

# Introduction to Maple

Third Edition

Springer Science+Business Media, LLC

André Heck

# Introduction to Maple

Third Edition

With 221 Illustrations



Springer

André Heck  
AMSTEL Institute  
Faculty of Science  
Universiteit van Amsterdam  
Kruislaan 404  
1098 SM Amsterdam  
The Netherlands  
heck@science.uva.nl

*Cover illustration:* Cover image © Kim McDevitt/SuperStock.

Mathematics Subject Classification (2000): 68Q40

Library of Congress Cataloging-in-Publication Data  
Heck, A. (André)

Introduction to Maple/André Heck. — 3rd ed.  
p. cm.

Includes bibliographical references and index.

ISBN 978-1-4612-6505-4 ISBN 978-1-4613-0023-6 (eBook)

DOI 10.1007/978-1-4613-0023-6

1. Maple (Computer file) 2. Algebra—Data processing. I. Title.

QA155.7.E4 H43 2003

512'.00285'5369—dc21

2002042743

ISBN 978-1-4612-6505-4 Printed on acid-free paper.

Maple is a registered trademark of Waterloo Maple, Inc.

© 2003 Springer Science+Business Media New York

Originally published by Springer-Verlag New York, Inc. in 2003

Softcover reprint of the hardcover 3rd edition 2003

All rights reserved. This work may not be translated or copied in whole or in part without the written permission of the publisher Springer Science+Business Media, LLC except for brief excerpts in connection with reviews or scholarly analysis. Use in connection with any form of information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed is forbidden.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

9 8 7 6 5 4 3 2

Typesetting: Pages created by the author using the Springer LaTeX 2 $\epsilon$  svsing6.cls macro.

[www.springer-ny.com](http://www.springer-ny.com)

# Preface to the Third Edition

The first two editions of this book have been very well received by the community, but so many revisions of the Maple system have occurred since then that simply reprinting the out-of-stock book would not do anymore. A major revision of the book was inevitable, too. The wording “major revision” must be taken seriously because I not only corrected typographical errors, rephrased text fragments, and updated many examples, but I also rewrote complete chapters and added new material. In particular, the chapter on differential equations now discusses Lie symmetry methods, partial differential equations, and numerical methods. Linear algebra is based throughout the book on the packages `LinearAlgebra` and `VectorCalculus`, which replace the deprecated package `linalg`. Maple users are strongly advised to do their work with the new packages. The chapter on simplification has been updated and expanded; it discusses the use of assumptions in more detail now. Last, but not least, a new chapter on Gröbner basis theory and the `Groebner` package in Maple has been added to the book. It includes many applications of Gröbner basis theory. Many of the Maple sessions have been rewritten so that they comply with the most recent version of Maple. As a result of all this work, hardly any section in the book has been left untouched.

## From the Preface of the Second Edition

The first edition of this book has been very well received by the community. The new Version 4 of Maple V contains so many new mathematical features and improvements in the user interface that Waterloo Maple, Inc., markets it as “the Power Edition.” These two facts have made it necessary to write a second edition within a short period of the first. I corrected typographical errors, rephrased text, updated and improved many examples, and added much new material. Hardly any chapter has been left untouched. Substantially changed or added sections and chapters address the assume facility, I/O, approximation theory, integration, composite data types, simplification, graphics, differential equations, and matrix algebra. Tables summarize features, command options, etc., and constitute a quick reference. The enlarged index of the book has been carefully compiled to make locating search items quick and easy. Many new examples have been included showing how to use Maple as a problem solver, how to assist the system during computations, and how to extend its built-in facilities.

## From the Preface of the First Edition

In symbolic computation on computers, also known as *computer algebra*, keyboard and display replace the traditional pencil and paper in doing mathematical computations. Interactive computer programs, which are called *computer algebra systems*, allow their users to compute not only with numbers, but also with symbols, formulas, equations, and so on. Many mathematical computations such as differentiation, integration, series expansion of functions, and inversion of matrices with symbolic entries can be carried out quickly, with emphasis on exactness of results and without much human effort.

Computer algebra systems are powerful tools for mathematicians, physicists, chemists, engineers, technicians, psychologists, sociologists—in short, for anybody who needs to do mathematical computations. Computer algebra systems are indispensable in modern pure and applied scientific research and education.

This book is a gentle introduction to one of the modern computer algebra systems, viz., Maple. Primary emphasis is on learning what can be done with Maple and how it can be used to solve (applied) mathematical problems. To this end, the book contains many examples and exercises, both elementary and more sophisticated. They stimulate you to use Maple and encourage you to find your way through the system. My advice is to read this book in conjunction with the Maple system, try the examples, make variations of them, and try to solve the exercises.

In this book, emphasis is on understanding the basic principles and ideas of Maple so that you can use it effectively to solve your mathematical problems. Factual knowledge or information about every built-in Maple facility can be obtained from the on-line help system or from the Maple documentation that comes along with the software. This book does not teach mathematics; it is understood that you know the theory behind the examples. By choosing a variety of problems and showing how Maple can be used to solve them, you should get an idea of the capabilities of the system.

In this book, the usage of Maple as a programming language is not discussed at a higher level than that of defining simple procedures and using simple language constructs. However, the Maple data structures are discussed in great detail because good understanding of them is necessary for manipulating and simplifying expressions effectively. This also forms a good starting point to acquaint you further with Maple as a programming language.

## About the Maple Version Used in the Third Edition

The third edition of this book is fully revised and updated to Maple 8. More precisely, the third edition of this book was produced with Maple 8 on a PC running Windows 2000, with a 1.7 Ghz Pentium 4 processor having 512 MB main memory. Maple 8 is available on many computer platforms, but most of the book should be system independent as it focuses on the mathematical contents of the system.

## About the Production of the Third Edition

The manuscript has been typeset in **LATEX**. The **LATEX** code of the Maple sessions has been produced by exporting Maple worksheets with the sessions into this typesetting format. In this way, you can reliably reproduce the results on your computer screen and print on paper.

## Source Code of Maple Sessions

Readers connected to the Internet can obtain the source code of all Maple sessions from the author's home page [www.science.uva.nl/~heck](http://www.science.uva.nl/~heck)

## Notation

The following notation is used throughout the book: Maple keywords are in **typewriter font**, and references to Maple procedures are in **bold type**.

## Acknowledgments

I am grateful to many people who have contributed to this book. My thanks go first to the developers of Maple. They have created a wonderful tool for scientists and engineers; Release 8 of Maple indeed allows you to command the brilliance of a thousand mathematicians, as commerce claims. Writing the new edition of the book has been an exciting tour through the new facilities. During the last years, Waterloo Maple, Inc., kindly provided me with updates of the Maple system so that I could keep abreast of developments in the software.

I would like to thank Achi Dosanjh from Springer-Verlag New York, Inc., for her interest in this work, her support for this book, and her patience with an author who is always short of time. The support of the production editor Francine McNeill is cordially acknowledged. I am indebted to Ton Ellermeijer, director of the AMSTEL Institute, who allowed me to free myself for a short period from daily work and to work fulltime on the new edition. The help of Martin Beugel and Leendert van Gastel in the process of creating the Index is cordially acknowledged. I also want to thank my friend Renato Doria for his enthusiasm for the book; parts of it were written during visits to him in Petrópolis, Brazil.

I wish to thank all readers of the first two editions who sent me corrections of errors, helpful ideas, or expressions of appreciation. Their e-mail reports and letters have encouraged me to write this new edition.

## Contact Address

Despite all the help I got, I am sure that users of this book will come up with remarks, suggestions, corrections, and so on. Please send them to

Universiteit van Amsterdam  
Faculty of Science, AMSTEL Institute  
Attn. André Heck  
Kruislaan 404, 1098 SM Amsterdam  
The Netherlands

or to the electronic mail address

[heck@science.uva.nl](mailto:heck@science.uva.nl)

# Contents

<b>Preface to the Third Edition</b>	v
<b>List of Tables</b>	xv
<b>1 Introduction to Computer Algebra</b>	1
1.1 What is Computer Algebra? . . . . .	1
1.2 Computer Algebra Systems . . . . .	2
1.3 Some Properties of Computer Algebra Systems . . . . .	5
1.4 Advantages of Computer Algebra . . . . .	11
1.5 Limitations of Computer Algebra . . . . .	23
1.6 Design of Maple . . . . .	29
<b>2 The First Steps: Calculus on Numbers</b>	33
2.1 Getting Started . . . . .	33
2.2 Getting Help . . . . .	36
2.3 Integers and Rational Numbers . . . . .	41
2.4 Irrational Numbers and Floating-Point Numbers . . . . .	46
2.5 Algebraic Numbers . . . . .	53
2.6 Complex Numbers . . . . .	58
2.7 Exercises . . . . .	63
<b>3 Variables and Names</b>	65
3.1 Assignment and Unassignment . . . . .	65
3.2 Evaluation . . . . .	73

3.3	Names of Variables . . . . .	77
3.4	Basic Data Types . . . . .	83
3.5	Attributes . . . . .	88
3.6	Properties . . . . .	89
3.7	Exercises . . . . .	93
<b>4</b>	<b>Getting Around with Maple</b>	<b>95</b>
4.1	Maple Input and Output . . . . .	95
4.2	The Maple Library . . . . .	101
4.3	Reading and Writing Files . . . . .	106
4.4	Importing and Exporting Numerical Data . . . . .	113
4.5	Low-level I/O . . . . .	116
4.6	Code Generation . . . . .	127
4.7	Changing Maple to Your Own Taste . . . . .	133
4.8	Exercises . . . . .	137
<b>5</b>	<b>Polynomials and Rational Functions</b>	<b>139</b>
5.1	Univariate Polynomials . . . . .	139
5.2	Multivariate Polynomials . . . . .	145
5.3	Rational Functions . . . . .	147
5.4	Conversions . . . . .	148
5.5	Exercises . . . . .	151
<b>6</b>	<b>Internal Data Representation and Substitution</b>	<b>153</b>
6.1	Internal Representation of Polynomials . . . . .	153
6.2	Generalized Rational Expressions . . . . .	159
6.3	Substitution . . . . .	161
6.4	Exercises . . . . .	174
<b>7</b>	<b>Manipulation of Polynomials and Rational Expressions</b>	<b>175</b>
7.1	Expansion . . . . .	175
7.2	Factorization . . . . .	178
7.3	Canonical Form and Normal Form . . . . .	181
7.4	Normalization . . . . .	183
7.5	Collection . . . . .	185
7.6	Sorting . . . . .	187
7.7	Exercises . . . . .	188
<b>8</b>	<b>Functions</b>	<b>189</b>
8.1	Mathematical Functions . . . . .	189
8.2	Arrow Operators . . . . .	193
8.3	Piecewise Defined Functions . . . . .	195
8.4	Maple Procedures . . . . .	201
8.5	Recursive Procedure Definitions . . . . .	204
8.6	<b>unapply</b> . . . . .	208

8.7	Operations on Functions . . . . .	209
8.8	Anonymous Functions . . . . .	210
8.9	Exercises . . . . .	211
<b>9</b>	<b>Differentiation</b>	<b>213</b>
9.1	Symbolic Differentiation . . . . .	213
9.2	Automatic Differentiation . . . . .	220
9.3	Exercises . . . . .	224
<b>10</b>	<b>Integration and Summation</b>	<b>225</b>
10.1	Indefinite Integration . . . . .	225
10.2	Definite Integration . . . . .	234
10.3	Numerical Integration . . . . .	239
10.4	Integral Transforms . . . . .	241
10.5	Assisting Maple's Integrator . . . . .	250
10.6	Summation . . . . .	255
10.7	Exercises . . . . .	260
<b>11</b>	<b>Series, Approximation, and Limits</b>	<b>265</b>
11.1	Truncated Series . . . . .	265
11.2	Approximation of Functions . . . . .	276
11.3	Power Series . . . . .	281
11.4	Limits . . . . .	285
11.5	Exercises . . . . .	287
<b>12</b>	<b>Composite Data Types</b>	<b>289</b>
12.1	Sequence . . . . .	289
12.2	Set . . . . .	292
12.3	List . . . . .	294
12.4	Arrays . . . . .	300
12.5	Table: <code>table</code> . . . . .	316
12.6	Last Name Evaluation . . . . .	319
12.7	Rectangular Table: <code>rtable</code> . . . . .	321
12.8	Record Data Structure . . . . .	325
12.9	Function Call . . . . .	326
12.10	Conversion between Composite Data Types . . . . .	328
12.11	Exercises . . . . .	331
<b>13</b>	<b>The Assume Facility</b>	<b>333</b>
13.1	The Need for an Assume Facility . . . . .	333
13.2	Basics of <code>assume</code> . . . . .	338
13.3	An Algebra of Properties . . . . .	342
13.4	Implementation of <code>assume</code> . . . . .	344
13.5	Exercises . . . . .	350
13.6	Hierarchy of Properties . . . . .	350

<b>14 Simplification</b>	<b>353</b>
14.1 Automatic Simplification . . . . .	354
14.2 <b>expand</b> . . . . .	356
14.3 <b>combine</b> . . . . .	364
14.4 <b>simplify</b> . . . . .	370
14.5 <b>convert</b> . . . . .	375
14.6 Trigonometric Simplification . . . . .	379
14.7 Simplification w.r.t. Side Relations . . . . .	382
14.8 Control Over Simplification . . . . .	386
14.9 Defining Your Own Simplification Routines . . . . .	391
14.10 Exercises . . . . .	397
14.11 Simplification Chart . . . . .	399
<b>15 Graphics</b>	<b>401</b>
15.1 Some Basic Two-Dimensional Plots . . . . .	403
15.2 Options of <b>plot</b> . . . . .	407
15.3 The Structure of Two-Dimensional Graphics . . . . .	418
15.4 The <b>plottools</b> Package . . . . .	422
15.5 Special Two-Dimensional Plots . . . . .	426
15.6 Two-Dimensional Geometry . . . . .	436
15.7 Plot Aliasing . . . . .	438
15.8 A Common Mistake . . . . .	439
15.9 Some Basic Three-Dimensional Plots . . . . .	441
15.10 Options of <b>plot3d</b> . . . . .	442
15.11 The Structure of Three-Dimensional Graphics . . . . .	448
15.12 Special Three-Dimensional Plots . . . . .	452
15.13 Data Plotting . . . . .	459
15.14 Animation . . . . .	469
15.15 List of Plot Options . . . . .	472
15.16 Exercises . . . . .	477
<b>16 Solving Equations</b>	<b>481</b>
16.1 Equations in One Unknown . . . . .	481
16.2 Abbreviations in <b>solve</b> . . . . .	483
16.3 Some Difficulties . . . . .	485
16.4 Systems of Equations . . . . .	492
16.5 The Gröbner Basis Method . . . . .	501
16.6 Inequalities . . . . .	508
16.7 Numerical Solvers . . . . .	510
16.8 Other Solvers in Maple . . . . .	512
16.9 Exercises . . . . .	519
<b>17 Differential Equations</b>	<b>521</b>
17.1 First Glance at ODEs . . . . .	522
17.2 Analytic Solutions . . . . .	524

17.3	Lie Point Symmetries for ODEs . . . . .	538
17.4	Taylor Series Method . . . . .	560
17.5	Power Series Method . . . . .	561
17.6	Numerical Solutions . . . . .	566
17.7	Graphical Methods . . . . .	580
17.8	Change of Coordinates . . . . .	586
17.9	Perturbation Methods . . . . .	590
17.10	Partial Differential Equations . . . . .	600
17.11	Lie Point Symmetries of PDEs . . . . .	615
17.12	Exercises . . . . .	617
<b>18</b>	<b>The LinearAlgebra Package</b>	<b>619</b>
18.1	Loading the LinearAlgebra Package . . . . .	619
18.2	Creating Vectors and Matrices . . . . .	621
18.3	Vector and Matrix Arithmetic . . . . .	629
18.4	Basic Matrix Functions . . . . .	634
18.5	Structural Operations . . . . .	641
18.6	Vector Operations . . . . .	645
18.7	Standard Forms of Matrices . . . . .	646
18.8	Numeric Linear Algebra . . . . .	656
18.9	Exercises . . . . .	660
<b>19</b>	<b>Linear Algebra: Applications</b>	<b>663</b>
19.1	Kinematics of the Stanford Manipulator . . . . .	663
19.2	A 3-Compartment Model of Cadmium Transfer . . . . .	669
19.3	Molecular-Orbital Hückel Theory . . . . .	680
19.4	Vector Calculus . . . . .	687
19.5	Moore-Penrose Inverse . . . . .	693
19.6	Exercises . . . . .	694
<b>20</b>	<b>A Bird's-Eye View of Gröbner Bases</b>	<b>697</b>
20.1	Introduction . . . . .	697
20.2	Elementary Solution Methods . . . . .	702
20.2.1	Heuristic Method . . . . .	702
20.2.2	Gaussian Elimination-Like Method . . . . .	702
20.2.3	Conclusion . . . . .	703
20.3	Basics of the Gröbner Basis Method . . . . .	703
20.3.1	Term Ordering . . . . .	704
20.3.2	Polynomial Reduction and Normal Form . . . . .	710
20.3.3	Characterization of a Gröbner Basis . . . . .	712
20.3.4	The Buchberger Algorithm . . . . .	714
20.3.5	Improvements of Buchberger's Algorithm . . . . .	716
20.4	Properties and Applications of Gröbner Bases . . . . .	719
20.4.1	Equivalence of Systems of Polynomial Equations . . . . .	720
20.4.2	Dimension, Hilbert Series and Hilbert Polynomial . . . . .	721

20.4.3 Solvability of Polynomial Equations . . . . .	725
20.4.4 Finite Solvability of Polynomial Equations . . . . .	729
20.4.5 Counting of Finite Solutions . . . . .	730
20.4.6 Converting a System of Polynomial Equations into Triangular Form . . . . .	732
20.4.7 Finding a Univariate Polynomial . . . . .	734
20.4.8 Decomposition of Ideals . . . . .	735
20.4.9 An Example From Robotics . . . . .	739
20.4.10 Implicitization of Parametric Objects . . . . .	740
20.4.11 Invertibility of Polynomial Mappings . . . . .	742
20.4.12 Simplification of Expressions . . . . .	742
20.4.13 Working over General Algebras . . . . .	743
20.5 Exercises . . . . .	745
<b>References</b>	<b>747</b>
<b>Index</b>	<b>761</b>

# List of Tables

1.1	Components of the Maple system.	30
2.1	The on-line Maple help system.	40
2.2	Mathematical constants in Maple.	48
2.3	Commonly used mathematical functions known to Maple.	50
2.4	Selectors on <b>RootOf</b> .	57
3.1	Utility functions for usage of names.	69
3.2	Evaluation for assignment and unassignment.	72
3.3	Reserved words in Maple.	78
3.4	Quotation marks and percentage symbols.	83
3.5	Commonly used surface data types	84
3.6	Tests for types.	86
3.7	Combinations of colon, semicolon, and equal sign.	87
4.1	<b>userinfo</b> -facility.	98
4.2	Subpackages of <b>stats</b> package.	104
4.3	Reading and writing of Maple files.	112
4.4	Maple streams: opening, and closing.	117
4.5	Low-level formatted I/O routines.	119
4.6	Flags in <b>printf</b> .	120
4.7	Conversion codes in <b>printf</b> .	121
4.8	Character escape codes.	122
4.9	Conversion codes in <b>fscanf</b> , <b>scanf</b> , and <b>sscanf</b> .	123

4.10 Values of <b>errorbreak</b> . . . . .	136
10.1 Some elliptic integrals in Maple. . . . .	238
10.2 Integral transforms in Maple . . . . .	241
12.1 Selection in sequence, set, or list. . . . .	300
12.2 Built-in special indexing functions. . . . .	310
12.3 Formats of call of special indexing function. . . . .	311
12.4 Comparison of arrays, matrices, Arrays, and Matrices. . . . .	315
12.5 Main conversions from composite data type <i>S</i> to <i>T</i> . . . . .	331
13.1 Commands of assume facility . . . . .	338
15.1 <b>Plottools</b> functions to alter graphics objects. . . . .	424
15.2 List plotting. . . . .	459
15.3 Dutch consumption of beverages per inhabitant. . . . .	459
15.4 Statistical plotting commands in <b>statplots</b> subpackage. . . . .	465
16.1 Options of <b>fsolve</b> . . . . .	510
17.1 Some solvable first order ODEs. . . . .	532
17.2 Some solvable second order ODEs. . . . .	533
17.3 Options of <b>dsolve(ODE, type=exact)</b> . . . . .	538
17.4 Lie's classification of second-order ODEs that admit a 2-dimensional symmetry algebra. . . . .	558
17.5 Main commands in <b>DEtools</b> package for Lie symmetry approach. . . . .	560
17.6 Numerical methods provided by <b>dsolve</b> for initial value problems. . . . .	577
17.7 Numerical methods provided by <b>dsolve</b> for boundary value problems. . . . .	577
17.8 Lie Point Symmetries of KdV-Equation. . . . .	617
18.1 Options of <b>Vector</b> and <b>Matrix</b> . . . . .	621
18.2 Procedures related to special vectors and matrices. . . . .	624
18.3 Basic functions in the <b>LinearAlgebra</b> package. . . . .	641
18.4 Structural operators. . . . .	641
18.5 Test functions for matrices. . . . .	642
18.6 Vector operations. . . . .	645
18.7 Test functions for matrices. . . . .	646
19.1 Some predefined 2D orthogonal curvilinear coordinates. . . . .	688
19.2 Some predefined 3D orthogonal curvilinear coordinates. . . . .	689

# 1

## Introduction to Computer Algebra

The goal of this chapter is to briefly describe what computer algebra is about, present a little history of computer algebra systems, give some examples of computer algebra usage, and discuss some advantages and limitations of this technological tool. We end with a sketch of the design of the Maple system.

The examples in this first chapter are sometimes of a rather advanced mathematical level. Starting with the second chapter, we give a detailed, step-by-step exposition of Maple as a symbolic calculator. The rest of the book does not depend much on this chapter. Thus, anyone who is so eager to learn Maple that he or she cannot wait any longer may skip this chapter and turn to it at any moment.

### 1.1 What is Computer Algebra?

Historically the verb “compute” has mostly been used in the sense of “computing with numbers.” Numerical computation is not only involved with basic arithmetic operations such as addition and multiplication of numbers, but also with more sophisticated calculations like computing numerical values of mathematical functions, finding roots of polynomials, solving systems of equations, and computing with matrices. It is essential in this type of computation that arithmetic operations are carried out on numbers and on numbers only. Furthermore, computations with numbers are in most cases not exact because in applications one is almost always dealing with floating-point numbers. Simple computations can be done with pencil and paper or with a pocket calculator; for large numerical computations, mainframes

serve as “number crunchers.” In the last fifty years numerical computation on computers flourished to such an extent that for many scientists mathematical computation on computers and numerical computation have become synonymous.

But mathematical computation has another important component, which we shall call *symbolic and algebraic computation*. In short, it can be defined as computation with symbols representing mathematical objects. These symbols may represent numbers like integers, rational numbers, real and complex numbers, and algebraic numbers, but they may also be used for mathematical objects like polynomials and rational functions, systems of equations, and even more abstractly for algebraic structures like groups, rings, and algebras, and elements thereof. Moreover, the adjective *symbolic* emphasizes that in many cases the ultimate goal of mathematical problem solving is expressing the answer in a closed formula or finding a symbolic approximation. By *algebraic* we mean that computations are carried out exactly, according to the rules of algebra, instead of using the approximate floating-point arithmetic. Examples of symbolic and algebraic computations are factorization of polynomials, differentiation, integration, and series expansion of functions, analytic solution of differential equations, exact solution of systems of equations, and simplification of mathematical expressions.

In the last thirty-five years great progress has been made regarding the theoretical background of symbolic and algebraic algorithms; moreover, tools have been developed to carry out mathematical computations on computers [26, 60, 91, 103, 154, 237]. This has lead to a new discipline, which is referred to by various names: symbolic and algebraic computation, symbolic computation, symbolic manipulation, formula manipulation, and computer algebra, to name a few. Tools for mathematical computation on a computer are given as many names as the discipline itself: symbolic computation programs, symbol crunchers, symbolic manipulation programs, symbolic calculators, and computer algebra systems. Unfortunately, the term “symbolic computation” is used in many different contexts, like logic programming and artificial intelligence in its broadest sense, which have very little to do with mathematical computation. To avoid misunderstanding, we shall henceforth adopt the term *computer algebra* and we shall speak of *computer algebra systems*.

## 1.2 Computer Algebra Systems

In this section, we shall give a very short, incomplete, and subjective overview of present-day computer algebra systems. For a more thorough overview we refer to [56, 108, 234] and to WWW-servers dedicated towards computer algebra [242]. Computer algebra systems can be conveniently

divided into two categories: *special purpose systems* and *general purpose systems*.

Special purpose systems are designed to solve problems in one specific branch of physics and mathematics. Some of the best-known special purpose systems used in physics are SCHOONSCHIP ([218] high-energy physics), CAMAL ([12] celestial mechanics), and SHEEP and STENSOR ([86, 133, 170] general relativity). Examples of special purpose systems in the mathematical arena are Cayley and GAP ([39, 42, 90] group theory), PARI, SIMATH, and KANT ([67, 132, 190, 196] number theory), CoCoA ([45, 101, 156] commutative algebra), Macaulay and SINGULAR ([110, 111, 112] algebraic geometry and commutative algebra), and LiE ([59] Lie theory). Our interest will be in the general purpose system Maple [173, 180, 181], but the importance of special purpose systems should not be underestimated: they have played a crucial role in many scientific areas [26, 41, 135]. Often they are more handsome and efficient than general purpose systems because of their use of special notations and data structures, and because of their implementation of algorithms in a low-level programming language.

General purpose systems please their users with a great variety of data structures and mathematical functions, trying to cover as many different application areas as possible (q.v., [122]). The oldest general purpose computer algebra systems still in use are MACSYMA [172] and REDUCE [124]. Both systems were born in the late sixties and were implemented in the programming language LISP. MACSYMA is a full-featured computer algebra system with a wide range of auxiliary packages, but its demands on computer resources are rather high and it is only available on a limited number of platforms. The release of the PC version of MACSYMA in 1992 has caused a revival of this system. REDUCE began as a special purpose program for use in high-energy physics, but gradually transformed into a general purpose system. Compared to MACSYMA the number of user-ready facilities in REDUCE is modest, but on the other hand it is a very open system (the complete source code is distributed!) making it easily extensible and modifiable. REDUCE is still under active development: REDUCE 3.7 is from 1999. It runs on a very wide range of computers and is well-documented.

In the eighties, MuMATH [239] and its successor DERIVE [157, 216] were the first examples of compact non-programmable symbolic calculators, designed for use on PC-type computers. DERIVE has a friendly menu-driven interface with graphical and numerical features. Considering its compactness and the limitations of DOS computers, DERIVE offers an amazing amount of user-ready facilities. Version 5 of 2000 also has limited programming facilities. Many of DERIVE's features have been built into the TI-92 calculator (Texas Instruments, 1995) and other high-end

calculators, thus making computer algebra available at small size computers. It also forms the computer algebra kernel of the computer learning environment called ‘TI Interactive!’ (Texas Instruments, 2000).

Most modern computer algebra systems are implemented in the programming language C. This language allows developers to write efficient, portable computer programs that really exploit the platforms for which they are designed. Many of these computer algebra systems work on a variety of computers, from supercomputers down to desktop computers.

In §1.6 we shall sketch the design of Maple [173, 180, 181]. Another notable modern general purpose system is *Mathematica* [238]. *Mathematica* was the first system in which symbolics, numerics, and graphics were incorporated in such a way that it could serve as a user-friendly environment for doing mathematics. There exists on most platforms the notebook interface, which is a tool (comparable with Maple worksheets) to create a structured text in which ordinary text is interwoven with formulas, programs, computations, and graphics. Another feature of *Mathematica* is the well-structured user-level programming language. With the publicity and marketing strategy that went into the production of *Mathematica*, commerce has definitely made its entry into the field of computer algebra, accompanied by less realistic claims about capabilities (q.v., [215]). On the positive side, the attention of many scientists has now been drawn to computer algebra and to the use of computer algebra tools in research and education. Another advantage has been the growing interest of developers of computer algebra systems in friendly user interfaces, good documentation, and ample user support. A wealth of information and user’s contributions can be found at the WWW server of Wolfram Research, Inc., and more specifically at the electronic library MathSource [174]. Version 4 of the system, launched in 1999, came with a new front end and with many new mathematical features. Later releases of *Mathematica* offer Internet facilities and connectivity with other programming languages.

The aforementioned general purpose systems manipulate formulae if the entire formula can be stored inside the main memory of the computer. This is the only limit to the size of formulae. The symbolic manipulation program FORM [128, 188, 225, 227] has been designed to deal with formulae of virtually infinite size (q.v., [226]). On the other hand, the size of the set of instructions in FORM is somewhat limited.

Magma [23, 24, 25, 43] is the successor of Cayley [39, 42], released in 1994, and designed around the algebraic notions of structure and morphism. Its aim is to support computation in algebra, number theory, geometry and algebraic combinatorics. This is achieved through the provision of extensive machinery for groups, rings, modules, algebras, geometric structures and finite incidence structures (designs, codes, graphs). Two basic design principles are the strong typing scheme derived from modern algebra whereby

types correspond to algebraic structures and the integration of algorithmic and database knowledge.

MuPAD [88, 100] stands for Multi Processing Algebra Data Tool. It is a system for symbolic and numeric computation, parallel mathematical programming and mathematical visualization. MuPAD is freely distributed for educational and non-commercial use from the website [www.mupad.de](http://www.mupad.de). It is still in active development at the University of Paderborn in cooperation with SciFace Software GmbH & Co. KG and its mathematical contents is growing.

A portable system for parallel symbolic computation through Maple exists as well: it is called  $\parallel$ MAPLE $\parallel$  (speak: parallel MAPLE) [209]. The system is built as an interface between the parallel declarative language Strand [84] and the sequential computer algebra system Maple, thus providing the elegance of Strand and the powerfulness of the existing sequential algorithms in Maple. More recently, a Java-based package for writing parallel programs in Maple and executing them on networks of computers has been developed at RISC Linz. It is called ‘Distributed Maple’ [208].

Last (but not least) in the row is AXIOM [70, 71, 140, 219]. It is a powerful general purpose system developed in the eighties at the IBM Thomas J. Watson Research Laboratory under the name of “Scratchpad.” In contrast to most other general purpose systems, which only allow calculations in a specific algebraic domain, e.g., the field of rational numbers or the ring of polynomials over the integers, AXIOM allows its users to define and handle distinct types of algebraic structures. But alas, this computer algebra system died in 2001, when NAG released the last version 2.3 to existing customers.

## 1.3 Some Properties of Computer Algebra Systems

Computer algebra systems differ from one another, but they share many properties. We shall illustrate common properties with examples from Maple.

Computer algebra systems are *interactive* programs that, in contrast to numerical computer programs, allow mathematical computations with *symbolic* expressions. Typically, the user enters a mathematical expression or an instruction, which the computer algebra system subsequently tries to execute. Given the result of the computation the user may enter a new instruction. This may lead to a fruitful computer algebra session. As an easy example of the use of Maple, we shall compute the stationary point of the real function

$$x \mapsto \arctan\left(\frac{2x^2 - 1}{2x^2 + 1}\right),$$

as well as the value of the function at this point. As we shall see later on in this section, Maple can compute the minimum on its own. Here, we only use the example to show some of the characteristics of computer algebra systems. Below is a screen dump of a complete work session with Maple 8, on a PC running the worksheet interface.

Let us take a closer look at this example. When Maple is started by choosing the Maple command from the appropriate menu or by double-clicking the corresponding icon on the desktop, an empty worksheet appears, except that the system prints the greater-than symbol “>” on the first line in order to prompt the user to enter an instruction. The symbol “>” is called the *Maple prompt*.

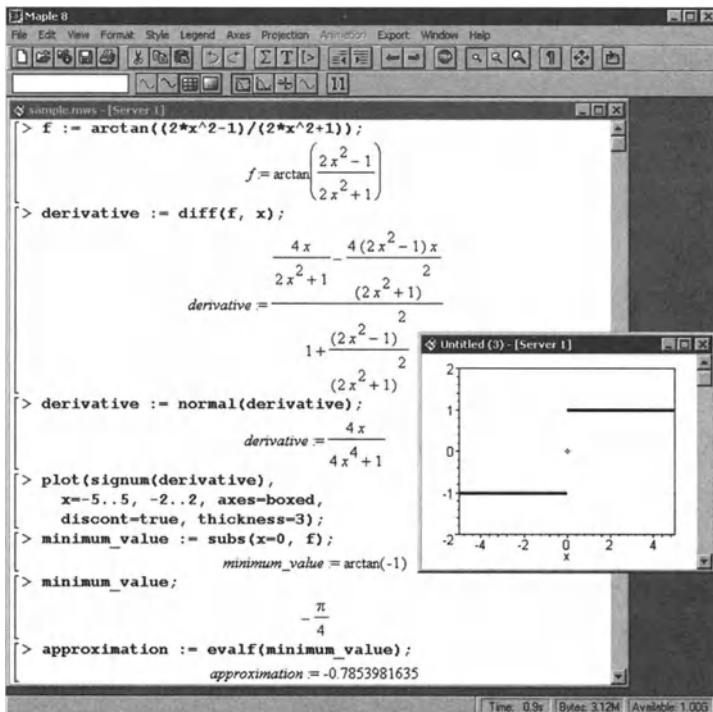


Figure 1.1. The Maple environment with a worksheet.

In the first command we enter the formula  $f$ , ending the input line with a semicolon, and pressing the RETURN key. The last two key strokes signal Maple to start to work. In this case, the formula is shown in two-dimensional mathematical notation of textbook quality. What strikes one most is that the system allows the use of symbols like  $x$ . In most numerical programming languages this would immediately cause an error; but not in systems for *symbolic* computations!

Each time Maple has executed an instruction, it prints the prompt and waits for another command. We decide to consider  $f$  as a function, differentiate it, and assign the result to the variable called *derivative*. Maple's answer is a rather complicated expression. So, we **normalize** the rational function. The answer is a simple expression of which the sign can be plotted. From this plot we immediately conclude that the original function has a minimum at  $x = 0$ . The minimum value  $-\frac{1}{4}\pi$  is obtained by **substitution** of  $x = 0$  in  $f$ . We obtain an approximate floating-point result by use of the procedure **evalf**. The name **evalf** — short for “**evaluate using floating-point arithmetic**” — is already the fourth example that shows Maple's general philosophy in choosing names: Use a short, easy-to-remember name for a procedure that describes its functionality. In addition to this, we have given meaningful names to variables, which describe their use.

We see that Maple leaves it to us to find our way through the computation. We must decide, on the basis of the second result, to try and find a less complicated formula for the derivative. One may wonder why the system itself does not perform this more or less obvious simplification. But remember, it is not always clear when and how to simplify. In many cases more than one simplification is possible and it is the mathematical context that actually determines which simplification is appropriate. For example, the rational expression

$$\frac{(x^2 - 1)(x^2 - x + 1)(x^2 + x + 1)}{(x - 1)^6}$$

can be transformed into the compact expression

$$\frac{x^6 - 1}{(x - 1)^6},$$

but also into a form suitable for integration, viz.,

$$1 + \frac{6}{(x - 1)^5} + \frac{15}{(x - 1)^4} + \frac{20}{(x - 1)^3} + \frac{15}{(x - 1)^2} + \frac{6}{x - 1}.$$

Another problem with automatic simplification is that in many computations one cannot predict the size and shape of the results and therefore must be able to intervene at any time. A procedure that works fine in one case might be a bad choice in another case. For example, one might think that it is always a good idea to factorize an expression. For example, the factorization of

$$x^8 + 8x^7 + 28x^6 + 56x^5 + 70x^4 + 56x^3 + 28x^2 + 8x + 1$$

is

$$(x + 1)^8.$$

However (surprise!) apart from being expensive, the factorization of the relatively simple

$$x^{26} + x^{13} + 1$$

yields

$$(x^{24} - x^{23} + x^{21} - x^{20} + x^{18} - x^{17} + x^{15} - x^{14} + x^{12} - x^{10} + x^9 - x^7 + x^6 - x^4 + x^3 - x + 1)(x^2 + x + 1).$$

For these reasons, Maple only applies automatic simplification rules when there is no doubt about which expression is simpler:  $x + 0$  should be simplified to  $x$ ,  $3x$  is simpler than  $x + x + x$ ,  $x^3$  is better than  $x \times x \times x$ , and  $\sin(\pi)$  should be simplified to 0. Any other simplification is left to the user's control; Maple only provides the tools for such jobs.

Automatic simplification sometimes introduces loss of mathematical correctness. For example, the automatic simplification of  $0 \times f(1)$  to 0 is not always correct. An exception is the case  $f(1)$  is undefined or infinity. The automatic simplification is only wanted if  $f(1)$  is finite but difficult to compute. In cases like this, designers of computer algebra systems have to choose between rigorous mathematical correctness and usability/efficiency of their systems (q.v., [77]). In Maple and many other systems the scales sometimes tip to simplifications that are not 100% safe in every case.

Another remarkable fact in the first example is that Maple computes the exact value  $-\frac{1}{4}\pi$  for the function in the origin and does not return an approximate value like 0.785398. This is a second aspect of computer algebra systems: the emphasis on *exact arithmetic*. In addition, computer algebra systems can carry out floating-point arithmetic in a *user-defined precision*. For example, in Maple the square  $\tan^2(\pi/12)$  can be computed exactly, but the numerical approximation in 25-digits floating-point notation can also be obtained.

```
> real_number := tan(Pi/12)^2;
real_number := tan( $\frac{\pi}{12}$ )^2
> real_number := convert(real_number, radical);
real_number := (2 -  $\sqrt{3}$ )^2
> real_number := expand(real_number);
real_number := 7 - 4 $\sqrt{3}$ 
> approximation := evalf[25](real_number);
approximation := .071796769724490825890216
```

Computer algebra systems like Maple contain a substantial amount of built-in mathematical knowledge. This makes them good mathematical assistants. In calculus they differentiate functions, compute limits, and

compute series expansions. Integration (both definite and indefinite), is one of the highlights in computer calculus. Maple uses non-classical algorithms such as the Risch algorithm for integrating elementary functions [31, 98, 203], instead of the heuristic integration methods that are described in most mathematics textbooks.

With the available calculus tools, one can easily explore mathematical functions. In the Maple session below we shall explore the previously defined function  $f$ . The *sharp symbol #* followed by text is one of Maple's ways of allowing comments during a session; in combination with the names of variables and Maple procedures, this should explain the script sufficiently. If an input line is ended by a colon instead of a semicolon, then Maple does not print its results. Spaces in input commands are optional, but at several places we have inserted them to make the input more readable.

```
> f := arctan((2*x^2-1)/(2*x^2+1)); # enter formula

$$f := \arctan\left(\frac{2x^2 - 1}{2x^2 + 1}\right)$$

> df := normal(diff(f, x)); # differentiate f

$$df := 4 \frac{x}{4x^4 + 1}$$

> F := integrate(df, x); # integrate derivative

$$F := \arctan(2x^2)$$

> normal(diff(F-f, x)); # verify F = f + Pi/4

$$0$$

> eval(F-f, x=0);

$$\frac{\pi}{4}$$

> extrema(f, {}, x, stationary_points);

$$\{\arctan(-1)\}$$

> %; # extra evaluation

$$\left\{\frac{\pi}{4}\right\}$$

> stationary_points;

$$\{x = 0\}$$

> # this value was assigned by the call of 'extrema'
> solve(f=0, x); # compute the zero's of f

$$\frac{\sqrt{2}}{2}, -\frac{\sqrt{2}}{2}$$

> series(f, x=0, 15);
```

```


$$-\frac{\pi}{4} + 2x^2 - \frac{8}{3}x^6 + \frac{32}{5}x^{10} - \frac{128}{7}x^{14} + O(x^{15})$$

> with(numapprox):
> pade(f, x, [6,4]);

$$\frac{-15\pi + 120x^2 - 36\pi x^4 + 128x^6}{12(5 + 12x^4)}$$

> chebpade(f, x, [2,2]);

$$\frac{-0.007904471007 T(0, x) + .4715125862 T(2, x)}{T(0, x) + .4089934686 T(2, x)}$$

> subs(simplify({T(0,x)=ChebyshevT(0,x),
> T(2,x)=ChebyshevT(2,x)}), %);

$$\frac{-0.4794170572 + .9430251724 x^2}{.5910065314 + .8179869372 x^2}$$

> limit(f, x=infinity);

$$\frac{\pi}{4}$$

> series(f, x=infinity);

$$\frac{\pi}{4} - \frac{1}{2x^2} + O(\frac{1}{x^6})$$

> plot([f, df], x=-5..5, linestyle=[0, 4], color=black,
> legend=["f", "f'"], title="graph of f and f'");

```

The graph is shown in Figure 1.2.

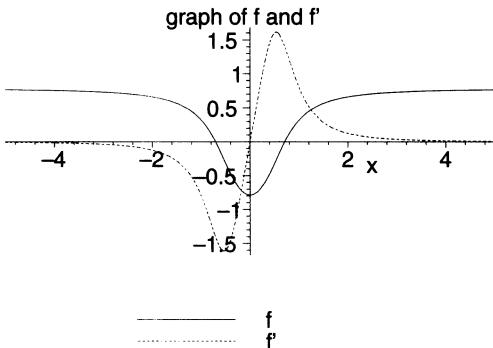


Figure 1.2. Graph of  $(x, y) \mapsto \arctan\left(\frac{2x^2-1}{2x^2+1}\right)$  and its derivative.

Other impressive areas of computer algebra are polynomial calculus, the solution of systems of linear and nonlinear equations, the solution of recurrence equations and differential equations, calculations on matrices with numerical and symbolic coefficients, and tensor calculus. Various tools for manipulation of formulae are present: selection and substitution of

parts of expressions, restricted simplification, simplification rules, pattern matching, and so on. We may call computer algebra systems *mathematical expert systems* with which mathematical problems can be solved in a more productive and accurate way than with pencil and paper.

In addition to functioning as symbolic and algebraic calculators, most computer algebra systems can be used as *programming languages* for implementing new mathematical algorithms. By way of illustration we write a Maple program that computes the Bessel polynomials  $y_n(x)$ . Recall [116] that they can be recursively defined by

$$\begin{aligned}y_0(x) &= 1, \\y_1(x) &= x + 1, \\y_n(x) &= (2n - 1)x y_{n-1}(x) + y_{n-2}(x), \quad \text{for } n > 1.\end{aligned}$$

```
> Y := proc( n::nonnegint, x::name )
>   if n=0 then 1
>   elif n = 1 then x+1
>   else Y(n,x) := expand( (2*n-1)*x*Y(n-1,x) + Y(n-2,x) )
>   end if
> end proc:
> Y(5,z);
```

$$945 z^5 + 945 z^4 + 420 z^3 + 105 z^2 + 15 z + 1$$

The Maple programming language is reminiscent of Algol68 without declarations, but also includes several functional programming paradigms.

## 1.4 Advantages of Computer Algebra

The long-term goal of computer algebra is to automate as much as possible the mathematical problem solving process. Although present computer algebra systems are far from being automatic problem solvers, they are already useful, if not indispensable, tools in research and education. Of course, it takes time to familiarize oneself with a computer algebra system, but this time is well-spent. In this section, some of the more important reasons for learning and using a computer algebra system will be illustrated with Maple examples, a few of which are rather advanced mathematically. All computations will be carried out with Maple 8, on a PC running Windows 2000, with a 1.7 Ghz Pentium 4 processor having 512 MB main memory. This does not imply that the same results could not have been obtained on a much smaller machine, but the timings would be different.

The main advantage of a computer algebra system is its ability to carry out large algebraic computations. Although many calculations are straightforward standard manipulations that can be calculated with pencil and paper, the larger the formulae, the harder the work and the less the chance

of success. For this kind of computation a computer algebra system is an excellent tool. The next three examples demonstrate this.

The first example is one of the problems posed by R. Pavelle [194] as a challenge for computer algebra systems. The object is to prove that

$$\frac{\sin\left(\frac{nz\sqrt{x^2+y^2+z^2}}{\sqrt{y^2+z^2}}\right)}{\sqrt{x^2+y^2+z^2}}$$

is a solution of the fourth order partial differential equation

$$\left(\frac{\partial^2}{\partial x^2}\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}\right) + n^2\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right)\right)f = 0.$$

The simplification procedures of Maple are powerful enough to solve this problem within a second. We shall use the procedure **radnormal** (**radical normalization**) to simplify the expression, which contains radicals.

```
> settime := time(); # start timing
> f := sin( n*z*sqrt(x^2+y^2+z^2)/sqrt(y^2+z^2) ) /
>     sqrt(x^2+y^2+z^2);
>
sin( n z sqrt(x^2 + y^2 + z^2)
      )
f := -----
                           sqrt(y^2 + z^2)
                           sqrt(x^2 + y^2 + z^2)
> radnormal(diff(diff(f,x$2) + diff(f,y$2) + diff(f,z$2),
> x$2) + n^2*(diff(f,x$2) + diff(f,y$2)));
0
> cpu_time = (time()-settime)*second; # computing time
cpu_time = 0.671 second
```

In the second example, the objective is find the generating function for dimensions of representations of the Lie group of type  $G_2$  (q.v., [57, 58]). So, the attempt is made to find a rational function  $F(x, y)$  such that

$$F(x, y) = \sum_{k,l \geq 0} G2(k, l) x^k y^l,$$

where  $G2(k, l)$  is the following polynomial expression.

```
> G2 := (k, l) -> 1/5!* (k+1)*(l+1)*(k+l+2)*(k+2*l+3)*
>     (k+3*l+4)*(2*k+3*l+5);
G2 := (k, l) ->
(k + 1) (l + 1) (k + l + 2) (k + 2 l + 3) (k + 3 l + 4) (2 k + 3 l + 5)
5!
```

Here, we have used Maple's arrow notation for functional operators. In this way  $G2$  is a function with values defined in terms of its arguments instead

of just a formula. Maple has a package called `genfunc` for manipulating rational generating functions. We use it to solve our problem.

```
> with(genfunc):      # load genfunc package
> settim := time(): # start timing
> F := rgf_encode(rgf_encode(G2(k,1), k, x), l, y):
> F := sort(factor(F));


$$F := (x^4 y^4 + 8 x^4 y^3 + x^3 y^4 + 8 x^4 y^2 - 26 x^3 y^3 + x^4 y - 41 x^3 y^2 + 15 x^2 y^3 - 6 x^3 y + 78 x^2 y^2 - 6 x y^3 + 15 x^2 y - 41 x y^2 + y^3 - 26 x y + 8 y^2 + x + 8 y + 1) / ((x - 1)^6 (y - 1)^6)$$

> cpu_time = (time() - settim)*second; # computing time
cpu_time = .501 second
```

An example taken from scientific life where Maple could have played the role of mathematical assistant can be found in [240]. In this paper the Laplace-Beltrami operator  $\Delta$  in hyperspherical coordinates is wanted. To this end, the Jacobian of the coordinate mapping, the metric tensor, and its inverse are calculated. Following are quotations from the paper:

“It is not difficult to compute  $\frac{\partial \mathbf{Y}}{\partial q_i}$ , and it is not difficult, but tedious, to compute the traces in Eq.(32B). After quite some algebra we find, ...”

“It is also tedious to invert  $\mathbf{g}$ . After several pages of computation of minors we find, ...”

These remarks are indeed true when one carries out these calculations with pencil and paper; but not if one lets Maple carry out the computations! Below is the Maple calculation, as much as possible in the notation of [240]. Don’t worry if you do not fully understand individual commands: details will be provided later in this book.

The first step in the computation is to define the coordinate mapping  $Y$  and to build the metric tensor  $G$ . This turns out to be the most time-consuming step in the computation.

```
> settim := time(): # start timing
> with(LinearAlgebra): # load the linear algebra package
> R[z] := x -> <<cos(x)| -sin(x)| 0>, 
>      <sin(x)| cos(x)| 0>, <0| 0| 1>:
> 'R[z](phi)' = R[z](phi);


$$R_z(\phi) = \begin{bmatrix} \cos(\phi) & -\sin(\phi) & 0 \\ \sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

> R[y] := x -> <<cos(x)| 0| -sin(x)>, <0| 1| 0>,
>      <sin(x)| 0| cos(x)>:
> 'R[y](phi)' = R[y](phi);
```

$$R_y(\phi) = \begin{bmatrix} \cos(\phi) & 0 & -\sin(\phi) \\ 0 & 1 & 0 \\ \sin(\phi) & 0 & \cos(\phi) \end{bmatrix}$$

```

> T := x -> <<cos(x)+sin(x)| 0| 0>,
>      <0| cos(x)-sin(x)| 0>, <0| 0| 0>>;
> 'T(phi)' = T(phi);


$$T(\phi) = \begin{bmatrix} \cos(\phi) + \sin(\phi) & 0 & 0 \\ 0 & \cos(\phi) - \sin(\phi) & 0 \\ 0 & 0 & 0 \end{bmatrix}$$


```

> macro(a=alpha, b=beta, c=gamma, f=phi, t=theta):
> Y := ScalarMultiply(R[z](a) . R[y](b) . R[z](c/2)
> . T(t/2) . R[z](f/2), r/sqrt(2))
> Y1 := map(diff, Y, r): Y2 := map(diff, Y, a):
> Y3 := map(diff, Y, b): Y4 := map(diff, Y, c):
> Y5 := map(diff, Y, t): Y6 := map(diff, Y, f):
> G := Matrix(6, 6, shape=symmetric):
> for i to 6 do for j from i to 6 do
>   G[i,j] := simplify(Trace(Transpose(Y||i) . Y||j))
> end do end do:
> intermediate_cpu_time = (time() - setttime)*seconds;

```



intermediate_cpu_time = 2.394 seconds



Now, we apply some simplification procedures to obtain the formulae in [240]. To shorten the output of the session, we continue to suppress most results. We also show a slightly polished session, admittedly not the first interactive session when the problem was solved.



```

> G := subs(cos(t/2)^2=1/2+1/2*cos(t),
>           cos(c/2)^2=1/2+1/2*cos(c), sin(t/2)=sin(t)/(2*cos(t/2))
>           sin(c/2)=sin(c)/(2*cos(c/2)), G):
> G[2,2] := normal(subs(cos(c)*sin(t)=(cos(b)^2+sin(b)^2)
>           *cos(c)*sin(t), normal(G[2,2]))):
> G[3,3] := normal(G[3,3]):
> G; # this the formula in the paper!
```


$$[1, 0, 0, 0, 0, 0]$$


$$\left[0, \frac{1}{2} r^2 (1 - \sin(\theta) \cos(\gamma) \sin(\beta)^2 + \cos(\beta)^2), \right.$$


$$\left.\frac{1}{2} r^2 \sin(\beta) \sin(\gamma) \sin(\theta), \frac{1}{2} r^2 \cos(\beta), 0, \frac{1}{2} r^2 \cos(\beta) \cos(\theta)\right]$$


$$\left[0, \frac{1}{2} r^2 \sin(\beta) \sin(\gamma) \sin(\theta), \frac{1}{2} r^2 (\cos(\gamma) \sin(\theta) + 1), 0, 0, 0\right]$$


$$\left[0, \frac{1}{2} r^2 \cos(\beta), 0, \frac{r^2}{4}, 0, \frac{1}{4} r^2 \cos(\theta)\right]$$


$$\left[0, 0, 0, 0, \frac{r^2}{4}, 0\right]$$


```

$$\left[ 0, \frac{1}{2} r^2 \cos(\beta) \cos(\theta), 0, \frac{1}{4} r^2 \cos(\theta), 0, \frac{r^2}{4} \right]$$

Due to the screen width and the length of expressions Maple was not able to produce a nice layout, but it can translate the formulae automatically into a format suitable for text processing programs like LATEX [159].

```
> latex(G, "metric_tensor");
```

The LATEX code is not shown, but the result after typesetting is

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{r^2(\cos(\beta)^2 + 1 - \sin(\theta)\cos(\gamma)\sin(\beta)^2)}{2} & \frac{r^2\sin(\gamma)\sin(\beta)\sin(\theta)}{2} & \frac{r^2\cos(\beta)}{2} & 0 & \frac{r^2\cos(\beta)\cos(\theta)}{2} \\ 0 & \frac{r^2\sin(\gamma)\sin(\beta)\sin(\theta)}{2} & \frac{r^2(1+\sin(\theta)\cos(\gamma))}{2} & 0 & 0 & 0 \\ 0 & \frac{r^2\cos(\beta)}{2} & 0 & \frac{r^2}{4} & 0 & \frac{r^2\cos(\theta)}{4} \\ 0 & 0 & 0 & 0 & \frac{r^2}{4} & 0 \\ 0 & \frac{r^2\cos(\beta)\cos(\theta)}{2} & 0 & \frac{r^2\cos(\theta)}{4} & 0 & \frac{r^2}{4} \end{bmatrix}$$

Let us compute the Jacobian.

```
> determinant := simplify(Determinant(G));
> determinant := normal(subs(cos(b)^2=1-sin(b)^2,
> determinant));
> determinant := normal(subs(cos(t)^2-1=-sin(t)^2,
> cos(t)=sin(2*t)/(2*sin(t)), determinant));
> Jacobian := sqrt(determinant) assuming 0<=r, 0<=t,
> t<=Pi/2, 0<=b, b<=Pi;
```

$$Jacobian := \frac{1}{32} r^5 \sin(2\theta) \sin(\beta)$$

This is formula (33) in the paper. The assumptions on the hyperspherical coordinates were necessary to have the square root expression automatically simplified by Maple.

Next, we compute the inverse of the metric tensor.

```
> GINV := map(simplify, MatrixInverse(G));
> GINV := subs(cos(t)^2=1-sin(t)^2,
> cos(b)^2=1-sin(b)^2, GINV);
> cpu_time = (time()-settime)*seconds; # computing time
```

$$cpu\_time = 3.425 \text{ seconds}$$

We do not show the inverse metric tensor, but all entries except GINV[4,4] are in the shape of formula (34) in the paper. Casting GINV[4,4] into good shape is not difficult; we skip this last part of the computation. Anyway, the calculation can easily be done in about 4 seconds of computing time and paper-ready formulae are obtained too!

Another example from science where computer algebra enables the researcher to finish off a proof of a mathematical result that requires a lot of straightforward but tedious computations can be found in [59]. There, a

purely algebraic problem, related to proving the existence of a certain finite subgroup of a Lie group, could be reduced to the problem of solving a set of linear equations in 240 variables with coefficients from the field of 1831 elements. This system of linear equations was easily shown to have a unique solution by computer. By pencil and paper this is almost impossible, but by computer this is quite feasible. The role of Maple in this computation is described in [167].

In the last two worked-out examples we have used Maple to determine a generating function and to compute a metric tensor and its inverse, respectively. Now, one may not think much of mathematical results obtained by a computer program, but remember that there are many independent ways to check the answers. One may even use Maple itself to check the answers or to enlarge one's confidence in the results: e.g., one can compute the first terms of the Taylor series and compare them with the original coefficients, and one can multiply the metric tensor and its inverse as an extra check on the answers.

Symbolic and algebraic computation often precedes numerical computation. Mathematical formulae are first manipulated to cast them into good shape for final numerical computation. For this reason, it is important that a computer algebra system provides a good interface between these two types of computation. Maple can generate C, FORTRAN, and Java expressions from Maple expressions. Double precision arithmetic and code optimization are optional. For example, the submatrix of the metric tensor  $G$  in the previous example consisting of the first two rows can be converted into FORTRAN as shown below. Two technicalities play a role: You have to get rid of the link between Euler's constant and the Greek character  $\gamma$  in Maple, e.g., by using the name **Gamma**, and you must use (named) arrays instead of nested lists for matrices.

```

> # recognition of Euler's constant
> CodeGeneration[Fortran](gamma);

cg = 0.5772156649D0

> H := SubMatrix(G, 2..3, 1..6):
> H := eval(% , gamma=Gamma); # replace gamma by Gamma

H :=

$$\begin{bmatrix} 0, \frac{1}{2} r^2 (1 - \sin(\theta) \cos(\Gamma) \sin(\beta)^2 + \cos(\beta)^2), \\ \frac{1}{2} r^2 \sin(\beta) \sin(\Gamma) \sin(\theta), \frac{1}{2} r^2 \cos(\beta), 0, \frac{1}{2} r^2 \cos(\beta) \cos(\theta) \end{bmatrix}$$


$$\begin{bmatrix} 0, \frac{1}{2} r^2 \sin(\beta) \sin(\Gamma) \sin(\theta), \frac{1}{2} r^2 (\cos(\Gamma) \sin(\theta) + 1), 0, 0, 0 \end{bmatrix}$$


> CodeGeneration[Fortran](H, optimize=true,
> functionprecision=double, coercetypes=false);

```

```

t1 = r ** 2
t2 = dsin(theta)
t3 = dcos(Gamma)
t4 = t2 * t3
t5 = dsin(beta)
t6 = t5 ** 2
t8 = dcos(beta)
t9 = t8 ** 2
t14 = dsin(Gamma)
t17 = t1 * t5 * t14 * t2 / 2
t18 = t1 * t8
t20 = dcos(theta)
cg(1,1) = 0
cg(1,2) = t1 * (1 - t4 * t6 + t9) / 2
cg(1,3) = t17
cg(1,4) = t18 / 2
cg(1,5) = 0
cg(1,6) = t18 * t20 / 2
cg(2,1) = 0
cg(2,2) = t17
cg(2,3) = t1 * (t4 + 1) / 2
cg(2,4) = 0
cg(2,5) = 0
cg(2,6) = 0

```

The answers obtained with a computer algebra system are either exact or in a user-defined precision. They can be more accurate than hand calculations [146]; and they can lead to many corrections to integral tables. Below, we give two examples of integrals incorrectly tabulated in one of the most popular tables, viz., Gradshteyn–Ryzhik [109]:

1. Formula 2.269

$$\int \frac{1}{x\sqrt{(bx+cx^2)^3}} dx = \frac{2}{3} \left( -\frac{1}{bx} + \frac{4c}{b^2} - \frac{8c^2x}{b^3} \right) \frac{1}{\sqrt{bx+cx^2}}.$$

2. Formula 3.828(19)

$$\int_0^\infty \frac{\sin^2(ax) \sin^2(bx) \sin(2cx)}{x} dx =$$

$$\frac{\pi}{16} (1 + \text{sgn}(c-a+b) + \text{sgn}(c-b+a) - 2\text{sgn}(c-a) - 2\text{sgn}(c-b)),$$

where  $a, b, c > 0$ .

