0\end{pmatrix},\begin{pmatrix}1&1&1&0\\0&1&0&0\\0&0&0&0\\0&0&0&0\end{pmatrix},\right.$$

$$\left.\left.\begin{pmatrix}1&1&1&0\\1&0&0&0\\0&0&0&0\\0&0&0&0\end{pmatrix},\begin{pmatrix}1&1&1&1\\0&0&0&0\\0&0&0&0\\0&0&0&0\end{pmatrix}\right\},\ 65536,\ 1820,\ 113,\ 19\right\}\right\}$$

Although this is the simplest way, it takes nearly an hour to complete for $n = 5$, which forces us to look for another approach. If, instead of selecting those binary numbers having n digits that are 1 out of all 2^{n^2} possible, we generate them directly (there are $\binom{n}{2}$ of them), we will save a substantial amount of time. The function `bin[n, b]` computes all lists of length b having n ones.

```
In[9]:= bin[n_, l_] := {} /; n > l
        bin[0, l_] := {Table[0, {l}]}
        bin[1, 1] := {{1}}
        bin[n_, l_] := bin[n, l] = Join[Map[Join[{0}, #] &, bin[n, l - 1]],
           Map[Join[{1}, #] &, bin[n - 1, l - 1]]]
```

```
In[13]:= bin[3, 5]

Out[13]= {{0, 0, 1, 1, 1}, {0, 1, 0, 1, 1}, {0, 1, 1, 0, 1},
          {0, 1, 1, 1, 0}, {1, 0, 0, 1, 1}, {1, 0, 1, 0, 1},
          {1, 0, 1, 1, 0}, {1, 1, 0, 0, 1}, {1, 1, 0, 1, 0}, {1, 1, 1, 0, 0}}
```

We get the 63 canonical forms for the case $n = 5$ more quickly.

```
In[14]:= Timing[
          n = 5;
          v = Select[Map[Partition[#, n] &, bin[n, n^2]], connectedQ[#] &];
          w = Union[standard /@ v];
          Length[w]]

Out[14]= {19.719 Second, 63}
```

However, some of these patterns are still equivalent under rotations and reflections, so more processing is needed. Although we have substantially reduced the computing time, we will not pursue this approach any further because there is a faster and more general alternative.

■ Polyominoes

The straightforward approach presented in the previous section has many disadvantages. The representation of polyominoes as 0-1 matrices is wasteful because they are very sparse. In this section we consider a polyomino to be a list of unit squares, where a square is specified by the Cartesian coordinates of its bottom-left vertex; further, these coordinates will be encapsulated as complex numbers. So, a polyomino will be a list of Gaussian integers; that is, complex numbers having integral real and imaginary parts. We thus define the following type.

```
In[15]:= polyominoQ[p_] := And @@ ((IntegerQ[Re[#]] && IntegerQ[Im[#]]) & /@ p)
```

In this definition we cannot use the pattern `_Complex..` because only numbers with nonzero imaginary parts have head `Complex`.

To determine whether two figures are equivalent, we consider the action of the dihedral group $D_4 = \{e, r, r^2, r^3, f, fr, fr^2, fr^3\}$, where r rotates the figure by 90° and f turns it over. Given a polyomino, we then have at most eight different equivalent polyominoes.

If we were using a matrix `m` to describe the positions of the squares forming a polyomino, we could perform a 90° rotation with `Reverse[Transpose[m]]` and a reflection with `Transpose[m]`.

The following functions act on a given polyomino `p`; `rot` and `ref` correspond to the transformations `r` and `f`. The function `cyclic` computes the list of the rotations of `p` and is used in the construction of D_4 produced by the function `dihedral`. (In this case `cyclic` is equal to `NestList[rot,p,3]`.) The function `canonical` computes a standard representation of `p` by removing repetitions and

sorting. The functions `liC` and `polC` convert a list of complex numbers into a list of points to be drawn as lines or as polygons. The arguments of the functions `cyclic` and `allPieces` are not restricted to polyominoes as they will also be used for the other families.

```
In[16]:= rot[p_?polyominoQ] := i p
         ref[p_?polyominoQ] := (# - 2 Re[#]) & /@ p

         cyclic[p_] := Module[{i = p, ans = {p}},
           While[(i = rot[i]) ≠ p, AppendTo[ans, i]]; ans]

         dihedral[p_?polyominoQ] := Flatten[{#, ref[#]} & /@ cyclic[p], 1]

         canonical[p_?polyominoQ] :=
          Union[(# - (Min[Re[p]] + Min[Im[p]] i)) & /@ p]

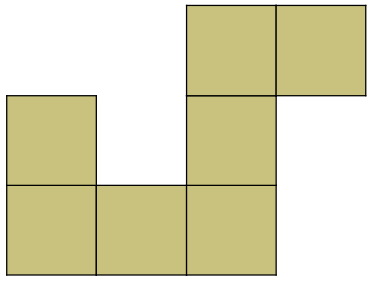
         allPieces[p_] := Union[canonical /@ dihedral[p]]

         liC[z_] := Line[{Re[#], Im[#]} & /@ z]
         polC[z_] := Polygon[{Re[#], Im[#]} & /@ z]

         draw[p_?polyominoQ, pr_: All] :=
          Graphics[{{DarkKhaki, polC[{#, # + 1, # + 1 + i, # + i}]},
              liC[{#, # + 1, # + 1 + i, # + i, #}]} & /@ p, PlotRange → pr]

In[25]:= polyomino = {0, 1, 2, i, 2 + i, 2 + 2 i, 3 + 2 i};
         pol = draw[polyomino];
         Show[pol]
```

From In[25]:=



The function `draw` generates the graphical object with an optional second argument specifying the plot range. Note also the change in the representation of a polyomino; the function `canonical` places the polyomino in the first quadrant touching both axes. Here is an example.

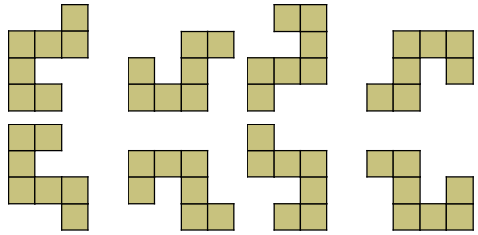
```
In[28]:= canonical[polyomino]

Out[28]= {0, i, 1, 2, 2 + i, 2 + 2 i, 3 + 2 i}
```

The function `allpieces` computes all eight equivalent figures (or less for some polyominoes).

```
In[29]:= pol = draw[#, {{0, 4.1}, {-0.1, 4}}] & /@ allPieces[polyomino];
         Show[GraphicsArray[Partition[pol, 4]]]
```

From In[29]:=

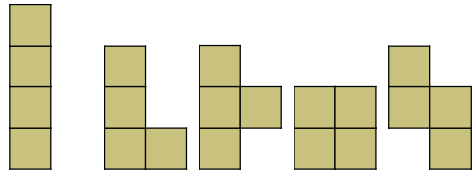


The following incremental method generates all n-ominoes. Having generated the set of all $(n-1)$-ominoes, we take each of its members and append a square to each of its squares in all four possible directions. Once we have this extended set containing all the n-ominoes, we obtain their canonical representation and proceed to eliminate the redundant ones.

```
In[31]:= polyominoes[1] := {{0}}
         polyominoes[n_] := polyominoes[n] = Module[{f, fig, ans = {}},
            fig = Map[(f = #; Map[{f, # + 1, f, # + i, f, # - 1, f, # - i} &, f]) &,
              polyominoes[n - 1]];
            fig = Partition[Flatten[fig], n];
            f = Select[Union[canonical /@ fig], Length[#] == n &];
            While[f ≠ {},
             ans = {ans, First[f]};
             f = Complement[f, allPieces[First[f]]]];
            Partition[Flatten[ans], n]]
```

```
In[33]:= pol = draw[#, {{0, 2.1}, {-0.1, 4}}] & /@ polyominoes[4];
         Show[GraphicsArray[pol]]
```

From In[33]:=



We can now compare the time taken to generate all pentominoes with the time it took in the previous section. Because *Mathematica* uses dynamic programming to retain the generated n-ominoes, the time it takes to recompute old values will only amount to how long it takes to retrieve them and hence would be negligible. So for a fair comparison, we have to start anew so that all previous values are cleared. Let us restart the *Mathematica* kernel and evaluate the following after all initialization cells are evaluated.

```
In[1]:= Timing[polyominoes[5];]

Out[1]= {0.047 Second, Null}
```

We can now obtain the number of free n-ominoes (free meaning equivalent under rotation and reflection) ([5, 6] seq. A000105).

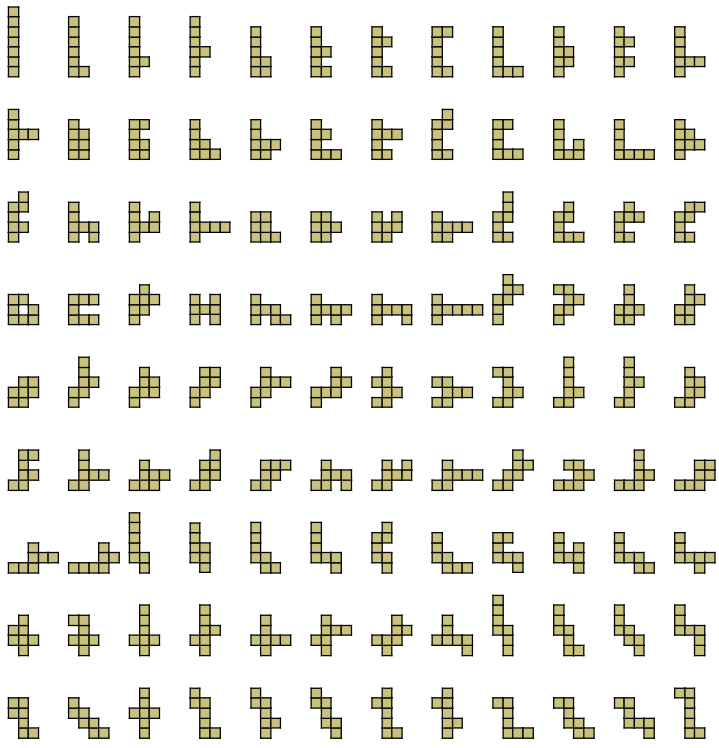
```
In[2]:= Timing[Table[Length[polyominoes[n]], {n, 9}]]

Out[2]= {13.625 Second, {1, 1, 2, 5, 12, 35, 108, 369, 1285}}
```

As an illustration, let us generate all 108 heptominoes, because it is not until $n = 7$ that a hole appears. According to our definition, a polyomino can have one or many interior holes. Can you spot the one having a hole among the following heptominoes? (That is the reason we color the squares; otherwise, it would be impossible to distinguish it.)

```
In[3]:= pol = draw[#, {{-0.5, 5}, {-0.5, 7}}] & /@ polyominoes[7];
        Show[GraphicsArray[Partition[pol, 12]]]
```

From In[3]:=



We can also obtain the one-sided (no reflections allowed) or "chiral" polyominoes by running the following cell, which redefines function `allPieces`, rerunning the previous function `polyominoes`, and computing the table as was done before ([5, 6] seq. A000988). At the end of the generation of this table, we have to recover the original definition of `allPieces` because it is needed in the next section. The easiest way is to restart the *Mathematica* kernel, evaluate the initialization cells, and continue with the cells in the next section.

```
In[1]:= allPieces[p_?polyominoQ] := Union[canonical /@ cyclic[p]]

In[2]:= Timing[Table[Length[polyominoes[n]], {n, 8}]]

Out[2]= {3.688 Second, {1, 1, 2, 7, 18, 60, 196, 704}}
```

■ Polyiamonds

Polyiamonds are figures built from congruent unit-side equilateral triangles in the same way that polyominoes are built from squares. Any triangle forming a polyiamond will be described by a pair $\{A, t\}$, wherein A codifies the complex coordinates $a + b\,e^{\frac{\pi i}{3}}$ of the leftmost vertex (anchor) of the triangle (we are assuming one of its sides rests horizontally) and t is its type taken from the set $\{1, -1\}$, corresponding to whether the apex is pointing up or down. The underlying grid on which to place the triangles is generated by all integral linear combinations of the numbers 1 and $e^{\frac{\pi i}{3}}$. Any triangle $\{A, t\}$ has three neighbors adjacent to it, namely, $\{A, -t\}$, $\{A + tv, -t\}$, and $\{A + t(v - u), -t\}$. As with polyominoes, the canonical representation of a polyiamond moves the piece so that its leftmost vertex touches the origin.

To construct the corresponding equivalent versions of a polyiamond, only rotations of multiples of 60° are allowed (a rotation of 60° implies the type of a triangle is changed) so that point (a, b) gets rotated to point $(-b, a + b)$. The reflection of a triangle is achieved simply by sending each of its vertices (a, b) to $(-a - b, b)$. (In this representation a reflection gives the reverse negative of a rotation!) The type of a polyiamond is tested as follows.

```
In[3]:= polyiamondQ[{{{_Integer, _Integer}, 1 | -1} ..}] := True
        polyiamondQ[_] := False
```

Here are the polyiamond functions.

```
In[5]:= rot[p_?polyiamondQ] :=
         {{-#[[1,2]], Plus @@ First[#]} + If[Last[#] == 1, {-1, 1}, {0, 0}],
            -Last[#]} & /@ p
        ref[p_?polyiamondQ] := {{-Plus @@ First[#], #[[1,2]]}, Last[#]} & /@ p

        dihedral[p_?polyiamondQ] := Flatten[{#, ref[#]} & /@ cyclic[p], 1]

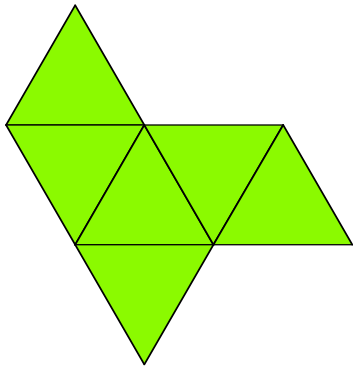
        canonical[p_?polyiamondQ] :=
         Sort[Map[{First[#] - {Min[First[First[#]] & /@ p],
               Min[Last[First[#]] & /@ p]}, Last[#]} & , p]]

        draw[p_?polyiamondQ, pr_: All] := Module[{{a, b, v, t, u = e^(π i/3)}},
          Graphics[{{{a, b}, t} = #; v = a + b u;
              {Chartreuse, polC[v + {0, If[t == 1, u, 1 - u], 1}]},
              liC[v + {0, If[t == 1, u, 1 - u], 1, 0}]} & /@ p, PlotRange → pr]]
```

Here is an example of a polyiamond and its conversion to canonical form.

```
In[10]:= polyiamond = {{{0, 0}, 1}, {{0, 0}, -1},
           {{1, -1}, 1}, {{1, -1}, -1}, {{1, 0}, -1}, {{2, -1}, 1}};
         pol = draw[polyiamond];
         Show[pol]
```