As shown below Maple does not only give correct answers but also more information on conditions of the parameters. In the example we shall use the percentage symbol to refer to the previous result.

```
> Integrate(1/(x*sqrt((b*x+c*x^2)^3)), x);
```

$$\int \frac{1}{x\sqrt{(bx+cx^2)^3}} dx$$

```
> value(%);

$$\frac{2(-\sqrt{bx+cx^2}b^2 + 4\sqrt{bx+cx^2}bcx + 5\sqrt{bx+cx^2}c^2x^2 + 3c^2\sqrt{x(b+cx)}x^2)\sqrt{x(b+cx)}}{(3xb^3\sqrt{x^3(b+cx)^3})}$$

> factor(%);

$$\frac{2(b+cx)(-b^2 + 4bcx + 8c^2x^2)}{3b^3\sqrt{x^3(b+cx)^3}}$$

```

Assume that all variables are positive and simplify the above result under this condition.

```
> simplify(%) assuming positive;
```

$$\frac{2(-b^2 + 4bcx + 8c^2x^2)}{3b^3x^{(3/2)}\sqrt{b+cx}}$$

It is a bit more work to get a look-alike of the tabulated result.

```
> subs(b+c*x=y/x, %);
```

$$\frac{2(-b^2 + 4bcx + 8c^2x^2)}{3b^3x^{(3/2)}\sqrt{\frac{y}{x}}}$$

```
> simplify(%) assuming positive;
```

$$\frac{2(-b^2 + 4bcx + 8c^2x^2)}{3b^3x\sqrt{y}}$$

```
> expand(%);
```

$$-\frac{2}{3bx\sqrt{y}} + \frac{8c}{3b^2\sqrt{y}} + \frac{16xc^2}{3b^3\sqrt{y}}$$

```
> collect(3/2*, sqrt(y));
```

$$\frac{-\frac{1}{bx} + \frac{4c}{b^2} + \frac{8xc^2}{b^3}}{\sqrt{y}}$$

```
> subs(y=b*x+c*x^2, 2/3*%);
```

$$\frac{2(-\frac{1}{bx} + \frac{4c}{b^2} + \frac{8xc^2}{b^3})}{3\sqrt{bx+cx^2}}$$

The mistake in the tabulated result is an incorrect minus sign. In the fourth edition of Gradshteyn–Ryzhik [109] in 1980 this has been corrected. Anyway, the result in an integral table you can only believe or not; in Maple you can verify the result by differentiation.

```
> simplify(diff(%,x) - 1/(x*sqrt((b*x+c*x^2)^3)))
> assuming positive;
```

0

The second example is more intriguing: the tabulated result is wrong and Maple finds a more general answer.

```
> Integrate(sin(a*x)^2*sin(b*x)^2*sin(2*c*x)/x,
> x=0..infinity);
```

$$\int_0^\infty \frac{\sin(a x)^2 \sin(b x)^2 \sin(2 c x)}{x} dx$$

```
> value(%);
```

$$\begin{aligned} & \frac{1}{8} \operatorname{csgn}(c) \pi + \frac{1}{16} \operatorname{csgn}(-2c+2b) \pi - \frac{1}{16} \operatorname{csgn}(2c+2b) \pi \\ & + \frac{1}{16} \operatorname{csgn}(-2c+2a) \pi - \frac{1}{16} \operatorname{csgn}(2c+2a) \pi \\ & + \frac{1}{32} \operatorname{csgn}(-2b+2c+2a) \pi + \frac{1}{32} \operatorname{csgn}(2b+2c+2a) \pi \\ & - \frac{1}{32} \operatorname{csgn}(-2b-2c+2a) \pi - \frac{1}{32} \operatorname{csgn}(2b-2c+2a) \pi \end{aligned}$$

Here, Maple gives the solution for general  $a$ ,  $b$ , and  $c$ . You can specialize to the case of positive variables, which is the only case tabulated in [109].

```
> % assuming positive; # all variables > 0
```

$$\begin{aligned} & \frac{\pi}{32} + \frac{1}{16} \operatorname{signum}(-2c+2b) \pi + \frac{1}{16} \operatorname{signum}(-2c+2a) \pi \\ & + \frac{1}{32} \operatorname{signum}(-2b+2c+2a) \pi - \frac{1}{32} \operatorname{signum}(-2b-2c+2a) \pi \\ & - \frac{1}{32} \operatorname{signum}(2b-2c+2a) \pi \end{aligned}$$

Let us get rid of the 2's in the signs and factorize.

```
> factor(eval(%), signum=(signum@primpart));
```

$$\begin{aligned} & \frac{1}{32} \pi (1 + 2 \operatorname{signum}(b - c) + 2 \operatorname{signum}(-c + a) + \operatorname{signum}(-b + c + a) \\ & - \operatorname{signum}(-b - c + a) - \operatorname{signum}(b - c + a)) \end{aligned}$$

So, a constant was wrong and a term was missing in the tabulated result.

In many cases of integration, conditions on parameters are important to obtain an exact result. Below is the example of the definite integral

$$\int_0^\infty \frac{t^{1/3} \ln(at)}{(b+2t^2)^2} dx, \quad a, b > 0.$$

```
> assume(a>0, b>0):
> normal(integrate(t^(1/3)*ln(a*t)/(b+2*t^2)^2,
> t=0..infinity));
```

$$\frac{1}{36} \frac{\pi 2^{(1/3)} (2 \ln(a) \sqrt{3} + \pi - 3 \sqrt{3} - \sqrt{3} \ln(2) + \sqrt{3} \ln(b))}{b^{(4/3)}}$$

The tildes after  $a$  and  $b$  in the above result indicate that these variables have certain assumed properties. Maple can inform its user about these properties.

```
> about(a);
Originally a, renamed a~:
is assumed to be: RealRange(Open(0),infinity)
```

Integration is also a good illustration of another advantage of computer algebra: it provides easy access to advanced mathematical techniques and algorithms. When input in Maple, the following two integrals

$$\int x^2 \exp(x^3) dx \quad \text{and} \quad \int x \exp(x^3) dx$$

give different kinds of response:

$$\frac{1}{3} \exp(x^3) \quad \text{and} \quad \int x \exp(x^3) dx.$$

This means more than just the system's incompetence to deal with the latter integral. Maple can decide, via the Risch algorithm [31, 98, 203], that for this integral no closed form in terms of elementary functions exists. This contrasts with the heuristic methods usually applied when one tries to find an answer in closed form. In the heuristic approach one is never sure whether indeed no closed form exists or that it is just one's mathematical incompetence. The Risch algorithm however is a complicated algorithm that is based on rather deep mathematical results and algorithms and that involves many steps that cannot be done so easily by pencil and paper. Computer algebra enables its user to apply such advanced mathematical results and methods without knowing all details.

Using a computer algebra system, one can concentrate on the analysis of a mathematical problem, leaving the computational details to the computer. Computer algebra systems also invite one to do "mathematical experiments". They make it easy to test mathematical conjectures and to propose new ones on the basis of calculations (q.v., [20, 165]). As an example of such a mathematical experiment, we shall conjecture a formula for the determinant of the  $n \times n$  matrix  $A_n$  defined by

$$A_n(i, j) := x^{\gcd(i,j)}.$$

The first thing to do is to compute a few determinants, and look and see.

```
> numex := 7: # number of experiments
> dets := Vector(numex):
> macro(det=LinearAlgebra[Determinant]):
> for n to numex do
>   A[n] := Matrix(n, n, shape=symmetric):
```

```

>      for i to n do
>          for j to i do
>              A[n][i,j] := x^igcd(i,j)
>          end do
>      end do:
>      dets[n] := factor(det(A[n])):
>      print(dets[n])
>  end do:

x

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


$$x^3(x+1)(x-1)^2$$


$$x^5(x+1)^2(x-1)^3$$


$$x^6(x^2+1)(x+1)^3(x-1)^4$$


$$x^7(x^2+1)(x^3+x-1)(x+1)^4(x-1)^5$$


$$x^8(x^2+x+1)(x^2-x+1)(x^2+1)(x^3+x-1)(x+1)^5(x-1)^6$$


```

At first sight, it has not been a success. But, look at the quotients of successive determinants.

```

>      for i from 2 to numex do
>          quo(dets[i], dets[i-1], x)
>      end do;


$$x^2 - x$$


$$x^3 - x$$


$$x^4 - x^2$$


$$x^5 - x$$


$$x^6 - x^3 - x^2 + x$$


$$x^7 - x$$


```

In the  $n^{\text{th}}$  polynomial only powers of  $x$  appear that have divisors of  $n$  as exponents. Thinking of Möbius-like formulae and playing a little more with Maple, the following conjecture comes readily to mind.

**Conjecture.**  $\det A_n = \prod_{j=1}^n \phi_j(x)$ , where the polynomials  $\phi_j(x)$  are defined by  $x^n = \sum_{d|n} \phi_d(x)$ .

### Some Properties.

(i) If  $p$  is a prime number, then

$$\phi_p(x) = x^p - x,$$

and for any natural number  $r$ ,

$$\phi_{p^r}(x) = \phi_p(x^{p^{r-1}}).$$

(ii) If  $p$  is a prime number, not dividing  $n$ , then

$$\phi_{pn}(x) = \phi_n(x^p) - \phi_n(x).$$

(iii) Let  $n = p_1^{r_1} \cdots p_s^{r_s}$  be a natural number with its prime factorization, then

$$\phi_n(x) = \phi_{p_1 \cdots p_s}(x^{p_1^{r_1-1} \cdots p_s^{r_s-1}}).$$

(iv) We have

$$\phi_n(x) = \sum_{d|n} \mu\left(\frac{n}{d}\right) x^d = \sum_{d|n} \mu(d) x^{(n/d)},$$

where  $\mu$  is the Möbius function such that  $\mu(1) = 1$ ,  $\mu(p_1 \cdots p_s) = (-1)^s$  if  $p_1, \dots, p_s$  are distinct primes, and  $\mu(m) = 0$  if  $m$  is divisible by the square of some prime number.

For the interested reader:  $\frac{1}{d} \phi_d(q)$  is equal to the number of monic irreducible polynomials of degree  $d$  in one variable and with coefficients in a finite field with  $q$  elements [176].

However much mathematical knowledge has been included in a computer algebra system, it is still important that an experienced user can enhance the system by writing procedures for personal interest. The author implemented in Maple several algorithms for inversion of polynomial mappings according to the methods developed in [76]. With these programs it is possible to determine whether a polynomial mapping has an inverse that is itself a polynomial mapping, and if so, to compute the inverse. We show an example of an invertible mapping in three unknowns.

```
> read "invtol.m"; # load user-defined package
> P := [ x^4 + 2*(y+z)*x^3 + (y+z)^2*x^2 + (y+1)*x
>       + y^2 + y*z, x^3 + (y+z)*x^2 + y, x + y + z ];
P := [x^4 + 2(y + z)x^3 + (y + z)^2x^2 + (y + 1)x + y^2 + yz,
      x^3 + (y + z)x^2 + y, x + y + z]
> settim := time(); # start timing
> invtol(P, [x,y,z]); # compute inverse mapping
[x - yz, y - x^2z + 2yz^2x - y^2z^3, z - x - y + yz + x^2z - 2yz^2x + y^2z^3]
> cpu_time = (time()-settim)*second; # computing time
cpu_time = .110 second
```

Computing the inverse of a polynomial mapping with pencil and paper is almost impossible; one needs a computer algebra system for the symbol

crunching. However, one cannot expect designers of computer algebra systems to anticipate all of the needs of their users. One can expect good programming facilities to implement new algorithms oneself.

## 1.5 Limitations of Computer Algebra

What has been said about computer algebra systems so far may have given the impression that these systems offer unlimited opportunities, and that they are a universal remedy for solving mathematical problems. But this impression is too rosy. A few warnings beforehand are not out of place.

Computer algebra systems often make great demands on computers because of their tendency to use up much memory space and computing time. The price one pays for exact arithmetic is often the exponential increase in size of expressions and the appearance of huge numbers. This may even happen in cases where the final answer is simply “yes” or “no.” For example, it is well-known that Euclid’s algorithm yields the greatest common divisor (gcd) of two polynomials. However, this “naive” algorithm does not perform very well. Look at the polynomials

```
> f[1] := 7*x^7 + 2*x^6 - 3*x^5 - 3*x^3 + x + 5;
f1 := 7 x7 + 2 x6 - 3 x5 - 3 x3 + x + 5
> f[2] := 9*x^5 - 3*x^4 - 4*x^2 + 7*x + 7;
f2 := 9 x5 - 3 x4 - 4 x2 + 7 x + 7
```

We want to compute the  $\text{gcd}(f_1, f_2)$  over the rational numbers by Euclid’s algorithm. In the first division step we construct polynomials  $q_2$  and  $f_3$ , such that  $f_1 = q_2 f_2 + f_3$ , where  $\text{degree}(f_3) < \text{degree}(f_2)$  or  $f_3 = 0$ . This is done by long division.

```
> f[3] := sort(rem(f[1], f[2], x, q[2]));
f3 :=  $\frac{70}{27} x^4 - \frac{176}{27} x^3 - \frac{770}{81} x^2 - \frac{94}{81} x + \frac{503}{81}$ 
> q[2];
 $\frac{7}{9} x^2 + \frac{13}{27} x - \frac{14}{81}$ 
```

Next, polynomial  $q_3$  and  $f_4$  are computed such that  $f_2 = q_3 f_3 + f_4$ , where  $\text{degree}(f_4) < \text{degree}(f_3)$  or  $f_3 = 0$ , etc. until  $f_n = 0$ ; then  $\text{gcd}(f_1, f_2) = f_{n-1}$ . The following Maple program computes the polynomial remainder sequence.

```

> Euclid_gcd := proc(f::polynom, g::polynom, x::name)
>   local r:
>   if g = 0 then sort(f)
>   else
>     r := sort(rem(f, g, x));
>     if r <> 0 then print(r) end if;
>     Euclid_gcd(g, r, x)
>   end if
> end proc:
> Euclid_gcd(f[1], f[2], x):

```

$$\begin{aligned}
& \frac{70}{27} x^4 - \frac{176}{27} x^3 - \frac{770}{81} x^2 - \frac{94}{81} x + \frac{503}{81} \\
& \frac{100881}{1225} x^3 + 72 x^2 - \frac{14139}{2450} x - \frac{98037}{2450} \\
& - \frac{16726864175}{10176976161} x^2 - \frac{5255280625}{10176976161} x + \frac{19754564375}{10176976161} \\
& \frac{35171085032244648729}{456796710414528050} x + \frac{6605604895087335357}{456796710414528050} \\
& \frac{240681431042721245661011901925}{121549387831506345564025862481}
\end{aligned}$$

The conclusion is that the polynomials  $f_1$  and  $f_2$  are relatively prime, because their greatest common divisor is a unit. But look at the tremendous growth in the size of the coefficients from 1-digit integers to rational numbers with thirty digits (even though the rational coefficients are always simplified). In [98, 148] one can read about more sophisticated algorithms for gcd computations that avoid blowup of coefficients as much as possible.

The phenomenon of tremendous growth of expressions in intermediate calculations turns up frequently in computer algebra calculations and is known as *intermediate expression swell*. Although it is usually difficult to estimate the computer time and memory space required for a computation, one should always do one's very best to optimize calculations, both by using good mathematical models and by efficient programming. It is worth the effort. For example, with the **LinearAlgebra** package, Maple computes the inverse of the  $9 \times 9$  matrix with  $(i, j)$ -entry equal to  $(iu + x + y + z)^j$  in 535 seconds requiring memory allocation of 6 Mbytes on a PC running Windows 2000, with a 1.7 Ghz Pentium 4 processor having 512 MB main memory. The obvious substitution  $x + y + z \rightarrow v$  reduces the computer time to 1 second and the memory requirements to about 0.5 Mbyte.

A second problem in using a computer algebra system is psychological: how many lines of computer output can one grasp? And when one is faced with large expressions, how can one get enough insight to simplify them? For example, merely watching the computer screen it would be difficult to

recover the polynomial composition

$$f(x, y) = g(u(x, y), v(x, y)),$$

where

$$\begin{aligned}g(u, v) &= u^3v + uv^2 + uv + 5, \\u(x, y) &= x^3y + xy^2 + x^2 + y + 1, \\v(x, y) &= y^3 + x^2y + x,\end{aligned}$$

from its expanded form.

While it is true that one can do numerical computations with a computer algebra system in any precision one likes, there is also a negative side of this feature. Because one uses software floating-point arithmetic instead of hardware arithmetic, numerical computation with a computer algebra system is 100 to 1000 times slower than numerical computation in a programming language like FORTRAN. Hence, for numerical problems, one must always ask oneself “Is exact arithmetic with rational numbers or high-precision floating-point arithmetic really needed, or is a numerical programming language preferable?”

In an enthusiastic mood we characterized computer algebra systems as mathematical expert systems. How impressive the amount of built-in mathematical knowledge may be, it is only a small fraction of mathematics known today. There are many mathematical areas where computer algebra is not of much help yet, and where more research is required: partial differential equations, indefinite and definite integration involving non-elementary functions like Bessel functions, contour integration, surface integration, calculus of special functions, and non-commutative algebra are just a few examples.

Another serious problem and perhaps the trickiest, is the wish to specify at an abstract level the number domain in which one wants to calculate. For example, there are an infinite number of fields, and one may want to write algorithms in a computer algebra system using the arithmetic operations of a field, without the need to say what particular field you work with. Moreover, one may want to define one’s own mathematical structures. AXIOM [70, 71, 140, 219] was the first computer algebra system making steps in this direction. In Maple, the `Domains` package allows its user to create domains in a similar way as AXIOM does.

As far as the syntax and the semantics are concerned, the use of a computer algebra system as a programming language is more complicated than programming in a numerical language like FORTRAN. In a computer algebra system one is faced with many built-in functions that may lead to unexpected results. One must have an idea of how the system works, how data are represented, how to keep data of manageable size, and so on.

Efficiency of algorithms, both with respect to computing time and memory space, requires a thorough analysis of mathematical problems and a careful implementation of algorithms. For example, if we had forgotten to expand intermediate results in the procedure for calculating Bessel polynomials as defined in §1.3, it would have resulted in unnecessarily large expressions. The rather unsuccessful implementation of **Y** would compute  $Y_5(z)$  as follows.

```
> Y := proc(n::nonnegint, x::name)
>   if n=0 then 1
>   elif n=1 then x+1
>   else Y(n,x) := (2*n-1)*x*Y(n-1,x) + Y(n-2,x)
>   end if
> end proc:
> Y(5,z);
```

$$9z(7z(5z(3z(z+1)+1)+z+1)+3z(z+1)+1)+5z(3z(z+1)+1)+z+1$$

Familiarity with the basic features of a computer algebra system, such as elementary data structures and built-in facilities, makes it easier to program efficiently and to foresee some of the many pitfalls in symbolical computation. A good understanding of the computer algebra systems that one has at one's disposal is also prerequisite to making the right choice for the system to use when studying a particular problem. For example, the formula manipulation system FORM [225] is better suited than Maple for doing computations in most non-commutative algebras because, in this system, non-commutative objects are of basic type and pattern matching is an amply supported basic operation. See [60] for a comparison of Maple and FORM in computing with quaternions.

Finally, we mention the technical difficulties in using computer algebra systems. Sometimes, input and output formats and the reuse of expressions are confusing and too far removed from standard mathematical notation. In comparison to global manipulations, local manipulations in formulae are often difficult to carry out. Interfaces with other programming languages are frequently absent or inadequate. And last but not least, bugs are a major issue. See [215] for an entertaining discussion on this issue. It is hoped that some of these remarks will soon be outdated. But systems like Maple will always surprise you, for the good or the bad. We end this section with two surprising results in Maple calculations where you have to assist the system a bit.

The first example: an easy integral that can be done without using a computer algebra system.

```
> Integrate(2*x*(x^2+1)^24, x);
```

$$\int 2x(x^2+1)^{24} dx$$

The answer is clear, but what does Maple give?

```
> value(%);
```

$$\begin{aligned} & 92x^6 + 12x^4 + x^2 + \frac{653752}{5}x^{30} + 178296x^{28} + 81719x^{18} + 178296x^{22} \\ & + \frac{653752}{5}x^{20} + 208012x^{24} + 208012x^{26} + 506x^{42} + \frac{10626}{5}x^{40} \\ & + \frac{10626}{5}x^{10} + 506x^8 + 7084x^{12} + 19228x^{14} + 43263x^{16} + \frac{1}{25}x^{50} \\ & + x^{48} + 12x^{46} + 92x^{44} + 7084x^{38} + 19228x^{36} + 43263x^{34} + 81719x^{32} \end{aligned}$$

Surprise! You may think that factorization helps.

```
> factor(%);
```

$$\begin{aligned} & x^2(x^8 + 5x^6 + 10x^4 + 10x^2 + 5)(x^{40} + 20x^{38} + 190x^{36} + 1140x^{34} \\ & + 4845x^{32} + 15505x^{30} + 38775x^{28} + 77625x^{26} + 126425x^{24} \\ & + 169325x^{22} + 187760x^{20} + 172975x^{18} + 132450x^{16} + 84075x^{14} \\ & + 43975x^{12} + 18760x^{10} + 6425x^8 + 1725x^6 + 350x^4 + 50x^2 + 5)/25 \end{aligned}$$

The problem is, as you may have guessed, in the choice of the integration constant.

```
> factor(% + 1/25);
```

$$\frac{(x^2 + 1)^{25}}{25}$$

You are expecting too much if you think that such problems can always be avoided.

The second example comes from a study in optics [19] and is about integrating expressions consisting only of terms

$$x^n \sin^i x \cos^j x \cosh^k x \sinh^l x,$$

where  $i, j, k, l, n \in \mathbb{N}$ . It can easily be shown that any such integral can be expressed completely in the same kind of terms. An example of what Maple does:

```
> Integrate(x*sin(x)^2*cos(x)*sinh(x)^2*cosh(x), x);
```

$$\int x \sin(x)^2 \cos(x) \sinh(x)^2 \cosh(x) dx$$

```
> value(%);
```

$$\begin{aligned} & \frac{1}{32} \left( \frac{3x}{10} - \frac{2}{25} \right) e^{(3x)} \cos(x) - \frac{1}{32} \left( -\frac{x}{10} + \frac{3}{50} \right) e^{(3x)} \sin(x) \\ & - \frac{1}{192} x e^{(3x)} \cos(3x) + \frac{1}{32} \left( -\frac{x}{6} + \frac{1}{18} \right) e^{(3x)} \sin(3x) - \frac{1}{64} x e^x \cos(x) \\ & + \frac{1}{32} \left( -\frac{x}{2} + \frac{1}{2} \right) e^x \sin(x) + \frac{1}{32} \left( \frac{x}{10} + \frac{2}{25} \right) e^x \cos(3x) \end{aligned}$$

$$\begin{aligned}
& -\frac{1}{32} \left( -\frac{3x}{10} + \frac{3}{50} \right) e^x \sin(3x) + \frac{1}{64} x e^{(-x)} \cos(x) \\
& + \frac{1}{32} \left( -\frac{x}{2} - \frac{1}{2} \right) e^{(-x)} \sin(x) + \frac{1}{32} \left( -\frac{x}{10} + \frac{2}{25} \right) e^{(-x)} \cos(3x) \\
& - \frac{1}{32} \left( -\frac{3x}{10} - \frac{3}{50} \right) e^{(-x)} \sin(3x) + \frac{1}{32} \left( -\frac{3x}{10} - \frac{2}{25} \right) e^{(-3x)} \cos(x) \\
& - \frac{1}{32} \left( -\frac{x}{10} - \frac{3}{50} \right) e^{(-3x)} \sin(x) + \frac{1}{192} x e^{(-3x)} \cos(3x) \\
& + \frac{1}{32} \left( -\frac{x}{6} - \frac{1}{18} \right) e^{(-3x)} \sin(3x)
\end{aligned}$$

To get the formula in the requested form you have to write the exponentials in terms of hyperbolic sines and cosines and work out the intermediate expression.

```

> convert(%, 'trig'): # exp -> trigonometric function
> expand(%);

```

$$\begin{aligned}
& \frac{1}{5} \cos(x) x \sinh(x) \cosh(x)^2 - \frac{1}{6} x \sinh(x) \cosh(x)^2 \cos(x)^3 \\
& - \frac{1}{6} x \cosh(x)^3 \sin(x) \cos(x)^2 + \frac{1}{5} x \cosh(x) \sin(x) \cos(x)^2 \\
& + \frac{1}{18} \sinh(x) \cosh(x)^2 \sin(x) \cos(x)^2 - \frac{1}{10} \cos(x) x \sinh(x) \\
& - \frac{13}{450} \sin(x) \sinh(x) \cosh(x)^2 - \frac{1}{50} \cos(x) \cosh(x)^3 \\
& - \frac{1}{10} \sin(x) x \cosh(x) + \frac{1}{15} \sin(x) x \cosh(x)^3 + \frac{1}{15} x \sinh(x) \cos(x)^3 \\
& - \frac{13}{450} \sinh(x) \sin(x) \cos(x)^2 + \frac{19}{450} \sin(x) \sinh(x) + \frac{1}{50} \cosh(x) \cos(x)^3
\end{aligned}$$

The final step might be to combine the commands into a procedure for further usage.

```

> trigint := proc()
>   integrate(args);
>   convert(%, 'trig');
>   expand(%)
> end proc;
> trigint(x*sin(x)^2*cos(x)^3, x);


$$\begin{aligned}
& \frac{1}{15} x \sin(x) \cos(x)^2 + \frac{2}{15} x \sin(x) + \frac{1}{45} \cos(x)^3 + \frac{2}{15} \cos(x) \\
& - \frac{1}{5} x \cos(x)^4 \sin(x) - \frac{1}{25} \cos(x)^5
\end{aligned}$$

> trigint(x*sin(x)*cos(x)^2, x=0..Pi);

```

This kind of adapting Maple to one's needs happens fairly often. Therefore, this book contains many examples of how to assist the system. Some of them are of a rather sophisticated level. They have been added because merely explaining Maple commands and showing elementary examples does not make you proficient enough at the system for when it comes to real work.

## 1.6 Design of Maple

The name **Maple** is not an acronym of **mathematical pleasure** — great fun as it is to use a computer algebra system — but was chosen to draw attention to its Canadian origin. Since 1980 a lot of work has gone into the development of the **Maple** system by the Symbolic Computation Group of the University of Waterloo and at ETH Zürich. Since 1992 it has been further developed and marketed by Waterloo Maple Software (since 1995, Waterloo Maple Inc.) in collaboration with the original developers.

**Maple** is a computer algebra system open to the public and fit to run on a variety of computers, from a mainframe computer down to desktop computers like Macintosh and PC. The user accessibility shows up best on a mainframe with a time-sharing operating system, where several users of **Maple** can work simultaneously without being a nuisance to each other or other software users, and without excessive load put on the computer. This is possible because of the modular design of **Maple**. It consists of several parts: the user interface called the *Iris*, the basic algebraic engine or *kernel*, the external *library*, and optionally the so-called *share library* with contributions of **Maple** users or other private libraries.

The Iris and the kernel form the smaller part of the system, which has been written in the programming language C; they are loaded when a **Maple** session is started. The Iris handles input of mathematical expressions (parsing and notification of errors), display of expressions (“prettyprinting”), plotting of functions, and support of other user communication with the system. There are special user interfaces called *worksheets* for the X Window System, Macintosh, and MS-Windows. *Maplets*, which are launched from a **Maple** session, provide customized graphical user interfaces, e.g., for setting kernel options or for building plots in an interactive way.

In a **Maple** worksheet you can combine **Maple** input and output, graphics, and text in one document. Further features of the worksheet interface are that it provides hypertext facilities inside and between documents, that it allows embedding of multi-media objects (on some platforms), that it uses typeset mathematics, and that subexpressions of mathematical output can be selected for further processing. A **Maple** worksheet consists of a hierarchy

of regions. Regions can be grouped into sections and subsections. Sections and subsections can be “closed” to hide regions inside. A worksheet can be exported as RTF (Rich Text format), HTML (with MathML), XML, or L<sup>A</sup>T<sub>E</sub>X format. You are referred to the Maple documentation for precise details about the worksheet interface.

The Maple kernel interprets the user input and carries out the basic algebraic operations such as rational arithmetic and elementary polynomial arithmetic. It also contains certain algebraic routines that are so often used that they must be present in the lower-level systems language, for efficiency reasons. To the latter category belong routines for manipulation of polynomials like **degree**, **coeff**, and **expand**. The kernel also deals with storage management. A very important feature of Maple is that the system keeps only one copy of each expression or subexpression within an entire session. In this way testing for equality of expressions is an extremely inexpensive operation, viz., one machine instruction. Subexpressions are reused instead of being recomputed over and over again.

Most of the mathematical knowledge of Maple has been coded in the Maple programming language and resides as functions in the external library. When a user needs a library function, Maple is smart enough to load the routine itself. Maple must be informed about the use of separate packages like the linear algebra, number theory, and statistics packages, to name a few. All this makes Maple a compact, easy-to-use system. But what is more important, in this setting Maple uses memory space only for essential things and not for facilities in which a user is not interested. This is why many people can use Maple on one computer simultaneously, and why Maple runs on computers with rather limited memory. Table 1.1 summarizes the design of the Maple system as just described.

Part	Function
Iris	parser
	display of expressions (“prettyprinting”)
	graphics
	special user interfaces
Kernel	interpreter
	memory management
	basic & time-critical procedures for computations in $\mathbb{Z}$ , $\mathbb{Q}$ , $\mathbb{R}$ , $\mathbb{C}$ , $\mathbb{Z}_n$ , $\mathbb{Q}[x]$ , etc.
Library	library functions
	application packages
	on-line help
Private Libraries	contributions of Maple users

Table 1.1. Components of the Maple system.

The Maple language is a well-structured, comprehensible, high-level programming language. It supports a large collection of data structures: functions, sequences, sets, lists, arrays, tables, etc. There are also plenty of easy-to-use operations on these data structures like type-testing, selection and composition of data structures, and so on. These are the ingredients of the programming language in which almost all mathematical algorithms of Maple are implemented, and which is the same language users will employ in interactive calculator mode. Furthermore, anyone who is interested in the algorithms that Maple uses and who is interested in the way these algorithms are implemented can look at the Maple code in the library; the procedures are available in readable form or can be reproduced as such inside a Maple session. If desired, a user can enlarge the library with self-written programs and packages. The *Maple Application Center* (URL: [www.mapleapps.com](http://www.mapleapps.com)) contains many user contributions, sample worksheets, and documentation. Maple also provides its users with facilities to keep track of the execution of programs, whether self-made or not.

The last advantage of Maple we shall mention is its user-friendly design. Many computer algebra systems require a long apprenticeship or knowledge of a low-level systems language if one really wants to understand what is going on in computations. In many systems one has to leaf through the manual to find the right settings of programming flags and keywords. Nothing of the kind in Maple. First, there is the help facility, which is basically the on-line manual for Maple procedures. Secondly, the secret to why Maple is so easy to use is the hybrid algorithmic structure of the computer algebra system, where the system itself can decide which algorithm is favorable. As an example we look at some invocations of the Maple procedure **simplify**, which does what its name suggests.

```

> trig_formula := cos(x)^6 + sin(x)^6
>           + 3*sin(x)^2*cos(x)^2;
> exp_ln_formula := exp(a+1/2*ln(b));
> radical_formula := (x-2)^(3/2)/(x^2-4*x+4)^(1/4);
> trig_formula = simplify(trig_formula);


$$\cos(x)^6 + \sin(x)^6 + 3 \sin(x)^2 \cos(x)^2 = 1$$


> exp_ln_formula = simplify(exp_ln_formula);


$$e^{(a+1/2 \ln(b))} = e^a \sqrt{b}$$


> radical_formula = simplify(radical_formula);


$$\frac{(x-2)^{(3/2)}}{(x^2 - 4x + 4)^{(1/4)}} = \frac{(x-2)^{(3/2)}}{((x-2)^2)^{(1/4)}}$$


```

Here, Maple does not assume that  $x \geq 2$  and cannot simplify the square root. The generic simplification of the square root can be forced by adding the keyword **symbolic**.

```
> radical_formula = simplify(radical_formula, symbolic);

$$\frac{(x - 2)^{(3/2)}}{(x^2 - 4x + 4)^{(1/4)}} = x - 2$$

```

But the key point is that just one procedure, viz., **simplify**, carries out distinct types of simplifications: trigonometric simplification, simplifications of logarithms and exponential functions, and simplification of powers with rational exponents. On the other hand, the concept of pattern matching and transformation rules (rewrite rules) is underdeveloped in Maple. It is rather difficult to program mathematical transformations that can be applied globally.

At many places Maple makes decisions which way to go; we mention four examples. Computation of the determinant of a matrix is done by the method of minor expansion for a small matrix, otherwise Gaussian elimination is used. Maple has essentially six numerical methods at its disposal to compute definite integrals over a finite interval: In the default case (no particular method specified), the integration problem is first passed to NAG integration routines if **Digits** is not too large. If the NAG routines cannot perform the integration, then some singularity handling may be performed and control may pass back to the NAG routines with a modified problem. The default hybrid numeric-symbolic integration method is Clenshaw-Curtis quadrature, but when convergence is slow (due to nearby singularities) the system tries to remove the singularities or switches to an adaptive double-exponential quadrature method. An adaptive Newton-Cotes method is available when low precision (e.g., **Digits**  $\leq 15$ ) suffices. Other numerical integration methods are the adaptive Gaussian quadrature method and the sinc quadrature method. By the way, generalized series expansions and variable transformations are two of the techniques used in the Maple procedure **evalf/int** to deal with singularities in an analytic integrand. The interested reader is referred to [93, 97]. At present, there are six algorithms coded into the Maple procedure **fsolve**: Newton, Secant, Dichotomic, inverse parabolic interpolation, a method based on approximating the Jacobian (for systems), and a method of partial substitutions (for systems again). As a user of Maple you normally do not need to know about or act on these details; the system finds its own way. This approach turns Maple into an easy-to-learn and easy-to-use system for mathematical computations on computers.

# 2

## The First Steps: Calculus on Numbers

In this chapter we shall cover some of the basics needed to start using Maple, to access the on-line help, and to compute with numbers. A discussion at length of how to interact with Maple will follow in Chapter 4. During the overview of the number fields that are supported by Maple, we shall also introduce you gently to the way Maple stores data internally.

We assume that Maple 8 is used in combination with the worksheet interface on a PC. Henceforth, all computations will be done on a PC running Windows 2000, with a 1.7 Ghz Pentium 4 processor having 512 MB main memory. Most of this book applies equally well to other computers and/or operating systems, but there may be implementation differences in areas like starting, interrupting, and quitting Maple, reading and writing of files, and plotting. For system specific aspects you should read the documentation that came with your software. And of course, timings differ from one computer to another.

### 2.1 Getting Started

To start Maple with the worksheet interface on a PC, choose the Maple command from the appropriate menu or double-click the corresponding shortcut on the desktop. On a Unix platform, type **maple** or **xmaple** from a Unix shell. Look for local assistance and documentation if this does not work. When Maple has been launched successfully with the worksheet interface, the Maple window appears and the system indicates with a *prompt*,

such as the greater-than symbol, “>”, in a worksheet document that it is waiting for a command. A series of commands may already have been executed automatically from an initialization file, e.g., from the file `maple.ini` in the Maple 8/lib directory or in the current directory. Now, you can use Maple as an ordinary pocket calculator.

```
> 2+100/5^2*3;
14
> 5!/21;
40
7
```

All arithmetic operations that you are familiar with are present: addition (+), multiplication (\*), division (/), exponentiation (^ or \*\*), **factorial** (!), and double factorial (!!). The usual precedence rules apply, and in doubtful cases, or just for the sake of clarity, you may use parentheses. But there are a few differences in the input mode when using Maple as a pocket calculator:

- Each command must be terminated by a semicolon or a colon. Don’t forget these punctuation marks, otherwise the system will sit waiting for more input. Termination by a semicolon informs Maple that input is complete, causes the instruction to be carried out by the system, and displays the result. A colon can be used instead of a semicolon if the output on the screen can be suppressed. This is convenient when you are interested in computing an intermediate result, but do not need to look at it. The process of carrying out an instruction is called *evaluation*.
- Maple only deals with an input line when the *newline* character, which is either the carriage return or the *line feed* character, is sent by pressing the RETURN or ENTER key, respectively. After the evaluation of the command Maple prints a prompt at the beginning of a new line, again indicating that it is waiting for input.

The Maple input mode and particularly the roles of the semicolon and the colon are exemplified below.

```
> 2*5; 2^5: 100/4; 100
> /6
> ;
10
25
50
3
> 12345\
> 6789;
```

123456789

As you see, more than one instruction may be entered in one line. Maple deals with them one by one, as if they were given separately. To enhance readability you may stretch a command across several lines and you may insert spaces. But be careful, this is not allowed everywhere. For example, you cannot type the number 1000000 as 1 000 000; but you may group the digits into groups of three by typing 1\000\000. The *backslash* is used in Maple as the *continuation* character, which is ignored. When you stretch a command across two or more lines, Maple warns you that it got an incomplete statement or that a semicolon is missing. Maple splits the parts of the command by newline characters, unless you end a line with the backslash; in this case both backslash and newline character are ignored in the final result.

If Maple does not understand your input because of syntactical errors, it will tell you so.

```
> this is a line Maple does not understand
Error, missing operator or ';'
```

In the worksheet interface, the cursor will blink at the token that was being read when the error occurred. Maple does not say anything more. You must find out yourself what went wrong; Maple just waits for a correction of the input. In the worksheet interface, you can correct or extend input by placing the cursor on the input line and then making all necessary changes and/or additions. The command line interface of Maple, which is launched by entering **maple** from a Unix shell or selecting **Command Line Maple** from the Maple program menu on a PC, has both an emacs-like command line editor available that allows you to edit previously entered input lines.

You quit Maple by successively clicking on the **File** and **Exit** menu items; the system will ask you to confirm this action. Alternatively, you press the corresponding acceleration keys. In the command line interface you leave Maple by entering the command **quit**, **stop**, or **done**, followed by the newline character.

You can interrupt a lengthy Maple calculation that is in progress by clicking on the **Stop** button in the tool bar of the Maple window so that the system receives the *interrupt* character (usually CONTROL-C). It is not always possible to stop computation instantly, but after the interrupt is recognized you may enter another command.

Inside Maple you can always start afresh by entering **restart**. The worksheet interface has a **restart** button in the toolbar for this purpose, too. This instruction will clear Maple's internal memory, clear the values of all variables, reset kernel options and interface settings, and reread the initialization file(s) so that it acts as if you had started Maple from scratch.

## 2.2 Getting Help

When you click on the **Help** button of a Maple worksheet, you get a menu of available help facilities. A good starting point is the item ‘Introduction’. You select it by clicking on **[Introduction]**. You get the help window that consists of two parts. At the top is the ‘Help Browser’ that offers an easy way of searching for information. At the bottom of the help window is a worksheet containing the information about some chosen topic. As an example, select in the Help Browser the item **Graphics...**, the subitem **2D...**, and finally click on the subitem **polar** to follow the hyperlink to the actual help page. You get the worksheet with the actual information on how to plot a parametric curve in polar coordinates at the lower part of the help window. Figure 2.1 is a screen dump of this search.

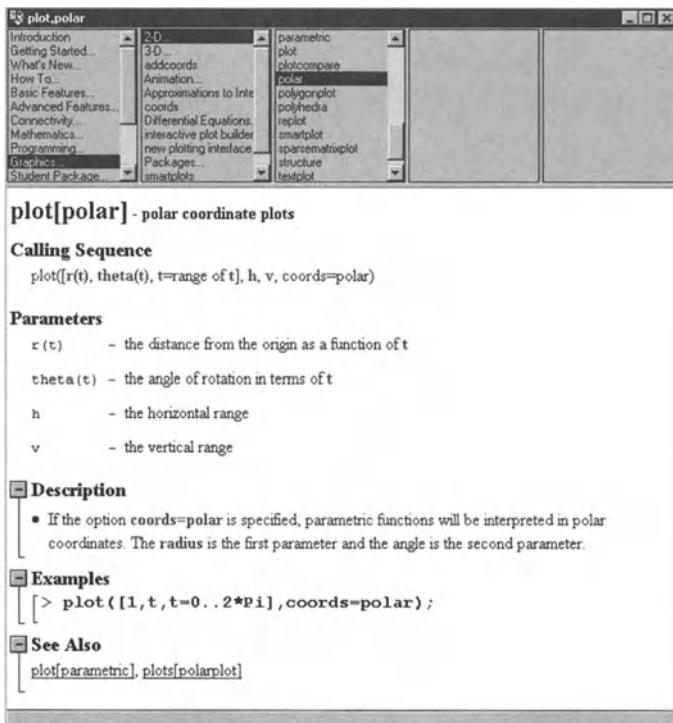


Figure 2.1. The Help Browser and help about graphs in polar coordinates.

The same help window appears when you enter

```
> ?plot,polar
```

The advantage of the Help Browser is that it allows you to explore or find your way through Maple without browsing through manuals. The **?topic**

command-line syntax is a faster way of getting information when you know what command you are looking for and only want to refresh your knowledge or look at an example. For example, if you want to see the definition of the dilogarithm and some examples of its use in Maple, just enter

```
> ?dilog
```

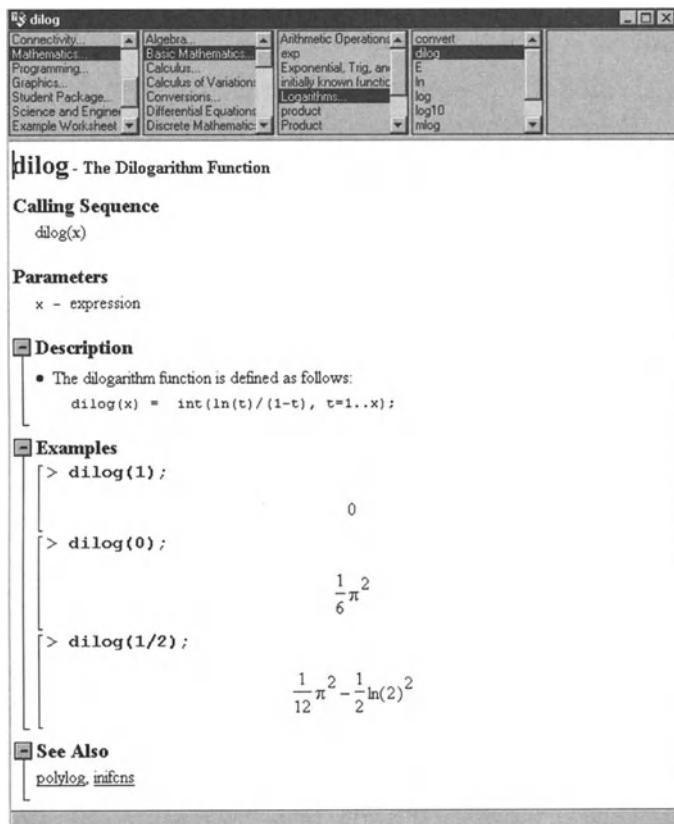


Figure 2.2. The Maple help window for **dilog**.

As another concrete example of available on-line help, let us ask for help about integer factorization via the long alternative for **?ifactor**.

```
> help(ifactor);
```

The corresponding help message is a clear illustration of the general format of Maple's descriptions of syntax, data types, and functions. Below, we have split the contents of the help window accordingly and at the same time demonstrate how to extract these parts from the help page in a Maple session (running the command line interface).

```
> info(ifactor);
```

```

FUNCTION: ifactor - integer factorization

CALLING SEQUENCE:
    ifactor(n)
    ifactor(n, method)

PARAMETERS:
    n      - integer or a rational
    method - (optional) name of base method for factoring

```

The procedure **ifactor** is explained first: what it is meant for and how you can use it. Obviously, you can factorize an integer with it, but the help text informs you that **ifactor** can also be used to factorize a rational number. If you desire a particular factorization method, then you can specify this with an extra argument in the procedure call.

Hereafter, the procedure and its options are described in more detail in the help file. There is also a built-in procedure, called **usage**, to extract the SYNOPSIS part.

```
> usage(ifactor);
```

**SYNOPSIS:**  
- ifactor returns the complete integer factorization of n.

- The answer is in the form:  
 $u * ((f_1)^{e_1} * \dots * (f_n)^{e_n})$  such that  
 $n = u * f_1^{e_1} * \dots * f_n^{e_n}$  where u equals  
 $\text{sign}(n)$ ,  $f_1, \dots, f_n$  are the distinct prime factors of n,  
and  $e_1, \dots, e_n$  are their multiplicities (negative in  
the case of the denominator of a rational).
- The expand function may be applied to cause the factors  
to be multiplied together again.
- If a second parameter is specified, the named method  
will be used when the front-end code fails to achieve  
the factorization. By default, the Morrison-Brillhart  
algorithm is used as the base method. Currently  
accepted names are:
  - 'squof' - D. Shanks' undocumented square-free  
factorization;
  - 'pollard' - J.M. Pollard's rho method;
  - 'lenstra' - Lenstra's elliptic curve method; and
  - 'easy' - which does no further work.
- If the 'easy' option is chosen, the result of the  
ifactor call will be a product of the factors that  
were easy to compute, and a name  $_c.m$  indicating  
an m-digit composite number that was not factored.
- The pollard base method accepts an additional optional  
integer: ifactor(n,pollard,k), which increases the  
efficiency of the method when one of the factors is of  
the form  $k*m+1$ .

What follows in a help page are a few examples. In many cases you can immediately learn from these examples how to do your work in Maple.

```
> example(ifactor);

EXAMPLES:
> ifactor( 61 );
                                         (61)
> ifactor( 60 );
                                         2
                                         (2)   (3) (5)
> ifactor( -144 );
                                         4      2
                                         - (2)   (3)
> expand(%);
                                         -144
> ifactor( 60, easy );
                                         2
                                         (2)   (3) (5)
> ifactor( 4/11 );
                                         2
                                         (2)
                                         -----
                                         (11)
> n := 8012940887713968000041:
> ifactor( n, easy );
                                         (13) (457) _c19_1
> ifactor( n );
                                         (13) (457) (473638939) (2847639359)
```

Finally, Maple refers to related matter.

```
> related(ifactor);

SEE ALSO: ifactors, isprime, factor, type[facint]
```

By the way, instead of **usage(topic)** and **example(topic)** you can use the shortcuts **??topic** and **???topic**, respectively.

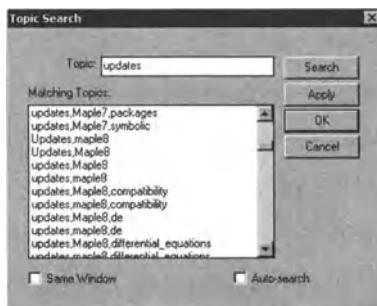


Figure 2.3. Searching by topic.

You can search by topic in the help system. Figure 2.3 shows the dialog box that appears when you choose **Topic Search** from the **Help** menu.

We use the word **updates** to find help topics whose names start with this string.

Searching for help topics whose contents contain specific words is possible as well. Figure 2.4 shows the dialog box that appears when you choose **Full-Text Search** from the **[Help]** menu. We use the word **fit** to find help topics whose contents contain this string.



Figure 2.4. Full-text searching.

So you see that you can get plenty of help on standard library functions directly from the system. Table 2.1 summarizes how to access the on-line Maple help system.

Type of Help Request	Maple Command
explain on-line help system	?help
explain a specific topic	?topic
explain a specific topic in a context	?context, topic ?context [topic] ?context:-topic
list all help categories	?index
list operators for forming expressions	?index,expression
list all library functions	?index,function
list miscellaneous facilities	?index,misc
list topics related to modules	?index,module
list all packages	?index,packages
list information about procedures and programming	?index,procedure
list all Maple statements	?type,statement
list basic Maple data types	?type,surface
summarize new features	?updates

Table 2.1. The on-line Maple help system.

When you do not use the Help menu, you can get a list of help categories when you enter **?index**. The categories *function*, *misc*, and *packages*

are listed among others. When you enter **?index,function** you get a list of all standard library functions present in Maple. After the command **?index,misc** you will see more rarely used facilities. Entering the command **?index,packages** will provide you with a list of all packages present in Maple. If you want to know what procedures are available in the linear algebra package, for example, just enter **?LinearAlgebra**. Or if you want to know more about the norm function in the linear algebra package, enter **?LinearAlgebra,Norm**, **?LinearAlgebra[Norm]**, or **?LinearAlgebra:-Norm**. If you omit the first part and simply enter **?Norm**, then you will get information about the function for computing the norm of an algebraic number; this procedure resides in the standard library. In cases of no ambiguity, e.g., **TridiagonalForm** for reducing a square matrix to tridiagonal form, you may use the abbreviation **?TridiagonalForm** instead of **?LinearAlgebra,TridiagonalForm**. Once a package is loaded, you can use abbreviations when you ask for help about a procedure in that package.

Good advice:

*Familiarize yourself with the help system because it will be your best assistant in learning and using Maple.*

Now we are ready for our tour through Maple, which will be in the form of short Maple sessions interleaved with explanatory remarks. Unless explicitly stated, we assume that in each example all history of previous examples has been forgotten by Maple, as if a new session has been started. Have a close look at the examples, try them yourself with Maple, and change the examples as you like. Only through direct confrontation with the computer algebra system you can acquire the right skills and experience the applicability and limitations of computer algebra personally. For the same reason we have added a set of exercises at the end of each session. Work through these exercises to become more and more proficient at Maple.

## 2.3 Integers and Rational Numbers

Being a true computer algebra system, Maple is averse to approximations in arithmetic computations. In contrast to pocket calculators, Maple never changes spontaneously arithmetic expressions into decimal numbers.

The quotient of two integers with nonzero denominator is only simplified. More precisely, the greatest common divisor is removed automatically from the numerator and the denominator of a rational number. For exact arithmetic on rational numbers, Maple must be able to deal with very large integers.

```
> 5/81491528324789773434561 - 101/10198346916138403856439;
```

```


$$\frac{-908850291802563899734274}{92342097398058352671328089792352035178332031}$$

> number := 4^(4^4);

number := 13407807929942597099574024998205846127479365820\
59239337772356144372176403007354697680187429816690\
34276900318581864860508537538828119465699464336490\
06084096

> length(number); # number of digits
155

```

Note that Maple spontaneously uses a backslash to indicate that output continues on the next line.

Of course, there is a maximum integer that can be represented by Maple, but it has a much higher value than in most other programming languages; the number of digits is limited to  $2^{28} - 8 = 268435448$  on a 32-bit machine.

```

> kernelopts(maxdigits); # maximum number of digits
268435448

```

See [107] for how to overcome the size limitation. The secret of Maple's capability of computing with large numbers up to  $10^{268435447}$  lies in the internal representation of large integers. Most programming languages make use of the hardware facilities for calculus on so-called *single-precision integers*. This limits the range of values for integers to those numbers that can be represented in one computer word. On a 32-bit machine, Maple uses single precision integers for numbers less than  $2^{30}$ . Maple has implemented *multiple-precision integer arithmetic* by using several consecutive computer words for the internal representation of a large integer [97, 181]. We call such a linear list a *dynamic data vector* and define the *length* of the data vector as the number of computer words used. The internal data structure in Maple representing a large positive integer has the format shown in Figure 2.5 below.

intpos	integer $i_0$	integer $i_1$	.....	integer $i_n$
--------	---------------	---------------	-------	---------------

Figure 2.5. Internal representation of a positive integer.

The first word in this vector encodes all the information about the data structure: it indicates that the vector represents a positive integer and that the vector length is equal to  $n + 2$ . The next  $n + 1$  words contain single-precision nonnegative integers  $i_0, i_1, i_2, \dots, i_n$ . Let  $B$  be the base of the number system used inside the computer, then the above vector represents

the integer

$$i_0 + i_1 B + i_2 B^2 + i_3 B^3 + \cdots + i_n B^n.$$

Maple uses as base  $B$  the largest power of 10 such that  $B^2$  can be represented in the single-precision integer arithmetic of the computer ( $B = 10^4$  on a 32-bit machine). Because the length of the vector may be chosen dynamically, instead of being of a pre-defined fixed length, very large integers can be represented in Maple. The only restriction is that the number of computer words necessary for representation of an integer must be specifiable in the first word of the dynamic data vector. Maple uses twenty-six bits for specification of the vector length, and this explains the “magic number”  $2^{28} - 8 = 4((2^{26} - 1) - 1)$ . Maple is clever enough to decide beforehand whether an integer can be represented or not.

```
> 123456789 ^ 987654321;
Error, numeric exception: overflow
```

For reasons of efficiency, small integers (with magnitude less than  $2^{30}$  on a 32-bit machine) are not stored as a dynamic data vector, but instead in a single computer word.

In Maple there are several procedures for doing calculus on integers. A few examples:

```
> number := 10^39 - 10^19 - 1;
number := number := 9999999999999999999989999999999999999999999
> isprime(number); # check whether the number is prime
false
```

The quality of Maple’s primality test is described in [243].

```
> settim := time(): # start timing
> ifactor(number); # factorize the integer
(59) (2122955154480399619) (325018590983) ((24563993)
> cpu_time := (time()-settim)*seconds; # computing time
cpu_time := 8.422 seconds
> nextprime(number); # determine the next largest prime
999999999999999999999900000000000000000000047
> # integer approximation to the square root
> isqrt(number);
31622776601683793320
```

Integer factorization is a time-consuming operation, and you may want to know the computing time. As you have seen before, this can be done with the help of the Maple procedure **time**, which gives back the computing time (in seconds) since the start of the Maple session. Enter the command

**time()** immediately before and after a computation. By taking the differences of these two numbers, you get the computing time of that particular computation.

Alternatively, you can use the command **time(<expression>)** to compute the time to evaluate the given *expression*. For example,

```
> time( assign(factored_number, ifactor(3!!!)) );
```

.120

computes the prime factorization of  $3!!! = 6!! = 720!$  and assigns the result to the variable **factored\_number** for later referencing. You cannot use the assignment operator  $:=$  here because the argument of **time** must be a Maple expression and may not be, for example, an assignment statement.

In all the above procedures, integer division (with remainder) and determination of greatest common divisor play a crucial role.

```
> a := 1234; b := 56;
> q := iquo(a, b); # quotient of integer division
q := 22
> r := irem(a, b); # remainder of integer division
r := 2
> testeq(a=q*b+r); # check identity
true
> igcd(a, b); # greatest common divisor of integers
2
```

By use of the procedure **igcdex** Maple can compute the extended greatest common divisor of integers; for any two integers  $a$  and  $b$ , two integers, say  $s$  and  $t$ , are computed such that  $as + bt = \gcd(a, b)$ .

```
> igcdex(a, b, 's', 't');
2
> 'a'=a, 'b'=b, 's'=s, 't'=t, 'a*s+b*t' = a*s+b*t;
a = 1234, b = 56, s = 1, t = -22, a*s + b*t = 2
```

The quotes around  $a$ ,  $b$ ,  $s$ ,  $t$ , and  $as + bt$  in the above examples are used to suppress evaluation of variables and expressions. This will be explained in more detail in the next chapter.

Modular arithmetic plays an important role in integer factorization and primality tests [148]. The operator **mod** gives, in modular arithmetic with respect to a nonzero integer  $n$ , an answer in the sequence  $0, 1, 2, \dots, |n| - 1$ . If you prefer an answer in the sequence

$$-\left\lfloor \frac{|n|-1}{2} \right\rfloor, \dots, -1, 0, 1, \dots, \left\lfloor \frac{|n|}{2} - 1 \right\rfloor, \left\lfloor \frac{|n|}{2} \right\rfloor,$$

i.e., symmetrically around zero. then you can specify this beforehand.

```
> 1/2345 mod 6;
      5
> 'mod' := mods: 1/2345 mod 6;
      -1
```

A drawback of Maple's modular arithmetic is that you cannot specify in one step that all computations are to be done modulo some fixed integer. In each separate command you need to call the **mod** routine.

As we have noted before, Maple simplifies a rational number spontaneously into a unique standard form. The system automatically removes the greatest common divisor from the numerator and the denominator of the rational number, and it guarantees that the denominator is a positive number. If you represent a rational number as a pair of integers (*numerator*, *denominator*) with positive denominator, this standard form, referred to as *canonical form*, is the most compact one. You would not like to represent  $\frac{1}{3}$  as

$$\frac{-41152263041152263041152263041152263041152263}{-123456789123456789123456789123456789123456789}.$$

Now, you might think that Maple represents a rational number internally as a data vector consisting of three components, the first component specifying the nature of the data vector, the second component representing the numerator, and the third component representing the positive denominator. However, because the developers of Maple wanted to make the system as memory-efficient as possible, they have chosen a different design, based on the following rule [48]:

*Every (sub)expression in Maple appears internally only once.*

Therefore, in a data vector representing a rational number, the last two components do not consist of the multiple-precision integers representing the numerator and denominator themselves, but only contain reference pointers to these integers. In this way, any integer that occurs in several rational numbers, appears internally only once. For example, the rational numbers  $-\frac{1}{2}$ ,  $\frac{2}{3}$ , and  $\frac{3}{5}$ , when used in the same Maple session, are internally represented as in Figure 2.6 below.

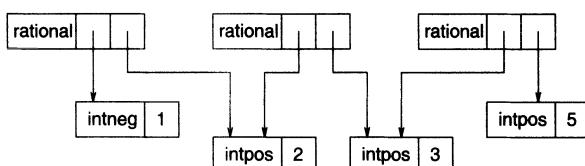


Figure 2.6. Internal representation of the fractions  $-\frac{1}{2}$ ,  $\frac{2}{3}$ , and  $\frac{3}{5}$ .

## 2.4 Irrational Numbers and Floating-Point Numbers

In the previous section you have seen that Maple simplifies rational numbers automatically. In general, Maple only carries out computations when it is specifically ordered to do so.

```
> 25^(1/6);
25^(1/6)
> simplify(%);
5^(1/3)
> evalf(%);
1.709975947
> convert(%%, float);
1.709975947
```

Here, the procedures **simplify** and **evalf** put Maple to work. With the percentage symbol in the command **simplify(%)** you refer to the previously evaluated expression, in this case  $25^{1/6}$ . Two percentage symbols return the second last expression evaluated; more than three percentage symbols for referring to previously evaluated expressions are not allowed. Percentage symbols that are used in the above sense are called *ditto operators*.

You might have expected in the above example that Maple would immediately yield an approximate result for the cube root, but this is against the principle of exact arithmetic. You may want to compute the third power of this cube root and expect the exact result. In approximate floating-point arithmetic you would get a different result.

```
> 25.0^(1/6);
1.709975947
> %^6;
25.00000003
```

Any number containing a dot is interpreted as a floating-point number, and henceforth Maple computes with it as such. In this case, Maple will also take care of automatic type conversions such as from integers to floating-point numbers.

```
> 90005*0.15;
13500.75
```

Furthermore, floating-point numbers are contagious in Maple, so **exp(1.)** automatically evaluates to **2.71828....**

Other Maple notations for a floating-point number, say 0.000001, are:  $0.1 * 10^{-5}$ , 1E-6, and `Float(10, -7)`. The latter notation is of the form `Float(mantissa, exponent)` and resembles Maple's internal representation of floating-point number: a data vector consisting of the header `FLOAT`, and pointers to the multiple-precision integers *mantissa*, and *exponent*. This vector represents the number *mantissa*  $\times 10^{\text{exponent}}$ . In other words, the significant digits of a floating-point number are stored in the *mantissa* and the overall magnitude is given by the *exponent*. As a consequence of the internal representation of floating-point numbers, the limit of the floating-point precision coincides with the maximum number of digits that an integer can have in Maple. However, the internal addition and multiplication routines that compute the integer values of the exponent are written in C. This means that exponent arithmetic is restricted to C's single-precision integer arithmetic.

The precision of floating-point arithmetic can be defined by setting different values to the Maple variable `Digits`, whose default value is equal to ten. There are several functions that make Maple compute in floating-point arithmetic, the most important being `evalf` (evaluate using floating-point arithmetic).

```
> evalf(sqrt(2));
1.414213562
> Digits;
10
> Digits := 20: evalf(sqrt(2));
1.4142135623730950488
```

You can evaluate the expression at a value of a fixed number of digits, say 6 digits.

```
> convert(%, float, 6);
1.41421
```

You can control the number of displayed decimal places of a floating-point number with the `interface` procedure and the interface variable `displayprecision`. Note that this does not change the precision in which Maple does numerical computations; it only changes the display of the floating-point numbers.

```
> interface(displayprecision=6):
> evalf(sqrt(2));
1.414214
> interface(displayprecision=-1): # reset to full precision
> evalf[150](Pi);
3.141592653589793238462643383279502884197169399375105820\
```

```
97494459230781640628620899862803482534211706798214\
808651328230664709384460955058223172535940813
```

The procedure **evalf** approximates its argument using the number of digits specified between square brackets. If there is no specification of the number of digits in the call of the **evalf** procedure, then Maple takes the value of **Digits** as the number of digits to be used in floating-point arithmetic. For a detailed description of Maple's software floating-point model and its consequences we refer to [181].

Maple knows, of course, some mathematical constants, like Euler-Mascheroni's constant  $\gamma$  and the number  $\pi$ . See Table 2.2 for a list of constants known to Maple.

Mathematical Constant	Maple Name	Value (approx.)
$\pi$ , the area of a unit circle	Pi	3.141592654
Catalan's number C $= \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)^2}$	Catalan	0.9159655942
Euler-Mascheroni's constant $\gamma$ $= \lim_{n \rightarrow \infty} \left( \left( \sum_{k=1}^n \frac{1}{k} \right) - \ln n \right)$	gamma	0.5772156649
Boolean values <i>true</i> , <i>untrue</i> , <i>fail</i> $\infty$	true, false, FAIL infinity	

Table 2.2. Mathematical constants in Maple.

Names of mathematical constants are protected in Maple so that you cannot assign values by accident.

```
> Pi := 3.14;
```

```
Error, attempting to assign to 'Pi' which is protected
```

You may add your own symbolic constants with name protection.

```
> with(ScientificConstants):
> electron_rest_mass := (GetValue*GetUnit)(Constant(m[e]));
electron_rest_mass := 0.910938188 10-30 [kg]
> protect('electron_rest_mass'):
> electron_rest_mass :=
> convert(electron_rest_mass, units, amu);
```

```
Error, attempting to assign to 'electron_rest_mass'
which is protected
```

You first have to **unprotect** the name before you can assign a new variable.

```

> unprotect('electron_rest_mass'):
> electron_rest_mass := 
>     convert(electron_rest_mass, units, amu);

$$\text{electron\_rest\_mass} := 0.0005485794250 \text{ [u]}$$

> protect('electron_rest_mass'):

```

Alternatively, you can associate the value  $9.109558 \times 10^{-31} \text{ kg}$  with the protected name `electron_rest_mass` by use of the procedure **macro**. In this way, `electron_rest_mass` is just an abbreviation of a specific value, which cannot be assigned a value by accident.

```

> macro(electron_rest_mass=9.109558*10^(-31)*Unit(kg)):
> electron_rest_mass;

$$0.9109558000 \text{ } 10^{-30} \text{ [kg]}$$


```

The latter mechanism may be used to work conveniently with the base  $e$  of the natural logarithm. In Maple this number is denoted by `exp(1)`. But it is cumbersome to type `exp(1)` each time you want to use the base  $e$  of the natural logarithm; you do not only want to see it displayed in the worksheet interface as the character  $e$ , but also want to use this short notation on the input side. To achieve this you can do the following.

```

> protect('e'):
> macro(e=exp(1)):
> ln(e);

```

1

With a scientific pocket calculator you can compute with mathematical functions like the exponential function, the natural logarithm, and trigonometric functions. In Maple, these functions are also present, but the system contains many more. In Table 2.3 we list commonly used mathematical functions and their names in Maple. A more complete list can be obtained by the command `?inifcns` (help about **initially known functions**).

Recall the definitions of

the Gamma function       $\Gamma(z) = \int_0^\infty t^{z-1} e^{-t} dt, \quad \Re(z) > 0,$

the Riemann zeta function       $\zeta(z) = \sum_{n=1}^{\infty} \frac{1}{n^z},$

the dilogarithm function       $\text{dilog}(x) = \int_1^x \frac{\ln t}{(1-t)} dt,$

and the error function       $\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt.$

Maple knows exact values of many of these functions. Trigonometric functions applied to multiples of  $\frac{\pi}{6}$  yield exact numerical results and rational

Mathematical Function	Maple Name
exponential function	exp
natural logarithm	ln, log
logarithm with base 10	log10
square root function	sqrt
absolute value	abs
trigonometric functions	sin, cos, tan, csc, sec, cot
inverse trigonometric functions	arcsin, arccos, arctan, arccsc, arcsec, arccot
hyperbolic functions	sinh, cosh, tanh, csch, sech, coth
inverse hyperbolic functions	arcsinh, arccosh, arctanh, arccsch, arcsech, arccoth
hypergeometric function	hypergeom
Bessel functions	BesselI, BesselJ, BesselK, BesselY
Gamma function	GAMMA
binomial coefficient	binomial
polygamma function	Psi
Riemann zeta function	Zeta
dilogarithm	dilog
error function	erf

Table 2.3. Commonly used mathematical functions known to Maple.

multiples of  $\pi$  with a denominator dividing 120 or 24 can be converted into exact numerical results with nested radicals. The Riemann zeta function yields exact numbers when applied to even, natural numbers less than fifty; for larger arguments, you must explicitly **expand** it.  $\lim_{x \rightarrow \infty} \text{erf}(x)$  is known to be equal to 1. And so on.

```
> sin(Pi/6), Zeta(2), limit(erf(x), x=infinity);

$$\frac{1}{2}, \frac{\pi^2}{6}, 1$$

> sin(Pi/10) = convert(sin(Pi/10), radical);

$$\sin\left(\frac{\pi}{10}\right) = -\frac{1}{4} + \frac{\sqrt{5}}{4}$$

> Zeta(50) = expand(Zeta(50));

$$\zeta(50) = \frac{39604576419286371856998202 \pi^{50}}{285258771457546764463363635252374414183254365234375}$$

```

Numerical approximations can be obtained with **evalf**.

```
> Zeta(3): % = evalf(%);

$$\zeta(3) = 1.202056903$$

```

The next examples give more food for thought about exact arithmetic versus floating-point arithmetic.

```
> sin(4)-2*sin(2)*cos(2); combine(%); evalf(%);
sin(4) - 2 sin(2) cos(2)
0
-0.1 10-9
> (1+sqrt(2))^2-2*(1+sqrt(2))-1; simplify(%); evalf(%);
(√2 + 1)2 - 3 - 2 √2
0
-.1 10-8
```

You may wonder how the Maple procedure **evalf** distinguishes all these mathematical functions. Maple shows its secrets when you set a higher value to the variable **printlevel**, whose default value is equal to 1.

```
> printlevel := 5: evalf(sin(1)+ln(2)); printlevel := 1:
{--> enter sin, args = 1
<-- exit sin (now at top level) = sin(1)}
{--> enter ln, args = 2
<-- exit ln (now at top level) = ln(2)}
{--> enter evalf/sin, args = 1
x := 1.

<-- exit evalf/sin (now at top level) = .8414709848}
{--> enter evalf/ln, args = 2
x := 2.

<-- exit evalf/ln (now at top level) = .6931471806}
1.534618165
```

This example indicates that there exists in the Maple library a procedure called **evalf/func** for each mathematical function *func* for which numerical values can be computed. Whenever you apply the procedure **evalf** on a Maple expression, the system finds out what functions are present and automatically applies the appropriate **evalf/func** procedures. In this way, it is only necessary for you to recall the name of *one* procedure for numerical approximation, instead of remembering several distinct procedures. We shall see that similar techniques are used in other areas, such as, integration, differentiation, and simplification.

Maple also has the procedure **evalhf** (evaluate using hardware floating-point arithmetic) for the purpose of gaining speed in numerical computations (e.g., in plotting graphs of functions) or for users who want hardware floating-point arithmetic. This procedure converts all its arguments to hardware floating-point numbers, computes the answer in double precision

(which is equivalent to a setting of `Digits` around fifteen), and converts it to a Maple floating-point result. We illustrate the use of `evalhf` with the calculation of the values of the function  $g$  defined by

```
> f := x -> arctan((2*x^2-1)/(2*x^2+1)):
> g := (x,y) -> f(x)*f(y):
> g(x,y);
```

$$\arctan\left(\frac{2x^2-1}{2x^2+1}\right) \arctan\left(\frac{2y^2-1}{2y^2+1}\right)$$

on a two-dimensional rectangular 50 by 50 grid at equally spaced points in the ranges  $[-3, 3]$  and  $[-3, 3]$ , respectively.

```
> settim := time(): # start timing
> for i to 100 do
>   for j to 100 do
>     evalf(g(-3+6*i/50, -3+6*j/50))
>   end do
> end do:
> cpu_time := (time()-settim)*seconds; # computing time
cpu_time := 1.913000 seconds

> settim := time():
> for i to 100 do
>   for j to 100 do
>     evalhf(g(-3+6*i/50, -3+6*j/50))
>   end do
> end do:
> cpu_time := (time()-settim) * second; # computing time
cpu_time := 0.130 second

> evalf(g(1,1)), evalhf(g(1,1));
.1035234193, .103523419254546598
```

Hardware floating-point arithmetic is used by Maple in plot routines, e.g., in `plot3d`, with which you can graph the function  $g$ .

```
> plot3d(g, -3..3, -3..3,
>         grid=[100,100], style=patchnogrid);
```

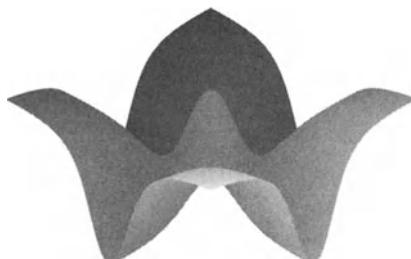


Figure 2.7. Graph of  $(x, y) \mapsto \arctan\left(\frac{2x^2-1}{2x^2+1}\right) \cdot \arctan\left(\frac{2y^2-1}{2y^2+1}\right)$ .

We end this section with the remark that Maple 8 implements a natural extension to arbitrary precision and exact arithmetic of the IEEE-754 and IEEE-854 standards. This means, among other things, that a floating-point zero with preserved sign exists. This enables more careful evaluation of mathematical functions that have branch cuts in the complex domain, as well as infinities and undefined answers. Enter `?numeric_overview` to learn more about the various components of the numerics of Maple.

## 2.5 Algebraic Numbers

We have already seen examples of radical numbers such as square roots and cube roots of integers. Maple has little problems in computing with them.

```
> (1/2+1/2*sqrt(5))^2;

$$\left(\frac{1}{2} + \frac{\sqrt{5}}{2}\right)^2$$

> expand(%);

$$\frac{3}{2} + \frac{\sqrt{5}}{2}$$

> 1/%;

$$\frac{1}{\frac{3}{2} + \frac{\sqrt{5}}{2}}$$

> simplify(%);

$$\frac{2}{3 + \sqrt{5}}$$

> rationalize(%);

$$\frac{3}{2} - \frac{\sqrt{5}}{2}$$

> (-1-3*Pi-3*Pi^2-Pi^3)^(1/3);

$$(-1 - 3\pi - 3\pi^2 - \pi^3)^{(1/3)}$$

> simplify(%);

$$\frac{(\pi + 1)(1 + \sqrt{3}I)}{2}$$

```

In the last example, Maple uses the principal branch of the complex, cube root function. The real-valued root can be found via the non-principal root function `surd`.

```
> convert(%%, surd);
```

```


$$-(1 + 3\pi + 3\pi^2 + \pi^3)^{(1/3)}$$

> simplify(%);

$$\frac{-\pi - 1}{\sqrt{3 + 1}}$$


```

Nested square roots of the form  $\sqrt{r_1 + r_2\sqrt{n}}$ , where  $n \in \mathbb{N}$  and  $r_1, r_2 \in \mathbb{Q}$ , are denested where possible by the **sqrt** and **simplify** commands.

```

> sqrt(4+2*sqrt(3));

$$\sqrt{3 + 1}$$

> (4+2*3^(1/2))^(1/2);

$$\sqrt{4 + 2\sqrt{3}}$$

> simplify(%);

$$\sqrt{3 + 1}$$


```

You can simplify more complicated nested radicals by **radnormal**.

```

> sqrt(25+5*sqrt(5)) - sqrt(5+sqrt(5)) - 2*sqrt(5-sqrt(5));

$$\sqrt{25 + 5\sqrt{5}} - \sqrt{5 + \sqrt{5}} - 2\sqrt{5 - \sqrt{5}}$$

> radnormal(%);

$$0$$


```

Radical numbers are instances of so-called algebraic numbers. In general, an *algebraic number* is a root of a univariate polynomial over the rational numbers:  $\sqrt{2}$  is a root  $\alpha$  of the polynomial  $x^2 - 2$ ,  $\sqrt{2} + \sqrt{3} + \sqrt{5}$  is a root  $\alpha$  of the polynomial  $x^8 - 40x^6 + 352x^4 - 960x^2 + 576$ , the nested radical  $\sqrt{1 + \sqrt{2}}$  is a root  $\alpha$  of the polynomial  $x^4 - 2x - 1$ . A root  $\alpha$  of the polynomial  $x^5 + x + 1$  cannot be expressed in terms of radicals (see [211] for a characterization of solvable quintics). Note that in these examples the symbol  $\alpha$  can be *any* of the roots of the polynomial, just as the square root of two can be approximately 1.4142 or -1.4142.

Computing with algebraic numbers is complicated and time-consuming [210] and this is no exception in Maple. An algebraic number is represented in the system by means of the procedure **RootOf**, which plays the role of a placeholder. For example,

```

> alpha := RootOf(z^2-2, z);

$$\alpha := \text{RootOf}(-Z^2 - 2)$$


```

represents any root of 2; in radical notation  $\sqrt{2}$  or  $-\sqrt{2}$ . Here, Maple reveals that it will use the underscore name `_Z` internally. The procedure **simplify** makes use of the fact that  $\alpha^2 = 2$  to simplify expressions containing  $\alpha$ .

```
> simplify(alpha^2);
```

```
> simplify(1/(1+alpha));
RootOf(_Z^2 - 2) - 1
```

These calculations become much clearer when an **alias** is used for the square root.

```
> alias( beta=RootOf(z^2-2, z) );
> 1/(1+beta) + 1/(beta-1); simplify(%);

$$\frac{1}{1+\beta} + \frac{1}{\beta-1}$$


$$2\beta$$

```

Actually, in the above example,  $\alpha$  and  $\beta$  can be any of the square roots of two, and the procedure **allvalues** does its very best to show them all.

```
> allvalues(beta);

$$\sqrt{2}, -\sqrt{2}$$

```

By default, **allvalues** treats all occurrences of the same **RootOf** expression as if they represent the same root. But you can request independent evaluation of **RootOfs** in an expression.

```
> beta + 1/beta;

$$\beta + \frac{1}{\beta}$$

> allvalues(%);

$$\frac{3\sqrt{2}}{2}, -\frac{3\sqrt{2}}{2}$$

> allvalues(%%, 'independent');

$$\frac{3\sqrt{2}}{2}, -\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}, -\frac{3\sqrt{2}}{2}$$

```

You can easily convert the representation of an algebraic number from **radical** into **RootOf**, and vice versa where possible.

```
> convert((-8)^(1/3), RootOf);

$$1 + \text{RootOf}(_Z^2 + 3, \text{index} = 1)$$

```

Here, an indexed **RootOf**-expression appears: the index  $i$  represents the  $i^{\text{th}}$  complex root of the polynomial equation, with respect to some ordering of roots. For details, enter **?RootOf, indexed**.

```
> convert(% , radical);

$$1 + \sqrt{-3}$$

```

In Maple you can do polynomial calculus over algebraic number fields. Below, we shall use this facility to check that  $\zeta = \sqrt{2} + \sqrt{3} + \sqrt{5}$  is a root of the polynomial  $x^8 - 40x^6 + 352x^4 - 960x^2 + 576$  and that in this case

$\sqrt{2} = \frac{1}{576}\zeta^7 - \frac{7}{144}\zeta^5 - \frac{7}{72}\zeta^3 + \frac{5}{3}\zeta$ . First, we use resultants to compute the defining polynomial for  $\zeta$  (q.v., [168]).

```
> polynomial := resultant(
>   resultant( x^2-5, (x-y)^2-3, x ), (y-z)^2-2, y );
polynomial := -960 z^2 + 352 z^4 - 40 z^6 + z^8 + 576
> expand( subs(z=sqrt(2)+sqrt(3)+sqrt(5), polynomial) );
0
```

So,  $\sqrt{2}+\sqrt{3}+\sqrt{5}$  is indeed a root of the polynomial. Introduce the algebraic number  $\zeta$  and factorize  $x^2 - 2$  over  $\mathbb{Q}(\zeta)$ .

```
> alias( zeta=RootOf(polynomial, z) ):
> factor(x^2-2, zeta);
```

$$\frac{(576x + 960\zeta - 56\zeta^3 - 28\zeta^5 + \zeta^7)(576x - 960\zeta + 56\zeta^3 + 28\zeta^5 - \zeta^7)}{331776}$$

You can also find the two roots of  $x^2 - 2$  over  $\mathbb{Q}(\zeta)$  by **roots**.

```
> roots(x^2-2, zeta);
[[-\frac{1}{576}\zeta^7 - \frac{5}{3}\zeta + \frac{7}{72}\zeta^3 + \frac{7}{144}\zeta^5, 1], [\frac{1}{576}\zeta^7 + \frac{5}{3}\zeta - \frac{7}{72}\zeta^3 - \frac{7}{144}\zeta^5, 1]]
```

The procedure **roots** computes the roots of the polynomial with multiplicities. In this case,  $\frac{1}{576}\zeta^7 + \frac{5}{3}\zeta - \frac{7}{72}\zeta^3 - \frac{7}{144}\zeta^5$  is a root of 2 in the field  $\mathbb{Q}(\zeta)$  with multiplicity 1, where  $\zeta$  satisfies  $\zeta^8 - 40\zeta^6 + 352\zeta^4 - 960\zeta^2 + 576 = 0$ . You can also do this computation in Maple with the **Primfield** procedure followed by **evala** (evaluate using algebraic numbers).

```
> map(convert, {sqrt(2), sqrt(3), sqrt(5)}, RootOf);
{RootOf(_Z^2 - 2, index = 1), RootOf(_Z^2 - 3, index = 1),
RootOf(_Z^2 - 5, index = 1)}
```

To get rid of the indexed RootOf-notation, you can apply the following rule:

```
> applyrule('RootOf(a::anything,b::anything)=RootOf(a)',%);
{RootOf(_Z^2 - 2), RootOf(_Z^2 - 3), RootOf(_Z^2 - 5)}
> evala(Primfield(%));
[[\zeta = RootOf(_Z^2 - 5) + RootOf(_Z^2 - 3) + RootOf(_Z^2 - 2)], [
RootOf(_Z^2 - 5) = \frac{95}{36}\zeta^3 - \frac{97}{288}\zeta^5 + \frac{5}{576}\zeta^7 - \frac{53}{12}\zeta,
RootOf(_Z^2 - 3) = \frac{15}{4}\zeta - \frac{61}{24}\zeta^3 + \frac{37}{96}\zeta^5 - \frac{1}{96}\zeta^7,
RootOf(_Z^2 - 2) = \frac{1}{576}\zeta^7 + \frac{5}{3}\zeta - \frac{7}{72}\zeta^3 - \frac{7}{144}\zeta^5]]
```

Note that because of an earlier alias of  $\zeta$ , the expression is in readable format. We replace it by the variable **Zeta**, which prints in the same way as **zeta**, but is left untouched in the following conversion to radical notation:

```
> convert(subs(zeta=Zeta, %), radical);
[[ $\zeta = \sqrt{5} + \sqrt{3} + \sqrt{2}$ ], [ $\sqrt{5} = \frac{95}{36} \zeta^3 - \frac{97}{288} \zeta^5 + \frac{5}{576} \zeta^7 - \frac{53}{12} \zeta, \sqrt{3} = \frac{15}{4} \zeta - \frac{61}{24} \zeta^3 + \frac{37}{96} \zeta^5 - \frac{1}{96} \zeta^7, \sqrt{2} = \frac{1}{576} \zeta^7 + \frac{5}{3} \zeta - \frac{7}{72} \zeta^3 - \frac{7}{144} \zeta^5]$ ]
> poly := subs(_Z=z, op(zeta));
poly :=  $-960 z^2 + 352 z^4 - 40 z^6 + z^8 + 576$ 
```

The above description of **RootOf** applies for manipulation of any root of a polynomial without specifying which root one actually has in mind. However, there is a selection mechanism for roots. A few examples.

```
> RootOf(x^2+9/10, x);
RootOf(10_Z^2 + 9)
> evalf(%);
.9486832980 I
> RootOf(x^2+9/10, x, -1.0*I);
RootOf(10_Z^2 + 9, -1.0 I)
> evalf(%);
-.9486832981 I
> RootOf(x^2+9/10, x, -1.0*I..1.0*I);
RootOf(10_Z^2 + 9, -1.0 I..1.0 I)
> evalf(%);
-.9486832981 I
```

The semantics of the two selectors in **RootOf** are given in Table 2.4.

Selector	Selected Root
<code>a + b*I</code>	closest in absolute value to <code>a + b*I</code>
<code>a + b*I .. c + d*I</code>	first root in the given range from the <b>fsolve</b> order
<code>index = a</code>	the <code>a</code> -th root in some ordering of roots
<code>label = a</code>	labeled by <code>a</code>

Table 2.4. Selectors on **RootOf**.

## 2.6 Complex Numbers

In contrast to algebraic numbers, complex numbers are of basic type. The complex number  $i$  (with  $i^2 = -1$ ) is by default represented in Maple by `I`. But a complex number can also be generated independently of the setting of the imaginary unit using the **Complex** constructor. In fact, the notation **Complex**(*real part*, *imaginary part*) resembles Maple's internal representation of a complex number: a data vector consisting of a header `COMPLEX` and pointers to the real and imaginary part. Numeric complex number arithmetic is automatic.

```
> Complex(0,1); Complex(2,3)
I
2 + 3 I
> (2+3*I) * (4+5*I);
-7 + 22 I
> Re(%), Im(%), conjugate(%), abs(%), argument(%);
-7, 22, -7 - 22 I, sqrt(533), -arctan(22/7) + pi
> 1/%;
-7/533 - 22/533 I
```

In Maple, many mathematical functions are regarded as complex functions. In case of multiple-valued complex functions Maple uses the principal branch.

```
> cos(I), ln(I), arccoth(0), sqrt(-8);
cosh(1), 1/2 I pi, -1/2 I pi, 2 I sqrt(2)
> sqrt((1.0+I)^2-1.0);
.7861513778 + 1.272019650 I
> Zeta(0.5+I), GAMMA(0.5+I);
0.1439364271 - 0.7220997435 I, 0.3006946173 - 0.4249678794 I
```

Two pictures of absolute values of complex functions are shown below.

```
> plot3d(abs(GAMMA(x+y*I)), x=-Pi..Pi, y=-Pi..Pi,
> view=0..5, grid=[30,30], orientation=[-120,45],
> axes=frame, style=patchcontour);
```

The surface plot of the absolute value of the gamma function in the complex plane is shown in Figure 2.8.

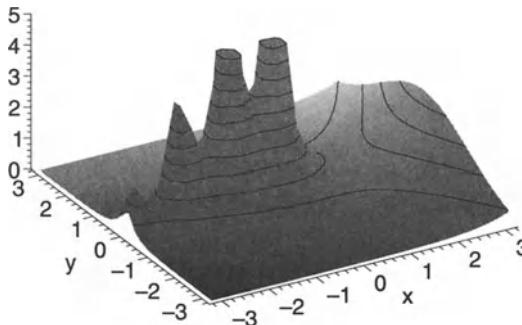


Figure 2.8. Plot of the absolute value of the gamma function.

In the above **plot3d** command, we have specified all options necessary to generate this picture. Many of these options can be manually changed after the surface has been drawn. For more details you are referred to Chapter 15, in which graphics is discussed in great detail.

The plot of the absolute value of the Riemann zeta function on the critical line  $\Re(z) = \frac{1}{2}$  is shown in Figure 2.9.

```
> plot(abs(Zeta(1/2+y*I)), y=0..36, numpoints=1000);
```

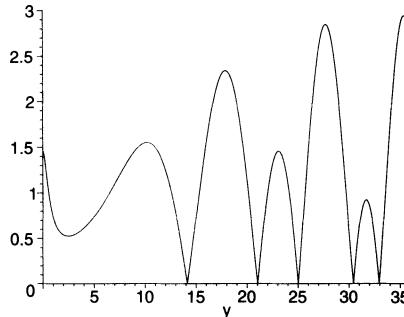


Figure 2.9. Plot of the absolute value of the Riemann zeta function on the critical line.

It shows the first few zeros of the Riemann function on the critical line. The famous Riemann hypothesis [138] states that all the complex zeros lie on the critical line. Many times the Riemann hypothesis has been numerically investigated; see [169, 187] for recent results.

To activate Maple's knowledge about complex numbers in *symbolic* expressions you must use the procedure **evalc** (evaluate using complex number arithmetic). **evalc** assumes that all variables in an expression represent real-valued quantities and puts a complex number in its canonical form  $a + bI$ , where  $a$  and  $b$  are real numbers.

```
> 1/(2+p-q*I);
```

$$\frac{1}{2 + p - q I}$$

```

> evalc(%);

$$\frac{2+p}{(2+p)^2+q^2} + \frac{qI}{(2+p)^2+q^2}$$

> abs(%);

$$\frac{1}{|2+p-qI|}$$

> evalc(%);

$$\frac{1}{\sqrt{4+4p+p^2+q^2}}$$


```

In the next example we shall see in the end how assumptions are helpful in obtaining simpler results.

```

> sqrt(p+q*I);

$$\sqrt{p+qI}$$

> evalc(%);

$$\frac{\sqrt{2\sqrt{p^2+q^2}+2p}}{2} + \frac{1}{2}I \operatorname{csgn}(q-pI) \sqrt{2\sqrt{p^2+q^2}-2p}$$


```

The complex sign function **csgn** is defined by

$$\operatorname{csgn}(z) = \begin{cases} 1 & \text{if } \Re(z) > 0 \text{ or } (\Re(z) = 0 \text{ and } \Im(z) > 0); \\ \operatorname{Envsignum0} & \text{if } z = 0 \text{ and } \operatorname{Envsignum0} \text{ is assigned;} \\ 0 & \text{if } z = 0 \text{ and } \operatorname{Envsignum0} \text{ is not assigned;} \\ -1 & \text{otherwise.} \end{cases}$$

Here, the value of **csgn(0)** is determined by the so-called *environment variable* **Envsignum0** as the following examples show.

```

> _Envsignum0 := 13: csgn(0);
13
> _Envsignum0 := 1: csgn(0);
1
> _Envsignum0 := '_Envsignum0': csgn(0);
0

```

Why is this special variable introduced? And why is it called **Envsignum0**? The latter question is easily answered: the environment variable controls the value of **signum(0)**. Of course, **signum** is used for the sign function of a real or expression, defined as

$$\operatorname{signum}(z) = \begin{cases} z / \operatorname{abs}(z) & \text{if } z \neq 0; \\ \operatorname{Envsignum0} & \text{if } z = 0 \text{ and } \operatorname{Envsignum0} \text{ is assigned;} \\ 0 & \text{if } z = 0 \text{ and } \operatorname{Envsignum0} \text{ is not assigned.} \end{cases}$$

The validity of simplification of expressions may depend on the value at 0. For example, the transformation

$$\text{signum}(\text{abs}(x)) \rightarrow 1$$

is wrong at  $x = 0$  if  $\text{signum}(0) = 0$ . Similarly, for real  $x$  and  $y$ , the transformation

$$\text{signum}(x y) \rightarrow \text{signum}(x) \text{signum}(y)$$

is wrong for  $x < 0$  and  $y = 0$  if  $\text{signum}(0) = 1$ . Maple should not do the simplification in this case; and indeed it does not simplify automatically.

```
> _Envsignum0 := 1: signum(0);
1
> signum(x*y);
signum(x y)
```

The value of `_Envsignum0` determines which transformations and simplifications can be applied as follows:

- `_Envsignum0` is not assigned a value.

In this case, any transformations that are valid everywhere except possibly at 0 can be applied.

- `_Envsignum0` is assigned a value.

In this case, only transformations that are valid everywhere, taking into consideration the assigned value of `signum(0)`, can be applied.

Automatic simplification is a subtle matter, but you see that you as a user have some control over it.

Let us continue with the `sqrt` example. When you make assumptions, then Maple may be able to simplify further. There are two ways of doing this: temporarily via the `assuming` operator or definitely via the procedure `assume`.

```
> evalc(sqrt(p+q*I)) assuming p>0, q>0;

$$\frac{\sqrt{2 \sqrt{p^2 + q^2} + 2 p}}{2} + \frac{1}{2} I \sqrt{2 \sqrt{p^2 + q^2} - 2 p}$$

> assume(p>0, q>0):
> evalc(sqrt(p+q*I));

$$\frac{\sqrt{2 \sqrt{p^{\sim 2} + q^{\sim 2}} + 2 p^{\sim}}}{2} + \frac{1}{2} I \sqrt{2 \sqrt{p^{\sim 2} + q^{\sim 2}} - 2 p^{\sim}}$$

```

The tildes after `p` and `q` indicate that assumptions hold for these variables. Maple can inform you about the properties via the procedure `about`. For a more detailed description of `assume` we refer to Chapter 13.

```
> about(p);
Originally p, renamed p~:
is assumed to be: RealRange(Open(0),infinity)
```

Let us finish with the notation of the square root of -1. Electrical engineers prefer  $j$  or  $J$  to denote this root so that the capital letter  $I$  is free to denote current in an electric circuit. This can be achieved in Maple as follows.

```
> interface(imaginaryunit=J);
```

Henceforth, complex numbers have standard notation  $a + bJ$ .

```
> J^2;
-1
> 1/(1+J);

$$\frac{1}{2} - \frac{1}{2} J$$

> 1/(x+y*J);

$$\frac{1}{x + y J}$$

> evalc(%);

$$\frac{x}{x^2 + y^2} - \frac{y J}{x^2 + y^2}$$

> solve(x^4=1, x);
-1, 1, J, -J
```

If you place the command `interface(imaginaryunit = J)` in your Maple initialization file, you can always use the  $J$  notation from scratch.

By the way, if you want to use different names for functions, say `tg` instead of `tan` and the Portuguese `sen` instead of `sin`, then you can achieve this with the `alias` procedure.

```
> alias(tg=tan, sen=sin):
> integrate(cos(x)+1/cos(x)^2, x);

$$\text{sen}(x) + \frac{\text{sen}(x)}{\cos(x)}$$

> convert(%-sen(x), tg);
tg(x)
```

## 2.7 Exercises

1. Consider the following Maple session.

```
> 3^2;
> 4^2;
16
> % + %%;
```

Does the last instruction make sense? If so, what is the result? If not, why?

2. Explain the different results of the following Maple commands.

- (a)  $x:y;$
- (b)  $x/y;$
- (c)  $x\backslash y;$

3. In this exercise you can practice your skills in using the help system of Maple.

- (a) Suppose that you want to select from an equation, e.g.,  $1 = \cos^2 x + \sin^2 x$ , only the left or right side. How can you easily do this in Maple?
  - (b) Suppose that you want to compute the continued fraction approximation of the exponential function; can Maple do this for you? If yes, carry out the computation.
  - (c) Suppose that you want to factor the polynomial  $x^8 + x^6 + 10x^4 + 10x^3 + 8x^2 + 2x + 8$  modulo 13. Can Maple do this? If yes, carry out this factorization.
  - (d) Suppose that you want to determine all subsets of the set  $\{1, 2, 3, 4, 5\}$ . How can you do this in Maple?
4. Load the `numtheory` package by entering `with(numtheory)`; You may recognize some functions from number theory; some of the routines in this package are useful in answering the following questions.
- (a) Build a list of all integers that divide 9876543210123456789.
  - (b) Find the prime number that is closest to 9876543210123456789.
  - (c) What is the prime factorization of  $5^{5^5}$ ?
  - (d) Expand the base `exp(1)` of the natural logarithm as a continued fraction up to 10 levels deep.
5. In Maple, what is the difference between  $1/3 + 1/3 + 1/3$  and  $1.0/3.0 + 1.0/3.0 + 1.0/3.0$ ?
6. Find the floating-point approximation of  $e^{\pi\sqrt{163}} - 262537412640768744$  using a precision of 15, 25, and 35 digits, respectively.
7. Calculate  $\pi^{(\pi^\pi)}$  to nine decimal places.
8. Compute this exercise in a floating-point precision of 8 decimal places. What is the result of

$$\frac{310.0 \times 320.0 \times 330.0}{-\sqrt{310.0 \times 320.0} \times \sqrt{320.0 \times 330.0} \times \sqrt{330.0 \times 310.0}}$$

9. Do you remember which of the numbers  $\frac{19}{6}$ ,  $\frac{22}{7}$ , and  $\frac{25}{8}$  is a fairly good rational approximation of  $\pi$ ? Use Maple to find the best of these three numbers. Find the best rational approximation  $a/b$  of  $\pi$ , where  $a$  and  $b$  are natural numbers less than 1000 (Hint: look at the continued fraction expansion of  $\pi$ ).
10. Check that  $\sqrt{2\sqrt{19549} + 286}$  is equal to  $\sqrt{113} + \sqrt{173}$ .
11. In Maple, transform  $\frac{1}{\sqrt{3} + 1}$  into an expression of the form  $a + b\sqrt{3}$ , with rational numbers  $a$  and  $b$ .
12. Let  $\theta$  be a root of the polynomial  $\theta^3 - \theta - 1$  and consider the extension of the field of rational numbers with  $\theta$ . So, we consider expressions of the form  $a + b\theta + c\theta^2$ , where  $a, b, c \in \mathbb{Q}$ , and in calculations with these expressions, we apply the identity  $\theta^3 = \theta + 1$ . Transform with Maple  $\frac{1}{\theta^2 + 1}$  into an expression of the form  $a + b\theta + c\theta^2$ , where  $a, b, c \in \mathbb{Q}$ .
13. Let  $\alpha = \sqrt{2}$ ,  $\beta = \sqrt{3}$ , and  $\gamma = \sqrt{5}$ . Use the procedure **Primfield** to compute a primitive element  $\zeta$  for the field extension  $\mathbb{Q}(\alpha, \beta, \gamma)$ , and compare the result with the last example of §2.5.
14. Show that Maple knows that the exponential power of a complex number can be written in terms of cosine and sine of the real and imaginary parts of that number. Also calculate  $e^{\pi i/12}$  in that form.
15. Show with Maple that

$$\tanh(z/2) = \frac{\sinh x + i \sin y}{\cosh x + \cos y},$$

for any complex number  $z = x + yi$  with  $x, y \in \mathbb{R}$ .

# 3

## Variables and Names

A Maple session usually consists of a series of statements in which values are computed, assigned names, and are used in further computations. In this chapter, we shall discuss what are valid names in Maple, how to assign a value to a name, how to unassign a variable, how to protect or unprotect a variable, and how to associate an attribute or a property to a variable. Unlike in programming languages such as FORTRAN, Algol, and C, there is in Maple no need to declare the types of variables. Maple figures out the type of an expression on the basis of how it is internally represented and how it is used. In this chapter, we shall have a look at the basic data types. Moreover, we shall describe the way symbolic expressions are normally evaluated, viz., by full evaluation.

### 3.1 Assignment and Unassignment

The secret behind the success of computer algebra systems in scientific computation is that you can manipulate formulae and solve mathematical problems where unknowns and parameters are involved. As an example of the use of such variables, we ask Maple about the general formulae for the solutions of a quadratic equation. Solving equations is usually done in Maple with **solve**.

```
> solve(a*x^2+b*x+c=0, x);  
-b + sqrt(b^2 - 4*a*c)  -b - sqrt(b^2 - 4*a*c)  
-----, -----  
      2 a            2 a
```

The variables **a**, **b**, and **c** are parameters and **x** is an unknown; they are used as symbols, neither more nor less. This use of *free* or *unbound variables*, which do not point to any value except their own name, is characteristic of computer algebra systems.

On the other hand, a variable can be bound to, or assigned, a value. The use of *assigned variables* is twofold: you often want to label a calculated result or some complicated expression for further reference, and when you use Maple as a programming language, you want to specify data in an algorithm by name instead of using the data themselves. When data only have some fixed value, we speak of *symbolic constants*. We have seen in §2.4 what symbolic constants Maple initially has available and how new constants can be introduced or removed. In Maple, when you want to use assigned variables, the two-character symbol **:=** is the assignment operator that is commonly used. The alternative **assign** will be described later on in this section.

```
> polynomial := 9*x^3 - 37*x^2 + 47*x - 19;
          polynomial := 9 x3 - 37 x2 + 47 x - 19
> # compute roots of the polynomial with multiplicities
> roots(polynomial);
[[1, 2], [19/9, 1]]
> # substitute x for the root 19/9 into the polynomial
> subs(x=19/9, polynomial);
0
> polynomial, x;
9 x3 - 37 x2 + 47 x - 19, x
```

In the above example, the effect of the first statement is that the variable **polynomial** is bound to  $9x^3 - 37x^2 + 47x - 19$ . Each time Maple encounters this variable, it takes its value; this is called *evaluation*. So, the instruction

```
roots(polynomial)
```

is read by Maple as

```
roots(9*x^3 - 37*x^2 + 47*x - 19)
```

In the meantime, the variable **x** has no value, except its own name. Through the instruction

```
subs(x=19/9, polynomial)
```

**x** in the expression labeled by **polynomial**, i.e., in  $9x^3 - 37x^2 + 47x - 19$  is replaced by  $19/9$ : a *substitution*. But **x** itself remains unchanged and still points to itself. The same holds for the polynomial; **polynomial**

has not been assigned a value through the substitution. You can also check the root  $19/9$  by assignment of  $x$ .

```
> x := 19/9;
x :=  $\frac{19}{9}$ 
> polynomial;
0
```

The way Maple carries out the last instruction can be understood as follows:

- Maple sees the variable **polynomial** and evaluates it first to  $9*x^3 - 37*x^2 + 47*x - 19$ ,
- then it evaluates each  $x$  in this expression to  $19/9$ ,
- and finally simplifies the result to 0, i.e., does the arithmetic.

The disadvantage of checking a root by assignment instead of substitution is that  $x$  now points to some number. If you want  $x$  to point to something else, you can achieve this by a new assignment.

```
> x := unknown: polynomial;
9 unknown3 - 37 unknown2 + 47 unknown - 19
```

When you assign a value, say 7, to the variable **unknown**

```
> unknown := 7;
unknown := 7
```

then  $x$  evaluates to **unknown**, and this for its part evaluates to 7.

```
> x;
7
```

You see that Maple evaluates as far as possible, using the values of variables as they are assigned at the state of computing. We refer to this general rule as *full evaluation*.

```
> polynomial;
1584
```

We shall come back to full evaluation of expressions in §3.2.

```
> x := 'x';
x := x
```

The last instruction gives  $x$  back its own name as its value. So, you can use *apostrophes* (also referred to as *forward quotes*, *right quotes*, and *acute accents*) to unassign variables. Whenever apostrophes surround an expression, Maple does not apply evaluation to the enclosed expression. In this particular case, evaluation of  $x$  is avoided.

Let us linger over assignment and unassignment of indexed names and concatenated names.

```
> i := 1; A[i] := 2; A[i] := 3; B[i] := 4;
      i := 1
      A1 := 2
      A1 := 3
      B1 := 4
```

You see that the expression to the left of the assignment operator `:=` is evaluated to a name. The index of an indexed name is evaluated, but the indexed name is then not further evaluated: in the third assignment, the left-hand side `A[i]` is evaluated to `A[1]`, but `A[1]` is not further evaluated to 2.

The unassignment-method using apostrophes does not work for indexed names and concatenated names.

```
> A[i] := 'A[i]'; # no unassignment of A[i]
      A1 := Ai
> A[i]; # but infinite recursion occurs
Error, too many levels of recursion
```

Under many operating systems, when Maple enters an infinite recursion, it will give up at some point, sending the above message.

Sometimes, Maple warns you for recursive definitions of names.

```
> x := x+1;
Error, recursive assignment
```

This does not mean that such an assignment is not possible in Maple. Only at top level this assignment is recognized as a troublesome statement. Inside a procedure this assignment is perfectly allowed, because at definition time it is not known whether the assignment involves recursive definition of names. A simple example: suppose that `x` is still a free variable and that the following procedure for adding 1 to a given argument has been defined.

```
> restart;
> add1 := proc(y) y := y+1 end proc;
      add1 := proc(y) y := y + 1 end proc
```

Then the next instruction gives it the new value `x+1`.

```
> add1(x);
      x + 1
```

But see what happens if you evaluate `x`:

```
> x;
```

**Error, too many levels of recursion**

In the last instruction, Maple is asked to evaluate the expression:

$$x \rightarrow x + 1 \rightarrow (x + 1) + 1 \rightarrow ((x + 1) + 1) + 1 \rightarrow \dots$$

Due to full evaluation, Maple re-evaluates **x** over and over again until a recursion limit is exceeded.

Unassignment of indexed names and concatenated names can be achieved by the procedure **evaln** (evaluate to a name).

```
> A[i] := evaln(A[i]); A[i]; # unassignment of A[i]
Ai := Ai
Ai
```

The evaluation of the argument of **evaln** is again special: it only evaluates the index and produces a name that is not further evaluated. Of course, the procedure **evaln** can also be used for variables with “ordinary” names like **x**, **y**, and **z**. It is a kind of sure way to unassign variables.

Because it is hardly possible to recall which variables have been assigned values and which not, the computer algebra system provides for this purpose with a few utility functions; they are tabulated in Table 3.1.

Procedure	Effect
<b>anames</b>	shows assigned names, i.e., the names of bound variables
<b>unames</b>	shows all unassigned names, i.e., the names of all free variables
<b>assigned</b>	checks whether a variable is bound to a value different from its name or not

Table 3.1. Utility functions for usage of names.

The next sample session illustrates the use of these Maple procedures.

```
> unames();
identical, anyfunc, equation, positive, Integer, restart, radical,
.....,
nonnegint, terminal, unknown, extended_rational, anything,
patternplus, comopt, setpath, literal, displayprecision, Vector
> nops({unames()});
# number of unassigned names
```

We have omitted most of the output of **unames**, but from the few names shown and from the number of names it is already clear that the procedure

is of little use because it gives too many details. Not only the user-defined names are listed, but also the unassigned names generated and used by Maple itself. Below is the **selection** of all initially known, unassigned four-character names.

```
> select(s->length(s)=4, {unames()});  
{even, list, name, TEXT, none, READ, args, FAIL, true}
```

For a description of the use of the anonymous function `s -> length(s)=4` in the selection process, you are referred to §8.8. Another example is the selection of all unassigned names that contain the substring `con` except the name `con` itself.

```
> select(s->SearchText(con,s)>0, {unames()}) minus {con};  
{constant, realcons}
```

Let us look more closely at the procedure **anames**.

```
> restart; p := q; r := q*s;  
p := q  
r := q s  
> anames();  
p, r,
```

The procedure **anames** returns a sequence of names that are currently assigned values other than their own name. Here, they are user-defined variables. But when you ask for names that are assigned values of specific type, you will also get names of variables that are assigned initial values in Maple.

```
> anames('*'); # variable whose value is a product  
r  
> anames(name); # variables whose value are a name  
UseHardwareFloats, Rounding, p, index/newtable, mod  
> anames(string); # variables whose value are a string  
libname  
> anames(integer); # variables with integer value  
Digits, printlevel, Order  
> anames(anything); # variables with any assigned value  
traperror, map2, normal, indets, ..., kernelopts, OrderedNE
```

The meaning of the left quotes and data types will be explained in §3.3 and §3.4.

With the procedure **assigned** you can get the status of individual variables.

```
> assigned(q), assigned(r);
false, true
```

Meanwhile, you have seen four exceptions of the general rule of full evaluation:

- expressions that are enclosed by apostrophes are not evaluated.
- the expression to the left of the assignment operator `:=` is only evaluated to a corresponding name.
- the argument of the procedure `evaln` is only evaluated to a corresponding name.
- the argument of the procedure `assigned` is only evaluated to a corresponding name.

We end this section with an application of the two Maple procedures `assign` and `unassign`. Their names suggest their functionality, but there are some differences from what we have previously learned about assignments and unassignments.

`assign(name, expression)`

has the same effect as

`name := expression`

except that the first argument of the procedure `assign` is fully evaluated, whereas this rule is not applied to the left operand of the assignment operator `:=`. Another difference is that `assign` does not check for recursive definition: a recursive definition `assign(x, x+1)` for a free variable `x` is possible without any complaint from the system. What makes `assign` worthwhile is for example that it can be applied to sets of equations returned by the procedure `solve` when it is desired to assign the solution values to the variables.

```
> eqns := {x+y=a, b*x-1/3*y=c}:
> vars := {x, y}:
> sols := solve(eqns, vars);

sols := { $x = \frac{a + 3c}{3b + 1}, y = \frac{3(-c + ab)}{3b + 1}$ }

> x, y;
x, y

> assign(sols);
> x, y;

 $\frac{3c + a}{3b + 1}, 3\frac{ba - c}{3b + 1}$ 
```

As you will see in §3.6, the command **assign** is also convenient, if not indispensable, in assigning values to names that have been associated certain properties.

**unassign** can be used to unassign several variables in one statement or to unassign a variable without throwing away its associated properties.

```
> unassign('x', 'y'); x, y;
x, y
```

You may be tempted to use it in combination with **anames** in order to clear all your assigned values, but look what a drastic effect this can have on the behavior of Maple.

```
> # compute some integral
> integral := integrate(1/(x^2+1), x);
integral := arctan(x)

> unassign(anames()); # unassign variables

Error, (in assign) attempting to assign to 'content'
which is protected

> int(1/x, x); # recompute another integral

Error, (in int/definite) int/definite uses a 4th argument,
bb, which is missing
```

The reason for Maple's malfunctioning is that we did not only remove values of user-defined variables like **integral**, but also of system-defined variables. You had better use the **restart** statement to clear internal memory and start Maple afresh without leaving the system.

Table 3.2 summarizes the various styles of assignment and unassignment in Maple.

Template	Left-Hand Side	Right-Hand Side
<code>x := y</code>	to the name	normal
<code>assign(x=y)</code>	normal	normal
<code>x := evaln(y)</code>	to the name	to the name
<code>x := 'y'</code>	to the name	one step
<code>assign(x=evaln(y))</code>	normal	to the name
<code>assign(x='y')</code>	normal	one step
<code>unassign(x)</code>	normal	—

Table 3.2. Evaluation for assignment and unassignment.

## 3.2 Evaluation

In general, when Maple encounters a name in an instruction, it searches for the object to which the name points: we say that Maple *evaluates* the name. When a name points to another name, the system will search again for the object to which that name points, and so on, until it arrives at an object that is not a name or that points to itself. The last object is now used as if it were substituted in the instruction. So, the normal process is for every expression to be *fully evaluated*. This explains the next sample session.

```
> a := b;  b := c;  c:= 3;
      a := b
      b := c
      c := 3
> a;  # evaluation of a
      3
```

To get a better understanding of full evaluation, we look at the effect of assignments in terms of internal data structures. In Maple, a variable is internally represented by a data vector with three components, viz., a first component indicating that the vector represents a variable, a second component pointing to the current value of the variable, and a third component to name the variable. When we assign a value to a variable, we simply set a pointer from the named data vector to the assigned value. In the above example, this results in the representation shown in Figure 3.1.

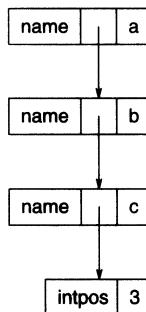


Figure 3.1. Internal representation after certain assignments.

The second component of the data vector representing the variable **a** points to the value of **a**: so, the value of **a** is equal to **b**. Similarly, the value of **b** is equal to **c**. We can check this internal representation by use of the **eval** procedure.

```
> a;  # evaluation of a
```

```

> eval(a,1); # value of a
      b
> eval(b,1); # value of b
      c
> eval(c,1); # value of c
      3
> eval(a,2); # two-level evaluation of a
      c
> eval(a,3); # three-level evaluation of a
      3

```

When we fully evaluate `a`, we walk down all pointers, starting at the data vector representing `a` and ending at the object of type “positive integer” and value 3. We say “`a` evaluates to 3.” When the variable `c` is now assigned a new value, say 5, then the variable `a` evaluates to 5. However, the value of `a` remains `b`; it can only be changed by an assignment.

```

> c := 5:
> a; # a now evaluates to 5
      5
> eval(a,1); # but the value of a remains the same
      b
> a := 4:
> a, eval(a,1);
      4, 4
> # the value of a and its full evaluation are both 4

```

In the process of full evaluation of expressions, variables are substituted by the values that they received in their most recent assignments. The next example illustrates this.

```

> x := y: y := 7:
> # the value of x is y, but x evaluates to 7
> eval(x,1), x;
      y, 7
> x := x;
      x := 7
> # now, the value of x is 7 and x also evaluates to 7
> y := 9:
> x; # the value and evaluation of x remain unchanged
      7

```

The instruction `x := x` suggests that `x` points to its own name again. This is not the case as you have seen before! If you want to achieve this, you

should have entered the command `x := evaln(x)` or `x := `x``. Here, the apostrophes prevent the full evaluation of `x`.

The easiest way to make sure that a name is passed through as an argument to a procedure is to use *apostrophes* to suppress evaluation. For example, the fourth parameter of the procedure `rem`, which is used for storage of the quotient, must be a name.

```
> # suppose that quotient has been assigned some value
> quotient := 0:
> rem(x^3+x+1, x^2+x+1, x, 'quotient');

$$2 + x$$

> quotient; # value has been set in previous command

$$x - 1$$

```

A more reliable way of preventing evaluation is to use the `evaln` procedure instead of apostrophes. If you really want to go safely, then you should reset the value of a variable first to its own name with the command

*variable* := `evaln(variable)`.

But what if you forgot the apostrophes in the above example?

```
> quotient := 0:
> rem(x^3+x+1, x^2+x+1, x, quotient);
Error, (in rem) illegal use of a formal parameter
```

What went wrong? Through the first instruction, the variable `quotient` gets the value 0. When the second instruction is carried out, Maple first evaluates its arguments, which leads to `rem(x^3+x+1, x^2+x+1, x, 0)`. Maple gives an error because it expects a name to which the computed quotient can be assigned as its fourth argument. If the variable `quotient` points to `x`, then an infinitely recursive structure would appear!

```
> quotient := x:
> rem(x^3+x+1, x^2+x+1, x, quotient);

$$2 + x$$

> eval(quotient,1), eval(quotient,2);

$$x, x - 1$$

```

So, internally an assignment `x := x - 1` is carried out, and this will cause an error when you later refer to `x`.

In other instances, you also want to or actually need to suppress full evaluation of expressions or of parts of expressions. Let us, for example, compute the sum of the first 5 prime numbers with the procedure `sum`.

```
> i := 999: # assume i has been assigned some value
> sum(ithprime(i), i=1..5);
Error, (in sum) summation variable previously assigned, second
argument evaluates to 999 = 1 .. 5
```

When Maple evaluates the argument `i=1..5` in the above function call it notices that it has gotten an invalid argument, viz., the natural number 999 instead of the name `i`. In our next try we use apostrophes to suppress premature evaluation.

```
> sum(ithprime(i), 'i'=1..5);
39535
```

No error message, but an unexpected result; the value is not equal to the sum of the first 5 prime numbers. You must also ensure that the summand evaluates to the proper expression.

```
> sum('ithprime(i)', 'i'=1..5);
28
```

In the first argument, the use of apostrophes is obligatory and you cannot use `evaln`. Because of the special evaluation rules for `evaln`, this would still evaluate to a function call with some integer as argument.

```
> i; # still assigned the value 999
999
> sum(evaln(ithprime(i)), evaln(i)=1..5);
5 ithprime(999)
> %;
39535
```

Good advice:

*In commands like `sum`, always use quotation marks around the summand and the summation index unless you are sure that they can be omitted.*

The result of evaluation of an expression enclosed by apostrophes can be best described as peeling off a pair of quotes.

```
> x := 1: # in this example x always has the value 1
> x+1;
2
> 'x'+1;
x + 1
> 'x+1';
x + 1
> ''x'+1''';
"x' + 1"
> %;
'x' + 1
```

```
> %;

$$x + 1$$

> %;

$$2$$

```

### 3.3 Names of Variables

In Maple, the simplest kind of names of variables and constants are *symbols*, i.e., sequences of letters, digits, and underscores, the first of which must be a letter or underscore. The internal representation of symbols is similar to that of integers: a dynamic data vector in which the first computer word encodes all the information including the indicator **NAME** and the vector length, and in which the other computer words contain the characters (maximum four characters per word on a 32-bit machine). The same size limitation for strings holds: a maximum length of  $2^{28} - 8 = 268435448$  characters (q.v., §2.3). A few examples of valid names:

```
> restart: # make sure that a fresh start is made
> x, a_long_name_containing_underscores, H2O;

$$x, a\_long\_name\_containing\_underscores, H2O$$

> unknown, UNKNOWN, UnKnOwN;

$$unknown, ?, UnKnOwN$$

```

Note that Maple uses both upper- and lower-case letters and that the system displays the name UNKNOWN in the worksheet interface as the question mark. Keep this in mind when using Maple names for mathematical constants:  $\pi$  is denoted in Maple by Pi, but the system does not associate the name pi with this constant.

```
> evalf([Pi, pi]);

$$[3.141592654, \pi]$$

```

Display of Greek characters in the worksheet interface and in the conversion to L<sup>A</sup>T<sub>E</sub>X format have some traps to fall into.

```
> [Pi, pi];

$$[\pi, \pi]$$

> latex([Pi, pi]);

$$[\backslash\pi, \backslash\pi]$$

```

You may have expected  $[\Pi, \pi]$  and  $[\backslash Pi, \backslash pi]$ , respectively.

```
> [Zeta, zeta];

$$[\zeta, \zeta]$$

```

$[Z, \zeta]$  would have been a better notation, but probably the notation is inspired by Maple's notation of the Riemann  $\zeta$ -function.

Also be careful with the Maple name for the complex square root of  $-1$ : not `i`, but `I`. Another confusing notation is used by Maple for the base of the natural logarithm: not `e` or `E`, but `exp(1)`. The next example shows how careful you must be

```
> [e^x, exp(1)^x, exp(x)] ;
[ex, (e)x, ex]
```

In the worksheet interface, the output sometimes only differs in the font choice. When this expression is converted into LATEX code, the generated items are as follows:

```
> latex(%);
[{\text{e}}^{\text{x}}, \left(\text{e}^{\{1\}}\right)^{\text{x}}, {\text{e}}^{\{\text{x}\}}]
```

But Maple distinguishes the two expressions really as the power of  $x$  with the name `e` (i.e., not with the base of the natural logarithm), and the exponential mapping applied to  $x$ . This distinction comes immediately to the surface when you substitute a floating-point number and evaluate the list.

```
> eval(% , x=0.5);
[e0.5, (e)0.5, 1.648721271]
> evalf(%);
[e0.5, 1.648721271, 1.648721271]
```

Names starting with an underscore are used by Maple itself (recall the result of `RootOf` in §2.5). There are also 46 reserved words that Maple does not accept right away as valid names because they have a special meaning in the Maple programming language. Table 3.3 can be obtained by entering `?reserved`.

and	assuming	break	by	catch
description	do	done	elif	else
end	error	export	fi	finally
for	from	global	if	implies
in	intersect	local	minus	mod
module	next	not	od	option
options	or	proc	quit	read
return	save	stop	subset	then
to	try	union	use	while
xor				

Table 3.3. Reserved words in Maple.