From In[10]:=



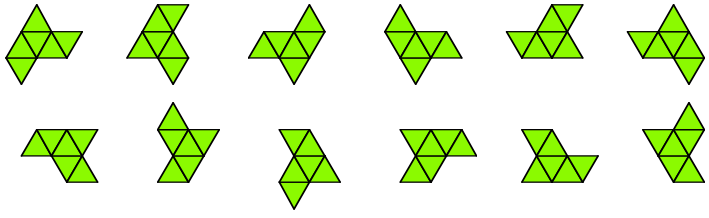
```
In[13]:= canonical[polyiamond]

Out[13]= {{{0, 1}, -1}, {{0, 1}, 1}, {{1, 0}, -1},
          {{1, 0}, 1}, {{1, 1}, -1}, {{2, 0}, 1}}
```

Here we obtain the polyiamonds equivalent to the previous one.

```
In[14]:= pol = draw[#, {{-0.1, 3.5}, {-1, 2.7}}] & /@ allPieces[polyiamond];
         Show[GraphicsArray[Partition[pol, 6]]]
```

From In[14]:=

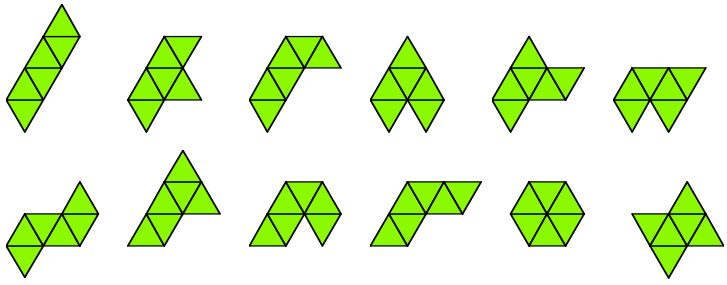


Finally, we generate all nonisomorphic polyiamonds. We make use of the fact that any triangle $\{\{a, b\}, 1\}$ has neighbors $\{\{a, b\}, -1\}$, $\{\{a, b+1\}, -1\}$, and $\{\{a-1, b+1\}, -1\}$, and that triangle $\{\{a, b\}, -1\}$ has neighbors $\{\{a, b\}, 1\}$, $\{\{a, b-1\}, 1\}$, and $\{\{a+1, b-1\}, 1\}$.

```
In[16]:= polyiamonds[1] := {{{{0, 0}, 1}}}
         polyiamonds[n_] :=
          polyiamonds[n] = Module[{f, A, t, x, y, z, fig, ans = {}},
            fig = Map[(f = #; Map[({A, t} = #; {f, {A, -t}, f, {A + {0, t}, -t},
                    f, {A + {-t, t}, -t}}) &, f]) &, polyiamonds[n - 1]];
            fig = Partition[Partition[Flatten[fig], 3] /.
               {x_Integer, y_, z_} → {{x, y}, z}, n];
            f = Union[canonical /@ Select[Union /@ fig, Length[#] == n &]];
            While[f ≠ {},
             ans = {ans, First[f]};
             f = Complement[f, allPieces[First[f]]]];
            Partition[
             Partition[Flatten[ans], 3] /. {x_Integer, y_, z_} → {{x, y}, z}, n]]
```

```
In[18]:= pol = draw[#, {{0, 3}, {-1, 2.6}}] & /@ polyiamonds[6];
         Show[GraphicsArray[Partition[pol, 6]]]
```

From In[18]:=



The number of different polyiamonds is given by the following table ([5, 6], seq. A000577).

```
In[20]:= Timing[Table[Length[polyiamonds[n]], {n, 8}]]
Out[20]= {0.969 Second, {1, 1, 1, 3, 4, 12, 24, 66}}
```

■ Polyhexes

Polyhexes are figures built from congruent unit-length hexagons. We represent a hexagon forming a polyhex simply by a pair (a, b) corresponding to the complex coordinates of its left-bottom corner $a + b\,e^{\frac{\pi i}{3}}$. For this reason, all the corresponding functions turn out to be straightforward adaptations of their polyiamond counterparts.

```
In[21]:= polyhexeQ[{{_Integer, _Integer} ..}] := True
         polyhexeQ[_] := False
```

```
In[23]:= rot[p_?polyhexeQ] := {-Last[#], Plus @@ #} & /@ p
         ref[p_?polyhexeQ] := {-Plus @@ #, Last[#]} & /@ p

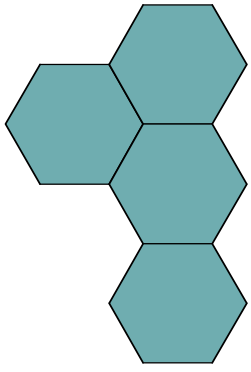
         dihedral[p_?polyhexeQ] := Flatten[{#, ref[#]} & /@ cyclic[p], 1]

         canonical[p_?polyhexeQ] :=
          Sort[Map[(# - {Min[First /@ p], Min[Last /@ p]}) & , p]]

         draw[p_?polyhexeQ, pr_: All] := Module[{a, b, v, t, u = E^(π i/3)},
           Graphics[{{a, b} = #; v = a + b u;
               {CadetBlue, polC[{v, v + 1, v + 1 + u, v + 2 u, v + 2 u - 1, v + u - 1}]},
               liC[{v, v + 1, v + 1 + u, v + 2 u, v + 2 u - 1, v + u - 1, v}]} & /@
             p, PlotRange → pr]]

In[28]:= polyhex = {{0, 0}, {1, 1}, {2, -1}, {3, -3}};
         pol = draw[polyhex];
         Show[pol]
```

From In[28]:=

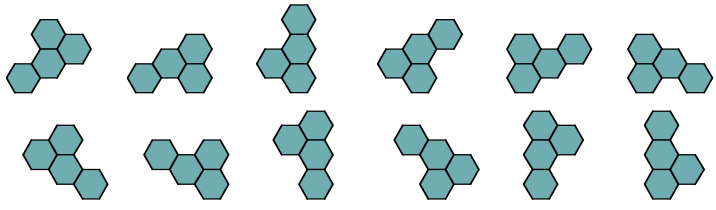


```
In[31]:= canonical[polyhex]

Out[31]= {{0, 3}, {1, 4}, {2, 2}, {3, 0}}

In[32]:= pol = draw[#, {{-0.5, 6}, {-0.2, 5.5}}] & /@ allPieces[polyhex];
         Show[GraphicsArray[Partition[pol, 6]]]
```

From In[32]:=



The neighbors of a hexagon (a, b) are $(a-1, b-1)$, $(a+1, b+2)$, $(a+2, b-1)$, $(a+1, b+1)$, $(a-1, b+2)$, and $(a-2, b+1)$. Thus, we can simulate a hexagonal cellular space by a rectangular cellular space in which the neighbors are these

nonadjacent ones. The following function generates all polyhexes of a given order.

```
In[34]:= polyhexes[1] := {{{0, 0}}}
         polyhexes[n_] := polyhexes[n] = Module[{f, a, b, fig, ans = {}},
            fig = Map[(f = #; Map[({a, b} = #; {f, {a - 1, b - 1}, f, {a + 1, b - 2},
                    f, {a + 2, b - 1}, f, {a + 1, b + 1}, f, {a - 1, b + 2},
                    f, {a - 2, b + 1}}) &, f]) &, polyhexes[n - 1]];
            fig = Partition[Partition[Flatten[fig], 2], n];
            f = Union[canonical /@ Select[Union /@ fig, Length[#] == n &]];
            While[f ≠ {},
             ans = {ans, First[f]};
             f = Complement[f, allPieces[First[f]]]];
            Partition[Partition[Flatten[ans], 2], n]]

In[36]:= pol = draw[#, {{-0.5, 6.2}, {-0.5, 5.5}}] & /@ polyhexes[4];
         Show[GraphicsArray[pol]]
```

From In[36]:=



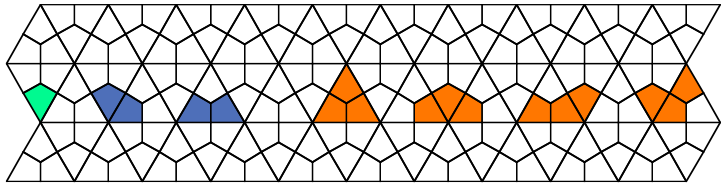
The number of different polyhexes is given by the following table ([5, 6], seq. A000228).

```
In[38]:= Timing[Table[Length[polyhexes[n]], {n, 8}]]

Out[38]= {18.891 Second, {1, 1, 3, 7, 22, 82, 333, 1448}}
```

■ Polykites

Polykites are figures built from the quadrilaterals forming the following lattice. On top of this lattice, we have shown the arrangements corresponding to all the possible shapes for the 1-polykite, 2-polykites, and 3-polykites using different colors. Larger-order polykites offer a lovely resemblance of faces, birds, and animal shapes in a way similar to tangrams. Their different balance and aesthetic properties motivated the author to include their generation as a proper sibling of polyominoes; however, the other Penrose piece, the Dart, being nonconvex, resisted all attempts at generating polydarts.



Just as we have two types of triangles forming polyiamonds, we have six types of kites forming polykites and we will describe them in a similar fashion. The

following comprises an adaptation of the functions we have previously designed. The type of a polykite is tested as follows.

```
In[39]:= polykiteQ[{{{_Integer, _Integer}, 1 | 2 | 3 | 4 | 5 | 6} ..}] := True
         polykiteQ[_] := False
```

Here are the polykite functions.

```
In[41]:= rot[p_?polykiteQ] :=
          {{-#[[1,2]], Plus @@ First[#]}, If[Last[#] == 6, 1, 1 + Last[#]]} & /@ p
         ref[p_?polykiteQ] := {{-Plus @@ First[#], #[[1,2]]},
             Switch[Last[#], 1, 3, 2, 2, 3, 1, 4, 6, 5, 5, 6, 4]} & /@ p

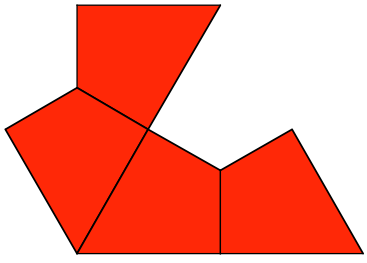
         dihedral[p_?polykiteQ] := Flatten[{#, ref[#]} & /@ cyclic[p], 1]

         canonical[p_?polykiteQ] :=
          Sort[Map[{First[#] - {Min[First[First[#]] & /@ p],
                Min[Last[First[#]] & /@ p]}, Last[#]} & , p]]

         draw[p_?polykiteQ, pr_: All] := Module[{a, b, v, t, u = e^(π i/3)},
           Graphics[{{{a, b}, t} = #; v = a + b u;
                {LightCadmiumRed, polC[v + {0, u^(t-1), (2 (u^t + u^(t-1)))/3, u^t}]},
                liC[v + {0, u^(t-1), (2 (u^t + u^(t-1)))/3, u^t, 0}]} & /@ p, PlotRange → pr]]
```

```
In[46]:= polykite = {{{0, 0}, 4}, {{0, -2}, 2}, {{0, -2}, 1}, {{2, -2}, 3}};
         pol = draw[polykite];
         Show[pol]
```

From In[46]:=

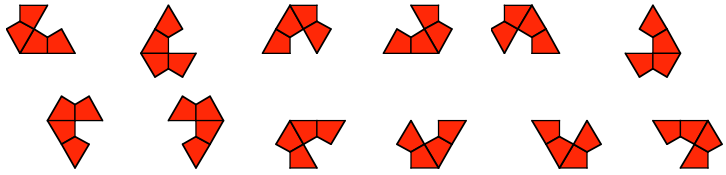


```
In[49]:= canonical[polykite]
```

```
Out[49]= {{{0, 0}, 1}, {{0, 0}, 2}, {{0, 2}, 4}, {{2, 0}, 3}}
```

```
In[50]:= pol = draw[#, {{-0.5, 3.5}, {-1, 2.8}}] & /@ allPieces[polykite];
         Show[GraphicsArray[Partition[pol, 6]]]
```

From In[50]:=

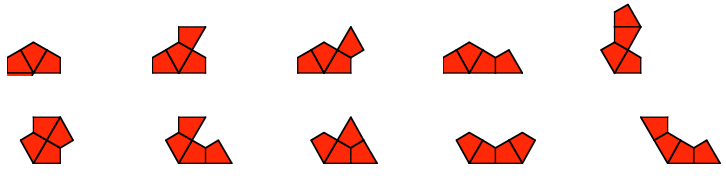


The polykite $\{A, t\}$ has neighbors depending on its anchor point and its type as shown in the following `Switch` instruction.

```
In[52]:= polykites[1] := {{{{0, 0}, 1}}}
         polykites[n_] :=
          polykites[n] = Module[{f, A, t, x, y, z, fig, ans = {}},
            fig = Map[(f = #; Map[({A, t} = #;
                    Switch[t,
                     1,
                     {f, {A + {2, 0}, 3}, f, {A + {0, 2}, 5}, f, {A, 2}, f, {A, 6}},
                     2, {f, {A + {0, 2}, 4}, f, {A + 2 {-1, 1}, 6},
                      f, {A, 3}, f, {A, 1}},
                     3, {f, {A + 2 {-1, 1}, 5}, f, {A + {-2, 0}, 1},
                      f, {A, 4}, f, {A, 2}},
                     4, {f, {A + {-2, 0}, 6}, f, {A + {0, -2}, 2},
                      f, {A, 5}, f, {A, 3}},
                     5, {f, {A + {0, -2}, 1}, f, {A + 2 {1, -1}, 3},
                      f, {A, 6}, f, {A, 4}},
                     6, {f, {A + {2, 0}, 4}, f, {A + 2 {1, -1}, 2}, f,
                      {A, 1}, f, {A, 5}}]) &, f]) &, polykites[n - 1]];
            fig = Partition[Partition[Flatten[fig], 3] /.
               {x_Integer, y_, z_} → {{x, y}, z}, n];
            f = Union[canonical /@ Select[Union /@ fig, Length[#] == n &]];
            While[f ≠ {},
             ans = {ans, First[f]};
             f = Complement[f, allPieces[First[f]]]];
            Partition[
             Partition[Flatten[ans], 3] /. {x_Integer, y_, z_} → {{x, y}, z}, n]]
```

```
In[54]:= pol = draw[#, {{-1, 4}, {-0.1, 3}}] & /@ polykites[4];
         Show[GraphicsArray[Partition[pol, 5]]]
```

From In[54]:=



The number of different polykites is given by the following table ([5, 6], seq. A057786).

```
In[56]:= Timing[Table[Length[polykites[n]], {n, 8}]]

Out[56]= {16.578 Second, {1, 2, 4, 10, 27, 85, 262, 873}}
```

We invite the reader to generate all six polykites that, besides having beautiful shapes, present for the first time a hollowed piece. Here is the corresponding code.

```
In[57]:= pol = draw[#, {{-1, 5}, {-1, 3}}] & /@ polykites[6];
         Show[GraphicsArray[Partition[pol, 17]]]
```

In the next part of this work, we will tile rectangles with polyominoes and introduce the family of *rep-tiles*, which presents a fascinating way of tiling the whole plane with specific polyominoes and polyiamonds.