Furthermore, there are names that already have a meaning in Maple, e.g., names of mathematical functions like **sin**, **cos**, **exp**, **sqrt**, ..., names of procedures like **copy**, **indices**, **lhs**, **rhs**, **type**, **coeff**, **degree**, **order**, ..., and names of data types like **set**, **list**, **matrix**, ... Most of these names are protected so that you cannot assign them values by accident. Use the procedures **protect** and **unprotect** if you want to add your own protected words or want to redefine names that are already protected.

An example: you can extend the built-in procedure **unames** so that the output is a sorted list of names.

```
> unames := subs(old_unames=eval(unames),
> proc() sort([old_unames()]) end proc):
> protect(unames):
> whattype(unames()); # check whether it is a list
list
```

Indexed names are a bit special when you want to protect them. When you assign an indexed name a value, you create a table (see §12.5) with the header of the indexed name as the name of the table. To prevent such an assignment, you must therefore protect the header instead of the indexed name.

```
> protect(A): A[1] := 2;
Error,
attempting to assign to array/table 'A' which is protected
```

A third category of names that cannot be assigned any arbitrary value are the so-called *environment variables*. Entering the help request **?environment** gives you a description of environment variables. The initially assigned environment variables can be found as follows:

```
> restart; anames(environment);
```

*Testzero, UseHardwareFloats, Rounding, %, %%%, Digits, index/newtable, mod, %%%, Order, printlevel, Normalizer, NumericEventHandlers*

For example, the environment variable **printlevel** allows you to control and customize the amount of information displayed on the screen during a computation. Its default value is 1, but you cannot make it half as verbose by the following assignment.

```
> printlevel := 1/2;
Error, printlevel must be a word size integer
```

An environment variable is Maple's analogue of a *fluid variable* in MuPAD, Reduce, and Lisp: such a variable is globally known inside a procedure and automatically re-set to its old value after a procedure has been executed. An example makes this clearer: we define a procedure that does numerical evaluation in 30 digits.

```

> Digits; # the current value
10
> evalf30 := proc(expr)
>   Digits := 30; # Digits temporarily set to 30
>   evalf(expr)
> end proc:
> evalf30(Pi); # Pi in 30 digits
3.14159265358979323846264338328
> Digits; # Digits is still bound to old value
10

```

Other environment variables in Maple have names that start with `_Env`, such as `_EnvExplicit` for notation of algebraic numbers and functions in the result of `solve`, or `_Envsignum0` for definition of `signum(0)`. You can also define your own environment variable: simply choose a name that starts with `_Env`. But be careful, names starting with an underscore are used by Maple itself at several places. Good advice:

*Avoid using names that start with an underscore if they are not absolutely necessary.*

Otherwise, you may get puzzled by Maple results like the one shown below.

```

> _Z := sqrt(2):
> RootOf(x^2-x, x);
Error, (in RootOf) expression independent of, 2^(1/2)

```

If you want to use spaces in a name for reasons of readability, or if you want to use special characters such as dots, colons, or slashes, then you must surround the name with *left quotes* (`, also referred to as *back quotes*, *grave accents*, and *reversed apostrophes*). These left quotes themselves are not part of the name and disappear when it is displayed; they only mark the boundaries of the name. Two consecutive left quotes in a name are parsed as an enclosed left quote character. Below are few more examples of valid names in Maple.

```

> 'exercise 1', '/usr/local/lib/maple/lib/C.m';
exercise 1, /usr/local/lib/maple/lib/C.m
> 'Answer:'; # the colon is part of the name
Answer:
> ``; # the empty name

> ````/usr/local/lib/maple/lib/C.m```;
`/usr/local/lib/maple/lib/C.m`

```

Don't associate Maple names enclosed by left quotes with *strings* in conventional programming languages like FORTRAN. The following example shows you this.

```
> miscellanea := 'apparent_text';
      miscellanea := apparent_text
> apparent_text := cos(Pi);
      apparent_text := -1
> miscellanea, 'miscellanea';
      -1, -1
```

As you see, these left quotes have no influence here. Occasionally, rather strange things happen with these quotes.

```
> 'Here starts a name that is
Warning, incomplete quoted name; use ' to end the name
> concatenated' := 0;

      Here starts a name that is\
      concatenated := 0
```

If you happen to type an opening left quote but forget about the closing one, Maple does not seem to accept input anymore. To get you out of this deadlock, follow Maple's advice: enter a closing left quote followed by a semicolon and everything should work fine again.

Objects of type *string* exist in Maple and can be created: they are sequences of characters between double quotes, mostly used for import and export of data between Maple and other computational systems, and for formatted I/O in programming. Some examples of creating and manipulating strings:

```
> "A very long string "
> "written in two lines and with funny characters %#($&Q ";
      "A very long string written in two lines and with funny characters
      %#($&Q "
```

You cannot give a string some value.

```
> " the number two " := 2;
      Error, invalid left hand side of assignment
```

You can select individual characters or substrings. Either use the **substring** procedure or square brackets and some range of indices to select a substring.

```
> substring(" the number two ", 13..15);
      "two"
> " the number two "[1..12];
```

```
" the number "
> " the number two "[2];
"t"
```

The **length** procedure determines the length of a string, i.e., the total number of characters.

```
> length(" the number two ");
16
```

More advanced and/or optimized string manipulations are provided by the **StringTools** package.

Now and then, but mostly when using Maple as a programming language, concatenation of names and strings is handy. A few examples of the use of the procedure **cat** and the concatenation operator (**||**) follow.

```
> libname, cat(libname, "/C.m");
"C:\Program Files\Maple 8/lib", "C:\Program Files\Maple 8/lib/C.m"
> # alternative way of concatenation
> "" || "C:\Program Files\Maple 8/lib" || "/" || "C.m" ;
" C:\Program Files\Maple 8/lib/C.m"
> X||Y, X||1;
XY, X1
> X||(Y, 1);
XY, X1
> X||(1..8);
X1, X2, X3, X4, X5, X6, X7, X8
> `` || (x, y) || (1..4);
X1, X2, X3, X4, Y1, Y2, Y3, Y4
> "" || (x, y) || (1..4);
"X1", "X2", "X3", "X4", "Y1", "Y2", "Y3", "Y4"
> i:=4: X||i, i||x;
X4, iX
```

The last example shows that, when using the concatenation operator, the first name does not get evaluated. The obvious design choice has been that, for example, **x||1** should always evaluate to **x1**, regardless whether **x** has been assigned a value or not. This explains why we have used in some of the above examples the empty name as first argument in concatenation.

Don't confuse the use of quotation marks and percentage symbols in Maple; we summarize their effects in Table 3.4.

Symbol	Purpose
' '	markers for a name with special characters
' '	delay of evaluation
" "	markers for a string in Maple
%	reference to previously evaluated expression
%%	reference to the second last expression evaluated
%%%	reference to the third last expression evaluated

Table 3.4. Quotation marks and percentage symbols.

## 3.4 Basic Data Types

Values of data are divided into classes: the *data types*. The data type fixes, among other things, what values can be taken or what operations are allowed on the data. The usual elementary data types like *integer*, *floating-point number*, and *string* are present in Maple, but there are many more. Maple can show you the basic type of data with the command **whattype**.

```
> whattype(5.0);
                           float
> whattype(I);
                           complex
> whattype('an example of a long name');
                           symbol
> whattype("an example of a long string");
                           string
> whattype({1, 2, 3});
                           set
> whattype(1, 2, 3);
                           exprseq
```

The result of **whattype** is a description of the header of the data vector. These data types that only require the header of the data vector are also called *surface data types*. In Table 3.5 we list the most common surface data types available in Maple. You get a more complete list with **?type,surface**.

Data	Type	Example
<i>Numbers and Strings:</i>		
integer	integer	1
fraction	fraction	1/2
floating-point number	float	0.33333
complex number	complex	1+I
alphanumeric text (symbol)	symbol	xvalue
indexed name	indexed	X[1]
string	string	"this is good"
<i>Arithmetic Expressions:</i>		
sum	`+`	x + y
product	`*`	x * y
power	`^` or `**`	x ^ y
<i>Relational Expressions:</i>		
equation	`=`, equation	x+1 = 1+x
inequality	`<>`	Pi <> pi
less-than-relation	`<`	2 < 3
less-than-or-equal	`<=`	E <= Pi
relation		
<i>Logical Expressions:</i>		
and-expression	`and`	P and Q
or-expression	`or`	P or Q
negation	`not`	not P
<i>Composite Expressions:</i>		
expression sequence	exprseq	a,b,c
set	set	{a,b,c}
list	list	[a,b,c]
table	table	table([a,b,c])
function call	function	f(x), where f is not defined
<i>Miscellanea:</i>		
unevaluated concatenation	`  `	a  (1..n), where n has no value
range	`...`, range	1 .. 3
generalized power series	series	x^-1 - gamma + O(x)
procedure definition	procedure	proc(x) x^3 end
unevaluated expression	uneval	'`x + y``

Table 3.5. Commonly used surface data types

You can also verify a data type in Maple with a name recognized in `type` procedure.

```
> type(x+1, '+');          true  
> type(a[1], symbol);      false  
> type(a[1], indexed);     true
```

There exist a hierarchy of atomic data types as shown in Figure 3.2.

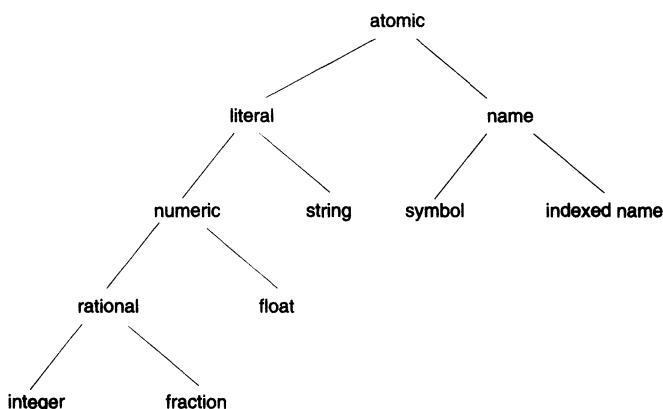


Figure 3.2. A hierarchy of atomic data types.

An object is of type *literal* if it evaluates to itself, i.e., a number or a string. The type *symbol* is used for simple names i.e, not for indexed names. The type *atomic* includes numbers, strings, symbols, and indexed names.

Maple provides two other tools for type testing, viz., **hastype** and **typematch**.

```

> hastype(x+1, '+');
                                         true

> typematch(x+1, (a::symbol) &+ (b::integer));
                                         true

> a, b;
                                         x, 1

> hastype(x+1/2*y, fraction);
                                         true

> hastype(x+2*y, fraction);
                                         true

```

*false*

The procedures **type**, **hastype**, and **typematch** test not only for surface data types, but also test more general types of expressions. They traverse expression trees and test for so-called *nested data types*. Furthermore, they can be used to test for so-called *structured data types*. These are expressions different from names that can be interpreted as data types. Some meaningful nested and structured data types:

```
> type(x^2+x+1, polynom);
                                         true
> type({x^2+x+1, x^2-x}, set(polynom));
                                         true
> type(x^2+x+Pi, polynom(integer,x));
                                         false
> type(x^2+x+1, quadratic(x));
                                         true
```

In Table 3.6 we have summarized how to test types in Maple.

Command	Tested Type
<b>whattype</b>	surface data type
<b>type</b>	nested or structured data type
<b>hastype</b>	existence of subexpression(s) of certain type
<b>typematch</b>	testing and matching a structured type

Table 3.6. Tests for types.

**typematch** is the most powerful command: when it succeeds in matching a structured type, all the pattern variables are assigned values according to this successful match. For example

```
> typematch(exp(x), a::exp(b::name));
                                         true
> a, b;
                                         ex, x
```

Note the use of the double colon (::) in the second part of the procedure **typematch**. They are not two command separators; it is an operator that pairs a symbol to a data type. We summarize the usage of names and operators that contain a colon, semicolon or equal sign in Table 3.7. The operator in the last row of the table is more meant for programming purposes. At interactive use, it only occurs when you do not want to load a package that has been implemented as a so-called module before calling a package function.

Operator	Purpose
:	command separator
;	command separator
=	equation or relation
::	association of type or property
:=	assignment
:-	module member selection

Table 3.7. Combinations of colon, semicolon, and equal sign.

In a conventional programming language like FORTRAN you have to specify the type of a variable and it cannot be changed within a program. Nothing of the kind in Maple: a variable has no fixed type. Otherwise, your interactive work would be overloaded with type declarations, which are almost always superfluous because of the mathematical context, and type conversions would become cumbersome. A few illustrative examples:

```
> number := 1: whattype(number);
          integer
> number := 0.75: whattype(number);
          float
> number := convert(number, fraction);
          number := 3/4
> convert(number, 'binary');
Error, invalid input: convert/binary expects its 1st argument,
n, to be of type {float, integer}, but received 3/4
```

The last statement illustrates that not all type changes are allowed in Maple: type changes must make sense to the system. For example, the rational number  $\frac{3}{4}$  can be written as the continued fraction

$$0 + \cfrac{1}{1 + \cfrac{1}{3}}$$

```
> convert(number, 'confrac');
[0, 1, 3]
```

If present, the apostrophes around the second argument of the procedure **convert** are used to prevent evaluation. **binary** and **confrac** are not protected names in Maple! You can find out whether a word is protected or not by the command **type(name, protected)**. More readable, two-dimensional output of a continued fraction is obtained by the procedure **numtheory[cfrac]** from the **numtheory** package for **number theory**.

```
> # load the function cfrac from the numtheory package
> with(numtheory, cfrac):
> cfrac(number);
```

$$\frac{1}{1 + \frac{1}{3}}$$

Maple also has the `padic` package to compute with *p-adic numbers*. Let us, for example, write the fraction  $\frac{3}{4}$  as a 3-adic number.

```
> with(padic): # load the p-adic number package
> # write the number as a 3-adic number in standard size
> evalp(number, 3);
```

$$3 + 2\cdot 3^2 + 2\cdot 3^4 + 2\cdot 3^6 + 2\cdot 3^8 + O(3^{10})$$

### 3.5 Attributes

The data structures in Maple that have constant length such as *name*, *list*, *set*, *procedure*, *function*, and *float* are allowed to have an extra field of information called the *attribute*. Any valid Maple expression is allowed as an attribute. The attribute cannot be seen nor modified by any Maple procedure, except for the two procedures **setattr** and **attributes**. **setattr** creates a new data structure with an attribute, and **attributes** returns the attributes of a structure. The Maple kernel recognizes certain attributes such as **protected** for names. We give some examples.

```
> setattr(A, "capital A");
A
> attributes(A);
"capital A"
> attributes(diff);
protected
> setattr(differentiate, protected);
differentiate
> differentiate := diff;
Error, attempting to assign to 'differentiate' which is protected
> S := setattr({2,3,5,7,11}, "first 5 primes");
S := {2, 3, 5, 7, 11}
> attributes(S);
"first 5 primes"
```

```

> S := setattribute( algebraicset(
>   [x*y*z^2-4, x^2+y^2+z^2-1, x-y-1],
>   [x, y, z] ), dimension=0, groebnerbasis=
>   [x-y-1, 2*y^2+z^2+2*y, 8+z^4] );
S := algebraicset([x y z^2 - 4, x^2 + y^2 + z^2 - 1, x - y - 1], [x, y, z])
> attributes(S);
dimension = 0, groebnerbasis = [x - y - 1, 2 y^2 + z^2 + 2 y, 8 + z^4]

```

## 3.6 Properties

You can associate properties to variables by the procedures **assume** and **additionally**. Maple tries to use the properties during calculations and does its best to deduce properties of expressions from given properties. Here we only give some introductory examples and describe some technicalities with respect to display and assignment. We assume that a variable with assumptions is distinguished by a trailing tilde in its name. This can be forced by the following command.

```
> interface(showassumed=1):
```

A deeper discussion of assumptions will follow in Chapter 13.

```

> (-1)^(m^2+n); # no assumptions
(-1)^(m^2+n)
> assume(m::odd, n::odd):
> [(-1)^m, (-1)^(m+n), cos(m*Pi)];
[(-1)^m, (-1)^(m+n), -1]
> simplify(%);
[-1, 1, -1]

```

Here, Maple “understands” all by itself that the cosine of an odd multiple of  $\pi$  is equal to  $-1$ . With **simplify**, Maple also uses the knowledge that the square of an odd number is odd again, and that the sum of two odd integers is even, from which it can draw correct conclusions about the powers with base  $-1$ .

Whether a property has been associated to a variable can be checked with the procedure **hasassumptions**; if yes, it can be found with **about**.

```

> hasassumptions(m);
true
> about(m);
Originally m, renamed m~:
is assumed to be: LinearProp(2,integer,1)

```

So, variables that have associated properties are renamed by adding a tilde. Let's check this for `m` and add another property to it by the procedure **additionally**.

```
> m;
m~
> additionally(m>0):
> about(m);
Originally m, renamed m~: is assumed to be:
AndProp(RealRange(Open(0),infinity),
LinearProp(2,integer,1))
```

So, indeed the knowledge of  $m > 0$  is added in the form of membership of the open interval  $(0, \infty)$ . But what about the name: is this the same as the previous one? The proof is in the eating of the pudding.

```
> m-%;
m~ - m~
```

Surprise! The secret behind it is that Maple generates a local name at each assumption. This has some consequences with respect to assignments.

```
> m := 3:
> about(m);
3:
All numeric values are properties as well as objects.
Their location in the property lattice is obvious,
in this case integer.

> about(m);
odd_number:
nothing known about this object
```

So, assigning a variable removes all properties known about the variable.

```
> assume(n::integer):
> expr := cos(n*Pi);
expr := (-1)^n~
> n := 2:
> expr;
(-1)^n~
```

What happened? `n` is assigned the value 2, which is indeed an integer. This assignment removes the previous association with the variable `n`. However, in the expression `expr` we still refer to the old variable, which was renamed `n~`. One solution: use **assign** as this will evaluate its arguments. If necessary, use **eval** to control the depth of evaluation. An example:

```
> assume(N::integer, M::integer):
> expr := N~ + M~;
expr := N~ + M~
```

```

> assign(N=2):
> expr;

$$2 + M^{\sim}$$

> assign(eval(N,1), 3);
> expr;

$$3 + M^{\sim}$$


```

Another solution is to first remove the assumptions on variables inside the expression with which one wants to work further on. In our example, we can remove the assumption on  $M$  inside `expr` as follows:

```

> expr := eval(expr, M='M');
expr := 3 + M
> M := 1: expr;

$$4$$

> M := 2: expr;

$$5$$


```

We end this session with three tricks to remove the tilde in the output of a variable with associated properties on the screen. The first trick uses the `alias` procedure.

```

> assume(p>0):
> p;

$$p^{\sim}$$

> alias('p'=p):
> p;

$$p$$

> about(p);

Originally p, renamed p^{\sim}:
is assumed to be: RealRange(Open(0),infinity)

```

Drawback of this method is that you must do this for every variable with assumptions. Another way of getting rid of the tilde is provided in Maple through the interface variable `showassumed`. It may have the values 0, 1, or 2: a setting of 0 hides the display of assumptions; a setting of 1 causes variables with assumptions to be displayed with a trailing tilde (this is the default setting); and a setting of 2 causes a list of assumed variables to be displayed at the end of the expressions. But be warned: it is only an optical illusion and the `Export As LaTeX` menu option will still print a tilde.

```

> interface(showassumed=0);
> assume(q>0);
> q;

$$q$$


```

```

> about(q);
Originally q, renamed q~:
  is assumed to be: RealRange(Open(0),infinity)

> interface(showassumed=2);
> assume(r>0);
> r;
r

> about(r);

Originally r, renamed r~:
  is assumed to be: RealRange(Open(0),infinity)

```

An even better way of getting rid of the tilde is to overrule Maple. First we look at the code of Maple that is responsible for the tilde.

```

> interface(showassumed=1); # reset display of assumptions
> interface(verboseproc=3); # make Maple communicative
> print('property/Rename'); # print responsible routine

proc(nm)
local tn, tmp, r, nam, t1;
global 'property/object';
.....
t1 := subs('DUMMY2' = "||nam||~", proc(nam)
  local DUMMY2;
  global 'property/OrigName';
  'property/OrigName' DUMMY2 := nam; nam := DUMMY2
  end proc);
nm = t1(nam)
end proc

```

We have omitted most of the source code because it is the same as in the code below, in which the tilde does not appear. After the dots appears the line of code that we change.

```

> 'property/Rename' := proc(nm)
>   local tn, tmp, r, nam, t1;
>   global 'property/object';
>   if type(nm, indexed) then
>     tn := op(0, nm);
>     if assigned('property/OrigName'[tn]) then
>       if assigned('property/object'[tn]) then
>         tmp := eval('property/object'[tn]);
>         r := 'property/Rename'(tn);
>         'property/object'[rhs(r)] := copy(tmp);
>         return r
>       else return tn = 'property/OrigName'[tn]
>     end if

```

```

>      else return 'property/Rename'(tn)
>      end if
> end if;
> if nargs = 1 and assigned('property/OrigName'[nm])
> then nam := eval('property/OrigName'[nm],1)
> else nam := nm
> end if;
> t1 := subs((‘DUMMY2’) = nam,
> proc(nam) local DUMMY2;
>     global ‘property/OrigName’;
>     ‘property/OrigName’[DUMMY2] := nam;
>     nam := DUMMY2
> end proc);
> nm = t1(nam)
> end proc:
> assume(s>0): s; # no tilde!

```

*s*

```

> about(s);

Originally s, renamed s:
is assumed to be: RealRange(Open(0),infinity)

```

## 3.7 Exercises

1. What is the difference in the use of the variables **a** and **b** in

```
> a := 'a'; b := 'b'; a := b; b := 10;
```

and

```
> a := 'a'; b := 'b'; b := 10; a := b;
```

Can you add one extra assignment at the end of the sentence below, so that the total effect of carrying out the instructions on this line is the same as the result of entering the first of the above sentences?

```
> a := 'a'; b := 'b'; b := 10;
```

2. Explain in detail what happens when you enter the following commands in sequence.

```
> p := 'p'; q := 'q';
> 7 * 5;
> p = 10 ;
> q : 3 ;
> %% ; %% ; %% ;
> p ; q ;
```

3. Explain the different result of the following Maple commands:

**x\*y;**, **x.y;**, **x||y;**, and **x\y;**.

4. Describe in detail the difference between the two Maple sessions below (e.g., by drawing pictures of the internal data structures).

```
> a := b;
> b := 3;
> a;
> b := 4;
> a;
```

and

```
> b := 3;
> a := b;
> a;
> b := 4;
> a;
```

5. Predict the results of entering the following commands and check your answers with Maple.

```
> X1 := 5; j := 1;
> X || j;
> 'X || j';
> 'X' || j;
> 'X || j';
```

6. Suppose that the variables `v1`, `v2`, and `v3` are bound. Why can't you unassign these variables by the following repetition statement (or do-loop)?

```
> for i to 3 do v||i := 'v||i' end do:
```

Find two ways for having the intended unassignment done in a loop?

7. Explain what goes wrong when you enter the following commands.

```
> gcd(x^2-1, x-1, x);
> x;
```

8. Carry out the following instructions, predict the Maple results and compare these with the actual answers of the system.

```
> i := 3: x := 4:
> sum(x^i, i=1..5);
> sum(x^i, 'i'=1..5);
> sum('x^i', 'i'=1..5);
> sum('x'^i, 'i'=1..5);
> sum(x^i, 'i'=1..5);
> sum('x'^i, 'i'=1..5);
> sum('x^i', 'i'=1..5);
> 'sum(x^i, i=1..5)';
```

9. Successively transform the expression  $a + b + c$  into the product  $a * b * c$ , and the list  $[a, b, c]$ .

10. Use Maple to find the continued fraction expansion of  $\sqrt{2}$ ,  $e - 1$ , and the golden ratio  $\frac{1 + \sqrt{5}}{2}$ , respectively.

# 4

## Getting Around with Maple

In this chapter we shall describe in detail how to get around with Maple: how Maple handles input and output, how to change Maple to your own taste (prompt, width of printout, labeling, etc.), how to edit inputs, how to read and write files, how to get more information about usage of computer resources, how to trace computations, and how to get user-information about chosen techniques or algorithms. Formatted I/O and code generation are examples of interaction between Maple and programming or typesetting languages. Moreover, the setup of the Maple library (standard library, packages, and share library of users' contributions) is explained.

Some of the topics in this chapter depend on the user interface and/or operating system: we shall assume that the worksheet interface is used on a PC-compatible machine. Occasionally we shall describe how things work in the text-based user interface or on non-PC platforms.

### 4.1 Maple Input and Output

Maple can react in several ways when you enter an instruction:

- The system understands the command and returns
  - ▷ either the result in one or more lines followed by a new prompt,
  - ▷ or a new prompt at once, indicating that Maple is waiting for another instruction.

In the former case the previous input line was closed by a semicolon; the latter reaction occurs when the input line was closed by a colon. When a colon is used Maple performs the calculation quietly. Later in this section we shall encounter some exceptions to this general rule.

- Your instruction is not finished yet, e.g., because you simply forgot to enter a semicolon or a colon to separate commands. In this case too, Maple is short in its response and only returns a new prompt. You may still finish your instruction.

```
> Digits := 25: evalf(
>     log(cos(Pi/12)) )
> ;
-0.03466823209753695510470895
```

- You entered an instruction that Maple does not understand and you are punished with a *syntax error* message; In the text-based interface, Maple points with the *caret* (^) to the place where the syntax error appeared. In a worksheet, a blinking cursor appears at the location where the syntax error was recognized by Maple. The only thing that you can do is to correct the instruction line and re-enter what you hope is a correct instruction.

```
> x*
> *3;
Error, `*` unexpected
> x**
> 3;
```

$$x^3$$

When you use the worksheet interface, you can place the cursor at any place to correct input (in most cases, to add brackets, to insert mathematical operators like \*, etc.) and re-execute it. When the *replace mode* option has been turned on, the newly generated output replaces the previous one. The text-based user interface of Maple has a command line editor built in which allows you to edit one hundred previous input lines.

- Although the command is syntactically correct, Maple refuses to carry out the instruction.

```
> read "a_non_existing_file";
Error, unable to read 'a_non_existing_file'
> 123456789 ^ 987654321;
Error, numeric exception: overflow
```

After an error message, Maple is usually ready for new input; and it certainly is after you have entered an empty statement in the form of a single semicolon. This does not apply if you have started a name with a left quote and forgotten about the closing left quote. In this case you must first finish the name with a left quote before Maple listens to you again.

```
> 'This is a long name;
```

**Warning, string contains newline character. Close strings with ' quote.**

```
> 2+3; @%!
```

**Warning, string contains newline character. Close strings with ' quote.**

```
> that contains funny characters';
```

*This is a long name;\*

*2 + 3; @%!\*

*that contains funny characters*

Here, the backslash is used as continuation character.

- By using the *sharp symbol*, #, you entered a comment. Maple does not echo the text after the sharp symbol.

```
> # this is a comment
> a := 2 # a := two ;
> # semicolon on previous line is part of a comment!
> :
> a;
```

2

- Although you change conditions under which a computation is carried out, Maple does not take this into account but merely picks up a previous result from a look-up table. An example, in which expansion of trigonometric formulas is under user control:

```
> expand(expandoff()): # enable use of expandoff/on
> expandoff(sin): # turn expansion of sines off
> expand(sin(x+y)+cos(x+y));
```

$$\sin(x + y) + \cos(x) \cos(y) - \sin(x) \sin(y)$$

So far, so good. But let us now turn expansion of sines on again.

```
> expandon(sin): # turn expansion of sines on
> expand(sin(x+y)+cos(x+y));
```

$$\sin(x + y) + \cos(x) \cos(y) - \sin(x) \sin(y)$$

Surprise! It does not work immediately as **expand** remembers its previously computed results. You must explicitly **forget** previous results to get the work done.

```
> forget(expand);
> expand(sin(x+y)+cos(x+y));
sin(x) cos(y) + cos(x) sin(y) + cos(x) cos(y) - sin(x) sin(y)
```

So, Maple's reaction to your input depends upon the way you finish your command and upon the history of the session. You have more influence on the output. First, there is the variable `printlevel` whose default value is equal to one. When you assign a negative value to this variable, then no results are shown irrespective of whether a command is finished with a semicolon or colon. On the other hand, when you set a higher value to `printlevel` Maple tells you more of what it is doing. Values in the range from two to four are for information such as explanatory remarks about the chosen algorithm or information about parameters, local variables, and the precise statement being executed if a run-time error occurs. The higher the value of `printlevel`, the more information you get. High values of `printlevel`, say in the hundreds or thousands, are not unusual when you want to debug a Maple program, even though Maple contains a debugger (`enter ?debugger` for more information). If you only want to see what technique or algorithm is chosen, then you should use the `userinfo`-facility, which can be invoked by the assignment `infolevel[function] := level` for the requested *function* or by the assignment `infolevel[all] := level` for all functions. User information is available at several levels as shown in Table 4.1.

Level	Purpose
1	all necessary information
2, 3	general information, including technique or algorithm being used
4, 5	detailed information about how the problem is being solved

Table 4.1. `userinfo`-facility.

The following examples clarify the use of `printlevel` and `infolevel`.

```
> printlevel := 1: # default: only error message
> dsolve(diff(y(x),x)=0, y);

Error, (in ODEtools/info) found wrong extra arguments: {y}

> printlevel := 2: # display routine that goes wrong
> dsolve(diff(y(x),x)=0, y);

dsolve called with arguments: diff(y(x),x) = 0, y
#(dsolve,28): 'ODEtools/odsolve'(args)

Error, (in ODEtools/info) found wrong extra arguments: {y}
```

```

locals defined as: Y = y(x), derivs = {diff(y(x),x)}, ode =
diff(y(x),x), f = f, y = y, x = x, diff_ord = 1, hint = (NULL),
WAY = [], l_args = {y}, methods = {}, zz = [false, false, false],
algorithms = [2, 3, 4, 5, 6, 7, pdsolve, patterns, abaco1,
exp_sym, abaco2, formal, all], Yn = Yn, i = i

> printlevel := 3: # give a summary of calling routines
> dsolve(diff(y(x),x)=0, y);

dsolve called with arguments: diff(y(x),x) = 0, y
#(dsolve,28): 'ODEtools/odsolve'(args)

ODEtools/odsolve called with arguments: diff(y(x),x) = 0, y
#('ODEtools/odsolve',60):
'ODEtools/info'(ARGS,ode,'FAIL',y,x,diff_ord,hint,WAY,extra,
int_scheme,methods);

ODEtools/info called with arguments: [diff(y(x),x) = 0, y], ode,
FAIL, y, x, diff_ord, hint, WAY, extra, int_scheme, methods
#('ODEtools/info',146): error "found wrong extra arguments: %1",
l_args

Error, (in ODEtools/info) found wrong extra arguments: {y}

locals defined as: Y = y(x), derivs = {diff(y(x),x)}, ode =
diff(y(x),x), f = f, y = y, x = x, diff_ord = 1, hint = (NULL),
WAY = [], l_args = {y}, methods = {}, zz = [false, false, false],
algorithms = [2, 3, 4, 5, 6, 7, pdsolve, patterns, abaco1,
exp_sym, abaco2,formal, all], Yn = Yn, i = i

```

When `printlevel` has been assigned the value 2, Maple is not much more communicative than usual, except when an error occurs: in that case it will inform you about what function is called and what values arguments and local variables have. These data may give you some insight in why things do not go as expected. When execution errors are encountered and `printlevel > 2` a summary of calling routines is also provided.

```

> printlevel := 1: # reset printlevel
> infolevel[integrate] := 1: # raise infolevel
> integrate(1/(sin(x)^2+1), x=0..Pi);

int/indef1: first-stage indefinite integration
int/indef2: second-stage indefinite integration
int/trigon: case of integrand containing trigs
int/trigon: trying a tan half angle substitution.
int/ratpoly: rational function integration
int/ratpoly: rational function integration
int/trigon: tan half angle substitution successful.

```

$$\frac{\sqrt{2} \pi}{2}$$

If you still want to see a result on the terminal screen while `printlevel` has been assigned a negative value, then you must say so. Use either one of the procedures `print` or `lprint`. The difference is in the display of results:

**print** yields a so-called two-dimensional layout, which resembles common mathematical notation as much as possible. **lprint** (linear **print**) displays results in one-dimensional, left-adjusted format. An example:

```

> printlevel := -1: # Maple becomes silent
> sols := solve(a*x^2+b*x+c, x);
> print(sols); # 2-dimensional output

$$\frac{-b + \sqrt{b^2 - 4ac}}{2a}, \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

> lprint(sols); # left-adjusted, 1-dimensional output
1/2/a*(-b+(b^2-4*a*c)^(1/2)), 1/2/a*(-b-(b^2-4*a*c)^(1/2))
> printlevel := 11: # Maple gives more details
> integrate(1/(x^5+1), x=0..infinity);

{--> enter int, args = 1/(x^5+1), x = 0 .. infinity
{--> enter int/int, args = [1/(x^5+1), x = 0 .. infinity], 10,
_EnvCauchyPrincipalValue
answer := [ $\frac{1}{x^5 + 1}$ , x, 0,  $\infty$ , [formula]]

$$\frac{1}{5} \frac{\pi}{\sin(\frac{1}{5}\pi)}$$

<-- exit int/int (now in int) = 1/5*Pi/sin(1/5*Pi)}

answer :=  $\frac{1}{5} \frac{\pi}{\sin(\frac{1}{5}\pi)}$ 

$$\frac{1}{5} \frac{\pi}{\sin(\frac{1}{5}\pi)}$$

<-- exit int (now at top level) = 1/5*Pi/sin(1/5*Pi)


$$\frac{1}{5} \frac{\pi}{\sin(\frac{1}{5}\pi)}$$

> printlevel := 1: # reset printlevel to default.

```

By the way, you cannot refer to the result of the procedures **print** and **lprint** by the ditto operators or in any other way, because both procedures work via side effect as display of formulae but in reality evaluate to the special name **NULL**. This special name is used in Maple to indicate an empty sequence of expressions or an empty statement. When you use the ditto operator to refer to a previous result, **NULL** is skipped. So, note the differences in the following commands.

```

> Example := 11/12, evalf(exp(-10)), NULL, (1-x)/(1+x);
Example :=  $\frac{11}{12}, 0.00004539992976, \frac{1-x}{1+x}$ 

```

```

> lprint(Example); # linear display
11/12, 0.4539992976e-4, (1-x)/(1+x)
> %; # reference to 2-dimensional display
      11
      --, 0.00004539992976,  $\frac{1-x}{1+x}$ 
> interface(prettyprint=0);
> %; # linear display
11/12, 0.4539992976e-4, (1-x)/(1+x)

```

The last two instructions show that you can set the value of the interface variable `prettyprint` to 0 so that henceforth Maple output is left-adjusted and in one-dimensional format. In §4.7 we shall see what other interface options you have in Maple to change the system to your own taste.

## 4.2 The Maple Library

The Maple library consists of four parts:

- the *standard library*,
- the *update library*,
- *packages*, and
- optionally the *share library* or *user-contributed libraries*.

The size of the manufacturer's Maple library, the share library exclusive, is about fifty megabytes, but the system loads only those functions into main memory that it actually needs.

Whenever you enter an instruction that uses a procedure from the *standard library* or the *update library* that has not been used before in the Maple session, the system is smart enough to load the requested procedure from the external memory into main memory itself and to carry out the computation. The following session illustrates this.

```

> f := exp(a*z)/(1+exp(z)); # some formula
      
$$f := \frac{e^{az}}{1 + e^z}$$

> member('residue', {anames('procedure')});
      false

```

So, the procedure **residue** for computing the residue of a complex function at a pole has not yet been loaded into main memory. Maple leaves it this way and waits for the moment that you call the function for the first time. Then it loads the code from the library.

```
> residue(f, z=Pi*I);
-e^(I a π)
> member('residue', {anames('procedure')});
true
```

There are also packages for more specialized purposes in Maple, e.g., the **LinearAlgebra** package for matrix-vector computations. To use a function from this package you have to mention the name of the package as well. For example, to compute a determinant of a given matrix  $A$  you enter

```
> A := <<1|2>,<3|4>>; # a matrix
A := 
$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

> LinearAlgebra[Determinant](A);
-2
```

Here, we have used the table-based method of calling a procedure from a package. However, new packages such as the **LinearAlgebra** package are implemented as so-called modules. The `:-` operator allows you to access an exported object from a module. So, an alternative way of computing the determinant of  $A$  is

```
> LinearAlgebra:-Determinant(A);
-2
```

There are four ways to avoid having to use such long names:

- Use the procedure **alias** to define a synonym.

```
> alias(det=LinearAlgebra:-Determinant):
```

```
> det(A);
```

```
-2
```

```
> det(A, method=modular[3]);
```

```
1
```

```
> 'LinearAlgebra:-Determinant'(A);
```

$$\det\left(\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}\right)$$

To remove the alias definition of **det** enter

```
> alias(det=det):
```

- Use the procedure **macro** to define an abbreviation.

```
> macro(det=LinearAlgebra:-Determinant):
```

```
> det(A);
```

```
-2
```

The difference between **macro** and **alias** is that the macro facility is a simple abbreviation mechanism working on the input side,

```
> 'det(A)';
```

$$\text{LinearAlgebra} : -\text{Determinant}\left(\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}\right)$$

whereas the alias facility affects both input and output (see also the examples in §2.5). To remove the macro definition of **det** enter

```
> macro(det=det):
```

- Explicitly load the **Determinant** procedure from the **LinearAlgebra** package.

```
> with(LinearAlgebra, Determinant);
```

[Determinant]

```
> Determinant(A);;
```

-2

Note that you cannot use abbreviated names for other procedures in the **LinearAlgebra** package than **Determinant**.

```
> Rank(A);
```

$$\text{Rank}\left(\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}\right)$$

- Tell Maple that you want to use the complete package.

If the package has been implemented as a module, then you can evaluate expressions using the module in the following style.

```
> use LinearAlgebra in
>   Rank(A);
>   Trace(A)
> end use;
```

2

5

You can also load the whole package via the procedure **with**.

```
> with(LinearAlgebra);
```

[Add, Adjoint, BackwardSubstitute, BandMatrix, Basis,  
BezoutMatrix, BidiagonalForm, BilinearForm,  
.....  
VectorAngle, VectorMatrixMultiply, VectorNorm,  
VectorScalarMultiply, ZeroMatrix, ZeroVector, Zip]

The result of invoking **with(LinearAlgebra)** is that a group of names (such as **Determinant**, **Rank**, and **MatrixNorm**) is defined to point to the same procedures as their corresponding package style

names (**LinearAlgebra[Determinant]**, **LinearAlgebra[Rank]**, and **LinearAlgebra[MatrixNorm]**, respectively). The complete code of all procedures is *not* loaded from the library.

```
> MatrixNorm(A);
```

7

Only at this point is the code for this particular procedure is loaded and executed for the given arguments.

Enter **?index,packages** to get a complete list of packages that are available in the Maple library. Packages can contain subpackages: the **LinearAlgebra** is an example of this set-up; it contains the subpackage **Modular**. You can access procedures from a subpackage in long or short format as the following example shows.

```
> LinearAlgebra:-Modular:-Mod(3, A, integer);
```

$$\begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix}$$

```
> LinearAlgebra[Modular][Mod](3, A, float[8]);
```

$$\begin{bmatrix} 1. & 2. \\ 0. & 1. \end{bmatrix}$$

```
> with(LinearAlgebra[Modular]):
```

**Warning, these names have been rebound:** `Adjoint`, `BackwardSubstitute`, `Basis`, `Determinant`, `ForwardSubstitute`, `LUDecomposition`, `Multiply`, `Transpose`

```
> Determinant(3, A);
```

1

The **stats** package is also an example of dividing a package into subpackage, but it uses the table-based implementation of packages. All package functions are grouped in the seven subpackages listed in Table 4.2.

Subpackage	Purpose
<code>anova</code>	Analysis of variance
<code>describe</code>	Data analysis functions
<code>fit</code>	Linear Regression
<code>random</code>	Random number generators
<code>statevalf</code>	Numerical evaluation of distributions
<code>statplots</code>	Statistical plotting functions
<code>transform</code>	Data manipulation functions

Table 4.2. Subpackages of **stats** package.

For example, computing the mean value of a list of numerical values can be done with the **mean** procedure from the subpackage **describe**. This procedure can be activated in several ways.

```
> data := [seq(i,i=1..6)];
      data := [1, 2, 3, 4, 5, 6]
> stats[describe,mean](data); # direct call
      7
      -
      2
> with(stats): # load stats package
> describe[mean](data); # semidirect call
      7
      -
      2
> with(describe): # load subpackage
> mean(data); # call after loading subpackage
      7
      -
      2
```

A large collection of user-created materials is available at the *Maple Application Center* ([www.mapleapps.com](http://www.mapleapps.com)). Many contributions to the good old *share library*, which is not shipped with the Maple software anymore, have been updated to a recent version of Maple and have been added to the Maple Application Center. The original share library is still accessible using the *Sharetool* package. For installation, we refer to the webpage [www.mapleapps.com/packages/whathappenedtoshare.html](http://www.mapleapps.com/packages/whathappenedtoshare.html) and references therein. In the example below, we assume that the share library has been properly installed so that you can read each procedure and load each package from the share library as if it were a regular Maple procedure or package. For example, you can load Dongming Wang's implementation of Ritt-Wu's characteristic sets method [228] and use it to solve a system of polynomial equations.

```
> readshare(charsets, algebra):
> polys := [x^2 - 2*x*z + 5, x*y^2 + y*z^3, 3*y^2 - 8*z^3];
      polys := [x^2 - 2 x z + 5, x y^2 + y z^3, 3 y^2 - 8 z^3]
> vars := [x, y, z]:
> _EnvExplicit := false:
> sols := csolve(polys, vars);

sols := {y = - $\frac{8}{3}$  %1, z =  $\frac{1}{2} \frac{\%1^2 + 5}{\%1}$ , x = %1},
{z = 0, y = 0, x = RootOf(_Z^2 + 5, label = _L20)},
{z = 0, y = 0, x = RootOf(_Z^2 + 5, label = _L28)}
%1 := RootOf(3 _Z^6 - 64 _Z^5 + 45 _Z^4 + 225 _Z^2 + 375, label = _L21)
```

```
> evalf[3](map(allvalues,{%}));
```

$$\begin{aligned}&\{y = 2.45 + 2.94I, z = -1.58 + 0.790I, x = -0.919 - 1.10I\}, \\&\{x = -0.919 + 1.10I, z = -1.58 - 0.790I, y = 2.45 - 2.94I\}, \\&\{x = 0.258 + 1.17I, z = 0.580 - 1.46I, y = -0.689 - 3.12I\}, \\&\{y = -0.689 + 3.12I, x = 0.258 - 1.17I, z = 0.580 + 1.46I\}, \\&\{x = 2.24I, z = 0., y = 0.\}, \{x = -2.24I, z = 0., y = 0.\}, \\&\{y = -5.50, z = 2.24, x = 2.06\}, \{z = 10.4, x = 20.6, y = -55.0\}\end{aligned}$$

## 4.3 Reading and Writing Files

First of all we distinguish between

- worksheet files;
- files containing Maple code in a format that can only be understood by the computer algebra system;
- Maple repositories;
- files containing Maple code in a user-readable and user-understandable format, i.e., code that has been written in the Maple programming language itself;
- files containing results of Maple calculations, which for their part cannot be used as input for further computations;
- files containing formatted input/output, which are created by or can be read by other applications.

The latter category of files will be dealt with in the next sections. Here, we shall concentrate on only the first five types of files.

Let us first look at worksheet files, which often have the extension `.mws` (Maple WorkSheet). These are ASCII files that contain the input, output, graphics, and text cells of worksheets together with style information. They can be loaded in a new Maple session and commands can be recomputed.

Next, we look at the fifth category of files, i.e., files used for display or storage of computational results. Normally, the result of a Maple command that has been terminated by a semicolon is embedded in the worksheet or shown on the terminal screen. However, you can also transfer the output to a file so that you can read through the results at your leisure or print the results on paper. For example, to transfer all results in a Maple session to the file `outputfile` it suffices to enter

```
> writeto("outputfile");
```

Henceforth, all Maple output is redirected to the file instead of to the worksheet or terminal screen. The prompt “>”, the given instructions, and the run-time messages are also written into the file. Suppose that you enter the commands

```
> number := 90;
> polynomial := x^2 + 81;
> subs(x=number, polynomial);
> writeto(terminal);
```

Everything from the last command is shown on the terminal screen again.

```
> %;
8181
```

The file **outputfile** looks as follows (in case you use the text-based interface, you must turn off printing of input statements and bytes-used message via **interface(echo=0)**).

```
> number := 90;
number := 90

> polynomial := x^2 + 81;
          2
polynomial := x  + 81

> subs(x=number, polynomial);
8181

> writeto(terminal);
```

Keep in mind that **writeto** overwrites an existing file. If you want to avoid this and just want to append output, then you should use the procedure **appendto**. As was remarked before, you cannot do very much else with the output file but reading and printing. You cannot use it as input for a new Maple session without further processing.

With the procedure **save** you can indeed save results of Maple calculations in such a way that they can be used again in a Maple session. You read data from such a file back into memory with the Maple procedure **read**. The data transfer into an output file can be done in two ways:

- in an internal format that can only be understood by Maple (when the filename ends with “.m”) or otherwise
- in a human-readable format or plain text format.

Let us look at an example.

```
> restart; # ensure a fresh start of Maple
> polynomial := x^2 + 2*x + 1; 'number four' := 4;
polynomial := x^2 + 2 x + 1
```

```

number four := 4
> save "datafile";
Error, must specify names to save
> save "datafile.m";
Error, must specify names to save

```

The last two instructions have been included to show that you must specify in the current version of Maple which variables you want to save.

```

> save polynomial, 'number four', "datafile";
> save polynomial, 'number four', "datafile.m";

```

Do not forget to use the quotes around the filename. If you work under Unix, you cannot use abbreviations like ~ (tilde) for your home directory, but must use full pathnames instead. Otherwise, reading and writing is always done relative to the directory from which you started the Maple session.

The files **datafile** and **datafile.m** both contain the variables **polynomial** and 'number four', and their values just before the data transfer. This can be the starting point for a new Maple session; you can read (and if desired change) the file **datafile** with an editor or look at it with a command like **type**, **cat** or **more**, depending on your operating system. Inside Maple you can supply such a command via **ssystem**. This command returns a list of two values: the first is an integer, the return code of the system, and the second is a Maple name with all the output. We shall only look at the second value.

```

> ssystem("more datafile") [2];
"polynomial := x^2+2*x+1;
'number four' := 4;
"

```

You see that readability leaves a lot to be desired because of the one-dimensional layout chosen by Maple and the fact that the result is represented as a string. The following procedure allows a platform-independent display of the real contents of a file.

```

> showfile := proc(filename::string)
>   local line;
>   do
>     line := readline(filename);
>     if line=0 then ''; break end if;
>     printf("%s\n", line)
>   end do
> end proc:
> showfile("datafile");

polynomial := x^2+2*x+1;
'number four' := 4;

```

The file `datafile.m` is clearly in internal Maple format:

```
> showfile("datafile.m");
M7R0
I+polynomial,(*$%"xG""#""F%F&F'F'6",
I,number~four%"F(
```

Let us read the file `datafile` in a new Maple session.

```
> read "datafile";
polynomial :=  $x^2 + 2x + 1$ 
number four := 4
> 'number four' * polynomial;
 $4x^2 + 8x + 4$ 
```

You see that the instructions in the file `datafile` are dealt with as if they were entered directly from the keyboard. Therefore, you can prepare in your favorite editor a file suitable for `read` by writing down Maple instructions one after another.

When you start Maple, the system always looks first for an initialization file that can be read in the above way. For example, under the Unix operating system, Maple searches first for a system-wide initialization file called `init` in the `src` subdirectory of the Maple library, and secondly searches for an initialization file `.mapleinit` in your home directory. Using this file you can automatically load library packages that you always want to use, reset interface variables (see the last section), and so on. On a PC, the Maple initialization file `maple.ini` is searched for in the `Maple 8/lib` directory or the current directory.

When all data have been read from a file by the Maple procedure `read`, Maple resumes the handling of instructions entered from the previous input stream; this may be from the Maple worksheet, the terminal screen, or from another file.

You see that all incoming data from the `read` instruction are redisplayed on the terminal screen. In some cases this is a handy reminder, but at others it is just too much information. You could have suppressed display of data by prepending to the user-readable file two lines in which you save the current value of `printlevel` and give it the new value `-1`, and by appending a line in which `printlevel` gets back its old value. But this is extra work. As you will see in the session below, echoing of input from a file does not happen when the file is in internal Maple format, i.e., a file with the name extension “`.m`.” However, the most important advantage of using internal Maple format is that reading and writing is done more efficiently than in user-readable format. The file `datafile.m` used below is not user-readable, but in internal Maple format. Nevertheless, the file is in ASCII characters, as you have seen, so that it can be transferred by email.

```

> restart;
> read "datafile.m";
> polynomial := factor(polynomial);


$$\text{polynomial} := (x + 1)^2$$


> squarenumber := subs(x='number four', polynomial);


$$\text{squarenumber} := 25$$


```

You may be tempted to save all variables by the following command.

```

> save anames(), "out.m";

Error, save can only save names

```

This does not work because of the special evaluation rules for the **save** command. But, with the help of the **codegen** package you could do the following:

```

> codegen[intrep2maple](Proc(Parameters(),
>   StatSeq(Save(ExprSeq(
>     op({anames()} minus{codegen}),
>     "out.m"))),
>   )),
> )():

```

The file **out.m** will have all currently assigned names and their values. They can be read in a new session.

```

> restart;
> read "out.m";
> polynomial, squarenumber;


$$(x + 1)^2, 25$$


> fourthpower := squarenumber^2;


$$\text{fourthpower} := 625$$


> codegen[intrep2maple](Proc(Parameters(),
>   StatSeq(Save(ExprSeq(
>     op({anames()} minus{codegen}),
>     "out.m"))),
>   )),
> )();
> restart;
> read "out.m";
> fourthpower;

```

625

Reading from and writing to a file always takes place with respect to the position where you were in the file structure when you launched Maple, unless you specify a full pathname. This is of course not very convenient for library functions. Therefore, when you invoke a procedure in a Maple session for the first time, the system searches its libraries for loading the definition. You can ask for the full pathname of the Maple libraries by

asking the value of the variable `libname` within a Maple session. The answer will look like

```
> libname;
"C:\Program Files\Maple 8/lib"
```

From there you can read for instance the file `mtaylor.m` by

```
> read cat(libname,"/",mtaylor,".m");
```

Of course it is cumbersome and error-prone to type such long names whenever you need something from the Maple library. Automatic loading of a procedure comes in handy. It would have sufficed to call the procedure for computing a multivariate Taylor series.

```
> mtaylor(sin(x+y), [x,y]);

$$\begin{aligned} &x + y - \frac{1}{6}x^3 - \frac{1}{2}yx^2 - \frac{1}{2}y^2x - \frac{1}{6}y^3 + \frac{1}{120}x^5 + \frac{1}{24}yx^4 + \frac{1}{12}y^2x^3 \\ &+ \frac{1}{12}y^3x^2 + \frac{1}{24}y^4x + \frac{1}{120}y^5 \\ > \text{sort}(\text{simplify}(%), \{z=x+y\}, [x,y,z])); \\ &\frac{1}{120}z^5 - \frac{1}{6}z^3 + z \\ > \text{subs}(z=x+y, %); \\ &\frac{1}{120}(x+y)^5 - \frac{1}{6}(x+y)^3 + x + y \end{aligned}$$

```

In Maple you can use more than one library, which allows users to override procedures in the standard library and to have private libraries. Not only procedures can be stored in an archive, but in fact any Maple expressions. This is why the term Maple repository is normally used. Below, we carry out the steps for creating a repository containing our own code. For further details, we refer to the help page about management of Maple repositories (enter `?repository,management`). Furthermore, Maple provides utility functions for manipulating libraries via the `LibraryTools` package. See the on-line help system for detailed information.

First, we create a new, empty directory in which to store our repository.

```
> restart;
> currentdir("C:/"); # make the C: the current directory
> mkdir("privatelib"); # make a new directory
```

Use the `create` command option of the `march` (`maple archive`) procedure to create an empty repository, specifying a suitable index size.

```
> march('create', "privatelib", 100):
```

Next, we define the Maple expression(s) that we want to save to the repository. In this case, we define the procedure `showfile` to display the contents of a file.

```
> showfile := proc(filename::string)
>   local line;
>   do
>     line := readline(filename);
>     if line=0 then ''; break end if;
>     printf("%s\n", line)
>   end do
> end proc:
```

We make sure that the procedure **savelib** will store contents in the private library. For this purpose, we assign the variable **savelibname** the name of our repository.

```
> savelibname := "C:/privatelib":
> savelib('showfile'):
```

The **list** command option of **march** can be used to verify the contents of the repository.

```
> march('list', "C:/privatelib");
[[“showfile.m”, [2002, 5, 10, 10, 33, 13], 1, 138]]
```

The only thing left to do is prepending the full name of the repository to **libname**. Below, we omit the full display of the procedure body.

```
> restart:
> libname := "C:/privatelib", libname:
> print(showfile);

proc(filename::string)
local line;
do
.....
end do
end proc
```

Table 4.3 summarizes the commands for reading and writing of Maple files.

<b>Command</b>	<b>Meaning</b>
<b>read "file"</b>	read Maple input from readable <i>file</i>
<b>read "file.m"</b>	read Maple input from internally formatted <i>file.m</i>
<b>save <i>varseq</i>, "file"</b>	save variables from the sequence <i>varseq</i> to readable <i>file</i>
<b>save <i>varseq</i>, "file.m"</b>	save variables from the sequence <i>varseq</i> to internally formatted <i>file.m</i>
<b>appendto("file")</b>	append all subsequent results to readable <i>file</i>
<b>writeto("file")</b>	write all subsequent results to readable <i>file</i>
<b>writeto(terminal)</b>	write all subsequent results to the screen

Table 4.3. Reading and writing of Maple files.

## 4.4 Importing and Exporting Numerical Data

### Importing Numerical Data

The procedures **readline** and **readdata** are utilities in Maple to read unbuffered (raw) data consisting of a sequence of ASCII characters (text) from a file or terminal screen. **readline** reads in one line from the specified file or terminal. **readdata** reads in numerical data arranged in columns from a file or terminal screen.

Consider a data file, say **numdata**, in the current directory and three lines long with the following tab-delimited numbers

```
1      .84      .54
2      .91      -.42
3      .14      -.99
```

On a PC, you can display the contents with the following user-defined command; in this section and the next one we shall assume that this procedure is always available.

```
> showfile := proc(filename::string)
>   local line;
>   do
>     line := readline(filename);
>     if line=0 then ``; break end if;
>     printf("%s\n", line)
>   end do
> end proc:
> showfile("numdata");

1      .84      .54
2      .91      -.42
3      .14      -.99
```

Below are a few examples of how this file or part of it can be read.

```
> readdata("numdata", 3); # read 3 columns of floats
[[1., 0.84, 0.54], [2., 0.91, -0.42], [3., 0.14, -0.99]]
> readdata("numdata", 2); # read 2 columns of floats
[[1., 0.84], [2., 0.91], [3., 0.14]]
> readdata("numdata"); # read 1st column of floats
[1., 2., 3.]
```

For **readdata**, the data must consist of integers or floats arranged in columns separated by white space. You can explicitly ask for reading integers

```
> readdata("numdata", integer);
[1, 2, 3]
```

or specify all data types.

```
> readdata("numdata", [integer,float,float]);
[[1, 0.84, 0.54], [2, 0.91, -0.42], [3, 0.14, -0.99]]
```

**readline** reads — nomen est omen — only one line. The first time when you call this command, the first line of a file is read.

```
> readline("numdata"); # read first line
"1\|t0.84\|t .54"
```

The second time, the second line is read, and so on. An empty line is read as an empty string ("").

```
> readline("numdata"); # read second line
"2\|t0.91\|t-.42"
> readline("numdata"); # read third line
"3\|t0.14\|t-.99"
```

When the end of file is encountered, **readline** returns the value 0 and further reading starts again at the beginning of the file.

```
> readline("numdata"); # end of file
0
> readline("numdata"); # read first line
"1\|t0.84\|t .54"
```

The last expression is of type *string*.

```
> whattype(%);
string
```

Data files frequently contain both numbers and text. These more sophisticated data files can be read only as formatted input, e.g., via the procedures **readline** and **fscanf**. We shall come back to this in the next section. Here, we only show how the last string can be decoded into a list of an integer and two floating-point numbers via **sscanf**.

```
> sscanf(%, "%d%f%f");
[1, 0.84, 0.54]
```

There exist two other general routines for importing numerical tabular data, viz., **importdata** from the **stats** package and **ImportMatrix**. The procedure **stats[importdata]** is similar to **readdata**, with the exception that it offers more possibilities for reading incomplete data, i.e., data with string entries, too. The procedure **ImportMatrix** reads tabular data as a matrix. It offers several options to deal with special formats like Matrix-Market, Matlab, or delimited format. In its simplest form, you only have to specify in the **ImportMatrix** command the filename where to read from.

```
> data := ImportMatrix("numdata");
```

$$data := \begin{bmatrix} 1 & 0.84 & 0.54 \\ 2 & 0.91 & -0.42 \\ 3 & 0.14 & -0.99 \end{bmatrix}$$

We export the tabular data in transposed order and as space-delimited lists to a file.

```
> ExportMatrix("outputfile", data, transpose=true,
> target=delimited, delimiter=" ");
```

32

32 bytes have been written into the file. We read its contents back to notice the difference.

```
> ImportMatrix("outputfile",
> source=delimited, delimiter=" ");
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 0.84 & 0.91 & 0.14 \\ 0.54 & -0.42 & -0.99 \end{bmatrix}$$

## Exporting Numerical Data

The procedure **ExportMatrix** is the opposite of **ImportMatrix**. The procedures **writeline** and **writedata** are other utilities in Maple to write raw data to a file or terminal screen. **writeline** writes the specified Maple expressions to the specified file, separated by newlines, and followed by a newline. **writedata** writes numerical data to a file or terminal screen.

We work again with the following data matrix.

```
> data := convert(data, matrix);
```

$$data := \begin{bmatrix} 1 & 0.84 & 0.54 \\ 2 & 0.91 & -0.42 \\ 3 & 0.14 & -0.99 \end{bmatrix}$$

The following command writes the numerical data as floating-point numbers on the terminal screen.

```
> writedata(terminal, data, float);
```

1	.84	.54
2	.91	-.42
3	.14	-.99

You can also write the numerical data to a file, say **outputfile**. You must close the file with the procedure **fclose**.

```
> writedata("outputfile", data, float);
> fclose("outputfile");
```

The file contents are as follows:

```
> showfile("outputfile");
```

```

1      .84      .54
2      .91      -.42
3      .14      -.99

```

You can also be more specific about the format of the data.

```

> writedata(terminal, data, [integer,float,float]);
1      .84      .54
2      .91      -.42
3      .14      -.99

```

**writedata** can also handle non-numeric data values, i.e., data items that are not integers, rationals, or floats. You can actually specify a Maple procedure to print a non-numeric data value in a format that you choose yourself. A typical example is to print an asterisk if a data value is not numeric.

```

> for i to 3 do data[i,1] := evaln(data[i,1]) end do;
> print(data);

$$\left[ \begin{array}{ccc} data_{1,1} & 0.84 & 0.54 \\ data_{2,1} & 0.91 & -0.42 \\ data_{3,1} & 0.14 & -0.99 \end{array} \right]$$

> writedata(terminal, data, float,
> proc(x) printf("*",x) end proc);
*
```

*	.84	.54
*	.91	-.42
*	.14	-.99

Thus, the general function call of **writedata** is

```
writedata("file", data, datatype, default)
```

Here, *file* is the name or the description of the file and *default* is a procedure to export non-numeric data. Use **writedata[APPEND]** instead of **writedata** when you want to add numerical data to an existing file without loosing previous data.

## 4.5 Low-level I/O

Maple allows you to read and write to general objects called *streams*. A stream is a source of input or output. Files, Unix pipes, and the terminal screen are examples of streams. In this section we shall give a brief description of basic facilities. More about I/O can be found in the Advanced Programming Guide [181]. Henceforth we shall assume that the procedure **showfile**, which was defined at the beginning of the previous section, can be used to display the contents of a file.

The basic low-level scheme for writing output to a stream is as follows. First, you open the stream for output, i.e., you inform Maple that you want to write output to a particular file or pipe. Having opened a stream you can write to it. Finally, you close the output stream.

Table 4.4 lists the kind of streams Maple distinguishes and the commands to open and close them.

Stream type	Meaning	Open/Close
STREAM	a buffered file	fopen, fclose
RAW	an unbuffered file	open, close
PIPE	a double-ended Unix pipe	pipe, close
PROCESS	a Unix pipe with one end attached to another process	popen, pclose
DIRECT	direct access to current (default) stream or top-level (terminal) I/O stream	not applicable

Table 4.4. Maple streams: opening, and closing.

When you open a stream, you must inform Maple how you intend to use it: **READ**, **WRITE**, or **APPEND**. Maple distinguishes between text and binary files: the types are **TEXT** for a stream of characters, and **BINARY** for streams of bytes.

A simple example:

```
> fd := fopen("outputfile", WRITE, TEXT);
          fd := 0
```

`fd` is the file descriptor, `outputfile` is the file name, `WRITE` is the file mode, and `TEXT` is the file type. Let us write two lines to the file and close it hereafter.

```
> writeline(fd, "this is some text");
          18
> writeline(fd, "this is some more text");
          23
> fclose(fd);
```

The output of `writeline` is the number of characters written to the file in the statement. The file `outputfile` indeed consists of two lines.

```
> showfile("outputfile");
          this is some text
          this is some more text
```

You can append a third line to this file in the following way.

```
> fd := fopen("outputfile", APPEND, TEXT):
> writeline(fd, "third line");
                                         11
> fclose(fd):
> showfile("outputfile");

this is some text
this is some more text
third line
```

An example of a two-sided Unix pipe (which does not work on a PC running MS Windows):

```
> with(process);
                                         [block, exec, fork, kill, launch, pclose, pipe, popen, wait]
```

When you want to use Unix pipes, you first have to load the **process** package for the utility functions. Let us open a two-sided pipe.

```
> pds := pipe(); # pipe descriptors
                                         pds := [0,1]
```

A list of two file descriptors is returned, the first of which is open for reading, and the second of which is open for writing. Pipes are opened with type **BINARY**. Let us write some text in binary format via the procedure **writebytes**.

```
> writebytes(pds[2], "This is a");
                                         9
> convert(" test", 'bytes');
                                         [32, 116, 101, 115, 116]
> writebytes(pds[2], %);
                                         5
```

You can read the text from the other end in binary format and convert it to an ASCII character string.

```
> readbytes(pds[1], 14);
                                         [84, 104, 105, 115, 32, 105, 115, 32, 97, 32, 116, 101, 115, 116]
> convert(%, 'bytes');
                                         "This is a test"
```

Finally, the pipe is closed.

```
> close(pds[1]):
> close(pds[2]):
```

Of course you can have more than one stream open at a time. The status of all open files can be asked for by entering **iostatus()**.

```
> restart;
> open("testFile", WRITE);
          0
> fopen("anotherFile", APPEND, TEXT);
          1
> process[popen]("rev", WRITE);
          2
> iostatus();
[3, 0, 7, [0, "testFile", RAW, FD = 6, WRITE, BINARY],
 [1, "anotherFile", STREAM, FP = 728220, READ, TEXT]
 [2, "rev", PROCESS, FP = 728240, WRITE, TEXT]]
```

The first three elements of the above list are the number of streams opened by the I/O library, the number of currently active nested read commands, and the the upper bound on the sum of both numbers. The additional elements describe the status of open streams.

Maple supports formatted input and output. This allows you to read data created by other applications and to write data for use in other applications in any way you want. The I/O procedures **printf**, **fprintf**, **nprintf**, **sprintf**, **scanf**, **fscanf**, and **sscanf** resemble their C equivalents. Their meanings are summarized in Table 4.5.

Command	Meaning
<b>fprintf</b> ( <i>stream, format, args</i> )	print the arguments <i>args</i> to <i>stream</i> based on the <i>format</i>
<b>nprintf</b> ( <i>format, args</i> )	write the arguments <i>args</i> into a Maple name based on the <i>format</i>
<b>printf</b> ( <i>format, args</i> )	print the arguments <i>args</i> to default output stream based on the <i>format</i>
<b>sprintf</b> ( <i>format, args</i> )	write the arguments <i>args</i> into a Maple string based on the <i>format</i>
<b>fscanf</b> ( <i>stream, format</i> )	read from <i>stream</i> based on the <i>format</i>
<b>scanf</b> ( <i>format</i> )	read from current input stream based on the <i>format</i>
<b>sscanf</b> ( <i>string, format</i> )	decode <i>string</i> based on the <i>format</i>

Table 4.5. Low-level formatted I/O routines.

So, the output procedure

**printf**(*format*, *arg*<sub>1</sub>, *arg*<sub>2</sub>, ...)

prints the arguments *arg*<sub>1</sub>, *arg*<sub>2</sub>, ..., under control of the conversion specification *format* to the default output stream. For example, to print the numerical approximation of  $\frac{5}{3}$  as a 15 digit number with 8 digits after the decimal point and 0 for the pad character you enter

```
> printf("%015.8f", 5/3);
```

```
000001.66666667
```

In general, the format specification is of the form

%[flags][width][.precision]code

The conversion specifications elements and their meanings are listed in Table 4.6 and 4.7.

Flag	Meaning
-	left adjustment within the field
0	use 0 as the pad character instead of a space (if padding is to the left of the converted value)
+	the result of a signed conversion will always start with a sign
space	a signed numeric value is output with either a leading – or a leading blank
#	if a number-sign flag is present, then an alternative output format is used
{}	a flag for parsing rtable objects see the help page ?rtable,printf

Table 4.6. Flags in **printf**.

Code	Printed as
a	Maple's lprint format
A	Maple's lprint format, but back quotes around names are omitted
c	single character
d	signed decimal number
e, E	signed decimal floating-point number in scientific notation
f	signed decimal floating-point number

*continued on next page*

<i>continued from previous page</i>	
Code	Printed as
g, G	signed decimal floating-point number in d, e, or f format, or in E format if G was specified, depending on the value to be printed
%	single percentage symbol (no conversion)
m	internal Maple format
o	unsigned octal number
p	machine address in human-readable format
q, Q	the same as a and A, respectively, but consumes all remaining arguments in Maple's lprint format
s	string
u	unsigned decimal number
x, X	unsigned hexadecimal number, using a,b,c,d,e,f, or A,B,C,D,E,F for 10, 11, ..., 15

Table 4.7. Conversion codes in **printf**.

The main differences with C's **printf** is that %a (for algebraic) will include any algebraic expression (with no line breaks) as printed by **lprint**, and that \* for field lengths are not handled. Below are some examples.

```
> printf("x=%d      y=%4.2f", 12, 34.3);
x=12      y=34.30
> printf("%a", 1/(x^2+1));
1/(x^2+1)
> printf("%c, %s", a, abc);
a, abc
> printf("%g, %g", 1/2, 1/2^16);
.5      , 1.525879e-05
> printf("%f", 10.^10);
1000000000.00000
> printf("%a", 10.^10);
.1000000000e11
```

Escape characters are also convenient when using the text-based user interface of Maple. For example, they allow you to add newline characters so that you don't need to use the **lprint()** command to get a new line. The complete list of character escape codes is listed in Table 4.8; precede them

by a backslash when you use them. A few examples will give you an idea of how to use them.

Escape Code	Meaning
a	bell ( <u>alert</u> )
b	<u>backspace</u>
e	<u>escape</u>
f	<u>formfeed</u>
n	<u>newline</u>
r	carriage <u>return</u>
t	horizontal <u>tabulate</u>
v	<u>vertical tabulate</u>
\	backslash
'	single quote
"	double quote
?	question mark

Table 4.8. Character escape codes.

```
>   printf("\n%s\n",
>         "string enclosed by newlines\nand a newline inside");

string enclosed by newlines
and a newline inside

> # in the textual interface only
> printf("\n%s\n", "backspacing\b\b\b\b\b\b\b\b\b\bxxxxx");

bacxxxxxxing

>   printf("\n%s\n%s\n", "x\ty\tnz", "w");

x      y
z
w
```

The procedure **sscanf**(*string, format*) is the inverse of **sprintf** and decodes the *string* according to the *format* into a list of Maple objects. The only difference to C's **sscanf** is that the Maple version returns a list of the scanned objects. Two examples:

```

> sscanf("x=123.45, y=6.7E-8, 9.10",
>        "x=%f, y=%g, %d.%d");
[123.45, 0.67 10-7, 9, 10]

> sscanf("f=x/(1+x^2)", "f=%a");
[x / (1 + x2)]
```

In general, the format specification is of the form

%[\*][width]code

The optional \* indicates that the object is to be scanned, but not returned as part of the result (i.e., it is discarded).

```
> sscanf("0-123", "%*d%d");
[ -123]
> sscanf("0-123", "%d%d");
[0, -123]
```

The width indicates the maximum number of characters to scan for this object. This can be used to scan one larger object as two or more smaller objects.

```
> sscanf("date=26062002", "date=%2d%2d%4d");
[26, 6, 2002]
```

The conversion specification elements and their meanings are given in Table 4.9. Keep in mind that the format specifications for rectangular tables of type `rtable` and for special formats like IEEE or XML have not been included; we refer for these formats to the on-line help pages.

Code	Converted into
a	unevaluated Maple expression
c	character
d, D, i	signed decimal number
e, f, g	signed decimal floating-point number
m	a Maple expression in internal format
n	number of characters read so far
s	string
o, O	unsigned octal number
p	a pointer value of a computer memory address
u, U	unsigned decimal number
x, X	unsigned hexadecimal number, using abcdef or ABCDEF for 10, 11, ...
%	single percent symbol
[...]	longest non-empty string of input characters from the set between brackets
[^...]	longest non-empty string of input characters <i>not</i> from the set between brackets

Table 4.9. Conversion codes in `fscanf`, `scanf`, and `sscanf`.

The `parse` command parses a string as a Maple statement. The option `statement` determines whether the full evaluation should take place or not.

```
> x := 1;
> parse("x + 1");
```

```

x + 1
> %;
2
> parse("x + 1", statement);
2

```

Together with the routines **filepos**, **feof**, **fflush**, and **fremove** for file information and control, the above low-level formatted I/O routines can be used as building blocks for input and output of data. A few examples will give you an idea of possibilities.

### Formatted Numerical Data

The first example is about exporting and importing numerical data in a specified format. Suppose that we have a matrix filled with randomly generated floating-point numbers in the range (-3,3) and that we want to write the matrix entries row-by-row into a file as 3 digit numbers with 2 digits after the decimal point (in scientific notation) separated by horizontal tabulates.

```

> data := matrix(4,3,
>      (i,j) -> stats[random,uniform[-3,3]]());
data := 
$$\begin{bmatrix} -0.435481985 & -1.073335840 & -0.938201558 \\ -0.154463138 & 0.350752314 & 1.480522983 \\ -2.807626668 & 1.337844731 & 0.625833683 \\ 1.473480224 & -1.441128284 & -1.139547077 \end{bmatrix}$$

> data := convert(data, listlist):
> fd := fopen("datafile", WRITE, TEXT):
> for row in data do
>   for item in row do
>     fprintf(fd, "%3.2e\t", item)
>   end do:
>   fprintf(fd, "\n"):
> end do:
> fclose(fd):

```

The contents of **datafile** are now:

```

> showfile("datafile");
-4.35e-01 -1.07e+00 -9.38e-01
-1.54e-01 3.51e-01 1.48e+00
-2.81e+00 1.34e+00 6.26e-01
1.47e+00 -1.44e+00 -1.14e+00

```

Let us import these data again but as two new lists of lists: the first list of lists contains the first two columns; the second list contains the last column of the original data.

```

> fd := fopen("datafile", READ, TEXT):
> line := table(): nline := 0:
> while not feof(fd) do
>   nline := nline+1:
>   line[nline] := fscanf(fd, "%e\t%e\t%*e")
> end do:
> newdata1 := [seq(line[k], k=1..nline-1)];

```

`newdata1 := [[-0.435, -1.07], [-0.154, 0.351], [-2.81, 1.34], [1.47, -1.44]]`

Or in matrix notation:

```
> setattribute(newdata1, matrix);
```

$$\begin{bmatrix} -0.435 & -1.07 \\ -0.154 & 0.351 \\ -2.81 & 1.34 \\ 1.47 & -1.44 \end{bmatrix}$$

Next, we return to the starting point of `datafile` and read the third column.

```

> filepos(fd, 0): # rewind file
> line := table(): nline := 0:
> while not feof(fd) do
>   nline := nline+1:
>   line[nline] := fscanf(fd, "%*e\t%*e\t%*e")
> end do:
> newdata2 := [seq(line[k], k=1..nline-1)];

newdata2 := [[-.938],[1.48],[.625],[-1.13]]

> fclose(fd):

```

## Mixed Numerical and Textual Data

An alternative way of importing data from a file is via `readline`. We illustrate this on data that are of both numerical and textual nature. First we generate the data and export them to a file.

```

> data := [seq([d||i, stats[random,uniform[0,1]]()]),
>          i=1..5)]:
> writedata("datafile", data, [string,float]);
> fclose("datafile");
> showfile("datafile");

d1 .42742
d2 .321111
d3 .343633
d4 .474256
d5 .558459

```

Next, we import the above data again.

```

> filepos("datafile", 0); # ensure starting point
> lines := table(): nline := 0:
> line := readline("datafile"):
> while line <> 0 do
>   nline := nline+1:
>   lines[nline] := sscanf(line, "%s%f"):
>   line := readline("datafile"):
> end do:
> newdata := [seq(lines[k], k=1..nline)];

newdata := [[“d1”, 0.42742], [“d2”, 0.321111], [“d3”, 0.343633],
[“d4”, 0.474256], [“d5”, 0.558459]]

```

### Nonuniform Data

Our final example of formatted I/O will be about importing nonuniform data. Many data files contain besides numerical data also headers that describe the file’s contents, creation time, author, and so on. Furthermore, data may not have the same format throughout the entire data set. The data set `agriculture` is extracted from a dataset about agriculture in the Netherlands obtained from Statistics Netherlands (URL: [www.cbs.nl](http://www.cbs.nl)). Its contents are

Title Livestock in Holland in Recent Years.  
Origin Statistics Netherlands (<http://www.cbs.nl>).

Livestock	unit	1985	1990	1995	2000
Cattle	mln	5.2	4.9	4.7	4.1
Pigs	mln	12.4	13.9	14.4	13.1
Sheep	mln	0.8	1.7	1.7	1.3
Chickens	mln	90	93	90	104
Horses	1000	62	70	100	118
Goats	1000	12	61	76	179

We import these data and process them.

```

> fd := fopen("agriculture", READ, TEXT):
> readline(fd); # read title
      “Title\tLivestock in Holland in Recent Years.”
> title := substring(% , 7..length(%));
      title := “Livestock in Holland in Recent Years.”
> readline(fd); # read origin
      “Origin\tStatistics Netherlands (http://www.cbs.nl)”
> origin := substring(% , 8..length(%));
      origin := “Statistics Netherlands (http://www.cbs.nl)”

```

```

> readline(fd): # read empty line
> fscanf(fd, "%s%s%d%d%d%d");
      ["Livestock", "unit", 1985, 1990, 1995, 2000]

> header := %:
> readline(fd): # read empty line
> livestock := table():
> while not feof(fd) do
>   line := fscanf(fd, "%s%a%f%f%f%f"):
>   if line <> [""] then
>     line := subs(mln=10^6, line):
>     fctr := line[2]:
>     livestock[line[1]] := map(z->fctr*z, line[3..6]):
>   end if
> end do:
> fclose(fd):
> livestock["Horses"], livestock["Sheep"]; # two examples

[62000., 70000., 100000., 118000.],
[800000.0, 0.17000000 107, 0.17000000 107, 0.13000000 107]

```

## 4.6 Code Generation

You can find the roots of a third degree polynomial with **solve**, Maple's general equation solver.

```

> polyeqn := x^3-a*x=1;
      polyeqn :=  $x^3 - a x = 1$ 
> sols := solve(polyeqn, x);

sols :=  $\frac{\sqrt[3]{\%1}}{6} + \frac{2 a}{\sqrt[3]{\%1}}$ ,
      -  $\frac{\sqrt[3]{\%1}}{12} - \frac{a}{\sqrt[3]{\%1}} + \frac{1}{2} I \sqrt{3} \left( \frac{\sqrt[3]{\%1}}{6} - \frac{2 a}{\sqrt[3]{\%1}} \right)$ ,
      -  $\frac{\sqrt[3]{\%1}}{12} - \frac{a}{\sqrt[3]{\%1}} - \frac{1}{2} I \sqrt{3} \left( \frac{\sqrt[3]{\%1}}{6} - \frac{2 a}{\sqrt[3]{\%1}} \right)$ 

\%1 :=  $108 + 12 \sqrt{-12 a^3 + 81}$ 

```

If you want to use such analytical Maple results in a FORTRAN program you can have Maple translate the formulae via the procedure **Fortran** from the **CodeGeneration** package. You cannot do this for all the solutions in the sequence simultaneously because the expression sequence data type does not exist in FORTRAN 77; you must translate each solution separately, or put the solutions in a vector or list.

```
> sol1 := sols[1]; # take 1st solution
sol1 := 
$$\frac{(108 + 12\sqrt{-12a^3 + 81})^{(1/3)}}{6} + \frac{2a}{(108 + 12\sqrt{-12a^3 + 81})^{(1/3)}}$$

```

When you add the option `optimize=true` in a call of the Maple procedure **Fortran** (reasonably) optimized FORTRAN 77 code is generated. When you want double precision FORTRAN code set the options `precision` and `functionprecision` to `double`. The `precision` option takes care of the translation of floating-point numbers, whereas the `functionprecision` option handles the translation of mathematical functions. The `declare` option is used to specify that the parameter *a* should be a float.

```
> with(CodeGeneration):
> Fortran(sol1, optimize=true, declare=[a::float],
> precision=double, functionprecision=double);

t1 = a ** 2
t5 = dsqrt(-0.12D2 * t1 * a + 0.81D2)
t8 = (0.108D3 + 0.12D2 * t5) ** (0.1D1 / 0.3D1)
t13 = t8 / 0.6D1 + 0.2D1 * a / t8
```

How well or how badly the code has been optimized can be checked with the procedures `cost` and `optimize` of the `codegen` package. This package is also for code generation, but it has been superseded by the `CodeGeneration` package. The user is advised to use the `CodeGeneration` package, whenever possible, for translation of Maple code to other languages like FORTRAN, C, and Java.

```
> # some abbreviations:
> macro(cnt=codegen[cost], opt=codegen[optimize]):
> cnt(sol1);

5 additions + 15 multiplications + 8 functions
```

*5 additions + 15 multiplications + 8 functions*

```
> opt(sol1);
```

*8 multiplications + 4 assignments + 4 functions + 3 additions + divisions*

Note that Maple only searches for common powers and products in terms of the internal representation. For example, the obvious code optimization for sums

$$(a + b + c)(b + c) \longrightarrow t \stackrel{\text{def}}{=} b + c; (a + t)t$$

is not found by Maple.

```
> opt((a+b+c)*(b+c));
```

$$t3 = (a + b + c)(b + c)$$

The option **tryhard** in the procedure **optimize** uses a different optimization algorithm. In general, the code optimization is improved, but it takes more computer resources. An example:

```
> opt(a*b+a*c);

$$t3 = a b + a c$$

> opt(a*b+a*c, tryhard);

$$t1 = (b + c) a$$

```

The procedure **C** in the **CodeGeneration** package generates ANSI C language code. An additional argument of the form **output = "file.c"** can be used to direct the output to the file *file.c*. By the way, the previously discussed **Fortran** routine also provides this storage facility.

```
> C(sol1, declare=[a::float], output="file.c"):
```

You can store more than one formula in a file.

```
> C(sol1, optimize=true, declare=[a::float],
>     output="file.c"):
```

The optimized C code for the first solution is appended to the end of the file as you can see from the contents below (Remark: we split the first line of this file so that it can be displayed in the current text width).

```
cg0 = pow(0.108e3 + 0.12e2*sqrt(-0.12e2*pow(a, 0.3e1) +
0.81e2), 0.1e1/0.3e1)/ 0.6e1 + 0.2e1*a*pow(0.108e3 +
0.12e2*sqrt(-0.12e2*pow(a, 0.3e1) + 0.81e2), -0.1e1/0.3e1);
t1 = a * a;
t5 = sqrt(-0.12e2 * t1 * a + 0.81e2);
t8 = pow(0.108e3 + 0.12e2 * t5, 0.1e1 / 0.3e1);
t13 = t8 / 0.6e1 + 0.2e1 * a / t8;
```

The language translation routines can also be applied to named arrays, lists of equations, and complete Maple procedures. The few examples below also demonstrate other options that can be used in the code generation routines.

A list of Maple expression translates to other languages like a computation sequence. Below is an example of translation a list of two Maple expression into the Java language.

```
> Java([x=Pi*ln(t), y=x^2-sqrt(gamma)]);
x = Math.Pi * Math.log(t);
y = x * x - Math.sqrt(0.5772156649e0);
```

Another example, but with code optimization before creating the computation sequence in the Java language:

```
> Java([p=x, q=x^2, r=p+q], optimize=true);
p = x;
q = x * x;
r = x + q;
```

The latter can be understood as part of the Java routine generated from the following Maple procedure.

```
> f := proc(x) global p,q,r: p:=x: q:=x^2: r:=p+q end proc:
> Java(f, optimize=true, returnvariablename="s",
>       defaulttype=integer);

class CodeGenerationClass {
    public static int f (int x)
    {
        extern int p;
        extern int q;
        extern int r;
        int s;
        p = x;
        q = x * x;
        r = x + q;
        s = r;
        return(s);
    }
}
```

As you see, you can use the **defaulttype** option to translate untyped parameters to integer variables. The translator creates for the Maple procedure a Java class called **CodeGenerationClass**, so that it is easy to incorporate the Java code into a larger Java program. The **returnvariablename** allows you to specify the name of the return variable.

But there is more flexibility in the code generation about the nature of variables, handling of arrays, printing of output, and error handling. We refer to the on-line help system for examples and detailed information about command options.

```
> F := proc(n::integer, v::Vector(3))
>   local nn, vv, new:
>   if n<1 then error "expecting a positive integer, "
>           "but received %1", n
>   end if:
>   nn := n: vv := v:
>   while nn>=1 do
>     new := vv[2]+vv[3]:
>     vv[1] := v[2]:
>     vv[2] := v[3]:
>     vv[3] := new:
>     nn := nn-1
>   end do:
>   vv[3]
> end proc:
> Java(F);

class CodeGenerationClass {
    public static double F (int n, double[] v)
    {
        int nn;
        double[] vv = new double[3];
    }
}
```

```

double new;
double cgret;
if (n < 1)
{
    System.err.println(
        "expecting a positive integer, but received " + n);
    System.exit(1);
}
nn = n;
vv[0] = v[0];
vv[1] = v[1];
vv[2] = v[2];
while (1 <= nn)
{
    new = vv[1] + vv[2];
    vv[0] = v[1];
    vv[1] = v[2];
    vv[2] = new;
    nn = nn - 1;
}
cgret = vv[2];
return(cgret);
}
}
}

```

We already gave the advice to use the `CodeGeneration` package instead of the `codegen` package for code generation whenever possible. One of the few occasions currently in Maple to use the latter package is for automatic differentiation (see §9.2). A simple example of computing the Jacobian matrix of a mapping:

```

> x := (r,phi) -> r*cos(phi); y := (r,phi) -> r*sin(phi);
> JACOBIAN([x,y]);

```

```

proc(r, φ)
local df, dfrθ, grd, t1, t2;
t1 := cos(φ);
t2 := sin(φ);
df := array(1..2);
dfrθ := array(1..2);
df1 := r;
dfrθ2 := r;
grd := array(1..2, 1..2);
grd1,1 := t1;
grd1,2 := -df1 * sin(φ);
grd2,1 := t2;
grd2,2 := dfrθ2 * cos(φ);
return grd
end proc

```

Maple provides the procedure **latex** to generate L<sup>A</sup>T<sub>E</sub>X code. When you want to generate L<sup>A</sup>T<sub>E</sub>X code for several formulae one after another and store them in one file use the function **appendto**. In the example below, we generate the L<sup>A</sup>T<sub>E</sub>X code for the first solution of the first example in this section. Here, we also show how the **latex** procedure can be adjusted to one's needs — we add the possibility of specifying L<sup>A</sup>T<sub>E</sub>X's math mode.

```
> 'latex/special_names['beginmath'] :=  
> '\begin{displaymath}':  
> 'latex/special_names['endmath'] :=  
> '\end{displaymath}':  
> # suppress messages, prompts, etc.  
> interface(quiet=true):  
> appendto("file.tex" ):  
> latex(beginmath); latex(sol1); latex(endmath);  
> writeto(terminal):  
> interface(quiet=false):
```

If the interface variable **screenwidth** equals the default value 80, then the file **file.tex** will look like

```
\begin{displaymath}  
1/6\sqrt[3]{108+12\sqrt{-12\{a\}^3+81}}+2\frac{a}{\sqrt[3]{108+12\sqrt{-12\{a\}^3+81}}}\\ \end{displaymath}
```

In typeset format, the formula looks like

$$\frac{1}{6}\sqrt[3]{108 + 12\sqrt{-12a^3 + 81}} + 2\frac{a}{\sqrt[3]{108 + 12\sqrt{-12a^3 + 81}}}.$$

When it bothers you to add the statements **latex(beginmath)** and **latex(endmath)** each time, you can write a small program like

```
> mylatex := proc(x)  
> global 'latex/special_names';  
> 'latex/special_names['beginmath'] :=  
> '\begin{displaymath}':  
> 'latex/special_names['endmath'] :=  
> '\end{displaymath}':  
> latex(beginmath); latex(x); latex(endmath);  
> end proc:
```

and call the procedure **mylatex** instead of the regular **latex**. After the statement

```
> macro(latex=mylatex):
```

you could even use the name **latex** instead of **mylatex** during the session.

```
> latex(sin(x**2));
```

```
\begin{displaymath}\sin \left( x^2 \right)
```

When formulae are too large to fit on one line, you have to edit the L<sup>A</sup>T<sub>E</sub>X file and insert breaks yourself. By the way, a complete Maple worksheet can be saved in L<sup>A</sup>T<sub>E</sub>X format, which typesets well under the style files delivered together with the software. This is even more convenient for large formulae as it will take care of splitting formulae into parts according to the text width, which can be set interactively or via the **interface** variable **latexwidth**. E.g., the instruction **interface(latexwidth=4.5);** will ensure that line-breaking occurs for a text width of 4.5in.

## 4.7 Changing Maple to Your Own Taste

In the text-based user interface, Maple will print during lengthy Maple computations *bytes-used* messages about the use of computer memory and computing time. These messages inform you about memory usage and computing time since the start of the session. The *bytes-used* message is of the following form:

*bytes used* = <*x*>, alloc = <*y*>, time = <*z*>

Here <*x*> is equal to the total computer memory used throughout the Maple session, whereas <*y*> measures the currently allocated amount of memory, both measured in bytes. <*z*> tells you the computing time since the beginning of the Maple session, measured in seconds. An example of *bytes-used* messages sent during a definite integration:

```
> integrate(1/(1+3*sin(t)^2), t=0..2*Pi));
bytes used=1016364, alloc=917336, time=0.10
bytes used=2016944, alloc=1507052, time=0.21
bytes used=3017884, alloc=2096768, time=0.39
bytes used=4018104, alloc=2293340, time=0.74
```

$\pi$

You can control the frequency of the *bytes-used* messages in the text-based interface with the procedure **kernelopts**.

```
> oldprintbytes := kernelopts(printbytes=false);
oldprintbytes := true
```

Hereafter no *bytes-used* messages appear. In the worksheet interface you yourself decide via menu selection whether the message regions at the bottom of the worksheet are present or not. In the above statement we set the new value of the kernel option **printbytes** and store the old value at the same time. This format is convenient for restoring the old value.

```
> kernelopts(printbytes=oldprintbytes):
```

Whereas the total number of computer words used steadily grows during a Maple session, this does not mean that the actual memory requirements grow as fast: reuse of memory takes place, and this cleaning up of memory is called *garbage collection*. Whenever this occurs Maple shows a *bytes-used* message (unless suppressed). You can explicitly request garbage collection with the procedure **gc**.

```
> gc():
```

With the kernel option **gcfreq** you can control the frequency of the garbage collection. When you enter

```
> oldfreq := kernelopts(gcfreq=10^6);
oldfreq := 250000
```

garbage collection takes place whenever one million extra computer words have been used during a Maple session; the default is, as you see, 250000 words (1 word = 4 or 8 bytes; you can ask for the word size of your machine via **kernelopts(wordsize)**).

If you mind these Maple messages in the text-based interface then you can silence the computer algebra system in two ways.

```
> kernelopts(printbytes=false):
```

This is the method you already learned. You can always find the status of the system with respect to the memory usage and computing time through the kernel options **bytesused**, **bytesalloc**, and **cputime** (enter **?kernelopts** to find out what other kernel options are available).

```
> kernelopts(bytesused, bytesalloc, cputime);
8937296, 3865916, 0.873
```

In the worksheet interface there are other ways of displaying and adjusting kernel options. The **KernelOpts** function in the **Examples** subpackage of the **Maplets** package displays a maplet that provides a graphical interface to the **kernelopts** routine. Just enter the following two commands and see what happens.

```
> with(Maplets[Examples]):
> KernelOpts();
```

A drastic way of making Maple less verbose in the text-based user interface is to enter

```
> interface(quiet=true):
```

A side effect is that the Maple prompt disappears and that, if you try to write results to a file with **writeto**, the input lines are not written to the file anymore. To Maple, silence means absolute silence!

With the same procedure **interface** you can arrange the output to appear in different ways. If desired you can change the prompt, stop

Maple's automatic use of labels for common subexpressions or force the Maple output into left-adjusted, one-dimensional format. A few examples:

```
> interface(prompt="---> "); # an arrow as a prompt
--> solve(x^3+x+a, x); # labels for common subexpressions


$$\frac{\%1^{(1/3)}}{6} - \frac{2}{\%1^{(1/3)}}, -\frac{\%1^{(1/3)}}{12} + \frac{1}{\%1^{(1/3)}} + \frac{1}{2} I \sqrt{3} \left( \frac{\%1^{(1/3)}}{6} + \frac{2}{\%1^{(1/3)}} \right),$$


$$-\frac{\%1^{(1/3)}}{12} + \frac{1}{\%1^{(1/3)}} - \frac{1}{2} I \sqrt{3} \left( \frac{\%1^{(1/3)}}{6} + \frac{2}{\%1^{(1/3)}} \right)$$


$$\%1 := -108a + 12\sqrt{12 + 81a^2}$$

--> interface(labeling=false); # no labels anymore
--> solve(x^3+x+a, x)[3];


$$-\frac{\%1^{(1/3)}}{12} + \frac{1}{\%1^{(1/3)}} - \frac{1}{2} I \sqrt{3} \left( \frac{\%1^{(1/3)}}{6} + \frac{2}{\%1^{(1/3)}} \right)$$


$$\%1 := -108a + 12\sqrt{12 + 81a^2}$$

--> Example := (1-x)/(1+x):
--> Example; # 2-dimensional layout


$$\frac{1-x}{1+x}$$

--> interface(prettyprint=0):
--> Example; # 1-dimensional layout


$$(1-x)/(1+x)$$

--> # reset prettyprint to high resolution
--> interface(prettyprint=2):
--> # show how many decimal places are displayed
--> interface(displayprecision);


$$-1$$

--> # -1 means, full display of floating-point numbers
--> evalf[30](Pi);


$$3.14159265358979323846264338328$$

--> # set displayprecision to low number of decimal places
--> interface(displayprecision=6):

--> evalf[30](Pi);


$$3.141593$$

--> # reset displayprecision
--> interface(displayprecision=-1);
--> interface(verboseproc=2): # make Maple verbose
--> print(unassign); # procedure body is printed
```

```

proc(nom)
description "remove an assignment from an assigned expression"
...
end proc
--> interface(verboseproc=3): # make Maple more verbose
--> print(ln);

proc(x::algebraic) ... end proc

# (1) = 0
# (infinity) = infinity
# (-infinity*I) = infinity - 1/2*I*Pi
# (-I) = -1/2*I*Pi
# (infinity*I) = infinity + 1/2*I*Pi
# (I) = 1/2*I*Pi

--> interface(prompt="> ");

```

The last two examples show that you can view the source code of library functions together with stored values in the remember tables. Only the C code of kernel functions is not viewable.

A handy interface variable is **errorbreak**. It determines what should happen when you read from a corrupted input file. Possible values are listed in Table 4.10.

<b>errorbreak</b>	
<b>Value</b>	<b>Purpose</b>
0	report error and continue reading
1	stop reading after syntax error
2	stop reading after any error

Table 4.10. Values of **errorbreak**.

Enter **?interface** to get a list of all interface variables, their possible values, and their purposes. Also note that some of the interface variables (e.g., **displayprecision** and **showassumed**) can be set in the worksheet interface through the **Preferences** menu-item.

You can limit the amount of time spent on a computation with the procedure **timelimit**.

```
> timelimit(0.1, factor(x^106+x^53+1));
```

```
Error, (in modp1/DistDeg) time expired
```

In the Unix operating system it is possible to inspect Unix environment variables like **HOME** and to find or set the name of the current working directory. The following session illustrates this.

```

> getenv("HOME");
          "/home/heck"
> currentdir();
          "/home/heck/Sheets"
> olddir := currentdir(getenv("HOME"));
          olddir := "/home/heck/Sheets"
> currentdir();
          "/home/heck"
> currentdir(olddir);
          "/home/heck"
> currentdir();
          "/home/heck/Sheets"

```

## 4.8 Exercises

- Find out where the Maple library resides on your computer and read the file `isolate.m` from the library with the `read` instruction. Hereafter, display the source code of this procedure.
- Study the interface variable `echo`.

- First, create the file `readfile` in your home directory and let the file contain the single Maple command  
`b := 2;`
- Secondly, create the file `testfile` in your home directory and let the file contain the following command lines.

```

interface( echo = X );
a := 1;
read readfile;
quit

```

- Thirdly, if your platform allows this, launch Maple from your home directory by

```
maple < testfile
```

so that all commands come from the particular file, while changing `X` in the first line of `testfile` into 0, 1, 2, 3, and 4, respectively.

- Finally, read the `testfile` in a Maple session with the procedure `read`, while changing `X` in the first line of `testfile` into 0, 1, 2, 3, and 4, respectively.

3. Create a file with the following contents

```
I := 1; # syntactically correct, but run-time error
x := 2; # correct input line
wrong name := 3; # syntax error
y := 4; # correct input line
```

For each possible value of the interface variable **errorbreak**, find out what happens when you read the file in a Maple session.

4. Consider a data file, three lines long with the following information

```
1    2
3    4
5    6
```

Read this file, convert the data into a matrix, transpose the matrix, and then printout the data in the format

```
1    2    3
4    5    6
```

5. Compute the second derivative of the expression  $x^{x^x}$  and translate the result into FORTRAN. Check the effect of code optimization.

# 5

## Polynomials and Rational Functions

Maple's favorite mathematical structures are polynomials and rational functions. This chapter is an introduction to univariate and multivariate polynomials, to rational functions, and to conversions between distinct forms. Keywords are: greatest common divisor (gcd), factorization, expansion, sorting in lexicographic or degree ordering, normalization of rational functions, and partial fraction decomposition.

### 5.1 Univariate Polynomials

Polynomials and rational functions (i.e., quotients of polynomials) are Maple's favorite data type. The computer algebra system manipulates them in full generality and at a high speed. For simplicity we start with polynomials in one indeterminate, say  $x$ ; these are symbolic expressions that are from mathematical point of view equivalent to

$$a_n x^n + a_{n-1} x^{n-1} + \cdots + a_2 x^2 + a_1 x + a_0 .$$

This representation is called the *expanded canonical form* with coefficients  $a_0, a_1, \dots, a_{n-1}$ , and  $a_n$ . If  $a_n \neq 0$  then we call  $a_n$  the *leading coefficient* and the natural number  $n$  the *degree* of the polynomial. Later on you will see that the coefficients of a polynomial are not subject to many restrictions, but for the moment we shall restrict ourselves to "ordinary" numbers like integers, rational numbers, real numbers, or complex numbers. We say that a polynomial in  $x$  is in *collected form*, when all the coefficients with the same power of  $x$  have been collected together. The only distinction from

the expanded canonical form is that in a polynomial in collected form all terms are not necessarily sorted in descending order with respect to the degree. An example:

```
> p1 := -3*x + 7*x^2 - 3*x^3 + 7*x^4; # collected form
      p1 :=  $-3x + 7x^2 - 3x^3 + 7x^4$ 
> type(p1, polynom);
      true
> leading_coefficient = lcoeff(p1);
      leading_coefficient = 7
> degree = degree(p1);
      degree = 4
> # p2 is in expanded canonical form
> p2 := 5*x^5 + 3*x^3 + x^2 - 2*x + 1;
      p2 :=  $5x^5 + 3x^3 + x^2 - 2x + 1$ 
> 2*p1 - 3*p2 + 3;
       $11x^2 - 15x^3 + 14x^4 - 15x^5$ 
> p1*p2;
       $(-3x + 7x^2 - 3x^3 + 7x^4)(5x^5 + 3x^3 + x^2 - 2x + 1)$ 
> expand(%);
       $-17x^6 + 11x^4 - 20x^3 + 13x^2 - 3x + 56x^7 + 4x^5 - 15x^8 + 35x^9$ 
```

Contrary to multiplication of numbers, the product of polynomials is not worked out automatically by Maple; you must give a separate command, viz., **expand**. This looks like a handicap, but really it is not. It is better to leave a factored form like  $(3x + 5)^{10}$  intact as long as possible in computations. This is clearer than the expanded form

$$59049x^{10} + 984150x^9 + 7381125x^8 + \dots + 58593750x + 9765625.$$

Although Maple computes the product of two polynomials correctly, the terms in the answer are mixed up and not in ascending or descending order with respect to the degree. This often leads to sloppy expressions that are difficult to read. Terms in polynomials are not sorted into some standard ordering for efficiency reasons (both with respect to computing time and memory storage). If you really want to rearrange terms of a polynomial in descending order with respect to the degree, then you should apply the procedure **sort**.

```
> sort(%);
       $35x^9 - 15x^8 + 56x^7 - 17x^6 + 4x^5 + 11x^4 - 20x^3 + 13x^2 - 3x$ 
```

Actually, **sort** will change the internal data structure. This is necessary because Maple only stores one copy for each expression or subexpression in memory, or more precisely in its *simplification table*. For any new expression Maple checks by “internal algebra” whether the expression can be simplified by elementary operations to an expression that has been used before and stored in the simplification table. If this is the case, then Maple skips the new expression and uses the old expression from the simplification table instead. Otherwise the expression, together with its subexpressions, is stored in the simplification table. Changing terms in a sum or factors in a product are examples of elementary simplifications that are used to test mathematical equivalence of expressions.

```
> p := 1+x+x^3+x^2; # random order

$$p := 1 + x + x^3 + x^2$$

> x^3+x^2+x+1; # no rearrangement, so still random order

$$1 + x + x^3 + x^2$$

> q := (x-1)*(x^3+x^2+x+1); # the same for subexpressions

$$q := (x - 1)(1 + x + x^3 + x^2)$$

> sort(p); # rearrange w.r.t. descending degree order

$$x^3 + x^2 + x + 1$$

> q; # subexpression of q has also changed

$$(x - 1)(x^3 + x^2 + x + 1)$$

```

Maple offers many procedures to manipulate polynomials; we shall have a look at some of them. Until further notice, p1 and p2 are equal to the previously defined polynomials.

```
> 'p1'=p1, 'p2'=p2;

$$p1 = -3x + 7x^2 - 3x^3 + 7x^4, p2 = 5x^5 + 3x^3 + x^2 - 2x + 1$$

```

The **coeff**icient of a power or monomial can be determined with the procedure **coeff**.

```
> coeff(p2, x^3), coeff(p2, x, 2);

$$3, 1$$

```

The leading and trailing **coefficients** can be found with **lcoeff** and **tcoeff**, respectively.

```
> lcoeff(p2, x), tcoeff(p2, x);

$$5, 1$$

```

You get a sequence of all **coefficients** and the corresponding powers with the procedure **coeffs**.

```
> coeffs(p2, x, 'powers');
```

```

1, 3, 1, -2, 5
> powers; # assigned in the previous command
1,  $x^3$ ,  $x^2$ ,  $x$ ,  $x^5$ 

```

Maple expects that polynomials be in collected form before you use **lcoeff**, **tcoeff**, **coeffs**, and **degree**.

```

> lcoeff(x^2-x*(x-1), x);
0
> coeffs((x^2-1)*(x^2+1), x);
Error, invalid arguments to coeffs
> coeffs(collect((x^2-1)*(x^2+1), x), x);
-1, 1
> degree(x^2-x*(x-1), x);
2

```

By the way, the procedure **CoefficientList** from the **PolynomialTools** package does not require polynomials to be in collected form. However, it will list all coefficients, whether zero or not.

```

> PolynomialTools[CoefficientList]((x^2-1)*(x^2+1), x);
[-1, 0, 0, 0, 1]

```

One of the most fundamental operations with polynomials is *division with remainder*. Maple has two procedures, viz., **quo** and **rem** — **quotient** and **remainder** — to do this.

```

> q := quo(p2, p1, x, 'r');

$$q := \frac{5x}{7} + \frac{15}{49}$$

```

The fourth argument in the call of **quo** is optional, but when used, it is assigned the remainder of the polynomial division.

```

> r;

$$1 - \frac{53}{49}x^3 + x^2 - \frac{53}{49}x$$

> testeq(p2=expand(q*p1+r)); # test equality
true
> rem(p2, p1, x, 'q');

$$1 - \frac{53}{49}x^3 + x^2 - \frac{53}{49}x$$

> q; # quotient has been set in previous command

$$\frac{5x}{7} + \frac{15}{49}$$

```

Mind the use of apostrophes around the fourth argument of these procedures to suppress unintended evaluation of this argument.

Another important function is **gcd** to compute greatest common divisors of polynomials over the rational numbers.

```
> gcd(p1, p2);
```

$$x^2 + 1$$

The computation of greatest common divisors is so important for so many mathematical calculations that developers of computer algebra systems (q.v., [98, 148]) have put great effort into the design and implementation of efficient gcd algorithms.

An even more complicated algorithm is factorization of polynomials; the interested reader is referred to [142, 144, 145]. The procedure **factor** writes polynomials with rational coefficients as a product of irreducible (over  $\mathbb{Q}$ ) polynomials.

```
> polynomial := expand(p1*p2);
```

*polynomial* :=

$$-17x^6 + 11x^4 - 20x^3 + 13x^2 - 3x + 56x^7 + 4x^5 - 15x^8 + 35x^9$$

```
> factor(polynomial);
```

$$x(7x - 3)(5x^3 - 2x + 1)(x^2 + 1)^2$$

Computations with polynomials over domains differing from the integers or rational numbers are possible, e.g., over finite fields or over algebraic number and function fields. Examples of factorizations of the above polynomial over such domains are:

- Factorization over  $\mathbb{Z}[i]$ , the ring of Gaussian integers.

```
> factor(polynomial, I);
```

$$x(5x^3 - 2x + 1)(7x - 3)(x + I)^2(x - I)^2$$

- Factorization over  $\mathbb{Z}_2$ , the field of two elements, 0 and 1.

```
> sort(polynomial mod 2); # polynomial over {0,1}
```

$$x^9 + x^8 + x^6 + x^4 + x^2 + x$$

```
> Factor(polynomial) mod 2;
```

$$(x^2 + x + 1)x(x + 1)^6$$

```
> Expand(%) mod 2;
```

$$x^9 + x^8 + x^6 + x^4 + x^2 + x$$

Note the use of the upper-case character in **Factor** and **Expand**. The purpose of this is to suppress factorization over the integers and expansion, respectively. **Factor(...)** and **Expand(...)** are just placeholders for

representing the factorization of the polynomial and the expansion of the polynomial. The **mod** operator actually gets the computation going. For example, with **factor(polynomial) mod 2**, the polynomial is first factored over the rational numbers, and then modulo arithmetic is applied.

```
> factor(polynomial) mod 2;

$$x(x+1)(x^3+1)(x^2+1)^2$$

• Factorization over the Galois field GF(4), in particular the algebraic extension of  $\mathbb{Z}_2$  with respect to the irreducible polynomial  $z^2+z+1$ .
> alias(alpha=RootOf(z^2+z+1,z));
> Factor(polynomial, alpha) mod 2;

$$(x+\alpha)x(x+1)^6(x+\alpha+1)$$

```

The mechanism of placeholders for manipulations of polynomials is used for other procedures like **Gcd**, **Divide**, **Expand**, and so on. A few examples:

```
> Gcd(p1,p2) mod 2;

$$x^3+x^2+x+1$$

> Expand(p1*p2) mod 2;

$$x^9+x^8+x^6+x^4+x^2+x$$

> q := Quo(p2, p1, x, 'r') mod 2;

$$q := x+1$$

> r;

$$x^3+x^2+x+1$$

> Expand(p2 - q*p1 - r) mod 2;

$$0$$

```

The **SNAP** package (Symbolic-Numeric Algorithms for Polynomials) provides routines for the algebraic manipulation of numerical polynomials in a reliable way. You can do division with remainder and numeric gcd computation. Let us see what happens when we do this for the numeric equivalents of the polynomials p1 and p2.

```
> np1 := evalf(p1); np2 := evalf(p2);

$$np1 := -3.x + 7.x^2 - 3.x^3 + 7.x^4$$


$$np2 := 5.x^5 + 3.x^3 + x^2 - 2.x + 1.$$

> nq := Quotient(np2, np1, x, 'nr');

$$nq := 0.714285714285714301x + 0.3061224490$$

> expand(np2 - nq*np1 - nr);

$$0.$$

```

```
> evalf(5/7*x+15/49); # compare with the exact result
0.7142857143 x + 0.3061224490
```

For the computation of a numeric gcd there are two algorithms available, viz., for computing an epsilon-gcd and a quasi-gcd [17].

```
> EpsilonGCD(np1, np2, x);
0.1250000000 x2 + 0.2304347830 10-9 x + 0.1250000001, 0.2306461025 10-5
```

Compare this and the next result with the exact gcd computed before.

```
> 8*fnormal(%[1]);
1.000000000 x2 + 1.000000001
```

## 5.2 Multivariate Polynomials

Multivariate polynomials are polynomials in more than one unknown. An example of a polynomial in two indeterminates  $x$  and  $y$  is

```
> polynomial := 6*x*y^5 + 12*y^4 + 14*y^3*x^3
> - 15*x^2*y^3 + 9*x^3*y^2 - 30*x*y^2 - 35*x^4*y
> + 18*y*x^2 + 21*x^5;
```

$$\begin{aligned} \text{polynomial} := & 6 x y^5 + 12 y^4 + 14 y^3 x^3 - 15 x^2 y^3 + 9 x^3 y^2 - 30 x y^2 \\ & - 35 x^4 y + 18 y x^2 + 21 x^5 \end{aligned}$$

You see that Maple has no personal feelings about term orderings or orderings of indeterminates within monomials; it just takes the ordering that it encounters first. Again, you can do something about it with the procedure **sort**.

```
> sort(polynomial, [x,y], 'plex');
21 x5 - 35 x4 y + 14 x3 y3 + 9 x3 y2 - 15 x2 y3 + 18 x2 y + 6 x y5
- 30 x y2 + 12 y4
```

Here, the terms are ordered in a **pure lexicographic ordering** defined by

$$x^i y^j \prec x^{i'} y^{j'} \iff i < i' \text{ or } (i = i' \text{ and } j < j').$$

So,

$$1 \prec y \prec y^2 \prec \dots \prec x \prec xy \prec x^2 \dots$$

You can also use the **total degree ordering** defined by

$$x^i y^j \prec x^{i'} y^{j'} \iff i + j < i' + j' \text{ or } (i + j = i' + j' \text{ and } i < i').$$

So,

$$1 \prec y \prec x \prec y^2 \prec xy \prec x^2 \prec y^3 \prec xy^2 \prec x^2 y \prec x^3 \prec \dots$$

In this term ordering, which is the default ordering used by **sort**, we get

```
> sort(polynomial);

$$14x^3y^3 + 6xy^5 + 21x^5 - 35x^4y + 9x^3y^2 - 15x^2y^3 + 12y^4
+ 18x^2y - 30xy^2$$


```

The above polynomial can also be considered as a polynomial in the indeterminate **x**, with polynomials in **y** as coefficients. You can bring the polynomial into this form with the procedure **collect**.

```
> collect(polynomial, x);

$$21x^5 - 35x^4y + (14y^3 + 9y^2)x^3 + (18y - 15y^3)x^2
+ (-30y^2 + 6y^5)x + 12y^4$$


```

Alternatively, you can consider **polynomial** as a polynomial in **y**.

```
> collect(polynomial, y);

$$6xy^5 + 12y^4 + (-15x^2 + 14x^3)y^3 + (9x^3 - 30x)y^2
+ (-35x^4 + 18x^2)y + 21x^5$$


```

Many of the Maple procedures for manipulation of univariate polynomials work as well for polynomials in more than one unknown. We only show a few examples, and invite you to experiment with multivariate polynomials yourself.

```
> coeff(polynomial, x^3), coeff(polynomial, x, 3);

$$14y^3 + 9y^2, 14y^3 + 9y^2$$

> coeffs(polynomial, x, 'powers'); powers;

$$12y^4, -30y^2 + 6y^5, -35y, 21, 14y^3 + 9y^2, 18y - 15y^3$$


$$1, x, x^4, x^5, x^3, x^2$$

> coeff(coeff(polynomial, x^3), y^2);

$$9$$

> settim := time(); # start timing
> factor(polynomial);

$$(3x^2 - 5xy + 2y^3)(7x^3 + 6y + 3xy^2)$$

> Factor(polynomial) mod 7;

$$2y(x^2 + 3xy + 3y^3)(xy + 2)$$

> cpu_time := (time() - settim)*seconds;

$$cpu\_time := .351\ second$$


```

No human can come near in speed to such factorizations; just ask your mathematical friends and colleagues to do such factorizations with pencil and paper! There is a lot of mathematical knowledge and complicated programming behind the procedures **factor** and **Factor**.

## 5.3 Rational Functions

A rational function in the unknown  $x$  is an expression that can be written in the form  $f/g$ , where  $f$  and  $g$  are polynomials in  $x$ , and  $g$  is not equal to 0. For example,

```
> f := x^2+3*x+2; g := x^2+5*x+6; f/g;
```

$$\frac{x^2 + 3x + 2}{x^2 + 5x + 6}$$

You can select the **numerator** and the **denominator** of a rational expression with the procedure **numer** and **denom**, respectively.

```
> numer(%), denom(%);
```

$$x^2 + 3x + 2, x^2 + 5x + 6$$

Note that contrary to calculations with rational numbers, Maple does not simplify rational expressions into a form where numerator and denominator are relatively prime by itself. Simplifications are carried out automatically only when Maple immediately recognizes common factors.

```
> ff := (x-1)*f; gg := (x-1)^2*g;
```

$$ff := (x - 1)(x^2 + 3x + 2)$$

$$gg := (x - 1)^2(x^2 + 5x + 6)$$

```
> ff/gg;
```

$$\frac{x^2 + 3x + 2}{(x - 1)(x^2 + 5x + 6)}$$

If you want to bring a rational expression into such form, then you should apply the procedure **normal**. This procedure leaves factors in numerator and denominator in factored form as much as possible for efficiency reasons.

```
> normal(f/g);
```

$$\frac{x + 1}{x + 3}$$

```
> normal(ff/gg);
```

$$\frac{x + 1}{(x + 3)(x - 1)}$$

There are three reasons for not doing **normalization** of rational expressions automatically:

- ▷ it does not always give simpler results, e.g.,  $(x^{1000} - 1)/(x - 1)$  is more compact than its normalization, which is a polynomial with ten thousand terms;
- ▷ it is too time-consuming to do normalization for every rational expression that appears in a Maple computation;

- ▷ a user may want other manipulations, e.g., partial fraction decompositions.

We end this section with an example of a rational expression in more than one unknown, in which we divide out a common factor in numerator and denominator (check this yourself by factoring these subexpressions).

```

> f := 161*y^3 + 333*x*y^2 + 184*y^2 + 162*x^2*y
>      + 144*x*y + 77*y + 99*x + 88;
> g := 49*y^2 + 28*x^2*y + 63*x*y + 147*y + 36*x^3
>      + 32*x^2 + 117*x + 104;
> ratexpr := f/g;

ratexpr :=

$$\frac{161 y^3 + 333 x y^2 + 184 y^2 + 162 x^2 y + 144 x y + 77 y + 99 x + 88}{49 y^2 + 28 x^2 y + 63 x y + 147 y + 36 x^3 + 32 x^2 + 117 x + 104}$$

> normal(ratexpr);


$$\frac{18 x y + 23 y^2 + 11}{4 x^2 + 7 y + 13}$$

```

## 5.4 Conversions

The *Horner form* of a polynomial is most suitable for numerical processing because in this representation the number of arithmetical operations is minimal. We illustrate this with the previously defined polynomial p1. The number of additions and multiplications can be tabulated with the procedure **cost** from the **codegen** package for **code generation**.

```

> with(codegen, cost):
> p1 := -3*x+7*x^2-3*x^3+7*x^4;
          p1 := -3 x + 7 x2 - 3 x3 + 7 x4
> cost(p1);
          3 additions + 10 multiplications
> convert(p1, 'horner');
          (-3 + (7 + (-3 + 7 x) x) x) x
> cost(%);
          4 multiplications + 3 additions
```

Conversion of rational expressions to their **continued fraction** forms can also speed up numerical computations.

```

> q := (x^3+x^2-x+1)/p1;
          
$$q := \frac{x^3 + x^2 - x + 1}{-3 x + 7 x^2 - 3 x^3 + 7 x^4}$$

```

```
> cost(q);
```

*6 additions + 13 multiplications + divisions*

```
> convert(q, 'confrac', x);
```

$$\cfrac{1}{7 \left( x - \frac{10}{7} + \cfrac{24}{7 \left( x + \frac{11}{6} + \cfrac{1}{9 \left( x - \frac{71}{24} + \cfrac{429}{64(x + \frac{17}{8})} \right)} \right)} \right)}$$

```
> cost(%);
```

*4 divisions + 7 additions*

The conversion to Horner form and to continued fraction form are also available in the **numapprox** package for numerical approximation as the procedures **hornerform** and **confracform**, respectively. See §11.2 for more details about this Maple package.

Partial fraction decomposition — **convert** to **partial fraction** form — can easily be done in Maple.