■ Tiling Rectangles with Polyominoes

In this section, we approach the problem of tiling a rectangle by using pieces taken from a set of polyominoes. Naturally, the area of the rectangle has to be equal to the sum of the areas of the individual pieces in the set. Thus, we immediately know that it would be impossible to fill a rectangle of area 11 with dominoes or one of area 10 with triominoes, although it is not immediately apparent that we could fill a rectangle of arbitrary area (greater than 2) using dominoes and triominoes.

This section comprises three parts. The first one discusses tiling a rectangle allowing repetitions of the pieces taken from a given list. The second part will consider those tilings having no fault lines. Finally, in the last part we will tile using all pieces of a given set.

The function `tess` uses the backtrack method to construct all possible tilings of an $n \times m$ rectangle with pieces taken from a given list `poly`. The optional argument `justOneSolution` is included in case we do not want all solutions. The function `tess` uses the recursive function `tessAux`, which implements the backtrack mechanism: all possible candidates for the placement of the next piece are computed and the lexicographically smallest one is chosen. Each piece, including its rotations and reflections, is placed, and then `tessAux` is called recursively. As the method works faster if $n > m$, their values are swapped before and after computing the solutions. The function `getLines` takes a tiling and computes the endpoints of the horizontal and vertical lines forming the rightmost boundaries of the pieces of the tiling. The function `tile` displays a graphic array of the solutions of given width r.

```
In[59]:= lexic[p_] := Sort[p,
           (Im[#1] < Im[#2]) || ((Im[#1] == Im[#2]) && (Re[#1] ≤ Re[#2])) &]

         tess[{n_, m_}, poly_, justOneSolution_: False] :=
          Module[{avail, pieces, i, j, ans = {}, tessAux, na, ma},

           tessAux[partial_] := Module[{f, c, candidates, newp, k},
             candidates = Complement[avail, Flatten[partial]];
             If[candidates == {},
              AppendTo[ans, partial]; If[justOneSolution, Throw[1]],
              k = First[lexic[candidates]];
              Map[(newp = k + # - First[#];
                 If[(Complement[newp, avail] == {}) &&
                   (f = Flatten[{partial, newp}]; Length[f] == Length[Union[f]]),
                  tessAux[Append[partial, newp]]]) &, pieces]]];

           {na, ma} = If[n < m, {m, n}, {n, m}];
           pieces = lexic /@ Union[Flatten[allPieces /@ poly, 1]];
           avail = Flatten[Table[i + j i, {j, 0, na - 1}, {i, 0, ma - 1}]];
           Catch[tessAux[{}]];
           If[n < m, Map[m - 1 + i # &, ans], ans]]

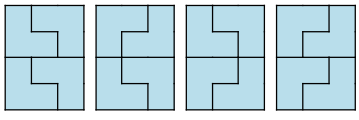
         getLines[tiling_] := Module[{p},
           Partition[Flatten[
             Map[(p = #; Map[{If[Not[MemberQ[p, # + 1]], {# + 1, # + 1 + i}, {}],
                    If[Not[MemberQ[p, # + i]], {# + i, # + 1 + i}, {}]} &,
                  p]) &, tiling]], 2]]

         tile[{n_, m_}, poly_, r_, justOneSolution_: False] := Module[{t, u, g},
           t = tess[{n, m}, poly, justOneSolution];
           g = Map[Graphics[Append[{{LightBlue, Rectangle[{0, 0}, {m, n}]},
                 Line[{{0, n}, {0, 0}, {m, 0}}]}, liC /@ getLines[#]]] &, t];
           Show[GraphicsArray[Partition[If[Mod[Length[t], r] == 0,
               g, Join[g, Table[Graphics[Point[{0, 0}]],
                 {r - Mod[Length[t], r]}]]], r]]];]
```

For instance, here is how to get all possible ways of tiling a 4×3 rectangle with the L-triomino.

```
In[63]:= tile[{4, 3}, {{0, i, 1}}, 4]
```

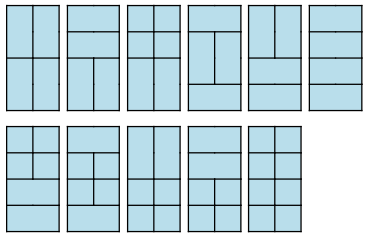
From In[63]:



Even nonpolyomino pieces are allowed. Consider, for instance, the "number 8" piece $\{0, 1 + i\}$, that is, two squares touching at a vertex, together with a domino $\{0, 1\}$ in the tiling of a 4×2 rectangle. Naturally, two number 8 pieces have to always be together forming a square in any tiling.

```
In[64]:= tile[{4, 2}, {{0, 1 + i}, {0, 1}}, 6]
```

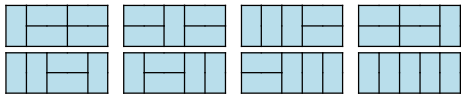
From In[64]:=



Here are all possible tilings of a 2×5 rectangle by dominoes.

```
In[65]:= tile[{2, 5}, polyominoes[2], 4]
```

From In[65]:=



In general, the number of tilings of $2 \times n$ rectangles by dominoes is well known to be the Fibonacci numbers.

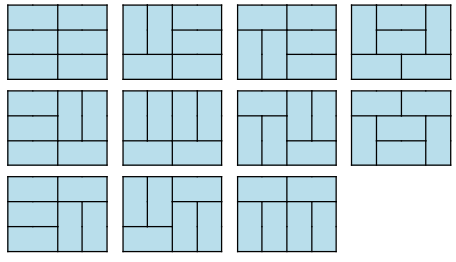
```
In[66]:= Timing[Table[Length[tess[{2, n}, polyominoes[2]]], {n, 15}]]
```

```
Out[66]= {2.547 Second,
           {1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987}}
```

Here are all possible tilings of a 3×4 rectangle by dominoes and the number of tilings of $3 \times n$ rectangles, $n \le 15$.

```
In[67]:= tile[{3, 4}, polyominoes[2], 4]
```

From In[67]:=



```
In[68]:= Timing[Table[Length[tess[{3, n}, polyominoes[2]]], {n, 15}]]
```

```
Out[68]= {50.141 Second, {0, 3, 0, 11, 0, 41, 0, 153, 0, 571, 0, 2131, 0, 7953, 0}}
```

Here is a generating function for these numbers [7].

```
In[69]:= Series[(3 - x)/(1 - 4 x + x^2), {x, 0, 6}]
```

$$\text{Out[69]= } 3 + 11\,x + 41\,x^2 + 153\,x^3 + 571\,x^4 + 2131\,x^5 + 7953\,x^6 + 0[x]^7$$

Or, more explicitly [7].

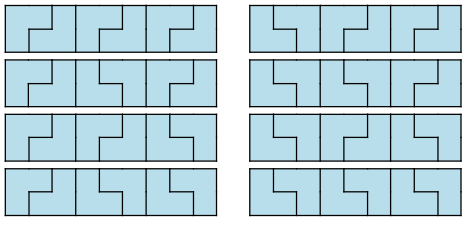
In[70]:=
$$\left\lfloor \frac{\left(2+\sqrt{3}\right)^{\text{Range}[10]}}{3-\sqrt{3}} \right\rfloor$$

Out[70]= {3, 11, 41, 153, 571, 2131, 7953, 29681, 110771, 413403}

Let us now explore the tiling of a 2×12 rectangle with the L-triomino.

In[71]:=
```
tile[{2, 9}, {Last[polyominoes[3]]}, 2]
```

From In[71]:=



The number of such tilings is somewhat expected.

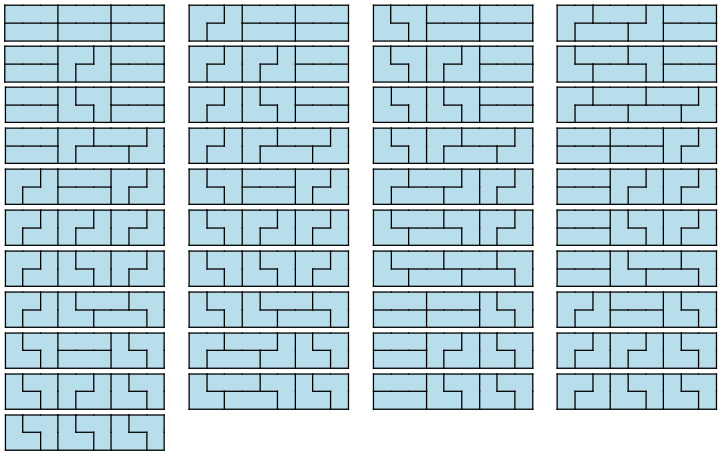
In[72]:=
```
Timing[
 Table[Length[tess[{n, 2}, {Last[polyominoes[3]]}]], {n, 15}]]
```

Out[72]= {0.312 Second, {0, 0, 2, 0, 0, 4, 0, 0, 8, 0, 0, 16, 0, 0, 32}}

Using the complete set of triominoes, we obtain the following ([5, 6], seq. A001835).

In[73]:=
```
tile[{2, 9}, polyominoes[3], 4]
```

From In[73]:=



In[74]:=
```
Timing[Table[Length[tess[{2, n}, polyominoes[3]]], {n, 15}]]
```

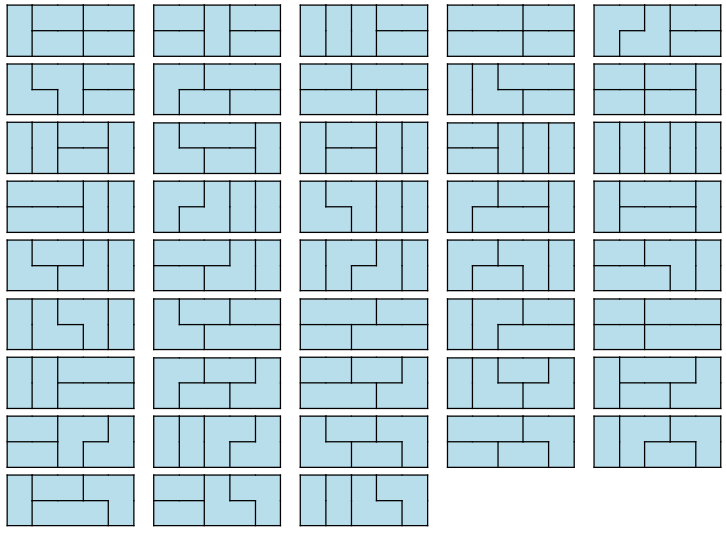
Out[74]= {2.235 Second, {0, 0, 3, 0, 0, 11, 0, 0, 41, 0, 0, 153, 0, 0, 571}}

Compare this sequence with the number of tilings of a $3\times n$ rectangle by dominoes.

We now consider both sets of dominoes and triominoes together.

```
In[75]:= tile[{2, 5}, Join[polyominoes[2], polyominoes[3]], 5]
```

From In[75]:=



```
In[76]:= Timing[Table[Length[
           tess[{2, n}, Join[polyominoes[2], polyominoes[3]]]], {n, 10}]]
```

```
Out[76]= {12.141 Second, {1, 2, 6, 17, 43, 108, 280, 727, 1875, 4832}}
```

Let us now compute the number of ways of tiling an $n\times m$ rectangle with dominoes. We know that $n\,m$ has to be even; the symmetry of the problem allows us to proceed as follows.

```
In[77]:= u = Table[0, {6}, {6}];
         Do[If[Mod[n m, 2] == 0, u[[n,m]] =
             u[[m,n]] = Length[tess[{n, m}, polyominoes[2]]]], {n, 6}, {m, n, 6}];
         MatrixForm[u]
```

Out[79]//MatrixForm=

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 2 & 3 & 5 & 8 & 13 \\ 0 & 3 & 0 & 11 & 0 & 41 \\ 1 & 5 & 11 & 36 & 95 & 281 \\ 0 & 8 & 0 & 95 & 0 & 1183 \\ 1 & 13 & 41 & 281 & 1183 & 6728 \end{pmatrix}$$

The elements of the diagonal correspond to [5, 6], seq. A004003. Here is the corresponding result for triominoes [8].

```
In[80]:= u = Table[0, {6}, {6}];
         Do[If[Mod[n m, 3] == 0, u[[n,m]] =
             u[[m,n]] = Length[tess[{n, m}, polyominoes[3]]]], {n, 6}, {m, n, 6}];
         MatrixForm[u]
```

Out[82]//MatrixForm=

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 3 & 0 & 0 & 11 \\ 1 & 3 & 10 & 23 & 62 & 170 \\ 0 & 0 & 23 & 0 & 0 & 939 \\ 0 & 0 & 62 & 0 & 0 & 8342 \\ 1 & 11 & 170 & 939 & 8342 & 80092 \end{pmatrix}$$

The contributions to this result by the individual triominoes are small.

```
In[83]:= u = Table[0, {6}, {6}];
         Do[If[Mod[n m, 3] == 0,
            u[[n,m]] = u[[m,n]] = Length[tess[{n, m}, {First[polyominoes[3]]}]]],
           {n, 6}, {m, n, 6}];
         MatrixForm[u]
```

Out[85]//MatrixForm=

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 2 & 3 & 4 & 6 \\ 0 & 0 & 3 & 0 & 0 & 13 \\ 0 & 0 & 4 & 0 & 0 & 22 \\ 1 & 1 & 6 & 13 & 22 & 64 \end{pmatrix}$$

```
In[86]:= u = Table[0, {6}, {6}];
         Do[If[Mod[n m, 3] == 0,
            u[[n,m]] = u[[m,n]] = Length[tess[{n, m}, {Last[polyominoes[3]]}]]],
           {n, 6}, {m, n, 6}];
         MatrixForm[u]
```

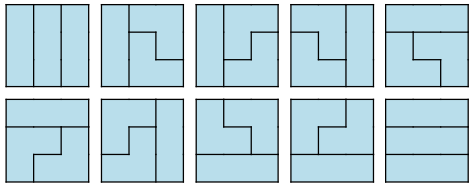
Out[88]//MatrixForm=

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 4 \\ 0 & 2 & 0 & 4 & 0 & 8 \\ 0 & 0 & 4 & 0 & 0 & 18 \\ 0 & 0 & 0 & 0 & 0 & 72 \\ 0 & 4 & 8 & 18 & 72 & 162 \end{pmatrix}$$

Although there are no 3×3 tilings by the L-triomino and only two by the straight triomino, together they manage to tile in 10 different ways.

```
In[89]:= tile[{3, 3}, polyominoes[3], 5]
```

From In[89]:=



In general, an $n \times m$ (m, $n \geq 2$) rectangle can be tiled with the L-triomino whenever $m\,n$ is divisible by 3, except when one is 3 and the other is odd [9]. Here are some more general results corresponding to other cases.

- An L-tetromino tiles an $n \times m$ rectangle iff n, $m > 1$ and 8 divides $n\,m$.
- The T-tetromino tiles an $n \times m$ rectangle iff 4 divides both n and m.
- If a rectangle can be tiled with k L-tetrominoes, then k is even.
- An $n \times m$ rectangle can be tiled by the straight p-omino iff p divides n or p divides m.

The next two examples show the time needed to get one solution versus all solutions.

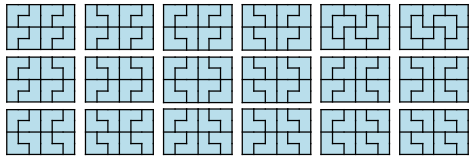
```
In[90]:= Timing[tile[{6, 11}, {{0, i, 2 i, 1 + 2 i, 3 i, 1 + 3 i}}, 4]]
Out[90]= {20.359 Second, Null}
In[91]:= Timing[tile[{6, 11}, {{0, i, 2 i, 1 + 2 i, 3 i, 1 + 3 i}}, 1, True]]
Out[91]= {1.219 Second, Null}
```

□ Fault-Free Tilings

As the number of different solutions grows, we would like to consider only those that have not appeared in smaller instances of the problem. For example, the tilings of a 4×6 rectangle by the L-triomino include all those appearing when tiling two separate 2×6 rectangles.

```
In[92]:= tile[{4, 6}, {{0, 1, i}}, 6]
```

From In[92]:=



In tiling a rectangle, we might generate fault lines. A fault line is any horizontal or vertical line that divides a tiling so that it can be regarded as the union of the tilings of two subrectangles [10]. The function `tileNF` computes all fault-free tilings of an $n \times m$ rectangle by a given set of pieces `poly`. As before, the graphic

array of the solutions is considered to be of a given width r. It generates the whole set of tilings and then selects the fault-free tilings, with the help of the predicate `noFaultLineQ`. This last function computes the length of the horizontal and vertical lines present in the tiling and verifies that no row or column has m or n elements, respectively.

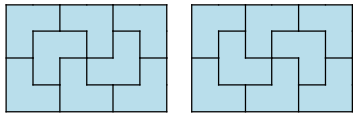
```
In[93]:= noFaultLineQ[{n_, m_}, tiling_] := Module[{l, h, v, c},
           l = getLines[tiling];
           h = Table[
             Length[Select[l, Im[First[#]] == Im[Last[#]] == c &]], {c, n - 1}];
           v = Table[Length[Select[l, Re[First[#]] == Re[Last[#]] == c &]],
             {c, m - 1}];
           Not[MemberQ[h, m] || MemberQ[v, n]]]

         tileNF[{n_, m_}, poly_, r_] := Module[{t, u, g},
           t = Select[tess[{n, m}, poly], noFaultLineQ[{n, m}, #] &];
           g = Map[Graphics[Append[{{LightBlue, Rectangle[{0, 0}, {m, n}]},
                 Line[{{0, n}, {0, 0}, {m, 0}}]}, liC /@ getLines[#]]] &, t];
           Show[GraphicsArray[Partition[If[Mod[Length[t], r] == 0,
               g, Join[g, Table[Graphics[Point[{0, 0}]],
                 {r - Mod[Length[t], r]}]]], r]];]
```

We use this predicate to select the tilings without fault lines appearing in the previous example.

```
In[95]:= tileNF[{4, 6}, {{0, 1, 1}}, 2]
```

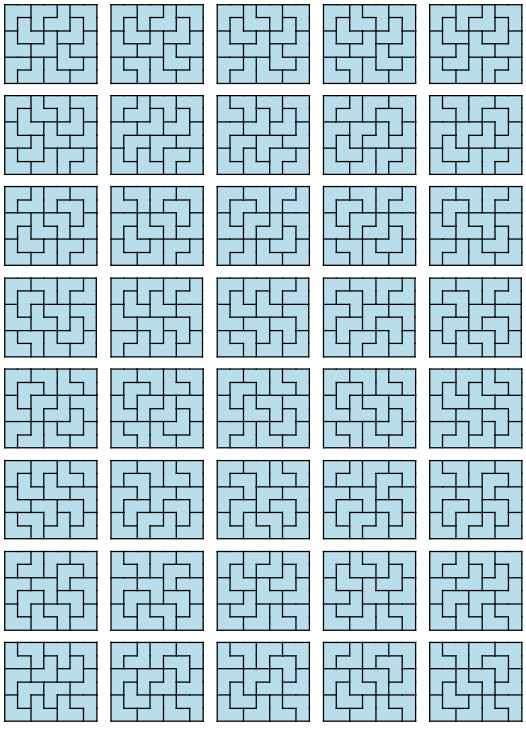
From In[95]:=



A more abundant example shows all fault-free tilings of a 6×7 rectangle using the L-triomino.

```
In[96]:= tileNF[{6, 7}, {Last[polyominoes[3]]}, 5]
```

From In[96]:=



Do tilings with dominoes always have fault lines? Not always.

```
In[97]:= Timing[u = Table[0, {7}, {7}];
          Do[If[Mod[n m, 2] == 0,
            u[[n,m]] = u[[m,n]] = Length[Select[tess[{n, m}, polyominoes[2]],
                noFaultLineQ[{n, m}, #] &]]], {n, 7}, {m, n, 7}];
          MatrixForm[u]]
```

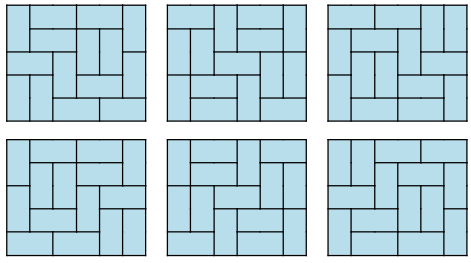
Out[97]=

$$\left\{425.109\,\text{Second},\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 6 & 0 \\ 0 & 0 & 0 & 0 & 6 & 0 & 124 \\ 0 & 0 & 0 & 0 & 0 & 124 & 0 \end{pmatrix}\right\}$$

Here are the smallest counterexamples.

```
In[98]:= tileNF[{5, 6}, polyominoes[2], 3]
```

From In[98]:=



Fault-free tilings produced by 3-ominoes are also interesting.

```
In[99]:= Timing[u = Table[0, {6}, {6}];
          Do[If[Mod[n m, 3] == 0,
            u[[n,m]] = u[[m,n]] = Length[Select[tess[{n, m}, polyominoes[3]],
                noFaultLineQ[{n, m}, #] &]], {n, 6}, {m, n, 6}];
          MatrixForm[u]]
```

$$\text{Out[99]=}\ \left\{1525.34\,\text{Second},\ \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 2 \\ 1 & 2 & 0 & 2 & 2 & 0 \\ 0 & 0 & 2 & 0 & 0 & 236 \\ 0 & 0 & 2 & 0 & 0 & 2060 \\ 0 & 2 & 0 & 236 & 2060 & 6312 \end{pmatrix}\right\}$$

Like the tilings without restrictions from the previous section, the contributions of the individual pieces to this total are minuscule.

```
In[100]:= Timing[u = Table[0, {9}, {9}];
          Do[If[Mod[n m, 3] == 0,
            u[[n,m]] = u[[m,n]] = Length[Select[tess[{n, m}, {First[polyominoes[3]]}],
                noFaultLineQ[{n, m}, #] &]], {n, 9}, {m, n, 9}];
          MatrixForm[u]]
```

$$\text{Out[100]=}\ \left\{1068.3\,\text{Second},\ \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 32 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 48 \\ 0 & 0 & 0 & 0 & 0 & 0 & 32 & 48 & 16 \end{pmatrix}\right\}$$

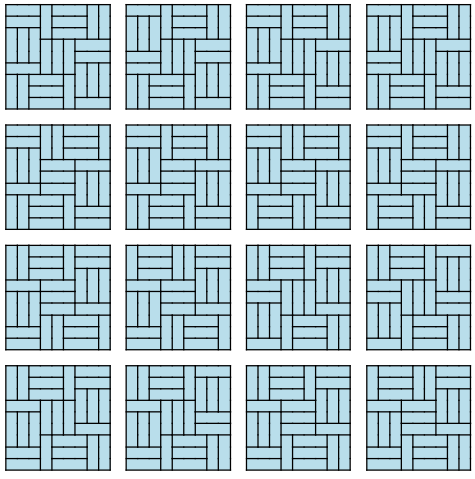
```
In[101]:= Timing[u = Table[0, {6}, {6}];
           Do[If[Mod[n m, 3] == 0,
             u[[n,m]] = u[[m,n]] = Length[Select[tess[{n, m}, {Last[polyominoes[3]]}],
                 noFaultLineQ[{n, m}, #] &]], {n, 6}, {m, n, 6}];
           MatrixForm[u]]
```

$$\text{Out[101]=}\ \left\{3.\ \text{Second},\ \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 8 \\ 0 & 0 & 0 & 2 & 8 & 2 \end{pmatrix}\right\}$$

The smallest examples of fault-free tilings of squares by each polyomino produce patterns of striking beauty.

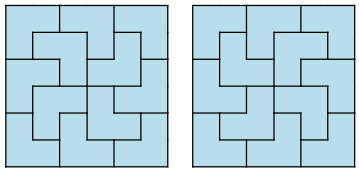
```
In[102]:= tileNF[{9, 9}, {First[polyominoes[3]]}, 4]
```

From In[102]:=



```
In[103]:= tileNF[{6, 6}, {Last[polyominoes[3]]}, 2]
```

From In[103]:=



□ Jigsaws

Let us now approach the problem of tiling an $n \times m$ rectangle using all the pieces of a list `poly` of given polyominoes once and only once. We naturally need to have `Length[Flatten[poly]]=n m`. The pieces appearing in `poly` do not need to be different and, with the inclusion of several equal ones, we can control their multiplicity. The functions `tessAll` and `jigsaw`, similar to our previous functions `tess` and `tile`, provide the necessary changes, which essentially amount to dropping a piece once used.

```
In[104]:= tessAll[{n_, m_}, poly_, justOne_: False] :=
           Module[{avail, i, j, ans = {}, tessAux, na, ma},

            tessAux[partial_, p_] :=
             Module[{f, c, i, candidates, newp, pieces, q, k},
              candidates = Complement[avail, Flatten[partial]];
              If[(p == {}) || (candidates == {}),
               AppendTo[ans, partial]; If[justOne, Throw[1]],
               k = First[lexic[candidates]];
               Do[
                q = lexic /@ allPieces[p[[i]]];
                Map[(
                   newp = k + # - First[#];
                   If[(Complement[newp, avail] == {}) && (f =
                       Flatten[{partial, newp}]; Length[f] == Length[Union[f]]),
                    tessAux[Append[partial, newp], Drop[p, {i}]]]
                  ) &, q], {i, Length[p]}]]];

            If[Length[Flatten[poly]] < n m, {},
             {na, ma} = If[n < m, {m, n}, {n, m}];
             avail = Flatten[Table[i + j I, {j, 0, na - 1}, {i, 0, ma - 1}]];
             Catch[tessAux[{}, poly]];
             If[n < m, Map[m - 1 + I # &, ans], ans]]]

          jigsaw[{n_, m_}, poly_, r_, justOneSolution_: False] :=
           Module[{t, u, g},
            t = tessAll[{n, m}, poly, justOneSolution];
            g = Map[Graphics[Append[{{LightBlue, Rectangle[{0, 0}, {m, n}]},
                  Line[{{0, n}, {0, 0}, {m, 0}}]}, liC /@ getLines[#]]] &, t];
            Show[GraphicsArray[Partition[If[Mod[Length[t], r] == 0,
                g, Join[g, Table[Graphics[Point[{0, 0}]],
                  {r - Mod[Length[t], r]}]]], r]];]

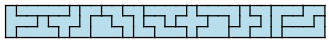
In[106]:= polyominoes[3]

Out[106]= {{0, I, 2 I}, {0, I, 1}}
```

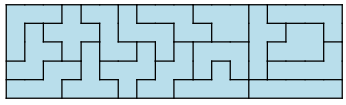
For instance, let us compute all possible tilings that also happen to be fault free using the set of pentominoes.

In[107]:=
```
n = 5;
p = n Length[polyominoes[n]];
s =
  Select[Map[{#, p/#} &, Divisors[p]], 1 < First[#] ≤ Last[#] &];
Timing[Map[jigsaw[#, polyominoes[n], 1, True] &, s];]
```

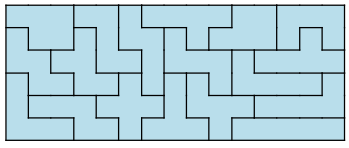
From In[107]:=



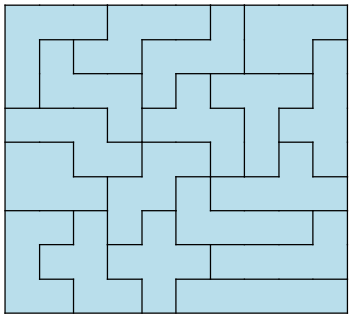
From In[107]:=



From In[107]:=



From In[107]:=

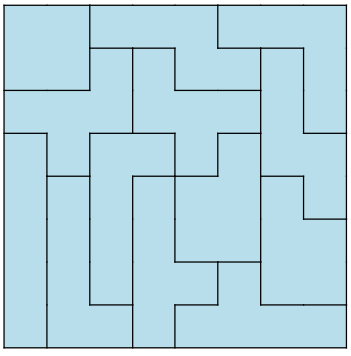


Out[110]=
```
{505.328 Second, Null}
```

We can generate a tiling of the 8×8 square with pentominoes by adding a square so that the sum of the areas is 64.

```
In[111]:= Timing[jigsaw[{8, 8}, Append[polyominoes[5], {0, 1, i, 1 + i}], 1, True]]
```

From In[111]:=



```
Out[111]= {209.859 Second, Null}
```

Unfortunately, the family of 35 hexominoes cannot tile a rectangle. In dealing with tilings with heptominoes, we have to consider that one of them has a hole and so we cannot tile a rectangle without a hole. For instance, in attempting to tile 12 8×8 squares (I thank the anonymous reviewer for this suggestion) in which a square has been removed from each one of them (so that their total area is 756, equal to the area covered by the 108 heptominoes), we just have to change the variable `avail` appearing in the function `tess` to construct the $8\times8-1$ squares appearing on the diagonal of a 96×96 square. This general approach takes too long and calls for another methodology, which the author hopes to report about in a future work.