```
> convert(q, 'parfrac', x);
```

$$\frac{3 + 7x}{29(1 + x^2)} + \frac{143}{87(-3 + 7x)} - \frac{1}{3x}$$

From the partial fraction decomposition you can almost immediately tell the indefinite integral of this rational expression.

```
> integrate(q, x);
```

$$-\frac{1}{3} \ln(x) + \frac{143}{609} \ln(7x - 3) + \frac{7}{58} \ln(1 + x^2) + \frac{3}{29} \arctan(x)$$

You can specify how the denominator in the rational expression is to be factored.

```
> convert(q, 'parfrac', x, sqrfree);
```

$$\frac{4 + 10x^2}{3(-3 + 7x - 3x^2 + 7x^3)} - \frac{1}{3x}$$

Note that in the above example Maple assumes that the field of definition is the rational number field. You can specify that you want floating-point real or floating-point complex partial fractions by adding a fourth keyword, **real** or **complex**, or some field extension.

```
> convert(q, 'parfrac', x, real);
```

```


$$-\frac{0.3333333334}{x} + \frac{0.2348111659}{x - 0.4285714286} + \frac{0.1034482759 + 0.2413793105 x}{x^2 + 1}.$$

> convert(q, 'parfrac', x, I);

$$\frac{\frac{7}{58} - \frac{3}{58}I}{x - I} + \frac{\frac{143}{609}(-\frac{3}{7} + x)}{-\frac{3}{7} + x} + \frac{\frac{7}{58} + \frac{3}{58}I}{x + I} - \frac{1}{3x}$$


```

You can also do partial fraction decomposition over the algebraic closure of the field of definition so that you get all linear factors of the denominator. A detailed description of the algorithm can be found in [30]. This full partial fraction decomposition — **convert** to **full partial fraction** form — can be done in the following way.

```

> convert(q, 'fullparfrac', x);


$$\left( \sum_{-\alpha=\%1} \frac{-\frac{3}{58}\alpha + \frac{7}{58}}{x - \alpha} \right) + \frac{\frac{143}{609}(-\frac{3}{7} + x)}{-\frac{3}{7} + x} - \frac{1}{3x}$$

%1 := RootOf(_Z^2 + 1)

```

You can have the roots of the 2nd degree polynomial in radical form and the summation carried out.

```

> convert(% , radical);


$$\frac{\frac{7}{58} - \frac{3}{58}I}{x - I} + \frac{\frac{7}{58} + \frac{3}{58}I}{x + I} + \frac{\frac{143}{609}(-\frac{3}{7} + x)}{-\frac{3}{7} + x} - \frac{1}{3x}$$


```

Like with partial fraction decomposition you can specify how the denominator of the rational expression is to be factored and whether rationalization of **RootOf** expressions should be done or not.

```

> convert(q, 'fullparfrac', x, sqrfree);


$$\left( \sum_{-\alpha=\%1} \frac{\frac{595}{2523} - \frac{3}{58}\alpha + \frac{581}{5046}\alpha^2}{x - \alpha} \right) - \frac{1}{3x}$$

%1 := RootOf(-3 + 7_Z - 3_Z^2 + 7_Z^3)
> convert(q, 'fullparfrac', x, sqrfree, false);


$$\left( \sum_{\alpha=\%1} \frac{4 + 10\alpha^2}{(21 - 18\alpha + 63\alpha^2)(x - \alpha)} \right) - \frac{1}{3x}$$

%1 := RootOf(-3 + 7_Z - 3_Z^2 + 7_Z^3)

```

Of course it is also not forbidden to help Maple when you know or guess a suitable extension of the field of definition; simply add the extension(s) as a fourth argument as we did before for the imaginary unit.

```
> 1/(x^4-5*x^2+6);

$$\frac{1}{x^4 - 5x^2 + 6}$$

> convert(%, 'parfrac', x, sqrt(2));

$$\frac{1}{x^2 - 3} - \frac{\sqrt{2}}{4(x - \sqrt{2})} + \frac{\sqrt{2}}{4(x + \sqrt{2})}$$

> convert(%%, 'parfrac', x, {sqrt(2),sqrt(3)});

$$-\frac{\sqrt{3}}{6(x + \sqrt{3})} - \frac{\sqrt{2}}{4(x - \sqrt{2})} + \frac{\sqrt{2}}{4(x + \sqrt{2})} + \frac{\sqrt{3}}{6(x - \sqrt{3})}$$

```

A final example of partial fraction decomposition of a rational expression in more than one indeterminate follows.

```
> ratfun := (x-a)/(x^5+b*x^4-c*x^2-b*c*x);
ratfun :=  $\frac{x - a}{x^5 + b x^4 - c x^2 - b c x}$ 
> convert(ratfun, 'parfrac', x);

$$\frac{b^2 c + b c a - b c x - c x a - x^2 a b^2 + c x^2}{(x^3 - c) c (c + b^3)} + \frac{a}{x b c} + \frac{-b - a}{(x + b) b (c + b^3)}$$

> # write numerators as polynomials in x
> map(collect, %, x);

$$\frac{(-a b^2 + c) x^2 + (-b c - c a) x + b^2 c + b c a}{(x^3 - c) c (c + b^3)} + \frac{a}{x b c} + \frac{-b - a}{(x + b) b (c + b^3)}$$

> convert(ratfun, 'parfrac', x, true);

$$\frac{a}{x b c} + \frac{b c + c a - a x^2 b - a x^3}{(x^4 + b x^3 - c x - b c) b c}$$

```

In the last command, we add the extra argument **true** to indicate that no application of the procedure **normal** on **ratfun** is needed and that the denominator can be considered to be already in the desired factored form.

## 5.5 Exercises

- Consider the rational expression  $\frac{x^4 + x^3 - 4x^2 - 4x}{x^4 + x^3 - x^2 - x}$ . Transform the expression into:
  - $\frac{(x+2)(x+1)(x-2)}{x^3 + x^2 - x - 1}$

- (b)  $\frac{x^4 + x^3 - 4x^2 - 4x}{x(x-1)(x+1)^2}$   
 (c)  $\frac{(x+2)(x-2)}{(x-1)(x+1)}$   
 (d)  $\frac{x^2}{(x-1)(x+1)} - 4 \frac{1}{(x-1)(x+1)}$

2. Consider the same rational expression as in the previous exercise.
  - (a) Write it as a continued fraction.
  - (b) Compute a partial fraction decomposition.
3. Let  $f$  be the polynomial  $x^7 + x^5 + 2x^3 + 2x^2 + 3x + 2$ .
  - (a) Factor  $f$  over  $\mathbb{Z}_5$ , and over  $\mathbb{Z}_7$ .
  - (b) How can you conclude from the former factorizations that  $f$  is irreducible over the ring over integers (Hint: look at the degrees of the factors). Check with Maple that  $f$  is indeed irreducible over  $\mathbb{Z}$ .
4. Some factorizations modulo a prime number:
  - (a) Factor  $x^2 - x$  over  $\mathbb{Z}_2$ .
  - (b) Factor  $x^3 - x$  over  $\mathbb{Z}_3$ .
  - (c) Factor  $x^5 - x$  over  $\mathbb{Z}_5$ .
  - (d) Factor  $x^{23} - x$  over  $\mathbb{Z}_{23}$ .
  - (e) By now you may have an idea how  $x^p - x$  factorizes for any prime number  $p$ ; check your conjecture for some prime number distinct from previous choices. Do you understand the result?
5. Let  $f = 2x^4 - 3x^2 + x + 4$  and  $g = 2x^5 - 6x^4 - 4x^3 + x^2 - 3x - 2$ . Consider them as polynomials over  $\mathbb{Z}_7$  and compute the greatest common divisor of  $f$  and  $g$ . Determine also polynomials  $s$  and  $t$  over  $\mathbb{Z}_7$  such that  $sf + tg = \gcd(f, g)$  ( $\gcd$  with respect to  $\mathbb{Z}_7$ , of course).
6. If you ask Maple to factor  $x^{2458} + x^{1229} + 1$  over  $\mathbb{Z}$ , then the system will complain about insufficient memory, or after several hours it will still not have found an answer. So, let us have a closer look at this problem and see if we can assist Maple. First, we note that 1229 is a prime number (e.g., with `isprime`). So, the polynomial is of the form  $x^{2p} + x^p + 1$ , where  $p$  is a prime number. Determine with Maple the factorization of this polynomial for low prime values of  $p$  and propose a conjecture about the form of the factored polynomial for the general case.

An experienced mathematician can tell you that the polynomial  $x^n - 1$ , where  $n$  is a natural number, can be factored over  $\mathbb{Z}$  in the *cyclotomic polynomials*  $\phi_k(x)$ .

$$x^n - 1 = \prod_{k|n} \phi_k(x)$$

These cyclotomic polynomials are irreducible over  $\mathbb{Z}$ . With this knowledge and under the assumption that your conjecture is correct, it should not be too difficult to prove the result.

# 6

## Internal Data Representation and Substitution

In this chapter we shall describe in detail the internal representation of polynomials and rational functions. This topic is not only of theoretical interest; it is also important to know about data representation when you want to fully understand conversion between distinct data structures and substitution.

We shall generalize the concept of rational expressions, and explain how the procedures defined for polynomial and rational functions extend to more general mathematical structures.

Maple's procedures for type checking and selection of parts of formulae will also be treated. These routines support you when studying or manipulating data structures.

Finally, we shall discuss sequential and simultaneous substitution, and how this can be used in the process of simplifying expressions.

### 6.1 Internal Representation of Polynomials

In the previous chapter, we have already seen some Maple procedures that transform polynomials from one shape into another. To get a better understanding of these manipulations and others to come, we dot the i's and cross the t's and have a closer look at the internal data representation of polynomials in Maple.

We take the polynomial  $x^4 + x^3 - x^2 - x$  as our example. The expanded canonical form is represented internally by the set of data vectors shown in Figure 6.1, which is, in graph-theoretical terms, a *directed acyclic graph (DAG)*.

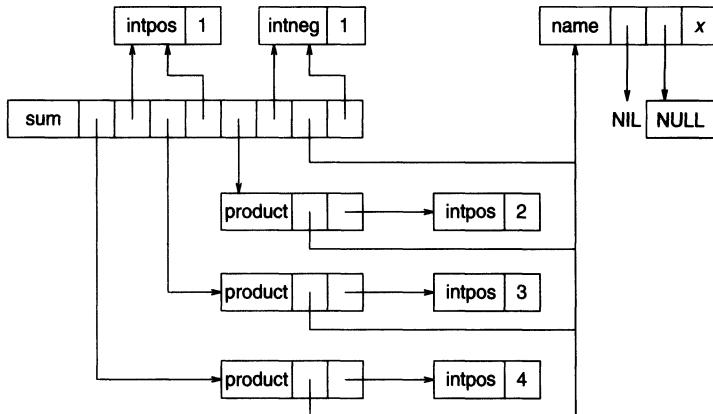


Figure 6.1. The internal representation of  $x^4 + x^3 - x^2 - x$ .

Here, you must interpret the data vector of Figure 6.2 as if it consists of pairs of expressions and their numerical coefficients: it represents the sum

$$\text{coeff}_1 \times \text{expr}_1 + \text{coeff}_2 \times \text{expr}_2 + \dots$$

sum	expr1	coeff1	expr2	coeff2	.....
-----	-------	--------	-------	--------	-------

Figure 6.2. The internal representation of a sum.

Similarly, Figure 6.3 shows the data vector that represents the product

$$\text{expr}_1^{\text{expon}_1} \times \text{expr}_2^{\text{expon}_2} \times \dots$$

product	expr1	expon1	expr2	expon2	.....
---------	-------	--------	-------	--------	-------

Figure 6.3. The internal representation of a product.

The NIL pointer in Figure 6.1 is used to indicate that the variable  $x$  is unbound. The pointer to the Maple expression NULL, which stands for an empty expression sequence, means that no attributes are associated with  $x$ .

At the first level, the internal representation of  $x^4 + x^3 - x^2 - x$  reflects the way Maple considers it, viz., as a sum of four terms,  $x^4$ ,  $x^3$ ,  $x^2$ , and  $x$ , with coefficients 1, 1, -1, and -1, respectively. These components are internally represented by data vectors. But, surprise, the term  $x^4$  is internally represented by a data vector consisting of the header product, a pointer to the unknown  $x$ , and a pointer to the exponent 4. You might have expected a data vector as in Figure 6.4.

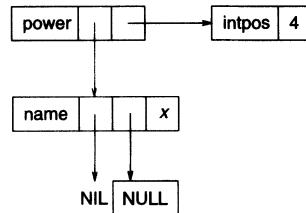


Figure 6.4. False internal representation of  $x^4$ .

This type of DAG indeed exists in Maple, but if the exponent is a numerical constant, it is changed to the product structure by Maple's simplifier.

Maple only recognizes  $x^4$ ,  $x^3$ ,  $x^2$ ,  $-x^2$ ,  $x$ , and  $-x$  as subexpressions of the polynomial  $x^4 + x^3 - x^2 - x$ ; as far as Maple is concerned,  $x^4 + x^3$  and  $x^4 - x$  are not subexpressions of  $x^4 + x^3 - x^2 - x$ . The fact that Maple does not recognize every mathematical subexpression plays an important role in substitution. Only this can explain substitutions like

```

> subs(1=7, x^4 + x^3 - x^2 - x);
    7 x4 + 7 x3 - x2 - x
> subs(1=3, Pi*x + x + 1);
    3 π3 x3 + 3 x + 9
  
```

Let us have a closer look at the last example.  $\pi x + x + 1$  is represented internally as in Figure 6.5.

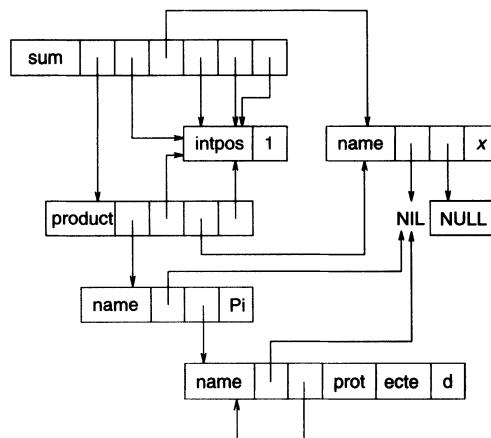


Figure 6.5. Internal representation of  $\pi x + x + 1$ .

The above directed acyclic graph clearly reflects that `Pi` has the attribute `protected`. This attribute is a valid Maple expression, viz., a name. The characters of a name are stored on a 32-bit machine in groups of at most four characters. The data structure `protected` has its own name as

attribute; so, this name is protected in Maple as well. If you replace the data vector that represents 1 by a vector representing 3, then you will understand the result of the substitution of 3 for 1.

The two examples above show you that the internal representation can differ significantly from the external representation, which looks more familiar and is actually what you see on the screen. For Maple however, it is only the internal representation that counts; two expressions are identical if and only if they have the same DAG, and simplification is for the system nothing more than a transformation from one DAG into another.

Before we continue, let us think about the reasons that the designers of Maple may have had for their choice of representing sums and powers. Let us consider the representation of the monomial  $x^2y^3z^4$ . Maple's internal representation is shown in Figure 6.6.

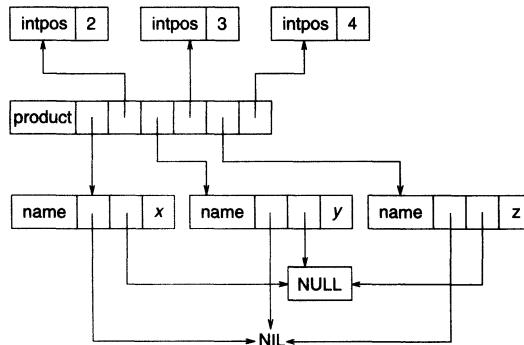


Figure 6.6. Internal representation of  $x^2y^3z^4$ .

An alternative could be to consider  $x^2y^3z^4$  as a product of the three powers  $x^2$ ,  $y^3$ , and  $z^4$ , with the internal representation as shown in Figure 6.7.

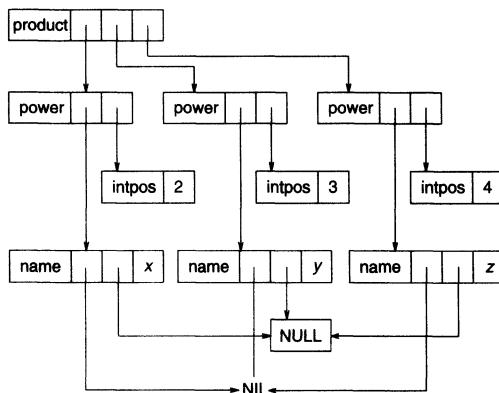


Figure 6.7. False internal representation of  $x^2y^3z^4$ .

In this picture, you see that more memory storage is needed in the latter representation. But what is more important, suppose that during a computation  $x^2y^3z^4$  is multiplied by like factors, say  $x^5z^6$ . Then Maple's simplifier should recognize the like factors as quickly as possible, add their exponents and build the new data structure. In the data representation chosen by Maple, this can be done easily compared to the alternative representation. Similarly, the internal representation of sums allows easy identification of like terms and addition of their coefficients. In short, Maple's internal data representation has been optimized to make polynomial arithmetic fast.

Let us forget for the moment that Maple stores (sub)expressions only once. Then the directed acyclic graph has the graph theoretical structure of a *tree*. The *representation tree* for our example is shown in Figure 6.8.

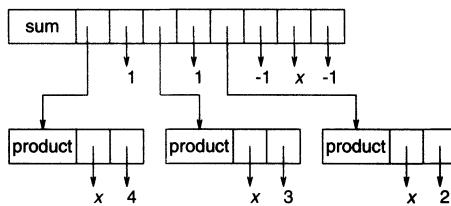


Figure 6.8. Representation tree of  $x^4 + x^3 - x^2 - x$ .

Here we have omitted the data vectors of the building blocks 1,  $-1$ , and  $x$ ; they are considered as atomic elements in the formula. The representation tree of  $x^4 + x^3 - x^2 - x$  also indicates that the polynomial consists of  $x^4$ ,  $x^3$ ,  $-x^2$ , and  $-x$ , and that  $x^2$  is a subexpression. The main advantage of the representation tree is that it is not as messy as the directed acyclic graph. An essential difference between the directed acyclic graph and the representation tree is that in the directed acyclic graph one node is used for identical subexpressions whereas in the representation tree they are represented by separate subtrees. If one keeps this in mind, then the representation tree is an adequate and easy-to-understand description of the internal structure of a Maple expression.

Perhaps your brain is reeling from all those data vectors, directed acyclic graphs, and representation trees? Are you wondering uneasily whether you will always be able to grasp the internal structure of a Maple expression? Well, let it be a relief to you that in most cases your intuition will not fail and that in doubtful cases the system itself can assist you in the determination of the internal data structure. The surface data type of an expression can be determined by the procedure **whattype**.

```

> p1 := x^4 + x^3 - x^2 - x; # expanded canonical form
      p1 := x^4 + x^3 - x^2 - x
> whattype(p1);
  
```

$$+$$

By the symbol “+”, the computer algebra system denotes that the expression is a **sum**. The procedure **nops** (number of operands) gives you the number of summands.

```
> nops(p1);
4
```

You get the sequence of components of a symbolic expression with the procedure **op** (extract operands).

```
> op(p1);
x4, x3, -x2, -x
```

You can unravel each subexpression separately. For example, the first term is a power of  $x$  with exponent 4.

```
> 'first term' := op(1, p1);
first term := x4
> whattype(%);
^
> op(%);
x, 4
```

Here, the symbol “^” is used as an indication for the data type power.

The third term,  $-x^2$ , is the product of  $-1$  and the power  $x^2$ . Maple denotes the product data type as “\*”.

```
> 'third term' := op(3, p1);
third term := -x2
> whattype(%);
*
> op('third term');
-1, x2
> op([3,2], p1); # 2nd operand of 3rd operand of p1
x2
```

In the same manner, you can unravel any Maple expression, not just polynomials. The only thing that you must keep in mind is that identical subexpressions are stored in Maple only once. In the above example, the character  $x$  occurs at four places, but internally it is only one Maple object. If you substitute  $y$  for  $x$  in the polynomial  $x^4 + x^3 - x^2 - x$ , you get the polynomial  $y^4 + y^3 - y^2 - y$ .

If you really want to dive into Maple and consider the direct acyclic graph instead of the representation tree, Maple can show you the details via the **dismantle** procedure. It displays the structure of a Maple expression, showing each subexpression and its length.

```
> dismantle(x^4+x^3-x^2-x);
```

```
SUM(9)
  PROD(3)
    NAME(4): x
    INTPOS(2): 4
    INTPOS(2): 1
  PROD(3)
    NAME(4): x
    INTPOS(2): 3
    INTPOS(2): 1
  PROD(3)
    NAME(4): x
    INTPOS(2): 2
    INTNEG(2): -1
    NAME(4): x
    INTNEG(2): -1
```

## 6.2 Generalized Rational Expressions

The following Maple session reveals the internal representation of a rational function.

```
> r := (y^2-1) / (y-1);
```

$$r := \frac{y^2 - 1}{y - 1}$$

```
> type(r, ratpoly); # check if rational expression
```

```
true
```

```
> whattype(r);
```

```
*
```

```
> op(r);
```

$$y^2 - 1, \frac{1}{y - 1}$$

```
> op(2, r);
```

$$\frac{1}{y - 1}$$

```
> whattype(%);
```

```
> op(%);
```

```

 $y - 1, -1$ 
> normal(r); # normal form
 $y + 1$ 

```

Here, you see again that the internal data structure may differ from the external form shown on the worksheet or terminal screen: the rational expression is a product of the numerator and the denominator raised to the power  $-1$ . If we consider the expression as an element of the field  $\mathbb{R}(y)$ , then it can be normalized to the polynomial  $y + 1$ . Considered as real functions,  $\frac{y^2 - 1}{y - 1}$  and  $y + 1$  are different. In other words: in the generic case ( $y \neq 0$ ) you can convert  $\frac{y^2 - 1}{y - 1}$  into  $y + 1$ .

Let us now look at the following expression:

```

> r := (sin(x)^2-1)/(sin(x)-1);
 $r := \frac{\sin(x)^2 - 1}{\sin(x) - 1}$ 
> type(r, ratpoly);
false

```

Maple agrees that this is not a rational function. But if you replace  $\sin x$  by  $y$ , then you get back the previous rational function. This is also revealed when you inspect the internal data structure of  $\frac{\sin^2 x - 1}{\sin x - 1}$  with the procedures **op**, **nops**, and **whattype**. Compare the two expression trees in Figure 6.9.

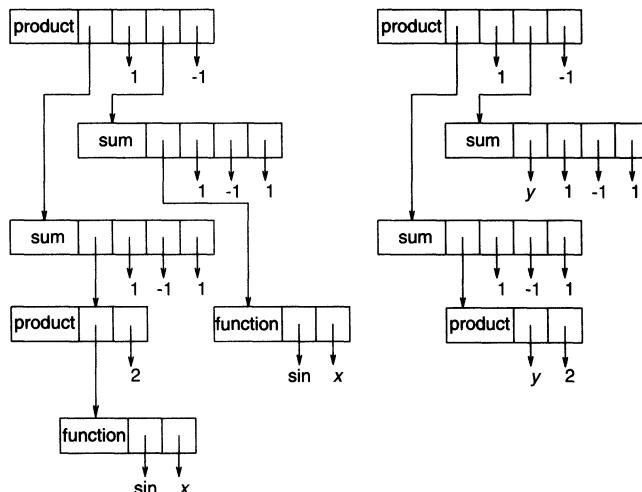


Figure 6.9. The representation trees of  $\frac{\sin^2 x - 1}{\sin x - 1}$  and  $\frac{y^2 - 1}{y - 1}$ .

So, we may say that  $\frac{\sin^2 x - 1}{\sin x - 1}$  is a rational expression in  $\sin x$  with integer coefficients. Maple agrees.

```
> type(r, ratpoly(integer, sin(x)));
                                         true
> normal(r);
                                         sin(x) + 1
```

It explains the following result.

```
> normal(r);
                                         sin(x) + 1
```

Maple considers  $\frac{\sin^2 x - 1}{\sin x - 1}$  as a *generalized rational expression*. When it applies the procedure **normal**, several things happen. First, the argument of every function call to an elementary transcendental function (trigonometric function, logarithm, exponential function, square root function, etc.) is recursively normalized before it is “frozen” to a unique name. Next, **normal** is applied to the rational expression. Finally, the frozen expressions are “thawed” again. This freezing and thawing of subexpressions is done automatically in Maple for the procedures **normal** and **factor**. Sometimes you have to help Maple in the process of considering a general expression as a rational expression via the procedure **frontend**.

```
> num := numer(r);   den := denom(r);
                                         num := sin(x)^2 - 1
                                         den := sin(x) - 1
> gcdex(num, den, sin(x));
Error, (in gcdex) invalid arguments
> frontend(gcdex, [num,den,sin(x)]);
                                         sin(x) - 1
```

The procedure **gcdex** expects polynomials over the rational numbers. **frontend** is employed to “temporarily freeze”  $\sin(x)$  in **num** and **den** to a unique name. On the intermediate results we can apply the procedure **gcdex**. Finally, the frozen names are thawed. In the next section, we shall get acquainted with another method of freezing subexpressions.

## 6.3 Substitution

An important tool to manipulate expressions is *substitution*. Huge expressions, which are difficult to read, can be simplified by replacement of large

subexpressions by smaller ones. For example, the matrix

$$\begin{pmatrix} a & b & e & f & a & b \\ c & d & g & h & c & d \\ -e & -f & a & c & e & f \\ -g & -h & b & d & g & h \\ -a & -b & -e & -f & e & g \\ -c & -d & -g & -h & f & h \end{pmatrix},$$

can be rewritten in block form as

$$\begin{pmatrix} X & Y & X \\ -Y & X^T & Y \\ -X & -Y & Y^T \end{pmatrix},$$

where

$$X = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \quad \text{and} \quad Y = \begin{pmatrix} e & f \\ g & h \end{pmatrix}.$$

On many occasions it will be more convenient to use the block matrix instead of the original matrix.

Maple introduces abbreviations in the printing of large formulae by labeling large subexpressions as %1, %2, and so on.

As a rule, you use the procedure **subs** for substitution. The simplest form of a substitution command is

**subs(*var* = *replacement*, *expression*).**

The effect is that *replacement* is substituted for all occurrences of the variable *var* in the *expression*. An example:

```
> kinetic_energy := momentum^2/(2*mass);
kinetic_energy :=  $\frac{\text{momentum}^2}{2 \text{mass}}$ 
> subs(momentum=mass*velocity, kinetic_energy);
 $\frac{\text{mass velocity}^2}{2}$ 
> kinetic_energy;
 $\frac{\text{momentum}^2}{2 \text{mass}}$ 
```

Note that the variable **kinetic\_energy** keeps its value after substitution. It can only be changed by an assignment.

```
> kinetic_energy := subs(momentum=mass*velocity,
> kinetic_energy):
> kinetic_energy;
```

$\frac{\text{mass velocity}^2}{2}$

The next two examples show that the result of a substitution is simplified but not evaluated. You have to take care of this yourself.

```
> # substitution, but no evaluation
> subs(x=0, cos(x)*(sin(x)+x^2+1));
cos(0) (sin(0) + 1)
> %; # extra evaluation
1
> sum(1/binomial(n,k), k=1..n);

$$\sum_{k=1}^n \frac{1}{\text{binomial}(n, k)}$$

> subs(n=3, %) = eval(subs(n=3,%));

$$\sum_{k=1}^3 \frac{1}{\text{binomial}(3, k)} = \frac{5}{3}$$

```

In this case, the result of the finite addition of terms is more easily done with the **eval** procedure alone, while specifying the condition  $n = 3$  as a second argument.

```
> eval(%%, n=3);
5
3
```

If you want evaluation to be performed automatically after substitution, then one way of achieving this is to enter the following command (or to put it in your initialization file).

```
> macro(subs=eval@subs):
```

See also §8.7.

As an example of multiple substitution, we consider a double substitution, which is specified in Maple as

**subs(*var1* = *replacement1*, *var2* = *replacement2*, *expression*).**

The effect of this double substitution is that first *replacement1* is substituted for all occurrences of *var1* in the *expression*, and then *var2* is replaced in the intermediate result by *replacement2*. More generally, in multiple substitutions the specified replacements are applied to intermediate expressions from left to right. We call this form of multiple substitution *sequential substitution*. Two examples:

```
> kinetic_energy := momentum^2/(2*mass);
kinetic_energy :=  $\frac{\text{momentum}^2}{2 \text{mass}}$ 
> subs(momentum=mass*velocity,
>       velocity=acceleration*time, kinetic_energy);
```

```


$$\frac{\text{mass acceleration}^2 \text{time}^2}{2}$$

> expression := 1+tan(x)^2;
      expression := 1 + \tan(x)^2
> subs(tan(x)=sin(x)/cos(x), sin(x)^2=1-cos(x)^2,
      expression);
      1 + \frac{1 - \cos(x)^2}{\cos(x)^2}
> normal(%);
      \frac{1}{\cos(x)^2}

```

Instead of sequential substitution you can also carry out so-called *simultaneous substitution*, simply by grouping the substitution equations together in a set or a list.

**subs( { var1 = replacement1, var2 = replacement2 }, expression )**

In simultaneous substitution, the replacements in the expression take place all at once. Better stated, Maple traverses the representation tree, inspects each subexpression, and going from left to right through the set of substitution equations, checks whether there is a match between the left-hand side of a substitution equation and the particular subexpression; if so, the replacement is carried out in the original expression and not in intermediate results. The following examples show you better the difference between sequential and simultaneous substitution.

```

> subs(x=y, y=z, x*y^2);    # sequential substitution
      z^3
> subs({x=y, y=z}, x*y^2); # simultaneous substitution
      y z^2
> subs(a=b, b=c, c=a, a+2*b+3*c);
      6 a
> subs({a=b, b=c, c=a}, a+2*b+3*c);
      b + 2 c + 3 a
> subs({p=q, q=p}, f(p,q)); # interchange p and q
      f(q, p)
> [a+b=d, a=b^2, b=c^2];
      [a + b = d, a = b^2, b = c^2]
> subs(% , (a+b)^2*b*a);

```

$$d^2 c^2 b^2$$

Not only variables can be replaced in a symbolic expression by the procedure **subs**, but also larger subexpressions. In this case, substitution is limited to subexpressions that are recognized as such by Maple. This means that only subexpressions that can be reached by (several applications of) the procedure **op** are allowed to play a role in substitution. For example,

```
> expr1 := x*y+z;   expr2 := x*y*z;   expr3 := (x*y)^2;
expr1 := x y + z
expr2 := x y z
expr3 := x2 y2
> subs(x*y=w, [expr1, expr2, expr3]);
[w + z, x y z, x2 y2]
> op(expr1);
x y, z
> op(expr2);
x, y, z
> op(expr3);
x2, y2
```

Because the product **x\*y** in the second and third expression is not recognized by Maple as a subexpression, the product cannot be easily replaced. If such a substitution is desired, you will just have to make do with solutions like

```
> subs(x=w/y, expr2);
w z
> subs(x=w/y, expr3);
w2
> subs({x=w, y=1}, expr2);
w z
> subs({x=w, y=1}, expr3);
w2
> subs(x*y*z=w*z, expr2);
w z
```

The last substitution only makes sense when **expr2** is part of a larger formula that contains **x**, and other terms involving **x** should stay intact. Alternatively you can use pattern matching, i.e., you specify the substi-

tutions as rules that must be applied repeatedly until no change occurs. Simply, use the procedure **applyrule**.

```
> applyrule(x*y=w, expr2);
```

 $w z$ 

In the same manner:

```
> expression := a+b+c;
```

 $expression := a + b + c$ 

```
> subs(a+b=d, expression);
```

 $a + b + c$ 

```
> subs(a=d-b, expression);
```

 $d + c$ 

```
> subs({a=d, b=0}, expression);
```

 $d + c$ 

```
> subs(a+b+c=d+c, expression);
```

 $d + c$ 

```
> applyrule(a+b=d, expression);
```

 $d + c$ 

Although **applyrule** is more powerful than the **subs**, it does not do “mathematical” transformations, but solely relies on syntactic substitution. See the example below:

```
> applyrule(a+b=d, a+2*b+3*c);
```

 $a + 2b + 3c$ 

Maple contains the procedure **algsubs** (algebraic substitution) for more advanced purposes. Applied to the above example, you get:

```
> algsubs(a+b=d, a+2*b+3*c);
```

 $d + b + 3c$ 

Algebraic substitution can be ambiguous. For example, what should the result of

 $\text{algsubs}(a + b = d, a + 2b + 3c)$ 

actually be?

 $b + d + 3c$ 

or

 $-a + 2c + 2d?$

And if both answers should be possible, how can you arrange this? This problem is partly solved in **algsubs** by the optional arguments in which you can specify

- the ordering of the variables
- one of the options **exact** and **remainder**

The variable ordering allows you to inform Maple which variables you prefer to see in the answer. In **remainder** mode (the default mode) a generalized polynomial division with remainder is used to find the occurrences of a pattern. In **exact** mode, if the pattern is a sum of terms, say  $f_1 + f_2$ , then replacement is only made in a subexpression, say  $t_1 + t_2$ , if and only if  $f_1 + f_2 = c \times (t_1 + t_2)$  for some coefficient  $c$ . Notice that we speak of polynomial division: **algsubs** only works for patterns and expressions that consist of monomials with integral exponents. Thus, a pattern or an expression like  $x^{\frac{1}{2}}$  is not taken into account by **algsubs**. Examples make things clearer.

```
> algsubs(a+b=d, a+2*b+3*c, 'exact');
          a + 2 b + 3 c
> algsubs(a+b=d, a+2*b+3*c, [a,b]); # a<b
          d + b + 3 c
> algsubs(a+b=d, a+2*b+3*c, [b,a]); # b<a
          -a + 2 d + 3 c
> algsubs(a^2+1=d, (a^2+1)^4+a^3+2*a^2+1);
          d^4 + (-1 + d) a - 1 + 2 d
> algsubs(a^2+1=d, (a^2+1)^4+a^3+2*a^2+1, 'exact');
          d^4 + a^3 + 2 a^2 + 1
> subs(a^2=d-1, (a^2+1)^4+a^3+2*a^2+1);
          d^4 + a^3 - 1 + 2 d
```

Truncating powers is an important application of **algsubs**.

```
> sum(x^k, k=-5..5);
          1 +  $\frac{1}{x^5} + \frac{1}{x^4} + \frac{1}{x^3} + \frac{1}{x^2} + \frac{1}{x} + x + x^2 + x^3 + x^4 + x^5$ 
> algsubs(x^3=0, %);
          1 +  $\frac{1}{x^5} + \frac{1}{x^4} + \frac{1}{x^3} + \frac{1}{x^2} + \frac{1}{x} + x + x^2$ 
> algsubs(1/x^3=0, %);
          1 +  $\frac{1}{x^2} + \frac{1}{x} + x + x^2$ 
```

**algsubs** also works for generalized rational expressions.

```
> sin(x)^2/(sin(x)^3+cos(x)^3);

$$\frac{\sin(x)^2}{\sin(x)^3 + \cos(x)^3}$$

> algsubs(sin(x)^2+cos(x)^2=1, %);

$$\frac{-\cos(x)^2 + 1}{\sin(x) - \cos(x)^2 \sin(x) + \cos(x)^3}$$

> algsubs(sin(x)^2+cos(x)^2=1, %%, [cos(x),sin(x)]);

$$\frac{\sin(x)^2}{\cos(x) - \sin(x)^2 \cos(x) + \sin(x)^3}$$

```

**algsubs** differs in some technical terms from **subs**:

- no sequential or simultaneous substitution in one command;
- automatic evaluation after substitution;
- no action on procedure names or indices.

Let us end with an example from polynomial calculus.

```
> algsubs(x*y^2=s, x^3*y^4);

$$s^2 x$$

> algsubs(x*y^2=x*y, x^3*y^4);

$$x^3 y^2$$

```

The last example shows that the replacement may contain variables that are part of the pattern. In our example, the expression  $x^3y^4$  can be considered as  $x(xy^2)^2$ , which explains the result of substitution. You could also make implicitly use of the Gröbner basis package **Groebner** by applying the procedure **simplify** with respect to side relations.

```
> simplify(x^3*y^2, {x*y=s}, [y,x,s]);

$$s^2 x$$

> simplify(x^3*y^4, {x*y^2=x*y}, [y,x,s]);

$$x^3 y$$

```

The last example shows that the side relation as a whole is taken into account, i.e., the right-hand side is part of the rule that is applied. We shall come back to simplification with respect to side relations in §14.7.

When Maple checks whether a certain expression occurs as a subexpression of a given symbolic expression, the computer algebra system traverses the entire representation tree. Substitution is of a global nature in Maple:

every occurrence of the subexpression in the original symbolic expression or intermediate results is substituted.

```
> expression := (x+y)^2+x;
          expression := (x + y)² + x
> op(expression);
          (x + y)², x
> op(op(1,(x+y)^2));
          x, y
> subs(x=z, expression);
          (z + y)² + z
```

So, in case of substitution, it is important to know how a Maple expression is built-up and which subexpressions are recognized by the system.

Keep in mind that Maple stores common subexpressions only once. During substitution with **subs** all occurrences of a subexpression are substituted. There is another possibility: by **subsop** you can substitute one or more operands of a Maple expression. For example, the effect of the command

**subsop( num1 = replacement1, num2 = replacement2, expression )**

is that the subexpressions **op(num1, expression)** and **op(num2, expression)** of the given *expression* are simultaneously replaced by *replacement1* and *replacement2*, respectively. The effect of the substitution

**subsop( [ i, j ] = replacement, expression )**

is that the *j*th operand of the *i*th operand in the internal representation of the given expression is replaced. A few examples to show how it works.

```
> expression := x^2+x+1/x;
          expression := x² + x +  $\frac{1}{x}$ 
> subsop(3=y, expression); # replace 3rd operand
          x² + x + y
> subsop(3=0, expression); # replace 3rd operand
          x² + x
> subsop(1=z, 2=y, expression);
          z + y +  $\frac{1}{x}$ 
> subsop([3,1]=y, expression);
```

```


$$x^2 + x + \frac{1}{y}$$

> 1 + 1/(1+cos(x^2+1/x^2));

$$1 + \frac{1}{1 + \cos(x^2 + \frac{1}{x^2})}$$

> subsop([2,1,2,1,2]=y, %);

$$1 + \frac{1}{1 + \cos(x^2 + y)}$$

> subsop(0=J, BesselJ(1,x));

$$J(1, x)$$


```

These examples show you the local nature of the procedure **subsop**. Only specified parts of an expression are replaced, and all other operands are left alone. This is opposite to the global nature of **subs**, with which you can replace one or more subexpressions, irrespective of where these subexpressions occur in the given symbolic expression. If you only have to manipulate part of a symbolic expression, you can exploit the local nature of **subsop**. Another advantage of the procedure **subsop** is that you do not have to retype the perhaps very large subexpression that you want to replace. Two examples:

```

> poly:= (x^2+y^2+2*x*y)*((x+y)^2+1);

$$poly := (x^2 + y^2 + 2 x y) ((x + y)^2 + 1)$$

> factor(poly);

$$(x + y)^2 (x^2 + y^2 + 2 x y + 1)$$

> subsop(1=factor(op(1, poly)), poly);

$$(x + y)^2 ((x + y)^2 + 1)$$

> expression := (x^2+2*x+1)^2 + (x^2-2*x+1)^2;

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

> factor(expression);

$$2 x^4 + 12 x^2 + 2$$

> subsop(1=factor(op(1,expression)),
> 2=factor(op(2,expression)), expression);

$$(x + 1)^4 + (x - 1)^4$$


```

Because such mathematical operations on operands of expressions are often needed, there are two library procedures that make life easier, viz., **applyop** and **map**. Let us illustrate them via the same examples:

```
> applyop(factor, 1, poly);
```

$$(x + y)^2 ((x + y)^2 + 1)$$

Indeed,

**applyop**(*func, index, expression*)

is the same as

**subsop**(*index = func(op(index, expression))*, *expression*)

In the second example you want to apply a function to all operands at level one of the expression. The procedure **map** is fit for this purpose.

```
> map(factor, expression);
```

$$(x + 1)^4 + (x - 1)^4$$

The command **map**(*procedure, expression*) has the following effect: apply the *procedure* to all operands of the *expression* separately, combine the results into an expression of the original data type, and carry out automatic simplification (which may change the type of the expression). In §8.8 we shall come back to the procedure **map**.

Let us note that the substitution in the first of the above two examples could also have been carried out as follows.

```
> subs(x+y=z, poly);
```

$$(x^2 + y^2 + 2xy)(z^2 + 1)$$

```
> factor(%);
```

$$(x + y)^2 (z^2 + 1)$$

```
> subs(z=x+y, %);
```

$$(x + y)^2 ((x + y)^2 + 1)$$

This technique of temporary replacement of a subexpression by an unknown, manipulation of the intermediate result, and back substitution of the “frozen” subexpression, is a frequently used simplification strategy. What follows is another illustration of this method.

```
> expression := (x+y)^2 + 1/(x+y)^2;
```

$$\text{expression} := (x + y)^2 + \frac{1}{(x + y)^2}$$

We want to change this expression into one of the form  $\frac{\text{numerator}}{\text{denominator}}$  without expansion of the powers  $(x+y)^2$ . The procedure **normal** does not quite do what we want.

```
> normal(expression);
```

$$\frac{x^4 + 6x^2y^2 + 4x^3y + 4xy^3 + y^4 + 1}{(x+y)^2}$$

If we temporarily replace the subexpression  $x+y$  in the powers by a new unknown, say  $z$ , normalize the intermediate rational function, and finally substitute back  $x+y$  for  $z$ , then we get the desired result.

```
> subs(x+y=z, expression);
```

$$z^2 + \frac{1}{z^2}$$

You could also use the procedure **applyrule** to replace powers of  $x+y$  by equivalent powers of  $z$ .

```
> applyrule((x+y)^n::integer=z^n, expression);
```

$$z^2 + \frac{1}{z^2}$$

```
> normal(%);
```

$$\frac{z^4 + 1}{z^2}$$

```
> subs(z=x+y, %);
```

$$\frac{(x+y)^4 + 1}{(x+y)^2}$$

Because it is not always easy in a long Maple session to recall which variables are already in use and which substitutions have already been applied, the Maple system offers you the procedures **freeze** and **thaw** as utility functions. When you use them, Maple itself chooses new names for subexpressions and keeps track of them. The above example would look like

```
> subs(x+y=freeze(x+y), expression);
```

$$freeze/R0^2 + \frac{1}{freeze/R0^2}$$

```
> normal(%);
```

$$\frac{freeze/R0^4 + 1}{freeze/R0^2}$$

```
> thaw(%);
```

$$\frac{(x+y)^4 + 1}{(x+y)^2}$$

We end this section with some simple examples that illustrate the problem of specialization.

```
> integrate(1/(x-1)^2, x=0..a);
```

$$-\frac{a}{a-1}$$

Maple has successfully solved a generic problem ( $a$  is an indeterminate with no value associated). When you specialize  $a$ , you may get trapped in strange results. For example, substituting  $a = 2$  in this integral gives a negative result of integration of a positive integrand

```
> subs(a=2, %);
-2
```

Maple would have gotten the correct answer if you had immediately asked for the  $\int_0^2 \frac{1}{(x-1)^2} dx$ .

```
> integrate(1/(x-1)^2, x=0..2);
∞
```

Fortunately, making the assumption in Maple that  $a$  is greater than 1 would already help. But making assumptions does not always help.

Consider the following equation, which depends on the parameter  $a$ .

```
> eq := (a^2-1)*x^2 + (2*a^2+a-3)*x + 2*a - 2 = 0;
eq := (a² - 1) x² + (2 a² + a - 3) x + 2 a - 2 = 0
```

For  $a = 1$  the equation is trivial.

```
> subs(a=1, %);
0 = 0
```

However, solving the generic case and substituting  $a = 1$  afterwards gives only two solutions.

```
> solve(eq, x);
-2, -1/(a+1)
> subs(a=1, %);
{-2, -1/2}
```

Consider the matrix  $M = \begin{pmatrix} 1 & a \\ 0 & 1 \end{pmatrix}$  and compute the Jordan form using the linear algebra package **LinearAlgebra**. By the way, the results are the same if you use the **linalg** package.

```
> with(LinearAlgebra):
> A := Matrix([[1,a],[0,1]]);
A := [ 1 a ]
      [ 0 1 ]
> JordanForm(%);
```

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

Obviously, this result does not hold for  $a = 0$ .

```
> B := Matrix([[a,1],[0,1]]);  
B :=  $\begin{bmatrix} a & 1 \\ 0 & 1 \end{bmatrix}$   
> JordanForm(%);  
 $\begin{bmatrix} 1 & 0 \\ 0 & a \end{bmatrix}$ 
```

This previous result is not valid for  $a = 1$ .

## 6.4 Exercises

1. Describe in detail the internal representation in Maple of the polynomial  $2x(y^2 + 1)^2$ . Draw the corresponding directed acyclic graph (DAG) and the representation tree. Write down all subexpressions that are recognized as such by Maple. Enlarge your trust in your answer by use of the procedures **whattype**, **nops**, and **op**. By use of the procedures **dismantle**, **addressof**, **pointto**, **disassemble**, and **assemble** you can completely check your answer.
2. Transform  $(x + y)^2 + \frac{1}{x + y}$  into  $\frac{(x + y)^3 + 1}{x + y}$  and vice versa.
3. Transform  $x^2 + 2x + 1 + \frac{1}{x^2 + 2x + 1}$  into  $\frac{(x + 1)^4 + 1}{(x + 1)^2}$  and vice versa.
4. Explain the result of the following substitution:

```
> x^2-x+1/x-1/x^2;
```

$$x^2 - x + \frac{1}{x} - \frac{1}{x^2}$$

```
> subs(-1=1, %);
```

$$x^2 + 2x + \frac{1}{x^2}$$

5. Transform  $\frac{(x + 1)^{10} - 2y}{(x + y)^{10}} + \frac{1}{(x + y)^9} - \frac{x}{(x + y)^{10}}$  into  $\frac{(x + 1)^{10} - y}{(x + y)^{10}}$ .

# 7

## Manipulation of Polynomials and Rational Expressions

In this chapter we shall systematically discuss manipulations of polynomials and rational expressions. The following manipulations pass under review: expansion, factorization, normalization, collection, and sorting of polynomials and rational expressions. We shall consider manipulation of expressions that are defined over the rational numbers as well as manipulation of expressions that are defined over other domains, like finite fields, algebraic numbers, and algebraic function fields. Furthermore, we shall give a short, theoretical introduction to canonical and normal form simplification.

### 7.1 Expansion

Suppose that you want to transform  $(x^2 - x)(x^2 + 2x + 1)$  from this factored form into the expanded canonical form  $x^4 + x^3 - x^2 - x$ . Then you can use the procedure **expand**, and if necessary **sort** the result. In **expand**, the factors are first multiplied out, and secondly similar terms are collected. In our example, the first step leads to  $x^4 + 2x^3 + x^2 - x^3 - 2x^2 - x$ , and then Maple simplifies this to  $x^4 + x^3 - x^2 - x$  (or eventually to a different ordering if the polynomial was already in memory in a different form; in this case you need the procedure **sort** to rearrange terms).

```
> partially_factored_form := (x^2-x)*(x^2+2*x+1);
```

```
partially_factored_form := (x2 - x) (x2 + 2 x + 1)
```

```
> expanded_form := expand(partially_factored_form);
expanded_form :=  $x^4 + x^3 - x^2 - x$ 
```

The way in which Maple distributes products over sums can be illustrated best by an easier example.

```
> factored_form := (a+b)*(c+d);
factored_form :=  $(a + b)(c + d)$ 
> expanded_form := expand(factored_form);
expanded_form :=  $a c + a d + b c + b d$ 
```

With the procedure **expand** all products are distributed over sums and like terms are automatically collected by Maple's simplifier. The terminology *full expansion* is more appropriate. For example, in our last example, the expansion of the factored form can be described as a two-step mechanism: first, the factored form is transformed to the expression  $a(c+d) + b(c+d)$ , and after that the intermediate expression is transformed to the final form  $ac + ad + bc + bd$ . If you want to avoid this second step, than you must specify in the call of the procedure **expand** that you want to keep the subexpression  $c + d$  intact.

```
> partially_expanded_form := expand(factored_form, c+d);
partially_expanded_form :=  $(c + d)a + (c + d)b$ 
```

The procedure **expand** also distributes powers with positive integer exponents over sums, simply by considering these powers as repeated products and by distributing them over sums.

```
> power := (x+1)^3;
power :=  $(x + 1)^3$ 
> expand(%);
 $x^3 + 3x^2 + 3x + 1$ 
```

Negative powers are left untouched by **expand**.

```
> power := (x+1)^(-2);
power :=  $\frac{1}{(x + 1)^2}$ 
> whattype(%);
 $\hat{}$ 
> op(power);
 $x + 1, -2$ 
> expand(power);
```

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

If you consider this expression as a rational one of which the denominator is equal to  $(x+1)^2$ , then you can expand the denominator separately.

```
> 1/expand(denom(%));
```

$$\frac{1}{x^2 + 2x + 1}$$

Maple does not expand powers with non-integer exponents.

```
> power := (x+1)^(3/2);
```

$$power := (x+1)^{(3/2)}$$

```
> expand(%);
```

$$(x+1)^{(3/2)}$$

The only effect of **expand** on rational expressions is that sums in the numerator are expanded.

```
> (x+1)^2/((x^2+x)*x);
```

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

```
> expand(%);
```

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

So far, we have only looked at expansion of polynomials and rational functions defined over the integers. When you manipulate polynomials that are defined over a finite ring, over algebraic numbers, or over algebraic function fields, then you must use the inert form of expansion, namely **Expand**, if necessary in combination with **evala** (evaluate in a context of algebraic numbers or function fields). A few examples will do.

- Expansion over  $\mathbb{Z}_8$ .

```
> Expand((x+1)^8) mod 8;
```

$$x^8 + 4x^6 + 6x^4 + 4x^2 + 1$$

- Expansion over  $\mathbb{Q}(\alpha)$ , where  $\alpha$  is a root of the polynomial  $z^5 + z + 1$ .

```
> alias(alpha=RootOf(z^5+z+1, z));
> (x+alpha)^5;
```

$$(x+\alpha)^5$$

```
> evala(Expand(%));
```

$$x^5 + 5x^4\alpha + 10x^3\alpha^2 + 10x^2\alpha^3 + 5x\alpha^4 - 1 - \alpha$$

- Expansion over  $\mathbb{Z}_5(\alpha)$ , where  $\alpha$  is a root of the polynomial  $z^5 + z + 1$ .

> `Expand(%%) mod 5;`

$$x^5 + 4\alpha + 4$$

- Expansion over the algebraic function field  $\mathbb{Q}(\sqrt{1+y})$ .

Recall that an algebraic function over  $\mathbb{Q}$  is a function that annihilates a univariate polynomial with coefficients in a rational function field. In our example,  $\sqrt{1+y}$  is defined as a root of the polynomial  $z^2 - 1 - y$  with coefficients in the rational function field  $\mathbb{Q}(y)$ .

> `alias(beta=RootOf(z^2-1-y, z)):`  
> `(x+beta)^2;`

$$(x + \beta)^2$$

> `evala(Expand(%));`

$$x^2 + 2x\beta + 1 + y$$

## 7.2 Factorization

The Maple procedure **factor** is **expand**'s big brother. It computes the factorization of a polynomial over the rationals in irreducible factors. The result is unique up to ordering of factors; we speak of a *factored normal form*. The procedure also works fine for rational expressions, in that common factors in numerator and denominator are first canceled before they are factored.

In §5.1 we have already seen that Maple can factor polynomials over various domains. Below are some more examples. In case of polynomials over finite fields and algebraic function fields, you may have to use the inert form **Factor**. The algorithms implemented in Maple are: Cantor-Zassenhaus [44] and Berlekamp [21] for univariate polynomials over finite fields, Hensel lifting for multivariate polynomials [245], and Lenstra [164] and Kronecker-Trager [223] for polynomials over algebraic number and function fields. Polynomial factorization of polynomials over rational numbers uses van Hoeij's knapsack algorithm [131].

- Factorization over  $\mathbb{Q}(\sqrt{6})$ .

> `factor(8*x^3-12*x, sqrt(6));`

$$2x(2x - \sqrt{6})(2x + \sqrt{6})$$

- Factorization over finite fields  $\mathbb{Z}_2$ ,  $\mathbb{Z}_3$ , and  $\mathbb{Z}_5$ .

> `x^4+1;`

$$x^4 + 1$$

```

> Factor(%) mod 2;

$$(x + 1)^4$$

> Factor(%%) mod 3;

$$(x^2 + 2x + 2)(x^2 + x + 2)$$

> Factor(%%%) mod 5;

$$(x^2 + 3)(x^2 + 2)$$


```

- Factorization over  $\mathbb{Z}_7(\alpha)$ , where  $\alpha$  is a root of the polynomial  $z^7 + z^3 + 1$ .

```

> alias(alpha=RootOf(z^7+z^3+1, z));
> x^7+6*alpha^3+6;

$$x^7 + 6\alpha^3 + 6$$

> Factor(%) mod 7;

$$(x + \alpha)^7$$


```

- Factorization over the algebraic function field  $\mathbb{Q}(\sqrt{1+y})$ .

```

> alias(beta=RootOf(z^2-1-y, z));
> x^2+2*beta*x+1+y;

$$x^2 + 2\beta x + 1 + y$$

> factor(x^2+2*beta*x+1+y, beta);

$$(x + \beta)^2$$


```

The extension of the field of rational numbers with the square root of six appeared more or less out of nothing to yield the complete factorization of  $8x^3 - 12x$ . But Maple itself can compute the splitting field of a univariate polynomial with the procedure **Split** from the **PolynomialTools** package.

```

> PolynomialTools:-Split(8*x^3-12*x, x);

$$8x(x + \frac{1}{2}\text{RootOf}(-Z^2 - 6))(x - \frac{1}{2}\text{RootOf}(-Z^2 - 6))$$

> convert(%, radical);

$$8(x + \frac{\sqrt{6}}{2})(x - \frac{\sqrt{6}}{2})x$$


```

An example of a multivariate polynomial that is irreducible over  $\mathbb{Q}$ .

```

> p := x^4+2*y^4;

$$p := x^4 + 2y^4$$


```

- Factorization over extension of  $\mathbb{Q}$  with  $\sqrt{2}$  and  $i (= \sqrt{-1})$ .

```

> factor(p, {sqrt(2), I});

$$-(-x^2 + Iy^2\sqrt{2})(x^2 + Iy^2\sqrt{2})$$


```

- Factorization over  $\mathbb{Z}_3$ .

```
> Factor(p) mod 3;
```

$$(x + 2y)(x^2 + y^2)(x + y)$$

- Factorization over Galois field GF(9), in particular the algebraic extension of  $\mathbb{Z}_3$  with respect to the irreducible polynomial  $x^2 - 2$ .

```
> alias(alpha=RootOf(z^2-2, z));
> Factor(p, alpha) mod 3;
```

$$2(y + 2\text{RootOf}(-Z^2 + 1)x)(y + 2x)(x + y)(y + \text{RootOf}(-Z^2 + 1)x)$$

```
> convert(%, radical);
```

$$2(y + 2Ix)(y + 2x)(x + y)(y + Ix)$$

**AFactor** does absolute factorization of multivariate polynomials over the complex numbers.

```
> evala(AFactor(x^3+y^3));
> convert(%, radical); # go from RootOf to radical notation
```

$$(x + (-\frac{1}{2} - \frac{\sqrt{-3}}{2})y)(x + (-\frac{1}{2} + \frac{\sqrt{-3}}{2})y)(x + y)$$

```
> simplify(%); # change into complex notation
```

$$\frac{(2x - y - y\sqrt{3}I)(2x - y + y\sqrt{3}I)(x + y)}{4}$$

One of the first steps in the factorization of a polynomial by **factor** is the *square-free factorization* of the polynomial. The square-free factorization of a non-constant polynomial is defined as a decomposition of the form  $cp_1 p_2^2 p_3^3 \dots$ , where  $c$  is a rational constant and the polynomials  $p_1, p_2, p_3, \dots$  are relatively prime and have no multiple factors. In Maple you can compute the square-free factorization by explicit conversion or by calling the procedure **sqrfree**.

```
> x^4+x^3-x^2-x;
```

$$x^4 + x^3 - x^2 - x$$

```
> convert(%, sqrfree);
```

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

```
> sqrfree(%);
```

$$[1, [[x, 1], [x - 1, 1], [x + 1, 2]]]$$

When you compute over the field  $\mathbb{Z}_2$  with two elements, the latter polynomial has a square-free factorization equal to  $x(x + 1)^3$ :

```
> Sqrfree(%%%) mod 2;
```

$$[1, [[x, 1], [x + 1, 3]]]$$

## 7.3 Canonical Form and Normal Form

So far the terms *canonical form* and *normal form* have appeared several times in the context of simplification or manipulation of expressions. It is worthwhile to linger a bit longer on them, even though it is a more theoretical intermezzo. A detailed survey can be found in [35].

The main problem with simplification is that a mathematical expression can be written in several equivalent forms whereas it is not always easy to recognize equivalence. For example, the third Hermite polynomial is equal to  $8x^3 - 12x$ , but you can easily write down four equivalent formulae:

$$\begin{aligned} &x(8x^2 - 12), \\ &4x(2x^2 - 3), \\ &2x(2x - \sqrt{6})(2x + \sqrt{6}), \\ &(2x)^3 - 6 \cdot (2x). \end{aligned}$$

What is the simplest form? If you want to stress the fact that it is a polynomial in  $2x$ , then  $2x(2x - \sqrt{6})(2x + \sqrt{6})$  and  $(2x)^3 - 6 \cdot (2x)$  are good candidates. If you use the number of terms as criterion, then  $8x^3 - 12x$  and  $(2x)^3 - 6 \cdot (2x)$  would be your favorites.

A more clearly specified problem is the so-called *zero equivalence* problem, i.e., to recognize whether an expression is equal to zero. But even this may turn out to be a hard problem as in

$$\ln \tan\left(\frac{1}{2}x + \frac{1}{4}\pi\right) - \operatorname{arcsinh} \tan x = 0.$$

Verification of this equality is non-trivial.

In general the simplification problem can be treated as follows. Let  $\mathcal{E}$  be a class of symbolic expressions (e.g., univariate polynomials over the integers) and let  $\sim$  be an equivalence relation defined on  $\mathcal{E}$ . The problem of finding an equivalent but simpler expression can be characterized as finding a computable transformation  $\mathcal{S}: \mathcal{E} \longrightarrow \mathcal{E}$  such that for any expression  $t$  in  $\mathcal{E}$  we have

$$\mathcal{S}(t) \sim t \text{ and } \mathcal{S}(t) \preceq t.$$

Here,  $\preceq$  denotes some simplification concept:  $s \prec t$  is our notation for “expression  $s$  is simpler in  $\mathcal{E}$  than expression  $t$ ,” and this means e.g., “expression  $s$  has less terms than  $t$ ,” “ $s$  uses less memory than  $t$ ,” or “ $s$  is more readable than  $t$ .” We say that expressions  $s$  and  $t$  are identical and denote this by  $s \equiv t$ , when  $s$  and  $t$  are built up in the same way from basic elements, or more precisely when they have the same directed acyclic graph (see e.g., the previous chapter for a description of Maple’s internal data representation).

A *canonical simplifier*  $\mathcal{S}$  on  $\mathcal{E}$  is a computable procedure such that for all  $s$  and  $t$  in  $\mathcal{E}$  we have

$$\begin{aligned}\mathcal{S}(t) &\sim t \text{ and} \\ s \sim t &\implies \mathcal{S}(s) \equiv \mathcal{S}(t).\end{aligned}$$

A canonical simplifier chooses a unique representative in each equivalence class, the *canonical form*.

For univariate polynomials, the expanded canonical form can be obtained as follows:

- (i) recursively distribute all products over sums,
- (ii) collect terms of the same degree, and
- (iii) remove superfluous terms and reorder the remaining terms in descending order of their degrees.

The requirements for canonical simplification are high and cannot always be met. A weaker form of simplification is *normal form simplification*. Suppose that there is a distinguished element, say 0, in  $\mathcal{E}$ . A *normal simplifier*  $\mathcal{S}$  on  $\mathcal{E}$  is a computable procedure such that for all  $s$  and  $t$  in  $\mathcal{E}$  holds

$$\begin{aligned}\mathcal{S}(t) &\sim t \text{ and} \\ t \sim 0 &\implies \mathcal{S}(t) \equiv \mathcal{S}(0).\end{aligned}$$

An expression  $t$  is called a *normal form* in  $\mathcal{E}$  when  $\mathcal{S}(t) \equiv t$ . So, the normal form need not be unique in each equivalence class except for the class to which 0 belongs in  $\mathcal{E}$ .

For univariate polynomials, the expanded normal form, which we called collected form, can be obtained as follows:

- (i) recursively distribute all products over sums,
- (ii) collect terms of the same degree, and
- (iii) remove superfluous terms.

Normal simplification is usually easier than canonical simplification. Other examples of normal forms of polynomials are the square-free form and the Horner form, which is a nested form often used for efficiency reasons.

```
> poly := expand((x+y+z)^2*(x+y+1));
poly := x^3 + 3 x^2 y + x^2 + 3 x y^2 + 2 x y + 2 x^2 z + 4 x z y + 2 x z
      + y^3 + y^2 + 2 y^2 z + 2 y z + z^2 x + z^2 y + z^2
> with(codegen, cost): cost(poly);
14 additions + 32 multiplications
> horner_form := convert(poly, 'horner', [x,y,z]);
```

```

horner_form := z2 + ((2 + z) z + (2 z + 1 + y) y) y
      + ((2 + z) z + (2 + 4 z + 3 y) y + (2 z + 1 + 3 y + x) x) x
> cost(horner_form);
14 additions + 13 multiplications

```

Partial fraction decomposition of rational functions is an example of a normal simplifier of rational expressions.

## 7.4 Normalization

The procedure **normal** provides the normal simplification for rational functions over  $\mathbb{Q}$  in Maple. A rational function is converted into the so-called *factored normal form*. This is the form *numerator / denominator*, where the numerator and denominator are relatively prime polynomials with integer coefficients; the numerator and denominator are both products of expanded polynomials and during the simplification to normal form common factors are kept intact as much as possible.

```

> (x-1)*(x+2)/((x+1)*x) + (x-1)/(1+x)^2;

$$\frac{(x - 1)(x + 2)}{(x + 1)x} + \frac{x - 1}{(x + 1)^2}$$

> normal(%);

$$\frac{(x - 1)(x^2 + 4x + 2)}{(x + 1)^2 x}$$

> (x^2+x-2)/((x+1)*x) + (x-1)/(1+x)^2;

$$\frac{x^2 + x - 2}{(x + 1)x} + \frac{x - 1}{(x + 1)^2}$$

> normal(%);

$$\frac{x^3 + 3x^2 - 2x - 2}{(x + 1)^2 x}$$

> normal(%%, expanded);

$$\frac{x^3 + 3x^2 - 2x - 2}{x^3 + 2x^2 + x}$$


```

You specify with the keyword **expanded** that you want both numerator and denominator in expanded normal form.

There exist several normal forms for rational expressions and you can produce some of them by application of the procedures **normal**, **expand** and **factor** upon numerator and denominator separately. Alternatives are:

```
> ratfunc := (x^4+x^3-4*x^2-4*x)/(x^3+x^2-x-1);
```

- $\text{ratfunc} := \frac{x^4 + x^3 - 4x^2 - 4x}{x^3 + x^2 - x - 1}$
- *factored normal form*
  - *factored normal form*

```
> factor(ratfunc);
```

$$\frac{(x - 2)(x + 2)x}{(x - 1)(x + 1)}$$
  - *factored normal form*
  - *expanded canonical form*

```
> factor(numer(ratfunc))/sort(expand(denom(ratfunc)));
```

$$\frac{x(x - 2)(x + 2)(x + 1)}{x^3 + x^2 - x - 1}$$
  - *expanded canonical form*
  - *factored normal form*

```
> sort(expand(numer(ratfunc)))/factor(denom(ratfunc));
```

$$\frac{x^4 + x^3 - 4x^2 - 4x}{(x - 1)(x + 1)^2}$$
  - *expanded canonical form*
  - *expanded canonical form*

```
> sort(normal(ratfunc, expanded));
```

$$\frac{x^3 - 4x}{x^2 - 1}$$

Above, we have applied **normal** on a rational function, but, as was already explained in §6.2, you can also normalize a *generalized* rational expression with this procedure. We confine ourselves to one example, which explains the recursive character of **normal**.

```
> sin(x+1/x) + 1/sin(x+1/x);
       $\sin\left(x + \frac{1}{x}\right) + \frac{1}{\sin\left(x + \frac{1}{x}\right)}$ 
> normal(%);
      
$$\frac{\sin\left(\frac{x^2 + 1}{x}\right)^2 + 1}{\sin\left(\frac{x^2 + 1}{x}\right)}$$

```

The procedure **normal** has a twin brother called **Normal**. This procedure is a normal simplification for various coefficient domains. Examples:

```
> Normal(ratfunc) mod 3; # normalization modulo 3
```

```

x
> (x^2-a)/(x-sqrt(a));
      
$$\frac{x^2 - a}{x - \sqrt{a}}$$

> convert(%, RootOf);
      
$$\frac{x^2 - a}{x - \text{RootOf}(-Z^2 - a, \text{index} = 1)}$$

> evala(Normal(%)); # normalization over Q(a)
      
$$x + \text{RootOf}(-Z^2 - a, \text{index} = 1)$$

> convert(%, radical);
      
$$x + \sqrt{a}$$


```

## 7.5 Collection

The procedure **collect** is used to group coefficients of like terms in a polynomial. The various ways of collecting terms are best shown by examples.

```

> poly := expand((x+y+z)^2*(x+y+1));
poly :=  $x^3 + 3x^2y + x^2 + 3xy^2 + 2xy + 2x^2z + 4xz y + 2xz + y^3 + y^2 + 2y^2z + 2yz + z^2x + z^2y + z^2$ 

```

This is an example of the *distributed* or *expanded form* of a polynomial in Maple. We use the notation  $\mathbb{Z}[x, y, z]$  to denote the set of distributed multivariate polynomials in the unknowns  $x$ ,  $y$ , and  $z$ , with integer coefficients.

Consider **poly** as a polynomial in  $z$  whose coefficients are distributed polynomials in  $x$  and  $y$ , i.e., consider **poly** as an element in  $\mathbb{Z}[x, y][z]$ .

```

> collect(poly, z);

$$(x + y + 1)z^2 + (4xy + 2x + 2x^2 + 2y^2 + 2y)z + x^3 + 3x^2y + x^2 + 3xy^2 + 2xy + y^2 + y^3$$


```

Now consider **poly** as a polynomial in  $z$  whose coefficients are polynomials in  $y$  whose coefficients are polynomials in  $x$  with integer coefficients, i.e., consider **poly** as an element of  $\mathbb{Z}[x][y][z]$ .

```

> collect(poly, [z, y], 'recursive');

$$(x + y + 1)z^2 + (2y^2 + (4x + 2)y + 2x + 2x^2)z + y^3 + (3x + 1)y^2 + (3x^2 + 2x)y + x^3 + x^2$$


```

Loosely speaking, we have considered here **poly** as a polynomial in  $z$  and  $y$  with preference for the indeterminate  $z$ .

Of course you can also consider **poly** as a polynomial in  $z$  and  $y$  without giving preference to any of the two unknowns, i.e., you can consider the **poly** as an element of  $\mathbb{Z}[x][y, z]$ .

```
> collect(poly, [z,y], 'distributed');
```

$$\begin{aligned} &x^3 + x^2 + (3x^2 + 2x)y + (2x + 2x^2)z + y^3 + (3x + 1)y^2 + (4x + 2)yz \\ &+ (x + 1)z^2 + 2y^2z + z^2y \end{aligned}$$

Finally, you can add as an argument to **collect** the name of a procedure that has to be applied to each coefficient separately.

```
> collect(poly, z, factor); # recursive order by default
```

$$(x + y + 1)z^2 + 2(x + y + 1)(x + y)z + (x + y + 1)(x + y)^2$$

```
> collect(poly, [z,y], 'distributed', factor);
```

$$\begin{aligned} &x^3 + x^2 + (3x^2 + 2x)y + (2x + 2x^2)z + y^3 + (3x + 1)y^2 + (4x + 2)yz \\ &+ (x + 1)z^2 + 2y^2z + z^2y \end{aligned}$$

The main reasons for using collected forms of multivariate polynomials in Maple are readability and efficiency with respect to memory and computing time. If a polynomial is dense in some variable, it is advantageous to collect with respect to this variable because this will reduce representation space compared to the expanded form. Collecting terms is a way to overcome the length limitation of internal Maple objects. In Maple, a sum with more than 65535 ( $= 2^{16} - 1$ ) terms cannot be represented; the system will print

**System Error, object too large**

By maintaining the expression collected in some of the indeterminates, you may be able to reduce the maximum size of the sums in the expression. In our example above, the expanded form of **poly** has 15 terms, whereas the collected form in  $\mathbb{Z}[x, y][z]$  with all coefficients in factored form has only 3 terms.

Like **normal**, the procedure **collect** can be applied to generalized rational expressions.

```
> r := ln(x)^3/a + ln(x)^2*x/(a^2+a) + a^2*ln(x)^2*x/(a^2+a)
>      + 2*x^2/(1+a)+a*x^2/(1+a) + a^3*ln(x)/(a^2+a)
>      + 2*ln(x)*a/(a^2+a) + ln(x)/(a^2+a) + a/(a^2+a);
```

$$\begin{aligned} r := &\frac{\ln(x)^3}{a} + \frac{\ln(x)^2 x}{a^2 + a} + \frac{a^2 \ln(x)^2 x}{a^2 + a} + \frac{2 x^2}{1 + a} + \frac{a x^2}{1 + a} + \frac{a^3 \ln(x)}{a^2 + a} \\ &+ \frac{2 \ln(x) a}{a^2 + a} + \frac{\ln(x)}{a^2 + a} + \frac{a}{a^2 + a} \\ > &\text{collect}(r, [x, ln(x)], 'distributed', normal); \end{aligned}$$

$$\frac{1}{1+a} + \frac{(1+a^2)x\ln(x)^2}{a(1+a)} + \frac{\ln(x)^3}{a} + \frac{(1+a^3+2a)\ln(x)}{a(1+a)} + \frac{(2+a)x^2}{1+a}$$

The **LargeExpressions** package offers the procedures **Veil** and **Unveil** to hide information and to reveal hidden information. The following example shows its functionality in relation with the collection of terms.

```
> with(LargeExpressions):
> collect(r, [x,ln(x)], Veil[C]);

$$C_1 x^2 + C_2 \ln(x)^2 x + C_3 \ln(x)^3 + C_4 \ln(x) + C_5$$

> [seq(C[i]=Unveil[C](C[i]), i=1..5)];
```

$$[C_1 = \frac{2+a}{1+a}, C_2 = \frac{1+a^2}{a(1+a)}, C_3 = \frac{1}{a}, C_4 = \frac{a^3+2a+1}{a(1+a)}, C_5 = \frac{1}{1+a}]$$

Note that you can only collect with respect to names and function calls, but not with respect to polynomial expressions. If the first few terms of

```
> x^5 - 5*x^4*y^2 + 10*x^3*y^4 - 10*x^2*y^6 + 5*x*y^8
> - y^10 - 2;
```

$$x^5 - 5x^4y^2 + 10x^3y^4 - 10x^2y^6 + 5xy^8 - y^{10} - 2$$

remind you of  $(x - y^2)^5$ , then you cannot simply collect with respect to  $x - y^2$ .

```
> collect(% , x-y^2);
Error, (in collect) cannot collect x-y^2
```

Instead, you can simplify with respect to the side relation  $x - y^2 = z$ .

```
> siderel := {z=x-y^2};
siderel := {z = x - y^2}
> simplify(%%, siderel, [x,y,z]);
-2 + z^5
> subs(siderel, %);
-2 + (x - y^2)^5
```

See §14.7 for more examples of such polynomial simplifications.

## 7.6 Sorting

The procedure **sort** is used to sort polynomials in some suitable ordering. We have already described this in §5.2. Here, we only give an example of a generalized rational function whose numerator and denominator are sorted.

```
> r := sort((cos(x)-sin(x))/(cos(x)+sin(x)),
> [cos(x),sin(x)], 'plex');
```

```

r :=  $\frac{\cos(x) - \sin(x)}{\cos(x) + \sin(x)}$ 
> sort(r, [sin(x), cos(x)], 'plex');

$$\frac{-\sin(x) + \cos(x)}{\sin(x) + \cos(x)}$$


```

## 7.7 Exercises

*Remark:* in your answers to the exercises below, the ordering of factors may differ.

1. Describe a canonical simplification of univariate rational functions with rational coefficients.
2. Consider the rational expression  $\frac{x^4 + x^3 - 4x^2 - 4x}{x^4 + x^3 - x^2 - x}$ . Transform this expression with Maple into:
  - (a)  $\frac{x^2 - 4}{x^2 - 1}$
  - (b)  $\frac{(x - 2)(x + 2)}{x^2 - 1}$
3. Consider the rational expression  $2\frac{x^3 - yx^2 - yx + y^2}{x^3 - yx^2 - x + y}$ . Transform it into:
  - (a)  $2\frac{x^2 - y}{x^2 - 1}$
  - (b)  $2\frac{x^2 - y}{(x - 1)(x + 1)}$
  - (c)  $2 - \frac{y - 1}{x - 1} + \frac{y - 1}{x + 1}$
  - (d)  $2 - 2\frac{y - 1}{x^2 - 1}$
4. Consider the polynomial  $(2x^2 - x)(2x^2 + x)$ . Transform it into:
  - (a)  $(-1 + 4x^2)x^2$
  - (b)  $x^2(2x - 1)(2x + 1)$
  - (c)  $(2x^3 + x^2)(2x - 1)$
5. Consider the polynomial  $(x^2 + xy + x + y)(x + y)$ . Transform it into:
  - (a)  $x^3 + 2x^2y + xy^2 + x^2 + 2xy + y^2$
  - (b)  $(x + 1)(x + y)^2$
  - (c)  $y^2 + (2y + y^2)x + (1 + 2y)x^2 + x^3$
  - (d)  $x^3 + x^2 + (2x^2 + 2x)y + (x + 1)y^2$

# 8

## Functions

In Maple, a functional relationship can be specified in three ways: as a formula, as an arrow operator (similar to common mathematical notation), and as a procedure. In this chapter we shall discuss all methods in detail. Special attention will be paid to recursively defined procedures and functions. The role of the `remember` option in defining efficient, recursive functions will be treated in detail.

This chapter is not about programming; we shall only discuss how mathematical functions can be defined in Maple. In this chapter, we shall focus on practical issues like “how to define a function,” “how to transform a formula into a function,” and “when to use anonymous functions.”

### 8.1 Mathematical Functions

In previous chapters we have already seen some commonly used mathematical functions that are available in Maple. You can get a complete list with the instruction `?inifcns` (help about `initially known functions`). This list also contains names of less known functions such as *Lambert's W function*. This function has been introduced as the solution of the following equation in [87].

```
> f(x)*exp(f(x)) = x;  
f(x) ef(x) = x  
> solve(%, f(x));
```

### LambertW( $x$ )

A more detailed description of Lambert's  $W$  function can be found in [63]. It is a multivalued complex function. The best plot of the two real branches of this function together in one picture can be obtained in Maple as a parametric plot. The graph is shown in Figure 8.1.

```
> plot([y*exp(y), y, y=-5..0.75]);
```

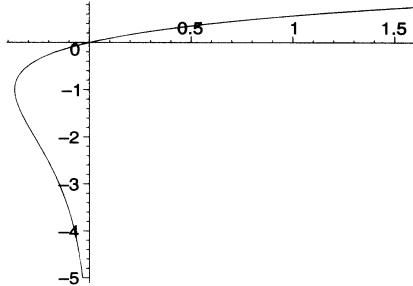


Figure 8.1. Graph of two real branches of Lambert's  $W$  function.

The ranges of the branches of the complex multiple-valued function  $W$  can be shown with the procedure **branches**; see Figure 8.2. Presently, this procedure also "knows" about the inverse trigonometric functions and the natural logarithm.

```
> branches(LambertW, thick);
```

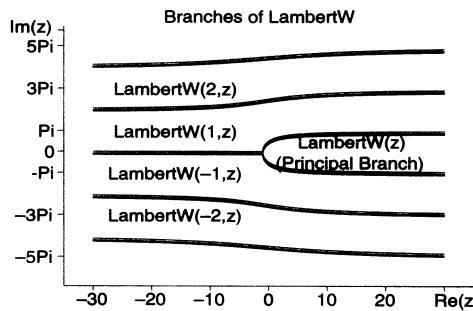


Figure 8.2. Ranges of branches of Lambert's  $W$  function.

Below are two examples of computations in which Lambert's  $W$  function plays a role.

```
> a*x+b^x=c;
```

$$a x + b^x = c$$

```
> solve(% , x);
```

$$\frac{\text{LambertW}\left(\frac{\ln(b) e^{(\frac{\ln(b) c}{a})}}{a}\right) a - \ln(b) c}{\ln(b) a}$$

```
> f(x) = solve(f(x)=x^f(x), f(x));
```

$$f(x) = -\frac{\text{LambertW}(-\ln(x))}{\ln(x)}$$

The latter result is nothing more than saying that the function

$$f: x \mapsto x^{x^x}$$

is equal to

$$f: x \mapsto -\frac{\text{LambertW}(-\ln x)}{\ln x}$$

for  $x \in (0, 1)$ . The graph of the function is shown in Figure 8.3.

```
> plot(rhs(%), x=0..1);
```

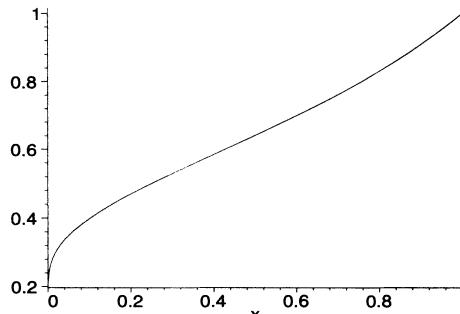


Figure 8.3. Graph of  $-\frac{\text{LambertW}(-\ln x)}{\ln x}$  on  $(0, 1)$ .

In Maple, you can specify functional relationship in three ways:

- **As a formula.**

The dependency of a quantity on variables can be specified in a formula. For example, if the temperature of a body decreases exponentially in time, then the following formula can be entered in Maple.

```
> T := T0*exp(-c*t);
```

$$T := T_0 e^{-ct}$$

To get the value of the temperature at a particular time you can replace  $t$  by some value or evaluate the expression  $T$  under a condition.

```
> subs(t=1/c, T);
```

$$T_0 e^{(-1)}$$

```
> eval(T, t=1/c); # evaluate under the condition t=1/c
```

$$T_0 e^{(-1)}$$

In two steps with the procedures **Eval** and **value**:

```
> Eval(T, t=1/c);

$$T0 e^{(-ct)} \Big|_{t=\frac{1}{c}}$$

> value(%);

$$T0 e^{(-1)}$$

```

- **As a function.**

You can introduce functions in a more mathematical way.

```
> T := t -> T0*exp(-c*t);

$$T := t \rightarrow T0 e^{(-ct)}$$

> T(2/c), T(time), T(0);

$$T0 e^{(-2)}, T0 e^{(-c time)}, T0$$

```

The so-called *arrow operator* will be discussed in more detail in the next section.

- **As a procedure.**

The above mathematical function can be defined as a procedure.

```
> T := proc(t) T0*exp(-c*t) end proc;

$$T := \text{proc}(t) T0 * \exp(-c * t) \text{end proc}$$

> T(2/c), T(time), T(0);

$$T0 e^{(-2)}, T0 e^{(-c time)}, T0$$

```

In fact, the arrow operator is a special case of a procedure definition. There will be ample discussion on procedures in §8.4 and §8.5.

Mathematical functions often come in handy, and you should be careful to distinguish in Maple between a variable that points to an expression in other unknowns and a variable whose value is a function. Referring to the above function definition of T:

```
> T; # evaluation to the name of the function

$$T$$

> T(t); # evaluation of function value at t

$$T0 e^{(-ct)}$$

> solve(T=100, t);
>
```

Maple does not show a result after the last instruction as a sign that no solution is found. If `infolevel[solve]` has a value greater than zero, then the system notifies that no solution is found.

```
> infolevel[solve] := 1;
> solve(T=100, t);

solve: Warning: no solutions found
```

Indeed, there is no value for  $t$  such that the *function*  $T$  is equal to the number 100. This was of course not what we intended to do; when we ask the correct question, Maple finds the answer.

```
> solve(T(t)=100, t);


$$-\frac{\ln\left(\frac{100}{T_0}\right)}{c}$$

```

## 8.2 Arrow Operators

In the previous section, you have seen one way of defining functions in Maple, viz., the *arrow operator* definition, which mimics the syntax for functions often used in mathematics:

$$\text{function\_name} := \text{parameter}(s) \rightarrow \text{expression}.$$

You may be tempted to define functions as follows:

```
> T(t) := T0*exp(-c*t);


$$T(t) := T_0 e^{-ct}$$

```

Maple accepts the instruction. It looks promising, but it does not really do what you think it means, as illustrated below.

```
> T(t), T(1/c), T(0);


$$T_0 e^{-ct}, T\left(\frac{1}{c}\right), T(0)$$

```

What happened? You created a function without a function description.

```
> print(T);

proc() option remember; 'procname(args)' end proc
```

Instead, you stored in the *remember table* of the procedure  $T$  the value of the function at  $t$ . When you ask for the function value at  $1/c$ , then Maple has no value in the remember table nor a function description. Therefore, the system can do nothing more than printing the function call; perhaps you will want to assign a function description to  $T$  later on in the session.

You have already seen the case of a function in one variable. Functions with more than one variable are defined similarly.

```
> f := (x,y) -> x^3-3*x*y^2;


$$f := (x, y) \rightarrow x^3 - 3xy^2$$

```

```

> f(3,2);
          -9
> plot3d(f, -1..1, -1..1, numpoints=2500,
>         style=patchcontour, axes=frame);

```

This graph is shown in Figure 8.4.

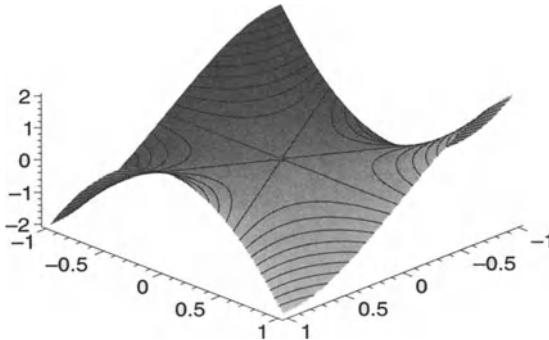


Figure 8.4. Surface plot of  $x^3 - 3xy^2$  on  $[0, 1] \times [0, 1]$ .

Mind the brackets around the function parameters in the definition; otherwise Maple will interpret it as

```

> f := x, (y -> x^3-3*x*y^2);
f := x, y → x^3 - 3 x y^2

```

i.e., you have created the expression sequence **f** consisting of the variable **x** and the anonymous function  $y \rightarrow x^3 - 3xy^2$ . Functions with zero parameters can be defined as well.

```

> f := x, (y -> x^3-3*x*y^2);
f := x, y → x^3 - 3 x y^2
> goldenratio := () -> (1+sqrt(5))/2; # constant function
goldenratio := () →  $\frac{1}{2} + \frac{1}{2}\sqrt{5}$ 
> goldenratio();
 $\frac{1}{2} + \frac{1}{2}\sqrt{5}$ 
> goldenratio(x); # no check on number of arguments
 $\frac{1}{2} + \frac{\sqrt{5}}{2}$ 

```

Be careful with the use of variables that are not parameters in the function definition. There might be side effects and you will not get a warning beforehand from Maple. An example:

```
> ERF := x -> 2/Pi^(1/2)*int(exp(-t^2), t=0..x);
```

$$ERF := x \rightarrow \frac{2}{\sqrt{\pi}} \int_0^x e^{(-t^2)} dt$$

```

> ERF(1); # indeed, ERF is nothing but the error function
          erf(1)
> t := 0; # let t be assigned a value!
> ERF(1); # all goes wrong
Error, (in int) wrong number (or type) of arguments

```

The only solution to this problem is to declare `t` as a local variable. Because the arrow-definition only allows one expression, one conditional statement, or one procedure definition, you have to use the long format of defining procedures.

```

> ERF := proc(x)
>   options operator, arrow;
>   local t;
>   2/Pi^(1/2)*int(exp(-t^2), t=0..x);
> end proc;

```

$$ERF := x \rightarrow \text{local } t; \frac{2}{\sqrt{\pi}} \int_0^x e^{(-t^2)} dt$$

Now there are no side effects anymore.

```

> ERF(1); # indeed, ERF is nothing but the error function
          erf(1)

```

In §8.4 we shall have a close look at the general format of defining a procedure.

## 8.3 Piecewise Defined Functions

Piecewise defined functions can easily be introduced with the arrow operator notation. There are two ways of defining such functions, the second of which is the best.

- **As a case-statement.**

Use a conditional statement to distinguish the cases. The general format is as follows:

```

parameter(s) ->
  if 1st condition then
    1st value
  elif 2nd condition then
    2nd value
  .....
  elif Nth condition then

```

```

Nth value
else
    default value
end if

```

- **Via the procedure piecewise.**

Mathematical knowledge for piecewise defined functions is available in Maple when using the procedure **piecewise**. The general format is as follows:

```

parameter(s) -> piecewise(
    1st condition, 1st value,
    2nd condition, 2nd value,
    . . .
    Nth condition, Nth value,
    default value
)

```

We shall look at both ways of defining piecewise defined functions. To start, let us consider a step function whose values are  $-1$  for real numbers less than  $1$ ,  $0$  at  $1$ , and  $1$  otherwise.

```

> step :=  $x \rightarrow$  if  $x < 1$  then  $-1$  elif  $x = 1$  then  $0$  else  $1$  end if;
      step := proc( $x$ )
      option operator, arrow;
          if  $x < 1$  then  $-1$  elif  $x = 1$  then  $0$  else  $1$  end if
      end proc
> step( $3/2$ ), step( $1$ ), step( $1/2$ );
      1, 0, -1
> plot(step,  $-1..Pi$ , discont=true,
      title="graph of step function", style=line);

```

In Figure 8.5 you see the Maple plot of this piecewise defined function.

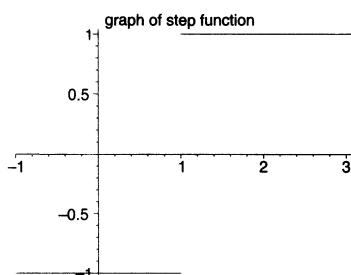


Figure 8.5. Graph of step function.

That such a function definition is already more delicate than at first sight becomes clear in the following instruction.

```
> printlevel := 2: # make Maple more communicative
> step(Pi);
step called with arguments: Pi
#(step,1): if x < 1 then ... elif x = 1 then ... else ... end if
Error, (in step) cannot determine if this expression is true or
false: Pi < 1
```

The problem here is that Maple cannot successfully execute the condition

$$\text{if } \text{Pi} < 1 \text{ then } \dots$$

Maple can only compare numbers of type *numeric*, i.e., integers, fractions, and floating-point numbers. Of course we know that  $1 < \text{Pi}$ . This inability to evaluate an expression in a boolean context is also the reason why we plotted the step function in functional notation and did not enter the command

```
> plot(step(x), x = -1 .. 3.14);
step called with arguments: x
#(step,1): if x < 1 then ... elif x = 1 then ... else ... end if
Error, (in step) cannot determine if this expression is true or
false: x < 1
```

Computing the sign of a real number can be seen as a membership test of the open interval  $(0, \infty)$ . You can use the **assume** facility for this purpose, and particularly call the procedure **is**. Then you do not have problems with evaluation and moreover you can work with indeterminates that have properties associated with them. The improved definition of the step function below relies on interval arithmetic in the current precision and on computing with properties in the **assume** facility. It uses the procedure **verify** instead of **is** to test the sign of difference between two objects.

```
> step := x -> if verify(x, 1, less_than) then
>           -1
>           elif verify(x, 1, equal) then
>             0
>           elif verify(x, 1, greater_than) then
>             1
>           else # nothing known or all fails
>             'procname'(x)
>           end if:
```

Within the procedure, we use the special name **procname** to refer to the name with which the procedure is invoked (in this case **step**). Let us see how the function works.

```
> step(Pi), step(1+sqrt(2)*sqrt(3)-sqrt(6));
1, 0
> step(exp(Pi/3*sqrt(163)) - 640319);
```

```

> step(s) assuming s>sqrt(2);
                                         1
> step(u);
                                         step(u)

```

For those who wondered why we introduced the **verify** command and did not simply use **is**: in the latter case you must use the test **is(x<=1)** and **is(x>=1)** instead of the simple statement **is(x=1)**, because the procedure **is** has some difficulty in testing equality.

```
> is(1+sqrt(2)*sqrt(3)-sqrt(6)=1);
```

*FAIL*

In the case-statement, the boolean tests are evaluated from left to right, and as soon as a condition is fulfilled, the corresponding value is returned. Therefore it is perfectly valid to define the following function and apply it to 0.

```
> f := x -> if x<=0 then x else 1/x end if:  
> f(0);
```

0

But a big disadvantage of the definition of a piecewise defined function by a conditional statement is that you cannot do much mathematics with it. Nothing of the following kind:

```

> diff(step(u), u);

$$\frac{d}{du} \text{step}(u)$$

> solve(step(u)=0, u);

$$\text{RootOf}(\text{step}(-Z))$$

> # try to solve the differential equation
> diff(g(u), u) = step(u);

$$\frac{d}{du} g(u) = \text{step}(u)$$

> dsolve(% - g(u));

```

$$g(u) = \int \text{step}(u) du + -C1$$

Many of the drawbacks of defining piecewise defined function as case-statements are overcome with the Maple procedure **piecewise**. Let us first look at the same example as before.

```
STEP := x → piecewise(x < 1, -1, x = 1, 0, 1 < x, 1, 'procname'(x))
> STEP(3/2), STEP(1), STEP(1/2), STEP(Pi);
1, 0, -1, 1
```

It works fine. You can also use unknowns.

```
> STEP(u);
```

$$\begin{cases} -1 & u < 1 \\ 0 & u = 1 \\ 1 & 1 < u \\ \text{STEP}(u) & \text{otherwise} \end{cases}$$

What you get is a description of the function call, which can be used for further processing.

```
> dsolve(diff(g(u), u) = STEP(u), g(u));
```

$$g(u) = \begin{cases} -u + _C1 & u < 1 \\ u + _C1 - 2 & 1 \leq u \end{cases}$$

The procedure **piecewise** also cascades the boolean tests. The following example shows that the boolean tests are evaluated from left to right, and that as soon as a condition is fulfilled, the corresponding value is returned.

```
> f := x → piecewise(x <= 0, x, x > 0, 1/x);
```

$$f := x \rightarrow \text{piecewise}(x \leq 0, x, 0 < x, \frac{1}{x})$$

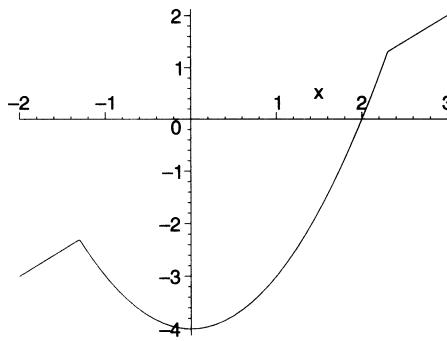
```
> f(0);
```

0

The advantage of **piecewise** is that Maple has mathematical knowledge built-in to work with these objects. You can differentiate and integrate such piecewise defined functions, solve equations and differential equations that contain such functions, compute Taylor series, and so on. Moreover, well-known mathematical functions like the absolute value function, the maximum and minimum function, and the Heaviside step function can be converted into **piecewise**-format for further processing. One example will do here; more examples you will encounter in chapters on differentiation and integration, series expansion, limits, and solving differential equations.

```
> f := min(x-1, x^2-4);
f := \min(x - 1, x^2 - 4)
> plot(f, x=-2..3);
```

The graph is shown in Figure 8.6.

Figure 8.6. Graph of  $\min(x - 1, x^2 - 4)$  on interval  $(-2, 3)$ .

Conversion to piecewise-definition:

```
> F := convert(f, piecewise);
```

$$F := \begin{cases} x - 1 & x \leq \frac{1}{2} - \frac{\sqrt{13}}{2} \\ x^2 - 4 & x < \frac{1}{2} + \frac{\sqrt{13}}{2} \\ x - 1 & \frac{1}{2} + \frac{\sqrt{13}}{2} \leq x \end{cases}$$

Now you can differentiate F with respect to x, solve equations, or compute a Taylor series around some expansion point.

```
> diff(F, x);
```

$$\begin{cases} 1 & x < \frac{1}{2} - \frac{1}{2}\sqrt{13} \\ \text{undefined} & x = \frac{1}{2} - \frac{1}{2}\sqrt{13} \\ 2x & x < \frac{1}{2} + \frac{1}{2}\sqrt{13} \\ \text{undefined} & x = \frac{1}{2} + \frac{1}{2}\sqrt{13} \\ 1 & \frac{1}{2} + \frac{1}{2}\sqrt{13} < x \end{cases}$$

```
> integrate(F, x);
```

$$\begin{cases} \frac{1}{2}x^2 - x & x \leq \frac{1}{2} - \frac{1}{2}\sqrt{13} \\ \frac{1}{3}x^3 - 4x + \frac{19}{12} - \frac{13}{12}\sqrt{13} & x \leq \frac{1}{2} + \frac{1}{2}\sqrt{13} \\ \frac{1}{2}x^2 - x - \frac{13}{6}\sqrt{13} & \frac{1}{2} + \frac{1}{2}\sqrt{13} < x \end{cases}$$

```
> plot(% , x=-2..5);
```

The graph is shown in Figure 8.7

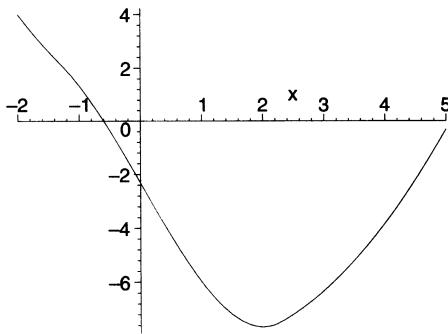


Figure 8.7. Graph of  $\int \min(x - 1, x^2 - 4) dx$  on interval  $(-2, 5)$ .

```

> solve(F=-3, x);
          -2, 1, -1
> series(F, x=1, 3);
          -3 + 2(x - 1) + (x - 1)^2
> convert(%, polynom);
          -5 + 2x + (x - 1)^2
> expand(%);
          x^2 - 4

```

## 8.4 Maple Procedures

Look again at the output of the arrow operator definition of the step function of the previous section: it is a *procedure*. In general, the definition of a Maple procedure has the following syntax:

```

proc( parameter_sequence )
  [ local name_sequence; ]
  [ global name_sequence; ]
  [ options name_sequence; ]
  [ description string; ]
  statements
end proc

```

where *name\_sequence* is a sequence of Maple names separated by commas. The *parameter\_sequence* is in many cases a sequence of Maple names too, but when the dynamic type checking facility is used, each name may be followed by its data type (separated by the double colon ::).

```

> sgn := proc(n::integer) (-1)^n end proc;
> sgn(Pi);
Error, invalid input: sgn expects its 1st argument, n, to be of type
integer, but received Pi
> sgn(-2);
1

```

The procedure **sgn** has one parameter. When it is called, Maple checks whether the given argument is of type *integer*. If not, or when no argument is supplied, the system returns an error message. Otherwise, it computes the power.

In general, the value returned by a procedure is the last value computed unless there is an explicit return via **return**. An example:

```

> MEMBER := proc(x::anything, L::list, pos::evaln)
>   local i;
>   description "membership test in a list";
>   for i to nops(L) do
>     if L[i] = x then pos := i; return true end if
>   end do;
>   false
> end proc;
> position := 0;
> MEMBER(2, [2,1,3,2], position);
true
> position;
1
> MEMBER(4, [2,1,3,2], position);
false

```

If the third parameter has not been declared in the procedure definition as **evaln**, but instead as **name**, then the name **position** must be enclosed in apostrophes.

When a variable inside a function definition is assigned a value and nothing about local or global definition of the variable has been stated, then Maple will automatically assume that it is a local variable. The next example illustrates this.

```

> f := proc(x)
>   y := 1 + sin(x);
>   y^2 + y + 1/y + 1/y^2
> end proc;

Warning, 'y' is implicitly declared local to procedure 'f'
f := proc(x) local y; y := 1 + sin(x); y^2 + y + 1/y + 1/y^2 end proc
> y := unspecified: f(Pi);

```

## 4

It goes without saying that you better specify the scope of the variables used in a procedure definition via **local** and **global** declarations. The declarations may be placed in any order, but each one may appear only once.

Maple has a *subscripted function* calling facility. For example, it is used for the logarithm with different bases. **log[b]** is used to specify the logarithm with base b. Here, we give only one example. It should give you an idea about the definition of subscripted functions. We extend Maple's implementation of Euler's beta function to the incomplete beta function, which is defined as

$$B_x(p, q) = \int_0^x t^{p-1} (1-t)^{q-1} dt.$$

$B_1(p, q)$  becomes the regular (complete) beta function. In Maple, a rudimentary, but not full-proof implementation of  $B_x(p, q)$  can be as follows.

```
> BETA := proc(p,q)
>   local x:
>   description "incomplete beta function";
>   if type(procname, indexed) then
>     x := op(procname);
>     int(t^(p-1)*(1-t)^(q-1), t=0..x)
>   else
>     Beta(p,q)
>   end if:
> end proc:
```

```
> BETA[1](2,3), Beta(2,3);
```

$$\frac{1}{12}, \frac{1}{12}$$

```
> BETA[2](3,4);
```

$$\frac{-4}{5}$$

```
> BETA(3/2,5/2);
```

$$\frac{\pi}{16}$$

When the procedure **BETA** is called, Maple will first have a closer look at the actual procedure name. It checks whether the variable **procname** inside the procedure is an indexed name of the form **BETA[x]** or not. If not, it is assumed that **BETA** is equal to the regular (complete) beta function. If an indexed name is used, the local variable **x** is assigned the value of the index, and the integral that defines the incomplete beta function is computed.

## 8.5 Recursive Procedure Definitions

Recursive definition of a function or procedure is possible in Maple. We shall illustrate this with the computation of Lucas numbers  $L_n$ , which are defined by the linear recurrence

$$L_1 = 1, L_2 = 3, \text{ and } L_n = L_{n-1} + L_{n-2}, \text{ for } n > 2.$$

In the arrow operator notation, it can be coded directly by

```
> L := n ->
>   if not type(n, posint) then
>     error "expecting a positive integer, "
>           "but received %1", n
>   elif n=1 then 1
>   elif n=2 then 3
>   else L(n-1) + L(n-2)
>   end if;

L := proc(n)
option operator, arrow;
if not type(n, posint) then
  error "expecting a positive integer, but received %1", n
elif n = 1 then 1
elif n = 2 then 3
else L(n - 1) + L(n - 2)
end if
end proc
> L(3);
4
> L(-3);
Error, (in L) expecting a positive integer, but received -3
```

Here, we have used the statement **error** to generate an error message from within the function when the given argument is not a **positive integer**. The standard way of defining Maple procedures is slightly more convenient.

```
> L := proc(n::posint)
>   if n = 1 then 1
>   elif n = 2 then 3
>   else L(n-1) + L(n-2)
>   end if
> end proc:
> L(6);
18
> L(-6);
Error, invalid input: L expects its 1st argument, n, to be of
type posint, but received -6
```

However, this is not an efficient way to compute the Lucas numbers. Let us have Maple count the number of procedure calls with **exprofile** (we refer to the help system for a detailed description).

```
> kernelopts(profile=true); # enable profiling
> writeto("output"); # redirect output to file
```

While in this mode and using the plain character user interface instead of the worksheet interface, no prompt will appear at the terminal.

```
> L(6):
> kernelopts(profile=false); # disable profiling
> writeto('terminal'); # redirect output to screen
> exprofile("output");

2 different functions, using .360 secs and 1097K words
name          #calls
====          =====
Main_Routine      1
L              15
```

Here, we have omitted information about computing time and memory usage. The information shown becomes clear when you realize how Maple computes the Lucas numbers. To compute  $L(6)$ , it is necessary to compute  $L(5)$  and  $L(4)$ . For each of these Lucas numbers two more function calls are needed, and this goes on until all needed Lucas numbers have been calculated. Figure 8.8 is the graph of function calls when computing  $L(6)$ .

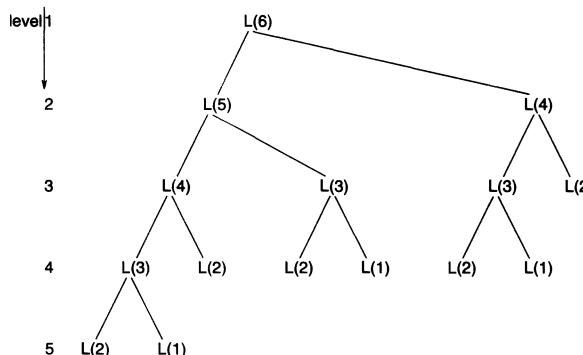


Figure 8.8. Recursive computation of  $L(6)$  without option **remember**.

The graph is in agreement with the result of **exprofile** and shows clearly that, for example,  $L(2)$  is computed five times. So, with the above procedure definition of **L**, it takes exponential time ( $2^n$ ) to compute the  $n$ th Lucas number, and it shall take almost forever to compute  $L(100)$ . But we can do better. It is clear that we should remember the function values as they are computed so that they can be used when they are needed. Maple provides the option **remember** for this purpose.

```

> L := proc(n::posint) Lucas(n) end proc;
> Lucas := proc(n)
>   option remember;
>   if n = 1 then 1
>   elif n = 2 then 3
>   else Lucas(n-1) + Lucas(n-2)
>   end if
> end proc;
> kernelopts(profile=true):
> writeto("output"):
> L(6):
> kernelopts(profile=false): # disable profiling
> writeto('terminal'); # redirect output to screen
> exprofile("output");

3 different functions, using .390 secs and 1139K words
name           #calls
=====
Main_Routine      1
Lucas            6
L                1

```

With the new definition of **L**, time-complexity is linear ( $2n$ ). What happens when **L** is called is the following: the type of the argument is checked, and if valid, the recursive procedure **Lucas** is called. In this manner, we check only once the type of the argument and not over and over again during the recursive computation that follows. Each Maple procedure has an associated *remember table*. In many cases it is non-existing in the sense that the pointer to the remember table is the special name **NULL**. But Maple's remember mechanism becomes activated when you add the option **remember** to the procedure definition. The entries of the remember table will be function values indexed by the arguments of the corresponding function calls. When, in our example, the procedure **Lucas** is called with some argument **n**, Maple will first look up the remember table of **Lucas** to see if the value **Lucas(n)** has been computed before. If it has, it returns the result from the remember table. Otherwise, it executes the code present in the procedure body of **Lucas**, and automatically stores the pair  $(n, \text{Lucas}(n))$  in the remember table of **Lucas**. In this way, each Lucas number is computed only once. In terms of the above graph of function calls, it can be understood as: "follow the depth-first strategy of traversing the graph, compute function values only once, store the results, and use them when needed."

The remember table of a procedure is printed along with the procedure body when the interface variable **verboseproc** has been set to the value 3. In our case:

```

> interface(verboseproc=3):
> print(Lucas);

```

```

proc(n)
option remember;
    if n = 1 then 1
    elif n = 2 then 3
    else Lucas(n - 1) + Lucas(n - 2)
    end if
end proc
# (1) = 1
# (2) = 3
# (3) = 4
# (4) = 7
# (5) = 11
# (6) = 18

```

The remember table of a procedure is accessible as the fourth operand of the procedure structure.

```

> op(4, eval(Lucas)); # remember table of Lucas





```

The remember table of a procedure can be explicitly updated by a function assignment of the form *procedure(argument) := value*. We illustrate this possibility of explicitly saving values in a remember table by using the so-called *functional assignment*. We start a new Maple session.

```

> restart:
> Lucas := proc(n)
>   Lucas(n) := Lucas(n-1) + Lucas(n-2)
> end proc:
> Lucas(1) := 1: Lucas(2) := 3:
> op(4, eval(Lucas)); # initial remember table





> Lucas(5): op(4, eval(Lucas)); # updated remember table





```

Maple provides the utility function **forget** to remove one or all entries from a remember table.

```

> forget(Lucas, 3); # forget the result of Lucas(3);
> op(4, eval(Lucas)); # updated remember table





```

To remove the remember table of **Lucas** and of all procedures whose names start with **Lucas/**, enter

```

> forget(Lucas):
> op(4, eval(Lucas)); # cleared remember table

```

Alternatively, you can empty the remember table **Lucas** and of all procedures whose names start with **Lucas/** by substituting **NULL** for the procedure's fourth operand.

```
> Lucas(1) := 1: Lucas(2) := 3: Lucas(5):
> op(4, eval(Lucas)); # updated remember table


```

You may wonder why **forget(<name>)** not only clears the remember table of the procedure *name*, but also the remember tables of procedures that start with *name/*. The reason is that, in this way, you can remove all tables for routines like **int** or **evalc** without knowing precisely which subfunctions starting with **int/** or **evalc/** make use of remember tables. Thus, **forget** really gives you a fresh start. For details about customizing the behavior of **forget** in this sense we refer to the on-line help system.

## 8.6 unapply

There exists a third way of making new functions, a way that is especially convenient when you encounter, in the midst of a Maple session, a symbolic expression that you want to use as a description of a function. This is done by **unapply**. As the name implies: it is the opposite of applying a function to symbolic parameters.

```
> formula := (b^2*x^2*sin(b*x) - 2*sin(b*x)
>           + 2*b*x*cos(b*x)*a*t) / b^3;
formula := 
$$\frac{b^2 x^2 \sin(b x) - 2 \sin(b x) + 2 b x \cos(b x) a t}{b^3}$$

> F := unapply(formula, x, t);
F := (x, t) → 
$$\frac{b^2 x^2 \sin(b x) - 2 \sin(b x) + 2 b x \cos(b x) a t}{b^3}$$

> F(0,1), F(Pi/b,5);
0, -
$$\frac{10 \pi a}{b^3}$$

```

In a definition of an arrow function, you may be tempted to refer to a previous formula by the ditto operator, but this does not work.

```
> formula := (b^2*x^2*sin(b*x) - 2*sin(b*x)
>           + 2*b*x*cos(b*x)*a*t) / b^3:
> F := (x,t) → %;
```

$$F := (x, t) \rightarrow \%$$

```
> F(u,v);
>
```

The problem is that the ditto operator actually is an *environment variable*, which works locally inside the procedure **F**, but does not refer to anything outside. The external value has been placed on the stack when the procedure was called and is restored when the procedure call is left.

```
> G := (x,t) -> formula;
G := (x, t) → formula
```

```
> G(1,2);

$$\frac{b^2 x^2 \sin(b x) - 2 \sin(b x) + 2 b x \cos(b x) a t}{b^3}$$

```

In this case, you get what you ask for: for any argument of **G**, the value is the evaluation of **formula**. So, you really have to type the function description in *parameter -> description* form. One alternative is to substitute the expression into the function description via a global variable, say **body**.

```
> H := subs(body=formula, (x,t) -> body);
H := (x, t) → 
$$\frac{b^2 x^2 \sin(b x) - 2 \sin(b x) + 2 b x \cos(b x) a t}{b^3}$$

> H(u,v);

$$\frac{b^2 u^2 \sin(b u) - 2 \sin(b u) + 2 b u \cos(b u) a v}{b^3}$$

```

Another alternative is to convert a computational sequence to a procedure.

```
> H := convert(CompSeq(params=[x,y],[formula]), procedure):
> H(U,V);

$$\frac{b^2 U^2 \sin(b U) - 2 \sin(b U) + 2 b U \cos(b U) a t}{b^3}$$

```

A more sophisticated approach would be to use the procedures **ToInert** and **FromInert** for converting a Maple expression into an inert form and backwards.

```
> HH := FromInert(_Inert_PROC(
>   _Inert_PARAMSEQ(_Inert_NAME("x"),_Inert_NAME("t")),
>   _Inert_STATSEQ(ToInert(formula)))
> ): HH(w,z);

$$\frac{b^2 w^2 \sin(b w) - 2 \sin(b w) + 2 b w \cos(b w) a z}{b^3}$$

```

## 8.7 Operations on Functions

Elementary operations on functions like addition, multiplication, and composition are easily done in Maple. A few examples:

```

> f := x -> ln(x)+1: g := y -> exp(y)-1:
> h := f+g: h(z);

$$\ln(z) + e^z$$

> h := f*g: h(z);

$$(\ln(z) + 1)(e^z - 1)$$

> h := f@g: h(z);

$$\ln(e^z - 1) + 1$$

> h := g@f: h(z);

$$e^{(\ln(z)+1)} - 1$$

> simplify(%);

$$ze - 1$$

> (f@@4)(z); # equivalent to f(f(f(f(z))))

$$\ln(\ln(\ln(\ln(z) + 1) + 1) + 1) + 1$$


```

The use of the @ operator in combination with the **macro** facility can be a powerful mechanism to introduce abbreviations in Maple. An example: substitution. Recall from §6.3 that **subs** only does substitution and no evaluation at the end.

```

> subs(n=2, Zeta(n)); # substitution

$$\zeta(2)$$

> %; # evaluation

$$\frac{\pi^2}{6}$$


```

Suppose that we always want to apply evaluation on the result of substitution, then we can do the following:

```

> macro(subs=eval@subs): # new version of subs
> subs(n=2, Zeta(n)); # with new version of subs

$$\frac{\pi^2}{6}$$


```

## 8.8 Anonymous Functions

You are not obliged to name a function. Such *anonymous functions* come in handy when you want to perform an operation only once and do not want to waste a name on it. Anonymous functions are used mostly in conjunction with procedures like **map**, **map2**, **foldl**, **foldr**, **select**, **remove**, **zip**, and **collect**. Some examples:

```
> map(x -> x+2, [1,2,3]); # add 2 to list elements
```

```

[3, 4, 5]
> map2((x,y) -> x+y^2, 2, [1,2,3]); # square and add 2
[3, 6, 11]
> map(x -> x^2, a+b+c); # square summands
a2 + b2 + c2
> # fold from the left
> foldl((x,y) -> [op(x),x[-1]+y], 0, 1,2,3,4,5,6,7);
[0, 1, 3, 6, 10, 15, 21, 28]
> # fold from the right
> foldr((x,y) -> [op(y),x+y[-1]], 0, 1,2,3,4,5,6,7);
[0, 7, 13, 18, 22, 25, 27, 28]
> data := [[1,1.0], [2,3.8], [3,5.1]]:
> # take the logarithhm of the 2nd element of each entry
> map(x -> applyop(ln,2,x), data);
[[1, 0.], [2, 1.335001067], [3, 1.629240540]]
> # compute sums of derivative and antiderivative
> map((f,x) -> diff(f,x)+int(f,x),
>      [cos(z), sin(z), exp(z), ln(z)], z);
[0, 0, 2 ez,  $\frac{1}{z} + z \ln(z) - z$ ]
> sum('x^i', 'i'=0..6);
1 + x + x2 + x3 + x4 + x5 + x6
> # select low anf high order terms of a polynomial
> select(t -> degree(t)<3, %);
1 + x + x2
> remove(t -> degree(t)<3, %);
x3 + x4 + x5 + x6
> # combine two lists of data
> zip((x,y) -> [x,ln(y)], [1,2,3], [1.0,3.8,5.1]);
[[1, 0.], [2, 1.335001067], [3, 1.629240540]]
> collect(x^2 + y*x^2 + 2*x + y, x, z -> z^2);
(1 + y)2 x2 + 4 x + y2

```

## 8.9 Exercises

1. Consider  $x^3 - (a - 1)x^2 + a^2x - a^3 = 0$  as an equation in  $x$ . Solve it, make a function out of the first solution, and compute the solution for  $a = 0$  and for  $a = 1$ . Give an approximate result for  $a = 2$ .

2. Define a function in Maple that is 1 on the segment  $[-1, 1]$ , and 0 otherwise. Also plot the graph of your function.
3. Define the function  $f: t \mapsto \sum_{n=1}^8 (-1)^{n+1} \frac{2}{n} \sin(nt)$ . Compute  $f(\frac{\pi}{10})$  and  $f(\frac{\pi}{6})$ . Plot the graph of your function.
4. After the following assignments,
 

```
> x := 1;
> f := proc(x) 2 end proc;
> f(x) := 3;
```

 what will be  $f(1)$ ,  $f(4)$ , and  $f()$ ?
5. Write a Maple procedure that computes the Legendre polynomials  $L_n(x)$ . These polynomials satisfy the recurrence relation  $L_0(x) = 1$ ,  $L_1(x) = x$ , and  $L_n(x) = \frac{n-1}{n} (x L_{n-1}(x) - L_{n-2}(x)) + x L_{n-1}(x)$ , for  $n > 1$ . Compute  $L_7(x)$ , and check your answer with the Maple procedure **LegendreP**. Can your procedure compute  $L_{50}(x)$ ?
6. Write an anonymous function that selects from a set of integers those values that are between 0 and 10. Use the procedure **rand** to generate a set of one hundred integers and apply your anonymous function to this set.
7. Write an anonymous function to drop from a polynomial in two unknowns (which can be created by the procedure **randpoly**) all terms with negative coefficients.

# 9

## Differentiation

In this chapter we shall explain the procedures **diff** and **D** for computing derivatives symbolically, give examples of implicit differentiation, and finally, briefly discuss Maple's automatic differentiation facility.

### 9.1 Symbolic Differentiation

With the Maple procedure **diff** you can differentiate a formula.

```
> 'diff(exp(-x^2), x)';

$$\frac{d}{dx} (e^{-x^2})$$

> %;

$$-2 x e^{-x^2}$$

```

The apostrophes around the first command are to postpone computation of the derivative and will print the input in two-dimensional layout. For this purpose we could have also used the inert **Diff** command, which simply returns unevaluated, and explicitly ask for its value afterwards

```
> Diff(ln(x/(x^2+1)), x): % = value(%);

$$\frac{d}{dx} \ln\left(\frac{x}{x^2 + 1}\right) = \frac{\left(\frac{1}{x^2 + 1} - \frac{2 x^2}{(x^2 + 1)^2}\right) (x^2 + 1)}{x}$$

> normal(%);
```

$$\frac{\partial}{\partial x} \ln\left(\frac{x}{x^2 + 1}\right) = -\frac{x^2 - 1}{x(x^2 + 1)}$$

In a real Maple session, you will probably first use the inert procedure **Diff** to check whether the formula entered is the one you had in mind. Next, if the formula is really the one you want to differentiate, you can use worksheet facilities (or, in Maple's text-based user interface, the built-in command line editor) to change **Diff** into the “active” procedure **diff** to compute the derivative or ask explicitly for its **value**. In the examples below, we shall use **Diff** to make results clearer and better understandable.

```
> Diff(x^(x^x), x); % = value(%);

$$\frac{d}{dx} x^{(x^x)} = x^{(x^x)} (x^x (\ln(x) + 1) \ln(x) + \frac{x^x}{x})$$

> collect(%, ln(x), simplify);

$$\frac{d}{dx} x^{(x^x)} = x^{(x^x+x)} \ln(x)^2 + x^{(x^x+x)} \ln(x) + x^{(x^x+x-1)}$$

```

Rigorous application of differentiation rules without simplification quickly leads to incomprehensible results. Taking account of this expression swell, higher derivatives can be computed without any difficulty.