■ Rep-Tiles

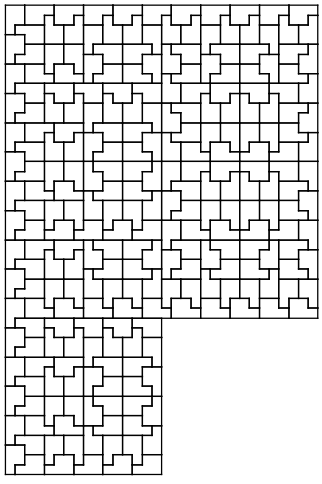
If a polyomino can be divided into a finite number of congruent copies similar to itself, we say that it is a *rep-tile*, or more specifically, an *n*-reptile. The so-called P-pentomino is a 4-reptile as we show later. Rep-tiles are self-similar pieces and so can be regarded as fractals. It can be proved that they tile the plane in a nonperiodic way [11], but it was only recently discovered, as reported by Roger Penrose in his fascinating and insightful book *Shadows of the Mind*, that there are three polyominoes that tile the plane only aperiodically [12].

```
In[112]:= Ppentomino[n_] := repP[{2, 0}, {0, 0}, {0, 6}, n]

          repP[a_, b_, c_, 1] :=
           {Line[{a, b, c, c + 2 (a - b), (6 a - 4 b + c)/3, a + (c - b)/3, a}]}
          repP[a_, b_, c_, n_] :=
           Join[repP[(a + c)/2, (b + c)/2, b, n - 1], repP[c + (a - b)/2, c, (b + c)/2, n - 1],
            repP[(a + c)/2, (3 a + b + 2 c)/6, (6 a - 4 b + c)/3, n - 1],
            repP[(3 a - 2 b + 5 c)/6, c + (a - b)/2, c + 2 (a - b), n - 1]]

In[115]:= Show[Graphics[Ppentomino[5]]]
```

From In[115]:=



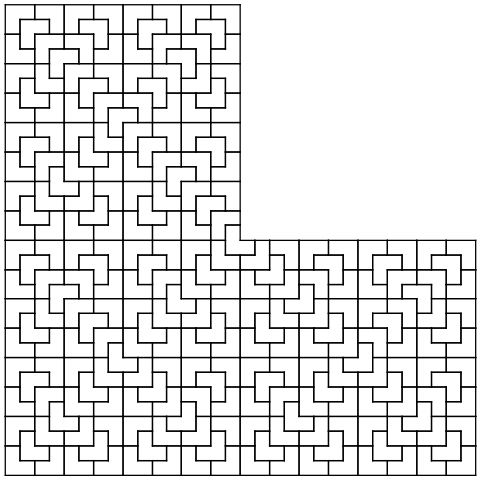
The following L-triomino is also a 4-reptile and therefore also a 4^n-reptile for all $n > 0$.

```
In[116]:= Ltriomino[n_] := repL[{2, 0}, {0, 0}, {0, 2}, n]

          repL[a_, b_, c_, 1] := {Line[{a, b, c, c + (a - b)/2, (a + c)/2, a + (c - b)/2, a}]}
          repL[a_, b_, c_, n_] :=
           Join[repL[(a + b)/2, b, (b + c)/2, n - 1], repL[(a + b)/2, a, a + (c - b)/2, n - 1],
            repL[(3 a + c)/4, (2 b + a + c)/4, (a + 3 c)/4, n - 1],
            repL[(b + c)/2, c, c + (a - b)/2, n - 1]]
```

In[119]:=
```
Show[Graphics[Ltriomino[5]]]
```

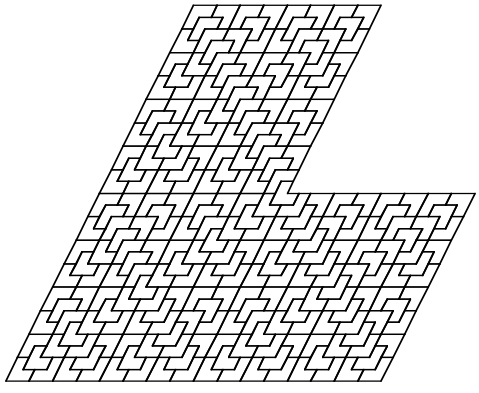
From In[119]:=



The value of c, given as an argument to the function `repL`, can be computed from those of a and b from $c = a + \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}.(b - a)$. Giving it explicitly lets us distort the piece at will.

In[120]:=
```
Show[Graphics[repL[{2, 0}, {0, 0}, {1, 2}, 5]]]
```

From In[120]:=



□ The Sphinx

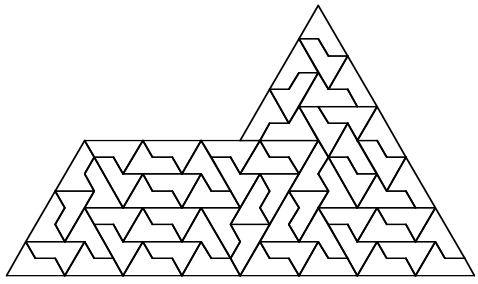
Rep-tiles also arise in the shape of polyiamonds. The sphinx is one of the most widely known 4-reptile polyiamonds.

```
In[121]:= sphinx[n_] := sph[{0, 0}, {12, 0}, {8, 4 Sqrt[3]}, n]

          sph[a_, b_, c_, 1] := {Line[{a, b, c, (2 a + b + 3 c)/6, (4 a - b + 3 c)/6, a}]}
          sph[a_, b_, c_, n_] := Join[sph[(a + b)/2, a, (4 a - b + 3 c)/6, n - 1],
            sph[b, (a + b)/2, (a + 2 b + 3 c)/6, n - 1], sph[(3 b + c)/4, c, (2 a + b + 3 c)/6, n - 1],
            sph[(4 a - b + 3 c)/6, (a + 2 b + 3 c)/6, (a + b)/2, n - 1]]

In[124]:= Show[Graphics[sphinx[4]]]
```

From In[124]:=



As indicated at the end of the previous section, more challenges are still present in the world of polyominoes. Even the seemingly simple task of finding out the number of tilings of an $n \times m$ rectangle using dominoes poses considerable difficulties (e.g., problem 7.51 in [7]). We can only guess as to the difficulty of these problems in the worlds inhabited by polyiamonds, polyhexes, and polykites. The advantages provided by the development of sophisticated languages like *Mathematica* yield a promising future for further investigations of this fascinating topic.

■ Acknowledgments

This work was completed while the author was a visiting scholar at Wolfram Research, Inc., whose assistance and enthusiastic support is gratefully acknowledged. I would also like to thank the University of Queretaro in Mexico for their continuous support.

■ References

[1] S. W. Golomb, *Polyominoes*: *Puzzles, Patterns, Problems, and Packings*, 2nd ed., Princeton, NJ: Princeton University Press, 1996.

[2] Eric W. Weisstein. "Polyomino." From *MathWorld*—A Wolfram Web Resource. http://mathworld.wolfram.com/Polyomino.html

[3] "*The Poly Pages*." (Feb 6, 2004) www.geocities.com/alclarke0/index.htm.

[4] M. Gardner, *Time Travel and Other Mathematical Bewilderments*, New York: W. H. Freeman & Co., 1987 pp. 175–187.

[5] N. J. A. Sloane. "The On-Line Encyclopedia of Integer Sequences." (Sep 2, 2004) www.research.att.com/~njas/sequences.

[6] N. J. A. Sloane and S. Plouffe, *The Encyclopedia of Integer Sequences*, San Diego: Academic Press, 1995.

[7] R. L. Graham, O. Ptashnik, and D. E. Knuth, *Concrete Mathematics*: *A Foundation for Computer Science*, Reading, MA: Addison-Wesley, 1989.

[8] T. P. Chu and R. Johnsonbaugh, "Tiling Boards with Triominoes," *Journal of Recreational Mathematics*, **18**, 1985–1986 pp. 183–193.

[9] G. E. Martin, *Polyominoes*: *A Guide to Puzzles and Problems in Tiling*, Washington, DC: The Mathematical Association of America, 1996.

[10] D. A. Klarner, "Polyominoes" *Handbook of Discrete and Computational Geometry*, Boca Raton, FL: CRC Press, 1997 pp. 225–239.

[11] B. Brunbaum and G. C. Shephard, *Tilings and Patterns*, New York: W. H. Freeman & Co., 1987.

[12] R. Penrose, *Shadows of the Mind: A Search for the Missing Science of Consciousness*, Oxford: Oxford University Press, 1994.

About the Author

Jaime Rangel-Mondragón earned M.Sc. and Ph.D. degrees in applied mathematics and computation from the University College of North Wales in Bangor, Great Britain. He has held research positions in the School of Computer Science at UCNW, the College of Mexico, the Center of Research and Advanced Studies, the Monterrey Institute of Technology, and the University of Queretaro in Mexico, where he is presently a member of the Computer Science faculty. As a prolific contributor to *MathSource*, his current research interests include recreational mathematics, combinatorics, the theory of computing, computational geometry, and functional languages.

Jaime Rangel-Mondragón
Facultad de Informática
Universidad Autónoma de Querétaro
Mexico
jrangel@uaq.mx

Algebraic Construction of Smooth Interpolants on Polygonal Domains

Elisabeth Anna Malsch
Gautam Dasgupta

A smooth and bounded interpolant can be constructed in explicit algebraic form within any polygon, convex or concave. The resulting function is not unique and accordingly can be adjusted to satisfy desired global conditions, such as linear fields. The closed-form representation is obtained by combining simple geometric descriptions, such as the side lengths and areas. The interpolant distributes values given at discrete nodes smoothly over the interior of the domain. On a convex polygon, the interpolant is a rational function of the product of areas. On a concave or multiply connected polygon, the interpolant is a function of areas and edge lengths, which introduces a square root term.

■ Introduction

There are many varied applications for a smooth and bounded interpolant. The most straightforward is for computer graphics, for example in rendering and coloring applications [1]. Smoothly distributing color defined at nodes over a domain requires a bounded interpolating function. Interpolants can also be employed as test functions for computational modeling [2]. For the finite element method, they act as shape functions [3, 4]. Similar interpolants have been applied to the analysis of biomedical growth and shape change [5].

Conventional methods for constructing interpolants do not distribute boundary values smoothly over arbitrary polygonal shapes. Most either require a mesh or apply only to very simple geometries [4, 6]. For example, the conventional finite element development does not specifically address concavity; instead, concave domains are tessellated into convex parts. An exception is the R-function construction that applies to any polygonal domain [7]. Unlike the proposed formulation, it can only represent homogeneous boundary conditions.

In general, any function $u(x, y)$ can be approximated as the sum of the products of the value u_i at given nodes (x_i, y_i) and an interpolant $N_i(x, y)$:

$$u(x, y) \simeq \hat{u}(x, y) = \sum_{i=1}^{n} u_i N_i(x, y). \tag{1}$$

The derived interpolants $N_i(x, y)$ are linearly independent:

$$N_i(x_j, y_j) = \delta_{i,j}. \tag{2}$$

Along any given boundary, only the values given at the end nodes prescribe the behavior along the line segment:

$$\hat{u}(t x_i + (1 - t) x_j, t y_i + (1 - t) y_j) = f(u_i, u_j, t). \tag{3}$$

Within the domain, the interpolants are bounded, normalized between zero and one, and smooth. The interpolant generated is not unique. Any function that satisfies an elliptic operator and the boundary conditions satisfies the interpolant requirements.

Using the same method to construct general bounded interpolants on any polygon, linear interpolants are constructed. A linear interpolant satisfies the additional requirements of boundary linearity:

$$\hat{u}(t x_i + (1 - t) x_j, t y_i + (1 - t) y_j) = t u_i + (1 - t) u_j, \tag{4}$$

and exact reproduction of linear fields:

$$x = \sum_{i=1}^{n} x_i N_i(x, y) \qquad \text{and} \qquad y = \sum_{i=1}^{n} y_i N_i(x, y). \tag{5}$$

The constant field is satisfied by construction:

$$1 = \sum_{i=1}^{n} N_i(x, y). \tag{6}$$

In most cases, the linearity requirements do not prescribe a unique interpolant. Therefore additional global conditions, such as higher-order fields, could also be imposed.

■ Input Data

The interpolant can be constructed on any two-dimensional slice of any polytope, including polygons, polyhedra, or higher-dimensional objects. Accordingly, the dimension of the data must be defined. The examples are in two dimensions.

```
In[1]:= xy = {x, y};
```

The list of vertices must also be defined; for example, here is a triangle with three nodes.

```
In[2]:= vertexConnect = {{1, 2, 3}};
```

Another example is a triangle enclosing a quadrilateral and an interior point.

```
In[3]:= vertexConnect = {{1, 2, 3}, {4}, {8, 7, 6, 5}};
```

For a simple interpolant, the quantity that is going to be distributed smoothly over the domain must be defined at every nodal point. The nodal points not only determine the structure of the domain, they are also the points at which the value being distributed is given. The vertices of the polygon are a subset of the nodal points. Nodal points and interpolation points are one and the same.

■ Geometric Measures and Support Functions

It is useful to construct the interpolant with respect to geometric measures as opposed to nodal locations. The benefit of such an approach is that it is invariant of coordinate system and dimension.

□ The Distance Function

The distance between two points p_1 and p_2 with coordinates (x_1, y_1) and (x_2, y_2) is:

$$| p_1\ p_2 | = l_{12} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}\ . \tag{7}$$

The function can be written as the norm of the vector connecting the points, `norm[`$p_1 - p_2$`]`. The built-in `Norm` function would be appropriate if the points were represented as complex numbers rather than as pairs of real numbers.

```
In[4]:= norm[a_] := Sqrt[a.a]
```

□ The Signed Area of a Triangle

The area of the triangle with vertices p_1, p_2, p_3 with coordinates (x_1, y_1), (x_2, y_2), (x_3, y_3) can be defined in terms of the determinant:

$$A = \Delta[p_1, p_2, p_3] = 1/2 \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}. \tag{8}$$

The area is signed: $\Delta[p_1, p_2, p_3] = -\Delta[p_2, p_1, p_3]$.

```
In[5]:= area[p_, q_, r_] :=
         1 / 2 Det[Join[Transpose[{p, q, r}], {{1, 1, 1}}]] /;
          Length[p] == Length[q] == Length[r] == 2
```

□ The Unsigned Area Function

The unsigned area of a triangle can be written in terms of side lengths using Heron's formula:

$$A = \sqrt{s\,(s-a)\,(s-b)\,(s-c)}\,, \tag{9}$$

where

$$s = \frac{1}{2}\,(a+b+c). \tag{10}$$

The resulting area is necessarily nonnegative. The area measure has no physical meaning for a node located on the edge of the domain. In such a case, defining the unsigned area to be one when the signed area is zero allows the concave element shape function routines to apply to elements with side nodes.

```
In[6]:= unsignedArea[p_, q_, r_] := Module[{a, b, c, s},
          a = norm[p - q]; b = norm[q - r]; c = norm[r - p]; s = 1 / 2 (a + b + c);
          Sqrt[s (s - a) (s - b) (s - c)]]

In[7]:= unsignedArea[p_, q_, r_] := 1 /; area[p, q, r] == 0
```

□ The Angle Functions

The cosine or sine of an angle defined by three vertices can be determined using line lengths and the unsigned area. Care is taken to choose the proper sign for the cosine and sine functions. For the angle β defined by the nodes p, q, r with the angle at q:

$$\cos(\beta) = \frac{|pq|^2 + |qr|^2 - |rp|^2}{2\,|pq|\,|qr|}, \tag{11}$$

$$\sin(\beta) = \frac{2\,|\Delta[p, q, r]|}{|pq|\,|qr|}. \tag{12}$$

The angle β never has to be evaluated.

```
In[8]:= cosAng[p_, q_, r_] := Module[{pq, qr, rp}, pq = norm[p - q];
          qr = norm[q - r]; rp = norm[r - p]; (pq^2 + qr^2 - rp^2) / (2 pq qr)]

In[9]:= sinAng[p_, q_, r_] := 2 unsignedArea[p, q, r] / (norm[p - q] norm[q - r])
```

■ Minimum Functions

The geometric definitions can be used to define minimum functions. For example, the distance from node p to node q, $|pq|$, is zero and minimized when $p = q$ and the area, defined by the three nodes p, q, r, is zero and minimized when p is on the line qr. The R-function formulation is similarly dependent on minimum functions [7].

Define a function of the variable point $z(x, y) = \{x, y\}$ that is strictly zero along the line qr:

$$|z(x, y) - q| + |z(x, y) - r| - |q - r|. \tag{13}$$

The equation applies in any dimension.

```
In[10]:= perim[xy_, q_, r_] := norm[xy - q] + norm[xy - r] - norm[q - r]

In[11]:= Show[#, DisplayFunction → $DisplayFunction,
           ImageSize → {200, Automatic}, PlotRange → All] & /@
         Thread[{ContourPlot[#[{x, y}, {-1 / 2, -1 / 2}, {1 / 2, 1 / 2}],
             {x, -1, 1}, {y, -1, 1}, DisplayFunction → Identity,
             PlotPoints → 33, PlotRange → {0, 1}] & /@
           {norm[#1 - {0, 0}] &, area, unsignedArea, perim},
          {Graphics[{PointSize[.02], RGBColor[0, 0, 1], Point[{0, 0}]}],
           Graphics[{PointSize[.02], RGBColor[0, 0, 1],
             Line[{{-1, -1}, {1, 1}}]}], Graphics[
            {PointSize[.02], RGBColor[0, 0, 1], Line[{{-1, -1}, {1, 1}}]}],
           Graphics[{PointSize[.02], RGBColor[0, 0, 1],
             Line[{{-1 / 2, -1 / 2}, {1 / 2, 1 / 2}}]}]}}]
```

From In[11]:=

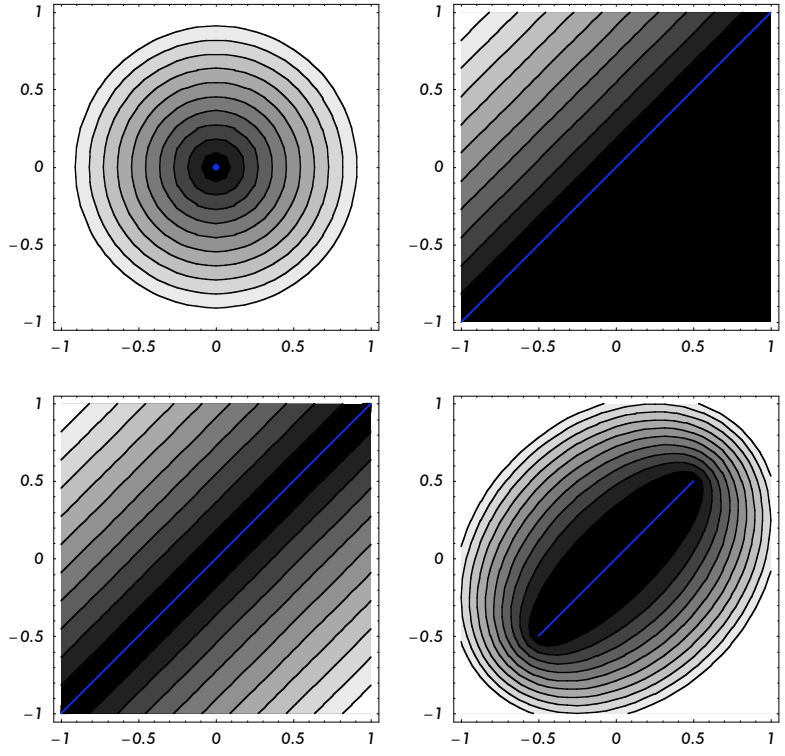


Figure 1. Contours that are minimum at a point, along a line, and along a line segment.

□ Linearity

The perimeter function can be modified so that the contour lines are linear along a broken line segment. Four nodal points are required: say, o, p, q, r. Construct a function that is zero along the path pq and linear along the paths op and qr, with angles $\alpha = \angle opq$ and $\beta = \angle pqr$. Again define the variable point $z(x, y) = \{x, y\}$:

$$A(x, y) = |pq|^2 - 2\cos(\beta)\,|qz(x, y)|\,|pq| - |pz(x, y)|^2 + |qz(x, y)|^2, \tag{14}$$

$$B(x, y) = |pq|^2 - 2\cos(\alpha)\,|pz(x, y)|\,|pq| + |pz(x, y)|^2 - |qz(x, y)|^2. \tag{15}$$

The functions $c_1(x, y)$, $c_2(x, y)$, and $c_3(x, y)$ are defined in terms of $A(x, y)$ and $B(x, y)$:

$$c_1(x, y) = \frac{1}{2}\,(A(x, y)\sin(\alpha) + B(x, y)\sin(\beta))\,|pq|, \tag{16}$$

$$c_2(x, y) = A(x, y) + B(x, y), \tag{17}$$

$$c_3(x, y) = (A(x, y)\cos(\alpha) + B(x, y)\cos(\beta))\,|pq|. \tag{18}$$

Here we combine all the pieces:

$$\frac{c_1(x, y)\,(|pz(x, y)| + |z(x, y)\,q|)^2 - |pq|^2}{c_2(x, y)\,(|pz(x, y)| + |z(x, y)\,q|) - c_3(x, y)}. \tag{19}$$

```
In[12]:= perimLin[xy_, o_, p_, q_, r_] :=
          Module[{A, B, pq, xyq, pxy, c1, c2, c3},
           pq = norm[p - q];
           xyq = norm[xy - q];
           pxy = norm[p - xy];
           A = pq^2 - 2 cosAng[p, q, r] xyq pq - pxy^2 + xyq^2;
           B = pq^2 - 2 cosAng[o, p, q] pxy pq + pxy^2 - xyq^2;
           c1 = 1 / 2 (A sinAng[o, p, q] + B sinAng[p, q, r]) pq;
           c2 = A + B;
           c3 = (A cosAng[o, p, q] + B cosAng[p, q, r]) pq;
           c1 ((xyq + pxy)^2 - pq^2) / ((xyq + pxy) c2 - c3)]
```

In[13]:=
```
Show[{ContourPlot[
     Evaluate[perimLin[{x, y}, #[[1]], {-1/2, 0}, {1/2, 0}, #[[2]]]],
     {x, -1, 1}, {y, -1, 1}, DisplayFunction → Identity,
     PlotPoints → 40], Graphics[{RGBColor[1, 0, 0],
      Line[{#[[1]], {-1/2, 0}}], Line[{{1/2, 0}, #[[2]]}],
      RGBColor[0, 0, 1], Line[{{-1/2, 0}, {1/2, 0}}]}]},
   DisplayFunction → $DisplayFunction, ImageSize → {200, Automatic},
   PlotRange → {{-1, 1}, {-1, 1}}] & /@
 {{{-1, -2}, {1, 1/2}}, {{0, 1}, {1/10, -1/10}}}
```

From In[13]:=

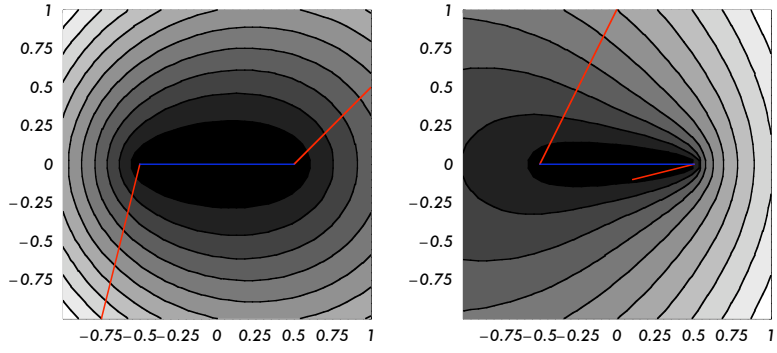


Figure 2. Contours of a function that are minimized along a line segment (in blue) and linear along two adjacent broken line segments (in red).

■ Functions for Displaying Interpolants

In[14]:=
```
outline[pts_] := Line[Append[#, #[[1]]] &[pts]]
name[pts_] :=
 Table[Text[ToString[i], pts[[i]], {-2, 0}, Background → Hue[.2],
   TextStyle → {FontWeight → "Bold"}], {i, Length[pts]}]
nodes[pts_] := Point /@ pts
mask[pts__] := Polygon[Flatten[
   Append[#, #[[1]]] & /@ {{{-.1, -.1}, {-.1, .1}, {.1, .1}, {.1, -.1}} +
      ({#[[1]], #[[2]], #[[4]], #[[3]]} &[Flatten[Outer[List, ##] & @@
          ({Min[#], Max[#]} & /@ Thread[pts]), 1]]), pts}, 1]]
output[i_, colorfunction_, f_, points_, cnct_, j_: 1] :=
 Module[{fc, pContour, pts}, {
   pts = Part[points, cnct[[1]]];
   fc = Compile[{{x, _Real}, {y, _Real}},
     Evaluate[f[{x, y}, points, cnct, j]]];
   pContour = ContourPlot[Check[fc[x, y][[i]],
      0., CompiledFunction::"cfn"],
     {x, Min[Thread[points][[1]]], Max[Thread[points][[1]]]},
     {y, Min[Thread[points][[2]]], Max[Thread[points][[2]]]},
     DisplayFunction → Identity, (*ContourShading→False*)
```

```
 ColorFunction → colorfunction, Contours → 20,
 PlotPoints → 60, PlotRange → {0, 1}]; Show[pContour,
Graphics[Flatten[{{RGBColor[1, 1, 1]}, {mask[pts]},
   Polygon /@ (points[[#]] & /@ Drop[cnct, 1])}]],
Graphics[Flatten[{{PointSize[.02]}, {outline /@
     (points[[#]] & /@ cnct), name[points], nodes[points]}}]],
PlotRange → All, Frame → False, AspectRatio → 1]}]
```

■ Interpolants for Convex Polygons

The interpolant is constructed as a combination of the geometric measures.

```
In[19]:= convex[{x_, y_}, points_, cnct_, i_] :=
          Module[{pts = Part[points, cnct[[i]]],
            areaTog, n, varArea, zeroArea}, n = Length[pts];
           areaTog = Apply[area, Partition[pts, 3, 1, {2}], {1}];
           varArea = Apply[area[##, {x, y}] &, Partition[pts, 2, 1, {1}], {1}];
           zeroArea = Apply[Times,
             Partition[varArea, Length[pts] - 2, 1, {Length[pts]}], {1}];
           (zeroArea * areaTog) / (zeroArea.areaTog)]
```

Interpolants are rational functions. For example, here is the set of functions for a triangle, rectangle, skew quadrilateral, and pentagon.

```
In[20]:= convex[{x, y}, {{0, 0}, {1, 0}, {1, 1}}, {{1, 2, 3}}, 1] // Simplify
```

```
Out[20]= {1 - x, x - y, y}
```

```
In[21]:= convex[{x, y}, {{0, 0}, {1, 0}, {1, 1 / 2}, {0, 1 / 2}},
           {{1, 2, 3, 4}}, 1] // Simplify
```

```
Out[21]= {(-1 + x) (-1 + 2 y), x - 2 x y, 2 x y, -2 (-1 + x) y}
```

```
In[22]:= convex[{x, y}, {{0, 0}, {1, 0}, {3 / 2, 1}, {-1 / 2, 1 / 4}},
           {{1, 2, 3, 4}}, 1] // Simplify
```

$$\text{Out[22]}= \left\{-\frac{(7+6x-16y)(-2+2x-y)}{14+12x+59y},\ \frac{2(7+6x-16y)(x+2y)}{14+12x+59y},\ \frac{26y(x+2y)}{14+12x+59y},\ -\frac{28(-2+2x-y)y}{14+12x+59y}\right\}$$

```
In[23]:= convex[{x, y}, {{0, 0}, {1, 0}, {3 / 2, 1 / 2}, {-1 / 4, 1}, {-1 / 2, 1 / 4}},
           {Range[5]}, 1] // Simplify
```

Out[23]=
$$\left\{\frac{(7+12x-4y)(-1+x-y)(-13+4x+14y)}{91-48x^2+487y+198y^2+2x(64+33y)},\right.$$
$$-\frac{(7+12x-4y)(x+2y)(-13+4x+14y)}{91-48x^2+487y+198y^2+2x(64+33y)},$$
$$\frac{18(7+12x-4y)y(x+2y)}{91-48x^2+487y+198y^2+2x(64+33y)},$$
$$-\frac{184(-1+x-y)y(x+2y)}{91-48x^2+487y+198y^2+2x(64+33y)},$$
$$\left.\frac{28(-1+x-y)y(-13+4x+14y)}{91-48x^2+487y+198y^2+2x(64+33y)}\right\}$$

In[24]:=
```
Show[GraphicsArray[
 Partition[Graphics[#[[1, 1]], PlotRange → All, AspectRatio → 1] & /@
   {output[1, (RGBColor[1, (1 - #), (1 - #)] &),
     convex, {{0, 0}, {1, 0}, {1, 1}}, {{1, 2, 3}}],
    output[1, (RGBColor[(1 - #), 1, (1 - #)] &), convex,
     {{0, 0}, {1, 0}, {1, 1 / 2}, {0, 1 / 2}}, {{1, 2, 3, 4}}],
    output[1, (RGBColor[(1 - #), (1 - #), 1] &), convex,
     {{0, 0}, {1, 0}, {3 / 2, 1}, {-1 / 2, 1 / 4}}, {{1, 2, 3, 4}}],
    output[1, (RGBColor[1, (1 - #), 1] &), convex, {{0, 0}, {1, 0},
      {3 / 2, 1 / 2}, {-1 / 4, 1}, {-1 / 2, 1 / 4}}, {Range[5]}]},
  2]], ImageSize → {350, Automatic}]
```

From In[24]:=

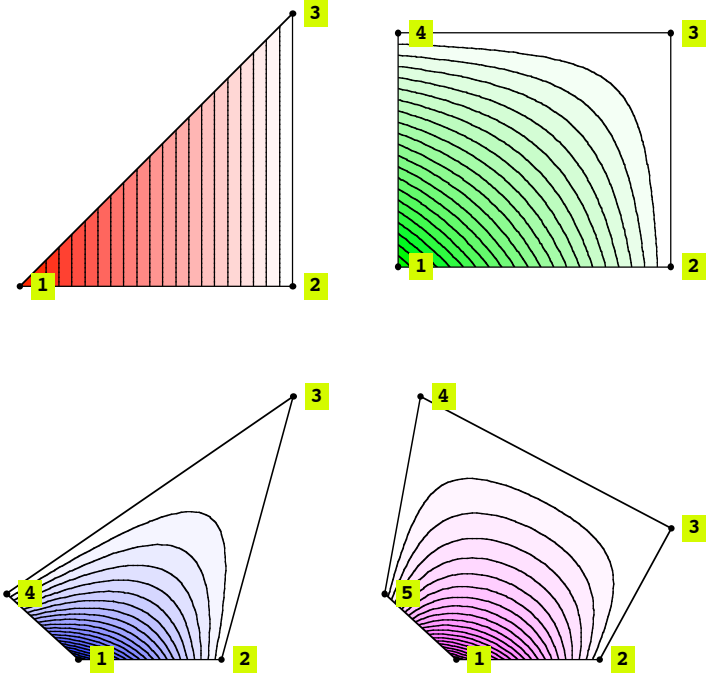


Figure 3. Interpolants on a triangle, rectangle, skew quadrilateral, and pentagon.

■ Interior Points

Points inside the domain are dealt with using the distance function.

In[25]:=
```
interior[{x_, y_}, points_, cnct_, 1] :=
 Module[{pts = points[[Flatten[Select[cnct, (Length[#] == 1 &)]]]]},
  zeroDist =
   Apply[Times, Partition[(norm[{x, y} - #] & /@ pts), Length[pts] - 1,
     1, {Length[pts]}], {1}]; zeroDist / (Plus @@ zeroDist)]
```

In[26]:=
```
points = {{-.1, -.1}, {1.1, -.1},
   {1.1, 1.1}, {-.1, 1.1}, {0, 0}, {1, 1}, {1, 0}};
cnct = {{1, 2, 3, 4}, {5}, {6}, {7}};
interior[{x, y}, points, cnct, 1][[1]]
Show[output[1, (RGBColor[1, 1, (1 - #)] &), interior, points, cnct],
 DisplayFunction → $DisplayFunction]
```

Out[28]=

$$\left(\sqrt{(-1+x)^2+(-1+y)^2}\ \sqrt{(-1+x)^2+y^2}\right) \Big/ \left(\sqrt{(-1+x)^2+(-1+y)^2}\ \sqrt{(-1+x)^2+y^2}+\sqrt{(-1+x)^2+(-1+y)^2}\ \sqrt{x^2+y^2}+\sqrt{(-1+x)^2+y^2}\ \sqrt{x^2+y^2}\right)$$

From In[26]:=

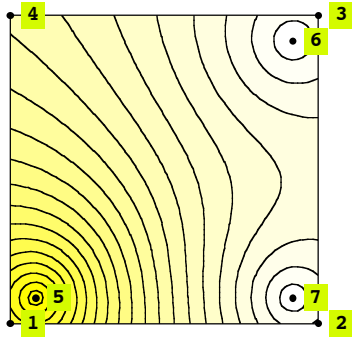


Figure 4. Interior points.

■ Concave Polygons

For concave polygons, the `zeroArea[`a`,`b`,`c`]` function is replaced with the `zeroPerim[`a`,`b`,`c`]` function.

In[30]:=
```
varPerim[xy_, points_, cnct_, i_] :=
  Module[{pts = Part[points, cnct[[i]]]},
   Apply[perim[xy, ##] &, Partition[pts, 2, 1, {1}], {1}]];
```

```
In[31]:= concave[{x_, y_}, points_, cnct_, i_] :=
          Module[{pts = Part[points, cnct[[i]]], n, zeroPerim},
           n = Length[pts];
          zeroPerim = Apply[Times,
              Partition[varPerim[{x, y}, pts, cnct, i], n - 2, 1, {n}], {1}];
          (zeroPerim) / Apply[Plus, zeroPerim]]
```

The shape functions for a concave quadrilateral can easily be constructed. The interpolant contains a square root. The branch points of the square root terms add a discontinuity in the gradient of the interpolant. This discontinuity is necessary for the modeling of re-entrant corners.

```
In[32]:= points = {{-1, -1}, {0, 0}, {1, -1}, {0, 1}};
         cnct = {{1, 2, 3, 4}};
         concave[{x, y}, points, cnct, 1] // Simplify
```

$$
\begin{aligned}
\text{Out[34]}= \Bigg\{ & \Big(\Big(-\sqrt{5}+\sqrt{x^2+(-1+y)^2}+\sqrt{(-1+x)^2+(1+y)^2}\Big) \\
& \Big(-\sqrt{2}+\sqrt{x^2+y^2}+\sqrt{(-1+x)^2+(1+y)^2}\Big)\Big) \Big/ \\
& \Big(-\sqrt{2}-\sqrt{5}+\sqrt{x^2+(-1+y)^2}+\sqrt{x^2+y^2}+ \\
& \quad \sqrt{(-1+x)^2+(1+y)^2}+\sqrt{(1+x)^2+(1+y)^2}\Big)^2, \\
& \Big(\Big(-\sqrt{5}+\sqrt{x^2+(-1+y)^2}+\sqrt{(-1+x)^2+(1+y)^2}\Big) \\
& \Big(-\sqrt{5}+\sqrt{x^2+(-1+y)^2}+\sqrt{(1+x)^2+(1+y)^2}\Big)\Big) \Big/ \\
& \Big(-\sqrt{2}-\sqrt{5}+\sqrt{x^2+(-1+y)^2}+\sqrt{x^2+y^2}+ \\
& \quad \sqrt{(-1+x)^2+(1+y)^2}+\sqrt{(1+x)^2+(1+y)^2}\Big)^2, \\
& \Big(\Big(-\sqrt{5}+\sqrt{x^2+(-1+y)^2}+\sqrt{(1+x)^2+(1+y)^2}\Big) \\
& \Big(-\sqrt{2}+\sqrt{x^2+y^2}+\sqrt{(1+x)^2+(1+y)^2}\Big)\Big) \Big/ \\
& \Big(-\sqrt{2}-\sqrt{5}+\sqrt{x^2+(-1+y)^2}+\sqrt{x^2+y^2}+ \\
& \quad \sqrt{(-1+x)^2+(1+y)^2}+\sqrt{(1+x)^2+(1+y)^2}\Big)^2, \\
& \Big(\Big(-\sqrt{2}+\sqrt{x^2+y^2}+\sqrt{(-1+x)^2+(1+y)^2}\Big) \\
& \Big(-\sqrt{2}+\sqrt{x^2+y^2}+\sqrt{(1+x)^2+(1+y)^2}\Big)\Big) \Big/ \\
& \Big(-\sqrt{2}-\sqrt{5}+\sqrt{x^2+(-1+y)^2}+\sqrt{x^2+y^2}+ \\
& \quad \sqrt{(-1+x)^2+(1+y)^2}+\sqrt{(1+x)^2+(1+y)^2}\Big)^2 \Bigg\}
\end{aligned}
$$

```
In[35]:= Show[GraphicsArray[
           Graphics[#[[1, 1]], PlotRange → All, AspectRatio → 1] & /@
            {output[1, (RGBColor[1, (1 - #), (1 - #)] &), concave,
              {{-1, -1}, {0, 0}, {1, -1}, {0, 1}}, {{1, 2, 3, 4}}],
             output[1, (RGBColor[1, 1, (1 - #)] &), concave,
              {{0, 1/2}, {1, 0}, {3/2, 1/2}, {-1/4, 1}, {-1/2, 1/4}},
              {{1, 2, 3, 4, 5}}]}], ImageSize → {350, Automatic}]
```

From In[35]:=

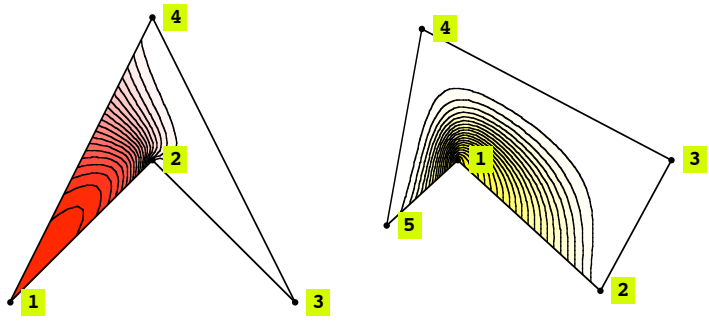


Figure 5. Interpolants on concave polygons do not require tesselation.

■ Concave Polygons with Linearity on the Sides

The `perim[a,b,c]` function is replaced with `perimLin[d,a,b,c,e]`. Similarly the `varPerim[x,y]` function is modified so that four vertices define the behavior of the minimum along a line. The constructed interpolants on the quadrilateral and pentagon are linear on the sides.

```
In[36]:= concaveLin[{x_, y_}, points_, cnct_, i_] :=
          Module[{pts = Part[points, cnct[[i]]], zeroPLin, varPLin, areaTog},
           areaTog = Apply[unsignedArea, Partition[pts, 3, 1, {2}], {1}];
           n = Length[pts];
           varPLin =
            Apply[perimLin[{x, y}, ##] &, Partition[pts, 4, 1, {2}], {1}];
         zeroPLin = Apply[Times, Partition[varPLin,
              Length[pts] - 2, 1, {Length[pts]}], {1}];
         (zeroPLin * areaTog) / (areaTog.zeroPLin)]
```

The interpolants for a quadrilateral can be constructed as follows.

```
In[37]:= Show[GraphicsArray[
           Graphics[#[[1, 1]], PlotRange → All, AspectRatio → 1] & /@
            {output[1, (RGBColor[(1 - #), (1 - #), 1] &), concaveLin,
              {{-1, -1}, {0, 0}, {1, -1}, {0, 1}}, {{1, 2, 3, 4}}],
             output[1, (RGBColor[(1 - #), 1, (1 - #)] &), concaveLin,
              {{0, 1/2}, {1, 0}, {3/2, 1/2}, {-1/4, 1}, {-1/2, 1/4}},
              {{1, 2, 3, 4, 5}}]}], ImageSize → {350, Automatic}]
```

From In[37]:=

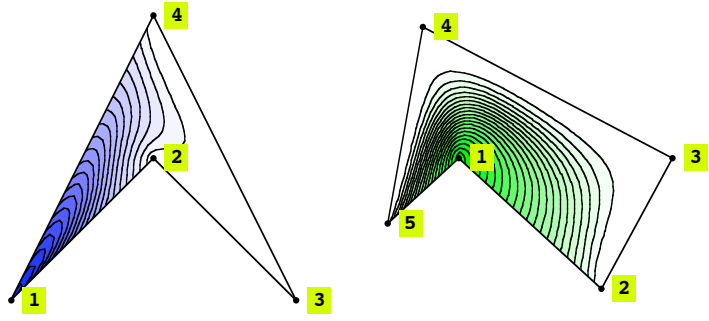


Figure 6. Interpolants that are smooth and linear on the sides of a concave polygon.

The formulation for a concave element applies to any shape. For linearity on a side the area function must be modified.

```
In[38]:= unsignedArea[a_, b_, c_] := 1 /; Abs[area[a, b, c]] < 10^-6
```

Otherwise, the side node would cause singular behavior along the boundary.

```
In[39]:= points = {{-1, -1}, {-1/3, -1},
            {0, -1}, {1/5, -1}, {1, -1}, {1/2, 0}, {0, 1}};
         cnct = {Range[Length[points]]};
         Plot[Evaluate[
           concaveLin[t points[[1]] + (1 - t) points[[4]], points, cnct, 1]],
          {t, 0, 1}, PlotRange → All]
```

From In[39]:=

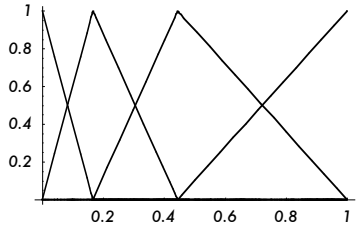


Figure 7. Linearity on the sides.

This works even with multiple side nodes.

```
In[42]:= Show[GraphicsArray[
           Graphics[#[[1, 1]], PlotRange → All, AspectRatio → 1] & /@
            {output[3, (RGBColor[(1 - #), 1, 1] &), concaveLin,
              {{-1, -1}, {0, -1}, {1, -1}, {0, 1}}, {{1, 2, 3, 4}}],
             output[5, (RGBColor[1, (1 - #), 1] &), concaveLin,
              {{-1, -1}, {-1/3, -1}, {0, -1}, {1/5, -1}, {1, -1}, {1/2, 0},
               {0, 1}}, {Range[7]}]}], ImageSize → {350, Automatic}]
```

From In[42]:=

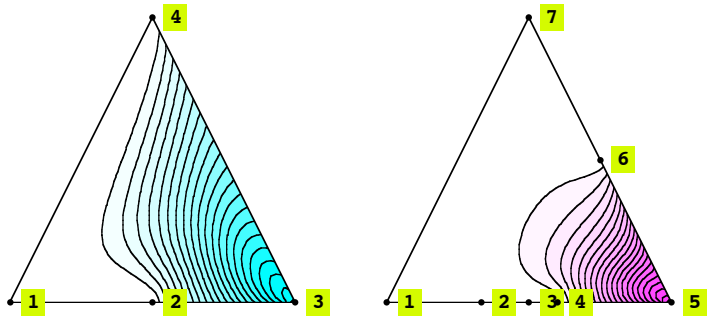


Figure 8. Side node interpolants that are smooth and linear on the sides.

■ Combining Parts

The boundary functions and interior point behaviors can be combined to describe shapes with holes or interior points. Shape functions that apply to separate boundaries and points can be combined.

```
In[43]:= combPrt[{x_, y_}, points_, cnct_, convxH_] :=
          Module[{funs, zero, xy = {x, y}}, funs = Table[Switch[cnct[[i]],
              a_?(# == convxH &), convex[xy, points, cnct, i],
              a_?(Length[#] ≠ 1 &), concaveLin[xy, points, cnct, i],
              _, {1}], {i, Length[cnct]}];
            zero = Apply[Times, Partition[Table[Apply[Times,
                (If[Length[cnct[[i]]] ≠ 1, varPerim[xy, points, cnct, i],
                  norm[xy - points[[cnct[[i]]][[1]]]])], {i, Length[cnct]}],
              Length[cnct] - 1, 1, {Length[cnct]}], {1}];
            Flatten[(funs * zero) / Apply[Plus, Flatten[funs zero]]]]
```

For example, construct a set of interpolants on a multiply connected domain that contains a triangle and two interior points.

In[44]:=

```
Show[GraphicsArray[Partition[
   Graphics[#[[1, 1]], PlotRange → All, AspectRatio → 1] & /@ {
     output[1, (RGBColor[1, 2/3 (1 - #), (1 - #)] &), combPrt,
      {{-1, -1}, {1, -1}, {1, 1}, {-1, 1}, {-1/2, -1/2}, {1/2, -1/2},
       {0, 1/2}}, {{1, 2, 3, 4}, {5, 6, 7}}, {1, 2, 3, 4}],
     output[8, (RGBColor[7/8 (1 - #), 1, (1 - #)] &), combPrt,
      {{-1, -1}, {1, -1}, {1, 1}, {-1, 1}, {-1/2, -1/2},
       {1/2, -1/2}, {0, 1/2}, {1/2, 1/2}, {-1/2, 1/2}},
      {{1, 2, 3, 4}, {5, 6, 7}, {8}, {9}}, {1, 2, 3, 4}],
     output[8, (RGBColor[(1 - #), 1, (2/3) (1 - #)] &),
      combPrt, {{-1, -1}, {0, -4/5}, {1, -1}, {1, 1}, {-1, 1},
       {-1/2, -1/2}, {0, 0}, {1/2, -1/2}, {0, 1/2}},
      {{1, 2, 3, 4, 5}, {6, 7, 8, 9}}, {1, 2, 3, 4}],
     output[7, (RGBColor[7/8 (1 - #), 5/6 (1 - #), 1] &), combPrt,
      {{-1, -1}, {1, -1}, {1, 1}, {-1, 1}, {-3/4, -3/4},
       {-1/4, -3/4}, {-1/4, -1/4}, {-3/4, -1/4},
       {3/4, 3/4}, {1/4, 3/4}, {1/4, 1/4}, {3/4, 1/4}},
      {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}}, {1, 2, 3, 4}]},
   2]], ImageSize → {350, Automatic}]
```

From In[44]:=

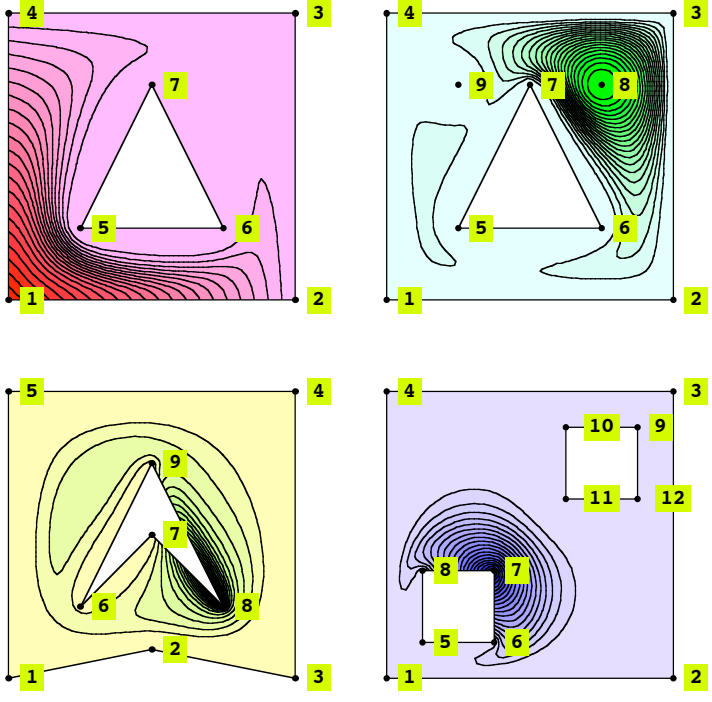


Figure 9. Interpolants that are smooth and linear on the sides of multiply connected polygons.

■ Enforcing Global Linearity

If some of the interpolating functions do not satisfy the same linearity conditions that the convex shape functions satisfy, they can be altered. Given the entire set of interpolating functions $N_j(x, y)$ corresponding to the n nodal points (x_j, y_j) and the set of functions $R_i(x, y)$ that correspond to the points (x_i, y_i) on the convex hull, the following requirement can be imposed:

$$\sum_{j=1}^{n} a(x_j, y_j)\, N_j(x, y) = a(x, y). \tag{20}$$

When the function $a(x, y)$ is either 1, x, or y, the convex interpolant satisfies the requirement exactly over the m convex hull points:

$$\sum_{i=1}^{m} a(x_i, y_i)\, R_i(x, y) = a(x, y). \tag{21}$$

The functions $N_i(x, y)$ can be modified to ensure linearity:

$$N_i(x, y) = R_i(x, y) - \sum_{k=1}^{p} R_i(x_j, y_j)\, N_j(x, y). \tag{22}$$

When the points (x_j, y_j) are not the convex hull points and $p = n - m$, boundedness is not necessarily preserved.

```
In[45]:= makeLin[{x_, y_}, points_, cnct_, cnvxH_] := Module[{nonLin, lin},
           {nonLin = combPrt[{x, y}, points, cnct, cnvxH];
             lin = convex[{x, y}, points, {cnvxH}, 1] -
               (ReplacePart[combPrt[{x, y}, points, cnct, cnvxH], 0, Map[List,
                   cnvxH]].(Map[convex[#, points, {cnvxH}, 1] &, points]));
             Fold[ReplacePart[#1, lin[[#2]], #2] &, nonLin,
              Range[Length[cnvxH]]]}[[1]]]
```

For example, construct linearized interpolations on a quadrilateral.

```
In[46]:= Show[GraphicsArray[
           Graphics[#[[1, 1]], PlotRange → All, AspectRatio → 1] & /@
            {output[1, (RGBColor[1, (1 - #), (1 - #)] &), makeLin,
              {{-1, -1}, {0, 0}, {1, -1}, {0, 1}}, {{1, 2, 3, 4}}],
             output[1, (RGBColor[(1 - #), (1 - #), 1] &), makeLin,
              {{0, 1/2}, {1, 0}, {3/2, 1/2}, {-1/4, 1}, {-1/2, 1/4}},
              {Range[5]}]}], ImageSize → {350, Automatic}]
```

From In[46]:=

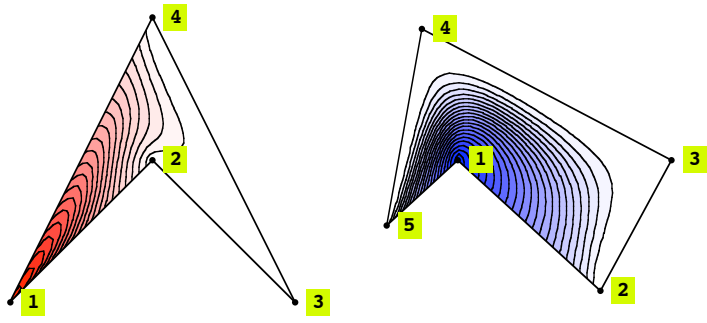


Figure 10. Linearized interpolants.

■ Conclusion

The resulting interpolants are smooth and bounded within their respective domains and the sum of any set of interpolants is one. The functions are linearly independent. Examples of the smooth and bounded behavior of the functions are shown in the figures. The convex polygon shape functions automatically satisfy constant and linear fields. This is not the case for the concave or multiply connected domain representation. Nevertheless, the linearity constraint can be imposed on the representation.

This type of interpolant is distinctly different from the available interpolant and shape function formulations, since each interpolant is constructed to satisfy the required conditions exactly: smoothness, boundedness, linearity on sides, and the linear field conditions. Using this algebraic construction, closed-form interpolants satisfying different field properties can also be derived.

■ Acknowledgment

This research was supported by a National Science Foundation grant (CMS:0202232). We are grateful to the reviewers for their careful reading and many helpful comments.

■ References

[1] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes, *Computer Graphics Principles and Practice*, 2nd ed., New York: Addison-Wesley, 1996.

[2] E. L. Wachspress, *A Rational Finite Element Basis*, New York: Academic Press, 1975.

[3] R. Courant, "Variational Methods for the Solution of Problems of Equilibrium and Vibrations," *Bulletin of the American Mathematical Society*, **49**, 1943 pp. 1–43.

[4] O. C. Zienkiewicz and R. L. Taylor, *The Finite Element Method*, 4th ed., New York: McGraw-Hill, 1989.

[5] G. Dasgupta and J. Treil, "Maxillo-Facial Frame: Finite Element Shapes," *The Mathematica Journal*, **8**(2), 2001 pp. 235–246.

[6] P. J. Davis, *Interpolation and Approximation*, New York: Dover Publications, 1963.

[7] V. L. Rvachev, T. I. Sheiko, V. Shapiro, and I. Tsukanov, "On Completeness of RFM Solution Structures," *Computational Mechanics*, **25**, 2000 pp. 305–316.

About the Authors

Elisabeth Anna Malsch completed her B.S. (1999), M.S. (2000), and Ph.D. (2003) from Columbia University, New York. Currently she is an Alexander von Humboldt fellow, Bonn, Germany. She has been using symbolic computation in her undergraduate, graduate and Ph.D. courses and research focused on engineering mechanics. For details about her recent publications on concave finite elements and boundary elements, see www.civil.columbia.edu/~malsch.

Gautam Dasgupta is a professor of civil engineering and engineering mechanics at Columbia University, New York. His areas of research include computer mathematics, waves in random media, earthquake engineering, and finite and boundary element analysis with *Mathematica*. For details on his recent work, see www.columbia.edu/~gd18.

Elisabeth Anna Malsch
Postdoctoral Researcher
Department of Civil Engineering and Engineering Mechanics
Columbia University
500 West 120th Street
New York, NY 10027
malsch@civil.columbia.edu

Gautam Dasgupta
Professor
Department of Civil Engineering and Engineering Mechanics
Columbia University
500 West 120th Street
New York, NY 10027
dasgupta@columbia.edu

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 submissions@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.

***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 offers 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* programs, isolating problems in large *Mathematica* programs, implementing custom features, and writing complete programs based on user specifications.

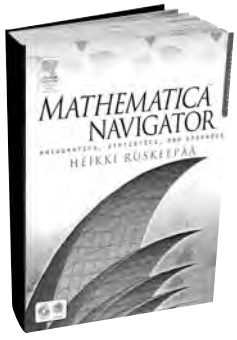
More info: www.wolfram.com/services/training

Online Calculus Solver—calc101.com supplies automatic step-by-step solutions for derivatives, elementary integrals, matrix algebra, and polynomial operations.

More info: www.calc101.com

Employ the full power of Mathematica 5.0 with these NEW resources!

All titles include a dual platform CD-ROM and are compatible with earlier editions of Mathematica



Mathematica Navigator
MATHEMATICS, STATISTICS, AND GRAPHICS

Second Edition

Heikki Ruskeepää
University of Turku, Finland

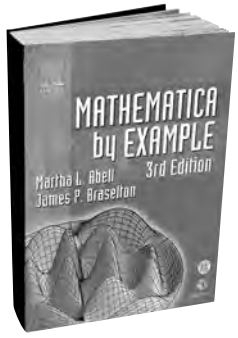
"Mathematica Navigator is packed with excellent examples...an invaluable companion to any textbook for most Mathematica-enriched courses." – Fred Szabo, Concordia University

Part tutorial and part handbook, the Second Edition of **Mathematica Navigator** shows how to navigate through the newest version of this popular computation software. It provides a general introduction to Mathematica 5.0 followed by specific applications in graphics, methods of applied mathematics and statistics, and programming.

Key Features:
- Serves as an all-in-one guide to the new Mathematica 5.0
- Covers the most basic information and leads up to advanced topics
- CD-ROM installs easily into the help system of Mathematica
- Studies several real-life data sets

Contents:
• Starting • Sightseeing • Notebooks • Files • 2D Graphics for Functions • Options for 2D Graphics • 3D Graphics for Functions • Options for 3D Graphics • 2D Graphics for Data • 3D Graphics for Data • Numbers • Expressions • Lists • Functions • Programs • Differential Calculus • Integral Calculus • Matrices • Equations • Optimization • Interpolation • Approximation • Differential Equations • Partial Differential Equations • Difference Equations • Probability • Statistics • References • Index

©2004, Paperback w/CD-ROM, 844 pp, ISBN: 0-12-603640-3
List Price: $69.95 / £39.99



Mathematica by Example
Third Edition

Martha L. Abell & James P. Braselton
Georgia Southern University, Statesboro, U.S.A.

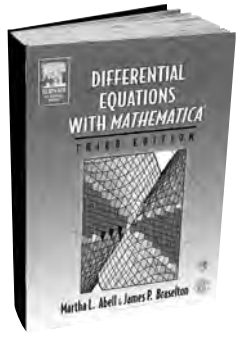
"The book does an excellent job of providing information that beginning Mathematica users need, I think the use of Mathematica...[and] graphical output is excellent."
– William Emerson, Metropolitan State College

Mathematica by Example is geared toward the beginning Mathematica user, and focuses on the most-often-used features of this powerful tool. It covers popular applications of mathematics within different areas including calculus, linear algebra, ordinary differential equations, and partial differential equations.

Key Features:
- Introduces users to the features and applications of the new Mathematica 5.0 in clear, understandable language
- Covers all the basics needed to get up and running
- Enclosed CD-ROM contains all inputs used in the book

Contents:
• Getting Started • Mathematical Operations on Numbers, Expressions, and Functions • Calculus • Introduction to Lists and Tables • Nested Lists • Applications Related to Ordinary and Partial Differential Equations • Some Graphics Packages • Selected References • Index

©2004, Paperback w/CD-ROM, 571 pp, ISBN: 0-12-041563-1
List Price: $69.95 / £39.99



Differential Equations with Mathematica
Third Edition

Martha L. Abell & James P. Braselton
Georgia Southern University, Statesboro, U.S.A.

"The topics covered are extensive. The order and organization are good, especially in terms of increasing sophistication of the mathematics involved and in the complexity of the Mathematica programming necessary."
– Stephen McDowall, Western Washington University

The Third Edition of **Differential Equations with Mathematica** integrates new applications from a variety of fields, especially biology, physics, and engineering. It is compatible with Version 5.0 and is a perfect introduction to the subject matter and the software. The book/CD-ROM package contains built-in commands that let the user solve problems directly using graphical solutions.

Contents:
• Introduction to Differential Equations • First-Order Ordinary Differential Equations • Applications of First-Order Ordinary Differential Equations • Higher-Order Differential Equations • Applications of Higher-Order Differential Equations • Ordinary Differential Equations with Nonconstant Coefficients • Laplace Transform Methods • Systems of Ordinary Differential Equations • Applications of Systems of Ordinary Differential Equations • Eigenvalue Problems and Fourier Series • Partial Differential Equations • Appendix: Getting Started

©2004, Paperback w/CD-ROM, 876 pp, ISBN: 0-12-041562-3
List Price: $69.95 / £40.95

AEG/LB/FR-25942-8/04

Order from your favorite bookseller, or directly from Elsevier:
In the US and Canada: 1-800-545-2522 • In other countries: +44 (0) 01865 474000 • http://books.elsevier.com
To request a review copy for a course, visit books.elsevier.com/textbooks or email textbook@elsevier.com

ELSEVIER ACADEMIC PRESS

RiskQ 4.2

Software for Probability and Statistics

***RiskQ* provides tools for developing empirical and theoretical distributions and using these for hypothesis testing. Applications include:**

○ **Experimental Data Analysis**

- Perform ANOVA, MANOVA, ANOCOVA, and GLM analyses
- Adjust *p*-values from multiple tests
- Apply common parametric and nonparametric statistical tests
- Compare correlation matrices

○ **Toxicology and Environmental Chemistry**

- Conduct statistical tests on data
- Characterize data variation graphically and analytically

○ **Epidemiology and Clinical Trials**

- Analyze associations exhibited by epidemiological data
- Perform adjusted chi-square tests of homogeneity and association
- Obtain the approximate or exact likelihood of observed contingency table data

○ **Risk and Uncertainty Analysis**

- Characterize distributions reflecting uncertainty
- Perform parametric and nonparametric sensitivity analysis
- Characterize Monte-Carlo estimation error of simulated output

Visit us at:
www.wolfram.com/products/applications/riskq

Loehle Enterprises
www.loehleenterprises.com
info@loehleenterprises.com