```
> Diff(x^(x^x), x, x); % = value(%):
> collect(%, ln(x), simplify);


$$\begin{aligned} \frac{d^2}{dx^2} x^{(x^x)} &= x^{(x^x+2x)} \ln(x)^4 + (x^{(x^x+x)} + 2x^{(x^x+2x)}) \ln(x)^3 \\ &\quad + (2x^{(x^x+x)} + 2x^{(x^x+2x-1)} + x^{(x^x+2x)}) \ln(x)^2 \\ &\quad + (3x^{(x^x+x-1)} + x^{(x^x+x)} + 2x^{(x^x+2x-1)}) \ln(x) + x^{(x^x+2x-2)} \\ &\quad + 2x^{(x^x+x-1)} - x^{(x^x+x-2)} \end{aligned}$$

> Diff(exp(-x^2), x$5); % = value(%);

$$\frac{d^5}{dx^5} e^{-x^2} = -120x e^{-x^2} + 160x^3 e^{-x^2} - 32x^5 e^{-x^2}$$

```

Here, we have used the sequence operator **\$** to shorten input.

```
diff(exp(-x^2), x$5)
```

is equivalent to

```
diff(exp(-x^2), x, x, x, x, x).
```

**Remark:** **diff(expression, [var\$n])** is an alternative notation for higher derivatives. It has been added to Maple to allow the empty list as notation for the 0th derivative, i.e., nothing else than the original expression.

```
> for i from 0 to 8 by 4 do
>   `||i||'th derivative' = diff(x^8/1680, [x|i])
> end do;
```

$$0th \ derivative = \frac{x^8}{1680}$$

$$4\text{th derivative} = x^4$$

$$8\text{th derivative} = 24$$

Differentiation of a function  $y$  of  $x$  that is implicitly defined by an equation can be done in the following elegant way.

```
> alias(y=y(x)): # consider y as a function of x
> eq := x^2+y^2=c; # equation defining y; c is constant
      eq :=  $x^2 + y^2 = c$ 
> diff(eq, x);
       $2x + 2y(\frac{\partial}{\partial x}y) = 0$ 
> dydx := solve(%, diff(y,x)); # 1st derivative
      dydx :=  $-\frac{x}{y}$ 
> diff(eq, x$2);
       $2 + 2(\frac{\partial}{\partial x}y)^2 + 2y(\frac{\partial^2}{\partial x^2}y) = 0$ 
> solve(%, diff(y,x$2)); # 2nd derivative
       $-\frac{1 + (\frac{\partial}{\partial x}y)^2}{y}$ 
> d2ydx2 := normal(subs(diff(y,x)=dydx, %));
      d2ydx2 :=  $-\frac{x^2 + y^2}{y^3}$ 
```

You can simplify further with respect to the original equation as a side relation.

```
> d2ydx2 := simplify(% , {eq}, [x,y,c]);
      d2ydx2 :=  $-\frac{c}{y^3}$ 
> d3ydx3 := diff(%, x);
      d3ydx3 :=  $\frac{3c(\frac{\partial}{\partial x}y)}{y^4}$ 
> d3ydx3 := normal(subs(diff(y,x)=dydx, %));
      d3ydx3 :=  $-\frac{3cx}{y^5}$ 
```

Higher derivatives can be computed in the same style. At the end of this section we shall show how the procedure **implicitdiff** can be used for implicit differentiation.

```
> alias(y=y): # unalias y for further usage
```

Partial derivatives cause no extra problems for **diff**. Two examples of functions in two unknowns:

```
> Diff(exp(a*x*y^2), x, y$2): % = factor(value(%));

$$\frac{\partial^3}{\partial y^2 \partial x} e^{(a x y^2)} = 2 a e^{(a x y^2)} (1 + 5 a x y^2 + 2 a^2 y^4 x^2)$$

> Diff(sin(x+y)/y^4, x$5, y$2): % = value(%);

$$\frac{\partial^7}{\partial y^2 \partial x^5} \frac{\sin(x+y)}{y^4} = -\frac{\cos(x+y)}{y^4} + \frac{8 \sin(x+y)}{y^5} + \frac{20 \cos(x+y)}{y^6}$$

> collect(% , cos(x+y) , normal);

$$\frac{\partial^7}{\partial y^2 \partial x^5} \frac{\sin(x+y)}{y^4} = -\frac{(y^2 - 20) \cos(x+y)}{y^6} + \frac{8 \sin(x+y)}{y^5}$$

```

If you want to differentiate a function instead of a formula, then you can use the **D** operator. **D** works on a function and computes the derivative as a function again. This is convenient when you want to compute or specify the derivative of a function at some point.

```
> g := x -> x^n*exp(sin(x)); # the function

$$g := x \rightarrow x^n e^{\sin(x)}$$

> D(g); # the derivative

$$x \rightarrow \frac{x^n n e^{\sin(x)}}{x} + x^n \cos(x) e^{\sin(x)}$$

> D(g)(Pi/6); # the derivative at Pi/6

$$\frac{6 (\frac{\pi}{6})^n n e^{(1/2)}}{\pi} + \frac{1}{2} (\frac{\pi}{6})^n \sqrt{3} e^{(1/2)}$$

```

Basically, **D(f)** is equivalent to **unapply(diff(f(x), x), x)**. A nice alternative for computing the derivative of a function at some point is the use of the procedures **Eval** and **value**:

```
> Eval(Diff(g(x), x), x=Pi/6);

$$\left. \left( \frac{\partial}{\partial x} (x^n e^{\sin(x)}) \right) \right|_{x=\frac{\pi}{6}}$$

> value(%);

$$\frac{1}{6} (\frac{1}{6} \pi)^n n e^{(1/2)} + \frac{1}{2} (\frac{1}{6} \pi)^n \sqrt{3} e^{(1/2)}$$

```

It is essential to make a clear distinction between **diff** and **D**; **diff** differentiates a *formula* and returns a formula, whereas **D** differentiates a *mapping* and returns a mapping. A few examples:

```
> diff(cos(t), t); # derivative of a formula
```

```

          -sin(t)
> D(cos); # derivative of a function
          -sin
> (D@@2)(cos); # 2nd derivative of a function
          -cos
> D(cos)(t); # derivative of a function at some point
          -sin(t)
> D(cos(t));
          D(cos(t))

```

In the last command, Maple does not consider `cos(t)` as the cosine function, nor as the composition of the cosine function and some function `t`. For the latter purpose you must follow Maple's syntax and semantics closely and use the composition operator `@`.

```

> D(cos@t);
          ((-sin)@t) D(t)

```

However, if we assume that `t` is a constant, then Maple is indeed willing to consider `cos(t)` as a function, viz., as a constant function.

```

> assume(t, constant): D(cos(t));
          0

```

It is easier to understand this for a familiar constant number.

```

> D(cos(1));
          0

```

`cos(1)` is a number and can be considered as a constant function whose derivative is equal to the zero function.

```

> t := 't': # unassign t, i.e., forget assumption
> diff(cos, t);
          0

```

In the last example, Maple considered `cos` as an expression in which `t` did not occur.

If you want to differentiate an implicitly defined function, it is convenient to manipulate the mapping as if it is an expression, and let `D` do the differentiation. The same example as was treated before with `diff` will be as follows in the new formalism.

```

> eq := x^2+y^2=c: D(eq);
          2 D(x) x + 2 D(y) y = D(c)

```

We specify that `x` is an independent variable and that `c` is a constant.

```

> D(x) := 1: D(c) := 0: dydx := solve(%%%, D(y));

```

```


$$\frac{dy}{dx} := -\frac{x}{y}$$

> (D@@2)(eq);

$$2 + 2(D^{(2)})(y)y + 2D(y)^2 = 0$$

> solve(%, (D@@2)(y));

$$-\frac{1 + D(y)^2}{y}$$

> d2ydx2 := normal(subs(D(y)=dydx, %));

$$d2ydx2 := -\frac{x^2 + y^2}{y^3}$$


```

And so on.

Use of the **D** operator is not restricted to univariate functions. An indexed function call to **D** will enable you to compute partial derivatives.

```

> h := (x,y,z) -> 1/(x^2+y^2+z^2)^(1/2);

$$h := (x, y, z) \rightarrow \frac{1}{\sqrt{x^2 + y^2 + z^2}}$$

> # partial derivative w.r.t. x
> 'D[1](h)' = D[1](h);

$$D_1(h) = ((x, y, z) \rightarrow -\frac{x}{(x^2 + y^2 + z^2)^{(3/2)}})$$


```

Here, **D[1](h)** means the partial derivative of **h** with respect to the first argument. **D[1](h)** is equivalent to **unapply(diff(h(x,y,z),x), x,y,z)**, but simplification differences may be present.

```

> # partial derivative w.r.t. x and y
> 'D[1,2](h)' = D[1,2](h);

$$D_{1,2}(h) = ((x, y, z) \rightarrow \frac{3yx}{(x^2 + y^2 + z^2)^{(5/2)}})$$


```

Here, **D[1,2](h)** is equivalent to **D[1](D[2](h))**.

```

> # 2nd partial derivative w.r.t. x
> 'D[1,1](h)' = D[1,1](h);

$$D_{1,1}(h) = ((x, y, z) \rightarrow \frac{3x^2}{(x^2 + y^2 + z^2)^{(5/2)}} - \frac{1}{(x^2 + y^2 + z^2)^{(3/2)}})$$


```

If we compute the Laplacian  $\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}\right)h$ , limitations of the operator method, due to lack of simplification of intermediate results, emerge.

```
> L[h] := (D[1,1]+D[2,2]+D[3,3])(h);
```

```


$$L_h := ((x, y, z) \rightarrow \frac{3x^2}{(x^2 + y^2 + z^2)^{(5/2)}} - \frac{1}{(x^2 + y^2 + z^2)^{(3/2)}})$$


$$+ ((x, y, z) \rightarrow \frac{3y^2}{(x^2 + y^2 + z^2)^{(5/2)}} - \frac{1}{(x^2 + y^2 + z^2)^{(3/2)}})$$


$$+ ((x, y, z) \rightarrow \frac{3z^2}{(x^2 + y^2 + z^2)^{(5/2)}} - \frac{1}{(x^2 + y^2 + z^2)^{(3/2)}})$$

> normal(L[h](x,y,z));
0

```

All roads lead to Rome. We have used **diff** and **D** to perform implicit differentiation. But the Maple library itself also contains the **implicitdiff** routine for this purpose. It is intended to make life easy for the user of the software. We shall apply this procedure for the same example as was treated with MACSYMA in section 1.5.4 of [69]. Let  $g(x, y)$  be an implicit form, where  $y$  is a function of  $x$ . Below, we shall determine formulae for the first, second, and third derivative of  $y$  with respect to  $x$ , in terms of partial derivatives of  $g$ .

```

> dydx := implicitdiff(g(x,y), y(x), x);
dydx := -  $\frac{D_1(g)(x, y)}{D_2(g)(x, y)}$ 

```

Perhaps, you are more familiar with the following notation.

```

> convert(dydx, diff);
-  $\frac{\frac{\partial}{\partial x} g(x, y)}{\frac{\partial}{\partial y} g(x, y)}$ 

```

Let us go on with computing higher derivatives of  $y$ .

```

> d2ydx2 := implicitdiff(g(x,y), y(x), x$2);
d2ydx2 := -(D_{1,1}(g)(x, y) D_2(g)(x, y)^2
- 2 D_{1,2}(g)(x, y) D_1(g)(x, y) D_2(g)(x, y)
+ D_{2,2}(g)(x, y) D_1(g)(x, y)^2) / D_2(g)(x, y)^3
> convert(% , diff);
-(( $\frac{\partial^2}{\partial x^2} g(x, y)$ ) ( $\frac{\partial}{\partial y} g(x, y)$ )^2 - 2 ( $\frac{\partial^2}{\partial y \partial x} g(x, y)$ ) ( $\frac{\partial}{\partial x} g(x, y)$ ) ( $\frac{\partial}{\partial y} g(x, y)$ )
+ ( $\frac{\partial^2}{\partial y^2} g(x, y)$ ) ( $\frac{\partial}{\partial x} g(x, y)$ )^2) / ( $\frac{\partial}{\partial y} g(x, y)$ )^3

```

The computation of the third derivative of  $y$  clearly shows the advantage of computer algebra usage over pencil and paper calculation.

```
> d3ydx3 := convert(implicitdiff(g(x,y), y(x), x$3), diff);
```

$$\begin{aligned}
d3ydx3 := & -((\frac{\partial^3}{\partial x^3} g(x, y)) (\frac{\partial}{\partial y} g(x, y))^4 \\
& - 3(\frac{\partial^3}{\partial y \partial x^2} g(x, y)) (\frac{\partial}{\partial x} g(x, y)) (\frac{\partial}{\partial y} g(x, y))^3 \\
& + 3(\frac{\partial}{\partial x} g(x, y))^2 (\frac{\partial}{\partial y} g(x, y))^2 (\frac{\partial^3}{\partial y^2 \partial x} g(x, y)) \\
& - 3(\frac{\partial^2}{\partial y \partial x} g(x, y)) (\frac{\partial}{\partial y} g(x, y))^3 (\frac{\partial^2}{\partial x^2} g(x, y)) \\
& + 6(\frac{\partial^2}{\partial y \partial x} g(x, y))^2 (\frac{\partial}{\partial y} g(x, y))^2 (\frac{\partial}{\partial x} g(x, y)) \\
& - 9(\frac{\partial^2}{\partial y \partial x} g(x, y)) (\frac{\partial}{\partial y} g(x, y)) (\frac{\partial^2}{\partial y^2} g(x, y)) (\frac{\partial}{\partial x} g(x, y))^2 \\
& - (\frac{\partial}{\partial x} g(x, y))^3 (\frac{\partial}{\partial y} g(x, y)) (\frac{\partial^3}{\partial y^3} g(x, y)) \\
& + 3(\frac{\partial}{\partial x} g(x, y)) (\frac{\partial^2}{\partial y^2} g(x, y)) (\frac{\partial^2}{\partial x^2} g(x, y)) (\frac{\partial}{\partial y} g(x, y))^2 \\
& + 3(\frac{\partial^2}{\partial y^2} g(x, y))^2 (\frac{\partial}{\partial x} g(x, y))^3) / (\frac{\partial}{\partial y} g(x, y))^5
\end{aligned}$$

Finally, we compute derivatives for  $g$  equal to the function  $\exp(x^2 + y^2)$ .

```

> g := (x,y) -> exp(x^2+y^2);
      g := (x, y) → e^(x²+y²)
> dydx;
      - x
      —
      y
> normal(d2ydx2);
      - x² + y²
      —
      y³
> normal(d3ydx3);
      -3 x (x² + y²)
      —
      y⁵

```

The formulae are in agreement with the results obtained earlier for the function  $y$  that was implicitly defined by  $x^2 + y^2 = c$ .

## 9.2 Automatic Differentiation

The **D** operator also addresses the problem of *automatic differentiation*, i.e., to differentiate procedures. Let us start with an example of a piecewise defined function that is presented as a case-statement.

```
> F := x -> if x>0 then sin(x) else arctan(x) end if;
```

```

F := proc(x)
option operator, arrow;
if 0 < x then sin(x) else arctan(x) end if
end proc

> Fp := D(F); # 1st derivative

Fp := proc(x)
option operator, arrow;
if 0 < x then cos(x) else 1/(1+x^2) end if
end proc

```

W plot F and its first derivative.

```
> plot({F, Fp}, -3*Pi..3*Pi);
```

The graphs of the function F and its derivative are shown in Figure 9.1.

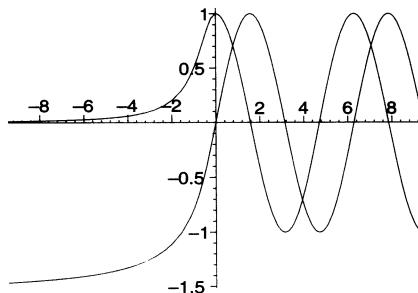


Figure 9.1. Graph of piecewise defined function and its derivative.

Maple can differentiate procedures that consist of more than one statement. A simple example:

```

> f := proc(x)
>   local s,t;
>   s := ln(x);
>   t := x^2;
>   s*t + 3*t
> end proc:
> fp := D(f);

fp := proc(x)
local sx, tx, s, t;
sx := 1/x; s := ln(x); tx := 2*x; t := x^2; sx*t + s*tx + 3*t
end proc

```

Does the procedure **fp** really compute  $f'$ ? In this case, we can prove that it does by executing the procedure on symbolic parameters, in effect converting the function represented by the procedure into a formula.

```
> diff(f(x), x) - fp(x);
```

```
0
```

Such a check is *not* possible for procedures that contain a conditional statement involving a formal parameter (like is the case in the above function **F**). By the way, this is also one of the reasons for introducing automatic differentiation: not every function can be (conveniently) described by a formula.

How is the procedure **fp** constructed? Comparison of the source codes of **f** and **fp** gives a clue to the following general method of automatic differentiation:

For each assignment statement  $v := f(v_1, \dots, v_n)$  that appears in the procedure, where the  $v_i$  are local variables or formal parameters, precede it by  $v_x := fp(v_1, \dots, v_n)$  where  $fp(v_1, \dots, v_n)$  is obtained by differentiating  $f(v_1, \dots, v_n)$  formally. Replace the last statement (or any return value) by its derivative.

This method is called *forward automatic differentiation*; the function and its derivative are constructed in one sweep through the procedure. This is the method implemented in Maple. There are restrictions to what is allowed inside the procedure body: no recursive procedures, no option remember, no assignments to global variables, and only constant loop variables, are a few of the restrictions.

There exists another method, called *reverse automatic differentiation*, in which there is a forward sweep in the procedure to compute the function and a backward sweep to compute the derivative. The reverse mode of automatic differentiation is not (yet) implemented in Maple. For a gentle introduction to automatic differentiation we refer to [113, 114]. A good reference, in which many algorithms, implementations, and applications are described, is [115].

We end this section by illustrating another reason to use automatic differentiation: cost-effectiveness with respect to computing time and memory space. Suppose we are interested in the iterated powers  $f_n$  recursively defined by

$$f_1 = x, \quad f_n = x^{f_{n-1}} \text{ for } n > 1,$$

and their derivatives. As we have seen in §9.1, the formulae for the derivatives become already quite large for small values of  $n$ . On the other hand, the derivative procedure obtained by automatic differentiation is very compact. We shall use the following iterative definition of  $f_n$ .

```

> f := proc(x,n)
>   local i,t;
>   t := 1;
>   for i to n do t := x^t end do;
>   t
> end proc:
> fp := D[1](f); # 1st derivative of f w.r.t. x

fp := proc(x, n)
local tx, i, t;
tx := 0;
t := 1;
for i to n do tx := x^t * (tx * ln(x) + t /x); t := x^t end do;
tx
end proc

```

We shall compute the third derivative of  $f_{22}$  in two ways: by automatic differentiation, and by symbolic differentiation. There will be big differences in computing time and memory usage.

- *automatic differentiation*

- *symbolic differentiation*

```

cpu_time = 7.091 seconds
> memory_used = evalf[5]( 
> (kernelopts(bytesused)-setbytes)/1024*kbytes);
memory_used := 73265. kbytes

```

### 9.3 Exercises

1. Let  $f, g$ , and  $h$  be the following multivariate real functions.

$$\begin{aligned} f(x, y) &:= \frac{\ln(1 + x^4 + y^4)}{\sqrt{x^2 + y^2}} \\ g(x, y, z) &:= \frac{1}{\sqrt{(x - a)^2 + (y - b)^2 + (z - c)^2}} \\ h(x, y, z) &:= \frac{z}{x^2 + y^2 + z^2} \end{aligned}$$

- (a) Determine all partial derivatives of  $f$  of order 2.  
(b) Check that  $g$  is a solution of the Laplace differential equation, i.e.,

$$\left( \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} \right) g = 0.$$

- (c) Check that  $h$  is a solution of the differential equation

$$\frac{\partial^2 h}{\partial x \partial y} + \left( \frac{4x}{x^2 + y^2 + z^2} \right) \frac{\partial h}{\partial y} = 0.$$

2. Compare the results of the following Maple commands:

```

> diff(f(x), x);
> convert(% , D);
> unapply(% , x);

```

3. Compute the derivative of the function  $f(x) := \max(x^3, x)$ .  
4. Let the function  $y(x)$  be implicitly defined by  $\sqrt{x} + \sqrt{y} = 1$ . Compute the derivative  $y'$  and the second derivative  $y''$ .  
5. Let the bivariate function  $z(x, y)$  be implicitly defined by  $h(x, y, z) = 0$ , for some trivariate function  $h$ . Determine formulae for  $\frac{\partial z}{\partial x}$  and  $\frac{\partial^2 z}{\partial x \partial y}$ . What are the results for  $h = \sqrt{x} + \sqrt{y} + \sqrt{z} - 1$ ?  
6. Consider the function  $f_n$  recursively defined by

$$f_0 = 0, \quad f_1 = x, \quad f_n = f_{n-1} + \sin(f_{n-2}) \text{ for } n > 1.$$

Determine (by automatic differentiation) a procedure to compute the first derivative of  $f_n$ .

# 10

## Integration and Summation

Integration (both definite and indefinite), is one of the highlights in computer algebra. Maple uses non-classical algorithms such as the Risch algorithm for integrating elementary functions, instead of heuristic integration methods, which are described in most mathematics textbooks. We shall briefly discuss Maple's strategy to compute integrals and give many examples so that you can get an idea of Maple's capabilities and of ways to assist the system. Examples of integral transformations like Laplace, Fourier, Mellin, Hilbert, and Hankel transforms will also be given. Their application in the field of differential equations will be treated in Chapter 17.

Summation, which can be seen as the discrete analogue of integration, is another important topic in calculus for which Maple uses advanced methods. A few examples will illustrate this.

### 10.1 Indefinite Integration

Maple has several powerful built-in algorithms for integration of functions. First, it tries traditional techniques taught at school and university: look-up tables and pattern matching, integration by parts, change of variables, application of the chain rule, and so on. When these heuristic methods fail, then the system proceeds to deterministic methods, and in particular to the so-called Risch algorithm [31, 98, 203].

Let us first look at an example of indefinite integration performed by the procedure **int**, which is short for **integrate**. We shall always use the long name for clarity.

$$\int \frac{x}{x^3 + 1} dx = \frac{1}{6} \ln(x^2 - x + 1) + \frac{1}{3} \sqrt{3} \arctan\left(\frac{(2x - 1)\sqrt{3}}{3}\right) - \frac{1}{3} \ln(x + 1)$$

Note that Maple leaves out the constant of integration; the reason is that this is more convenient in the manipulation of the results. There is no harm in some common distrust; so let us check the answer.

$$\begin{aligned} > \text{diff(rhs(\%), x);} \\ & \frac{2x - 1}{6(x^2 - x + 1)} + \frac{2}{3(1 + \frac{(2x - 1)^2}{3})} - \frac{1}{3(x + 1)} \\ > \text{normal(\%, 'expanded');} \\ & \frac{x}{x^3 + 1} \end{aligned}$$

Later on in this chapter, you will see examples where such verifications of integration results are not so easily done.

In the above example, we used the inert form procedure **Integrate** for display reasons (we could also have used the short name **Int**. Henceforth we shall use **Integrate** to show the integrand and integration limits, and we shall use the procedure **value** to actually compute the integral.

One of the most handsome tricks you learn at school for integration of rational functions is partial fraction decomposition. It works well for the integral above. However, it becomes a more tedious and error-prone trick when you have to use it in cases like the following.

$$\begin{aligned} > \text{Integrate}(x/(x^5 + 1), x); \% = \text{value}(\%); \\ \int \frac{x}{x^5 + 1} dx &= \frac{1}{20} \ln(2x^2 - x - \sqrt{5}x + 2) - \frac{1}{20} \ln(2x^2 - x - \sqrt{5}x + 2)\sqrt{5} \\ &+ \frac{2}{5} \frac{\arctan\left(\frac{4x - 1 - \sqrt{5}}{\sqrt{10 - 2\sqrt{5}}}\right)\sqrt{5}}{\sqrt{10 - 2\sqrt{5}}} + \frac{1}{20} \ln(2x^2 - x + \sqrt{5}x + 2)\sqrt{5} \\ &+ \frac{1}{20} \ln(2x^2 - x + \sqrt{5}x + 2) - \frac{2}{5} \frac{\arctan\left(\frac{4x - 1 + \sqrt{5}}{\sqrt{10 + 2\sqrt{5}}}\right)\sqrt{5}}{\sqrt{10 + 2\sqrt{5}}} - \frac{1}{5} \ln(x + 1) \\ > \text{normal}(\text{diff}(\text{rhs}(\%), x), 'expanded'); \# \text{ check the answer} \\ & \frac{x}{x^5 + 1} \end{aligned}$$

And the trick of partial fraction decomposition does not work any more when the integrand is only changed a little bit.

```
> infolevel[integrate] := 2: # more information
> Integrate(x/(x^5+2*x+1), x); % = value(%);

int/indef1: first-stage indefinite integration
int/ratpoly: rational function integration
int/ratpoly/horowitz: integrating


$$\frac{x}{x^5 + 2x + 1}$$


int/ratpoly/horowitz: Horowitz' method yields


$$\int \frac{x}{x^5 + 2x + 1} dx$$


int/risch/ratpoly: Rothstein's method - factored resultant is


$$[[z^5 - \frac{500}{11317} z^3 + \frac{4}{11317} z + \frac{1}{11317}, 1]]$$


int/risch/ratpoly: result is


$$\begin{aligned} & \sum_{R=\%1} -R \ln(-X + \frac{65376045600}{64828679}) R^4 - \frac{4270752875}{64828679} R^3 \\ & - \frac{625000000}{64828679} R^2 + \frac{447235682}{64828679} R - \frac{21514240}{64828679} \end{aligned}$$

%1 := RootOf(11317 Z^5 - 500 Z^3 + 4 Z + 1)


$$\int \frac{x}{x^5 + 2x + 1} dx = \sum_{R=\%1} -R \ln(x + \frac{65376045600}{64828679}) R^4$$


$$- \frac{4270752875}{64828679} R^3 - \frac{625000000}{64828679} R^2 + \frac{447235682}{64828679} R - \frac{21514240}{64828679}$$

%1 := RootOf(11317 Z^5 - 500 Z^3 + 4 Z + 1)

> normal(diff(rhs(%), x), 'expanded'); # check the answer


$$\frac{x}{x^5 + 2x + 1}$$


> infolevel[integrate] := 0: # default information level
```

In the above example, we have assigned `infolevel[integrate]` the value two, so that we see what method Maple actually uses. Maple's general approach for integration of rational functions is as follows.

- In the “first-stage indefinite integration,” Maple tries simple methods such as table look-ups and selects on the basis of the integrand what method should be tried further on.
- If simple methods fail, Maple enters “the second stage indefinite integration.” Maple tries now heuristic methods like “derivative-divides”

(integrand of the form  $\frac{p}{q}$  where  $q'$  divides  $p$ ), substitutions, integration by parts, and partial fraction decomposition (for denominators of degree less than seven). Furthermore, Maple treats special forms such as  $f(ux + v)\frac{p(x)}{q(x)}$  and  $Bessel(x) \cdot p(x)$ , where  $f$  is either  $\exp$ ,  $\ln$ ,  $\sin$ ,  $\cos$ ,  $\sinh$ , or  $\cosh$ , and  $p$  and  $q$  are polynomials.

- If the heuristic methods of the second-stage indefinite integration fail, Maple enters the Risch algorithm. This goes as follows for rational functions.
  - (i) Horowitz-Ostrogradsky reduction [134] is applied to express the integral in the form  $\frac{c}{d} + \int \frac{a}{b}$ , where  $a$ ,  $b$ ,  $c$ , and  $d$  are polynomials,  $b$  is square-free, and  $\text{degree}(a) < \text{degree}(b)$ . In this form,  $\frac{c}{d}$  is called the rational part because the remaining integral can be expressed only by introducing logarithms. Thus,  $\int \frac{a}{b}$  is called the logarithmic part.
  - (ii) The Rothstein/Trager method [205, 223] is used to express the logarithmic part in the form  $\sum_i c_i \log v_i$ , where  $c_i$  are nonzero constants and  $v_i$  are monic, square-free, relatively prime polynomials of positive degree. Actually, the Lazard/Rioboo/Trager improvement [160], in which algebraic extensions and factorizations are avoided in the computation of the logarithmic part, has been implemented.

Of course, you may doubt the usefulness of the answer of the above example because it contains roots of a fifth degree polynomial that cannot be computed analytically. But the same algorithm is applied in cases like the following one taken from [222].

```
> Integrate((7*x^13+10*x^8+4*x^7-7*x^6-4*x^3-4*x^2+3*x+3)
>           / (x^14-2*x^8-2*x^7-2*x^4-4*x^3-x^2+2*x+1), x):
> % = value(%);


$$\int \frac{7x^{13} + 10x^8 + 4x^7 - 7x^6 - 4x^3 - 4x^2 + 3x + 3}{x^{14} - 2x^8 - 2x^7 - 2x^4 - 4x^3 - x^2 + 2x + 1} dx =$$


$$\frac{1}{2} \ln(x^7 - \sqrt{2}x^2 + (-1 - \sqrt{2})x - 1)$$


$$+ \frac{1}{2} \ln(x^7 - \sqrt{2}x^2 + (-1 - \sqrt{2})x - 1) \sqrt{2}$$


$$+ \frac{1}{2} \ln(x^7 + \sqrt{2}x^2 + (\sqrt{2} - 1)x - 1)$$


$$- \frac{1}{2} \ln(x^7 + \sqrt{2}x^2 + (\sqrt{2} - 1)x - 1) \sqrt{2}$$

> normal(diff(rhs(%),x), 'expanded'); # check the answer

$$\frac{7x^{13} + 10x^8 + 4x^7 - 7x^6 - 4x^3 - 4x^2 + 3x + 3}{x^{14} - 2x^8 - 2x^7 - 2x^4 - 4x^3 - x^2 + 2x + 1}$$

```

The strength of the Risch algorithm [31, 98, 203] comes into prominence when you apply it to a larger class of functions, viz., the class of *elementary functions*. Elementary functions of  $x$  can be roughly described as follows.

- Start with a set of constants, e.g.,  $\mathbb{Q}$ ,  $\mathbb{R}$ , or  $\mathbb{C}$ .
- Make polynomials  $p(x)$  with constant coefficients.
- Make rational functions  $\frac{p(x)}{q(x)}$ .
- Add exponentials of these (this will include the trigonometric and hyperbolic functions, and their inverses if the complex number  $i$  is one of the constants).
- Add logarithms of these (if we just go this far we have the *elementary transcendental functions*).
- Add algebraic functions, i.e., solutions of polynomial equations whose coefficients are functions of the types already introduced (e.g., add  $\sqrt{x^2 + 1}$ , which solves the equation  $y^2 - x^2 - 1 = 0$ ).

It is assumed in the construction that every step (except the choice of constants) can be iterated, so we can form a rational function of polynomials of logarithms and so on.

Risch [203] described an algorithm that, given an elementary transcendental function, decides whether its integral can be expressed as an elementary function, and if so, computes the integral. It is based on:

**Liouville's Principle** *If an elementary function  $f(x)$  has an elementary integral, then it has one of the form*

$$\int f(x) dx = v(x) + \sum_{i=1}^n c_i \log(u_i(x))$$

where the  $c_i$ 's are constants, and  $v$  and the  $u_i$ 's involve no quantity not already in  $f$  and the  $c_i$ 's.

Liouville's theorem bounds, so to speak, the search space for elementary integrals. A further generalization would be to take *Liouvillian functions*, defined as elementary functions together with anything of the form  $\int f(x) dx$  for any function  $f(x)$  already in the class considered; again the constructions can be iterated. This class includes, for example, the error function  $\text{erf}$  and the exponential integral function  $\text{Ei}$ , defined by

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x \exp(-t^2) dt \text{ and } \text{Ei}(x) = \int_x^\infty \frac{\exp(-t)}{t} dt, \text{ respectively.}$$

For details about the Risch algorithm, the interested reader is referred to [27, 28, 31, 68, 69, 96, 98, 183, 203]. Here, we only sketch Maple's general approach to integration of an elementary function:

- If the heuristic methods have failed, Maple applies first the “Risch–Norman front end” [96] to avoid introduction of complex exponentials and logarithms for integrands in which trigonometric and hyperbolic functions are involved.
- If still no answer is found, the Risch algorithm is entered.

In the final step, the most difficult part is the integration of algebraic functions. Trager’s method [224] for algebraic functions in **RootOf** notation has been implemented. But there are some restrictions in the current implementation.

- In the case where the coefficient field is not an algebraic number field, the transcendental part of the integral is not computed. In practice this means that when the integrand contains parameters, Maple may fail to find an answer even though it exists.
- The algorithm is slow due to the complexity.
- The integrand must contain a single **RootOf** involving the integration variable. The **evala@Primfield** procedure may be used to achieve this condition.
- Integration methods for nested **RootOfs** are not available.

Let us look at some examples to get a better idea of Maple’s integration capabilities.

```
> Integrate(ln(x-1)^2/x, x); % = value(%);


$$\int \frac{\ln(x-1)^2}{x} dx =$$


$$\ln(x-1)^2 \ln(x) + 2 \ln(x-1) \operatorname{polylog}(2, -x+1) - 2 \operatorname{polylog}(3, -x+1)$$

> simplify(diff(rhs(%), x)); # check the answer

$$\frac{\ln(x-1)^2}{x}$$

> integrate(ln(x-1)^2/x^2, x);

$$\int \frac{\ln(x-1)^2}{x^2} dx$$

```

The first of these integrals is computed by a table-driven integration by parts in the integrator’s front end. The Risch algorithm decides that the second integral cannot be expressed in closed form, i.e., in elementary functions. In such cases, Maple simply returns the input command or introduces a new function, i.e., extends the class of admissible functions (with functions such as the error function, exponential integral, sine integral, and so on).

```
> Integrate(exp(-x^2), x); % = value(%);
```

$$\int e^{(-x^2)} dx = \frac{1}{2} \sqrt{\pi} \operatorname{erf}(x)$$

```
> Integrate(exp(-x)/x, x); % = value(%);
```

$$\int \frac{e^{(-x)}}{x} dx = -\operatorname{Ei}(1, x)$$

In addition to heuristic methods, Maple uses Hermite reduction for integrands consisting of logarithms alone.

```
> Integrate(ln(1-b*x/(a+c*x^2))/x, x); % = value(%);
```

$$\begin{aligned} \int \frac{\ln(1 - \frac{bx}{ax + cx^2})}{x} dx &= \ln(x) \ln(\frac{ax + cx^2 - bx}{ax + cx^2}) + \ln(x) \ln(\frac{\sqrt{-ac} - cx}{\sqrt{-ac}}) \\ &+ \ln(x) \ln(\frac{\sqrt{-ac} + cx}{\sqrt{-ac}}) + \operatorname{dilog}(\frac{\sqrt{-ac} - cx}{\sqrt{-ac}}) + \operatorname{dilog}(\frac{\sqrt{-ac} + cx}{\sqrt{-ac}}) \\ &- \ln(x) \ln(\frac{b + \%1 - 2cx}{b + \%1}) - \ln(x) \ln(\frac{-b + \%1 + 2cx}{-b + \%1}) \\ &- \operatorname{dilog}(\frac{b + \%1 - 2cx}{b + \%1}) - \operatorname{dilog}(\frac{-b + \%1 + 2cx}{-b + \%1}) \end{aligned}$$

$$\%1 := \sqrt{b^2 - 4ac}$$

```
> Integrate(sin(x)/(x^3+x+1), x); % = value(%);
```

$$\int \frac{\sin(x)}{x^3 + x + 1} dx = \sum_{_R1=\%1} \frac{-\operatorname{Si}(-x + _R1) \cos(_R1) + \operatorname{Ci}(x - _R1) \sin(_R1)}{3 _R1^2 + 1}$$

$$\%1 := \operatorname{RootOf}(_Z^3 + _Z + 1)$$

Above, Maple noticed that the integrand involves a sine function and applied Hermite reduction for expressions consisting of trigonometric functions alone.

```
> Integrate(1/(x*(x^2+1)^(1/3)), x); % = value(%);
```

$$\begin{aligned} \int \frac{1}{x(x^2 + 1)^{(1/3)}} dx &= \frac{1}{4} \sqrt{3} \Gamma(\frac{2}{3}) \left( \frac{2}{3} \frac{(-\frac{1}{6} \pi \sqrt{3} - \frac{3}{2} \ln(3) + 2 \ln(x)) \pi \sqrt{3}}{\Gamma(\frac{2}{3})} \right. \\ &\quad \left. - \frac{2}{9} \frac{\pi \sqrt{3} x^2 \operatorname{hypergeom}([1, 1, \frac{4}{3}], [2, 2], -x^2)}{\Gamma(\frac{2}{3})} \right) / \pi \end{aligned}$$

```
> simplify(%);
```

$$\int \frac{1}{x(x^2+1)^{(1/3)}} dx = -\frac{1}{12} \pi \sqrt{3} - \frac{3}{4} \ln(3) + \ln(x) - \frac{1}{6} x^2 \text{hypergeom}([1, 1, \frac{4}{3}], [2, 2], -x^2)$$

In this case, Maple could also compute the integral without using special functions, but only with the use of radicals in **RootOf** notation. So, let us introduce  $\alpha$  as a cube root of  $x^2 + 1$ . From the result it will be clear that it is also convenient to introduce a root  $\beta$  of the polynomial  $z^2 + z + 1$ .

```
> alias(alpha=RootOf(z^3-x^2-1, z, index=1),
>       beta=RootOf(z^2+z+1, z));
> convert(Integrate(1/(x*(x^2+1)^(1/3)), x), RootOf);
```

$$\int \frac{1}{x \alpha} dx$$

In this notation, Maple can compute the integral.

```
> settime := time(); # start timing
> value(%); # evaluate integral
```

$$\begin{aligned} & \frac{1}{2} \ln\left(\frac{2x^2 + 5 - 9\beta x^2 - 19\beta + 4\beta^2 x^2 - 4\beta^2 - 24\alpha - 9\beta\alpha + 15\alpha^2 + 24\alpha^2\beta}{x^2}\right) \\ & + \frac{1}{2}\beta \ln\left(\frac{-8 - 6x^2 + 6\beta + \beta x^2 - \beta^2 + \beta^2 x^2 + 14\alpha + 7\beta\alpha - 14\alpha^2\beta - 7\alpha^2}{x^2}\right) \\ & > \# simplify RootOf's in the algebraic context \\ & > evala(Simplify(%)); \\ & \frac{1}{2} \ln\left(\frac{2x^2 - 9 + 13\beta x^2 + 15\beta + 24\alpha + 9\beta\alpha - 15\alpha^2 - 24\alpha^2\beta}{x^2}\right) \\ & + \frac{1}{2}\beta \ln\left(-\frac{7(1 + x^2 - \beta - 2\alpha - \beta\alpha + 2\alpha^2\beta + \alpha^2)}{x^2}\right) \\ & > \# check the answer \\ & > evala(Normal(diff(% , x) - 1/(x*alpha))); \\ & 0 \\ & > \text{cpu\_time} = (\text{time}() - \text{settime}) * \text{seconds}; \# computing time \\ & \text{cpu\_time} = 7.931 \text{ seconds} \end{aligned}$$

In [68], the Chebyshev integral [55]

$$\int \frac{2x^6 + 4x^5 + 7x^4 - 3x^3 - x^2 - 8x - 8}{(2x^2 - 1)^2 \sqrt{x^4 + 4x^3 + 2x^2 + 1}}$$

was computed with REDUCE. In Maple, the computation and the result are as follows.

```

> alias(beta=RootOf(z^2-x^4-4*x^3-2*x^2-1, z)):
> settim := time():
> Integrate((2*x^6+4*x^5+7*x^4-3*x^3-x^2-8*x-8)/
> ((2*x^2-1)^2*beta), x): % = value(%);


$$\int \frac{2x^6 + 4x^5 + 7x^4 - 3x^3 - x^2 - 8x - 8}{(2x^2 - 1)^2 \beta} dx = \frac{\beta x}{2x^2 - 1} + \frac{\beta}{2(2x^2 - 1)}$$


$$+ \frac{1}{2} \ln((1025x^{10} + 6138x^9 + 12307x^8 + 10188x^7 + 4503x^6 + 3134x^5$$


$$+ 1589x^4 + 140x^3 + 176x^2 + 2 - 4104\beta x^7 - 28\beta x - 10\beta x^2$$


$$- 624\beta x^3 - 805\beta x^4 - 2182\beta x^5 - 5084\beta x^6 - 1023\beta x^8) /$$


$$(2x^2 - 1)^5)$$

> # check the answer
> evala(Normal(diff(rhs(%), x)-op(1, lhs(%))));

0

> cpu_time=(time()-settim)*seconds; # computing time
cpu_time = 1.953 seconds

```

The computation of the transcendental part is most time-consuming.

Algebraic functions appear in the following two integrals.

```

> Integrate(x/(x^3+x+a), x): % = value(%);


$$\int \frac{x}{x^3 + x + a} dx = \sum_{-R=\%1} \frac{-R \ln(x - -R)}{3 - R^2 + 1}$$

%1 := RootOf(_Z^3 + _Z + a)
> Integrate(1/(x^16+a), x): % = value(%);


$$\int \frac{1}{x^{16} + a} dx = \sum_{-R=\%1} \left( \frac{1}{16} \frac{\ln(x - -R)}{-R^{15}} \right)$$

%1 := RootOf(_Z^16 + a)

```

The next example, which is a parameterization of one of the previous integrals, shows that the current implementation of integration of algebraic functions is limited to cases where the coefficient field is an algebraic number field. Below, we capitalize the message that says so.

```

> alias(gamma=RootOf(z^3-x^2-a, z)):
> infolevel[integrate] := 1: # set higher information level
> integrate(1/(x*gamma), x);

int/indef1: first-stage indefinite integration
int/indef2: second-stage indefinite integration
int/rischnorm: enter Risch-Norman integrator
int/rischnorm: exit Risch-Norman integrator
int/algrisch/int: Risch/Trager's algorithm for algebraic function

```

```

int/algrisch/int: computation of the algebraic part:
start time 18.025
int/algrisch/int: computation of the algebraic part:
end time 18.025
int/algrisch/int: computation of the transcendental part:
start time 18.025
int/algrisch/transcpar: PARAMETRIC CASE NOT HANDLED YET
int/algrisch/int: computation of the transcendental part:
end time 18.025
int/algrisch/int: could not find an elementary antiderivative
int/risch: enter Risch integration
int/algrisch/int: Risch/Trager's algorithm for algebraic function
int/algrisch/int: computation of the algebraic part:
start time 18.115
int/algrisch/int: computation of the algebraic part:
end time 18.115
int/algrisch/int: computation of the transcendental part:
start time 18.115
int/algrisch/transcpar: parametric case not handled yet
int/algrisch/int: computation of the transcendental part:
end time 18.115
int/algrisch/int: could not find an elementary antiderivative

```

$$\int \frac{1}{x^\gamma} dx$$

```
> infolevel[integrate] := 0: # back to default value
```

## 10.2 Definite Integration

Definite integration is also performed in Maple by **int** (and by the alias **integrate**). Again, we shall use the inert form **Integrate** in combination with the procedure **value** for display reasons.

```

> Integrate(x/(x^3+1), x=1..a): % = value(%);


$$\int_1^a \frac{x}{x^3 + 1} dx = -\frac{1}{3} \ln(1 + a) + \frac{1}{6} \ln(a^2 - a + 1)$$


$$+ \frac{1}{3} \sqrt{3} \arctan\left(\frac{1}{3} (2a - 1)\sqrt{3}\right) + \frac{1}{3} \ln(2) - \frac{\sqrt{3}\pi}{18}$$

> Integrate(ln(t)/(1-t), t=0..x): % = value(%);


$$\int_0^x \frac{\ln(t)}{1-t} dt = \operatorname{dilog}(x) - \frac{\pi^2}{6}$$

> Integrate(1/((1+x^2)*(1+2*x^2)), x=0..1): % = value(%);


$$\int_0^1 \frac{1}{(1+x^2)(1+2x^2)} dx = -\frac{\pi}{4} + \sqrt{2} \arctan(\sqrt{2})$$


```

Definite integration does not always compute the corresponding indefinite integral and substitute in the limits of integration as the next example, in which  $\int_{-1}^1 \frac{1}{x^2} dx$  is computed, illustrates.

```
> Integrate(1/x^2, x): % = value(%);
```

$$\int \frac{1}{x^2} dx = -\frac{1}{x}$$

```
> subs(x=1, rhs(%)) - subs(x=-1, rhs(%));
```

$$-2$$

The problem here is that the analytic conditions for application of the fundamental theorem of analysis are not fulfilled: there is a non-removable singularity at  $x = 0$  in the integrand within the interval  $(-1, 1)$ . Maple checks continuity of the integrand over the given interval and, in case of possible errors in the answer, simply returns the command. By the way, you can force continuity of the integrand over the interval by adding the keyword `continuous`. In the example chosen, Maple itself can prove that the definite integral diverges.

```
> Integrate(1/x^2, x=-1..1): % = value(%);
```

$$\int_{-1}^1 \frac{1}{x^2} dx = \infty$$

A more difficult example is

$$\int_0^{2\pi} \frac{1}{1 + 3 \sin^2 t} dt$$

```
> Integrate(1/(1+3*sin(t)^2), t=0..2*Pi): % = value(%);
```

$$\int_0^{2\pi} \frac{1}{1 + 3 \sin(t)^2} dt = \pi$$

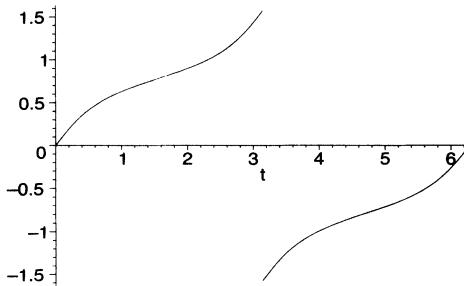
The answer is correct. But again, it is not the result of computing the indefinite integral and substituting the limits. The indefinite integral computed by Maple itself is even discontinuous over the interval of definite integration (see Figure 10.1) although there exists a continuous antiderivative, viz.,

$$\frac{t - \arctan \left( \frac{\sin(2t)}{\cos(2t) - 3} \right)}{2}$$

```
> Integrate(1/(1+3*sin(t)^2), t): % = value(%);
```

$$\int \frac{1}{1+3\sin(t)^2} dt = \frac{2 \arctan\left(\frac{2\tan(\frac{t}{2})}{4+2\sqrt{3}}\right)}{4+2\sqrt{3}} + \frac{\arctan\left(\frac{2\tan(\frac{t}{2})}{4+2\sqrt{3}}\right)\sqrt{3}}{4+2\sqrt{3}} - \frac{\arctan\left(\frac{2\tan(\frac{t}{2})}{4-2\sqrt{3}}\right)\sqrt{3}}{4-2\sqrt{3}} + \frac{2 \arctan\left(\frac{2\tan(\frac{t}{2})}{4-2\sqrt{3}}\right)}{4-2\sqrt{3}}$$

> plot(rhs(%), t=0..2\*Pi, discont=true);

Figure 10.1. Graph of discontinuous antiderivative of  $\frac{1}{1+3\sin^2 t}$ .

In many cases Maple uses look-up tables and pattern matching [95, 207], and differentiation of special functions with respect to a parameter to compute definite integrals.

```
> Integrate(exp(-sqrt(t))/(t^(1/4)*(1-exp(-sqrt(t)))),  
> t=0..infinity): % = value(%);
```

$$\int_0^\infty \frac{e^{-\sqrt{t}}}{t^{(1/4)} (1 - e^{-\sqrt{t}})} dt = \sqrt{\pi} \zeta\left(\frac{3}{2}\right)$$

```
> evalf(rhs(%));  
4.630314748
```

```
> Integrate(t^4*ln(t)^2/(1+3*t^2)^3, t=0..infinity):  
> % = value(%);
```

$$\int_0^\infty \frac{t^4 \ln(t)^2}{(1+3t^2)^3} dt = \frac{\sqrt{3}\pi}{216} + \frac{\pi^3 \sqrt{3}}{576} - \frac{1}{108}\pi\sqrt{3}\ln(3) + \frac{1}{576}\pi\sqrt{3}\ln(3)^2$$

These are examples of integrals of the general form

$$\int_0^\infty \frac{\exp(-u_1 t^{s_1} - u_2 t^{s_2}) t^w \ln(b t^{d_1})^m \begin{Bmatrix} \cos \\ \sin \end{Bmatrix} (c t^r)}{(a_0 + a_1 f^d)^p} dt,$$

where

$f = t$  or  $\exp(t^s)$ ,  
 $p$  and  $m$  are nonnegative integers,  
 $\text{signum}(a_0/a_1) > 0$ ,  
 $s_1$  and  $s_2$  are nonzero real numbers,  
 $d$  and  $d_l$  are any real numbers,  
 $b$ ,  $u_1$ , and  $u_2$  are positive real numbers,  
 $w$  is a complex number such that  $\Re\left(\frac{w+1}{s}\right) > 0$ , and  
 $r = 0$  or  $r = s$  or  $r = 2s$  or  $s = 2r$ .

Other types of definite integrals that are computed by pattern matching are

$$\int_0^\infty \exp(-u t^s) t^w \ln(b t^{d_l})^m \begin{Bmatrix} \text{erf} \\ \text{erfc} \end{Bmatrix} (f t^v + g) dt,$$

$$\int_0^\infty \exp(-u t^s) t^w \text{BesselJ}(b, c t^{s_p})^{k_J} \text{BesselY}(\pm b, c t^{s_p})^{k_Y} dt,$$

and

$$\int_{t_0}^{t_1} \begin{Bmatrix} \sin \\ \cos \end{Bmatrix} \left( z \begin{Bmatrix} \sin \\ \cos \end{Bmatrix} (a t) \right) \begin{Bmatrix} \sin \\ \cos \end{Bmatrix} (b t) dt.$$

Two more examples:

```
> Integrate(t*exp(-t^2)*erf(2*t+1), t=0..infinity):
> % = value(%);
```

$$\int_0^\infty t e^{-t^2} \operatorname{erf}(2t+1) dt =$$

$$\frac{1}{2} \operatorname{erf}(1) + \frac{1}{5} e^{-(-1/5)} \sqrt{5} - \frac{1}{5} e^{(-1/5)} \sqrt{5} \operatorname{erf}\left(\frac{2\sqrt{5}}{5}\right)$$

$$\int_0^\infty x e^{-x^2} \text{BesselJ}(0, x) \text{BesselY}(0, x) dx = -\frac{1}{2} \frac{e^{(-1/2)} \text{BesselK}(0, \frac{1}{2})}{\pi}$$

$$\int_0^\infty \sin(z \sin(x)) \sin(3x) dx =$$

```
> Integrate(sin(z*sin(x))*sin(3*x), x=0..Pi):
> % = value(%);
```

$$\int_0^\pi \sin(z \sin(x)) \sin(3x) dx = \\ \frac{8\pi \text{BesselJ}(1, z)}{z^2} - \frac{4\pi \text{BesselJ}(0, z)}{z} - \pi \text{BesselJ}(1, z)$$

Elliptic integrals of the first, second, and third kind are available both in algebraic form (Jacobi's notation) and in Legendre's form. Two notations exist: one that follows the Handbook of Elliptic Integrals by Byrd and Friedman [40] and one that follows the Handbook of Mathematical Functions [2]. See Table 10.1.

Description	Definition	Maple function
incomplete elliptic integral of the 1st kind	$\int_0^x \frac{1}{\sqrt{(1-t^2)(1-k^2t^2)}} dt$	$\text{LegendreF}(x, k)$ $\text{EllipticF}(x, k)$
incomplete elliptic integral of the 2nd kind	$\int_0^x \frac{\sqrt{1-k^2t^2}}{\sqrt{1-t^2}} dt$	$\text{LegendreE}(x, k)$ $\text{EllipticE}(x, k)$
incomplete elliptic integral of the 3rd kind	$\int_0^x \frac{1}{(1-at^2)\sqrt{(1-t^2)(1-k^2t^2)}} dt$	$\text{LegendrePi}(x, a, k)$ $\text{EllipticPi}(x, a, k)$
complete elliptic integral of the 1st kind	$\int_0^1 \frac{1}{\sqrt{(1-t^2)(1-k^2t^2)}} dt$	$\text{LegendreKc}(k)$ $\text{EllipticK}(k)$
complete elliptic integral of the 2nd kind	$\int_0^1 \frac{\sqrt{1-k^2t^2}}{\sqrt{1-t^2}} dt$	$\text{LegendreEc}(k)$ $\text{EllipticE}(k)$
complete elliptic integral of the 3rd kind	$\int_0^1 \frac{1}{(1-at^2)\sqrt{(1-t^2)(1-k^2t^2)}} dt$	$\text{LegendrePic}(a, k)$ $\text{EllipticPi}(a, k)$
associated complete elliptic integral of the 1st kind	$\int_0^1 \frac{1}{\sqrt{(1-t^2)(1-c^2t^2)}} dt$	$\text{LegendreKc1}(k)$ $\text{EllipticCK}(k)$
associated complete elliptic integral of the 2nd kind	$\int_0^1 \frac{\sqrt{1-c^2t^2}}{\sqrt{1-t^2}} dt$	$\text{LegendreEc1}(k)$ $\text{EllipticCE}(k)$
associated complete elliptic integral of the 3rd kind	$\int_0^1 \frac{1}{(1-at^2)\sqrt{(1-t^2)(1-c^2t^2)}} dt$	$\text{LegendrePic1}(a, k)$ $\text{EllipticCPi}(a, k)$

Table 10.1. Some elliptic integrals in Maple.

Here,  $0 < k < 1$ , and  $c = \sqrt{1 - k^2}$ .

The notation of the Handbook of Mathematical Functions [2] is the one Maple actually prefers most. A few examples:

```
> Integrate(1/sqrt((t^2-1)*(t^2-2)), t=2..infinity):
> % = value(%);
```

$$\int_2^\infty \frac{1}{\sqrt{(t^2-1)(t^2-2)}} dt = \frac{1}{2} \sqrt{2} \text{EllipticF}\left(\frac{1}{2} \sqrt{2}, \frac{1}{2} \sqrt{2}\right)$$

```
> Integrate(1/sqrt((t-1)*(t-2)*(t-3)*(t-4)),
> t=4..infinity): % = value(%);
```

$$\int_4^\infty \frac{1}{\sqrt{(t-1)(t-2)(t-3)(t-4)}} dt = \frac{2}{3} \text{EllipticK}\left(\frac{1}{3}\right)$$

> **Integrate**(1/sqrt(t\*(1-t)\*(1+t)), t=0..1); % = value(%);

$$\int_0^1 \frac{1}{\sqrt{t(1-t)(t+1)}} dt = \sqrt{2} \text{EllipticK}\left(\frac{1}{2}\sqrt{2}\right)$$

> **Integrate**(1/(1-1/9\*sin(t)^2)^(5/2), t=0..Pi/2);

> % = value(%);

$$\int_0^{1/2\pi} \frac{1}{(1 - \frac{1}{9} \sin(t)^2)^{(5/2)}} dt = -\frac{3}{8} \text{EllipticK}\left(\frac{1}{3}\right) + \frac{51}{32} \text{EllipticE}\left(\frac{1}{3}\right)$$

## 10.3 Numerical Integration

A computer algebra system aims at getting exact results, but Maple also has several numerical facilities, e.g., numerical integrators.

> **integrate**(exp(arcsin(x)), x=0..1);

$$\int_0^1 e^{\arcsin(x)} dx$$

> **evalf**(%);

$$1.905238690$$

> **Integrate**(exp(-2\*t)\*t\*ln(t), t=0..infinity);

> [% , value(%)];

$$\left[ \int_0^\infty e^{-2t} t \ln(t) dt, -\frac{1}{4} \ln(2) + \frac{1}{4} - \frac{1}{4} \gamma \right]$$

> **evalf**[20](%);

$$[-0.067590711365369542506, -0.06759071136536954251]$$

Note that we have used above the inert form **Integrate** instead of the active **integrate**. This is to avoid computing of exact results but going straight on to numerical approximation via the composition **evalf@Integrate** (or more specifically via the procedure **evalf/int**). The numerical integration problem is first passed to NAG integration routines if **Digits** is not too large and no particular integration method is mentioned. If the NAG routines cannot perform the integration, then the next invoked numerical integration method will be by default an adaptive Clenshaw-Curtis quadrature method. When convergence is slow (due to nearby singularities) the system tries to remove the singularities or switches to an adaptive double-exponential quadrature method. An adaptive variable-order Gaussian quadrature, an adaptive sinc quadrature, and an adaptive Newton-Cotes method (only if low precision, e.g., **Digits <= 15** suffices)

are available. Generalized series expansions and variable transformations are two of the techniques used in the Maple procedure **evalf/int** to deal with singularities in an analytic integrand. The interested reader is referred to [93, 97].

The optional argument **method** indicates the preferred integration method. Via the optional arguments **digits** and **epsilon** you can control the number of digits of precision for the computation and the relative error tolerance for the computed result, respectively. In the help page **?evalf,int** you can find details about available methods and options in the numerical integration facility.

```
> evalf(Integrate(1/sqrt(x), x=0..1, epsilon=0.5e-3,
>     method=_Dexp));
1.999998382
> evalf[4](%);
2.000
> evalf(Integrate(1/sqrt(x), x=0..1,
>     epsilon=0.5e-3, method=_d01akc)); # apply NAG routine
1.999989075
```

Double and triple integrals can of course be computed by repeatedly calling the integration procedure. In case of numerical integration you may also use a special multiple integration notation with a list as the second argument instead of nested one-dimensional integrals. The following examples illustrate this.

```
> Integrate(Integrate(x^2+y^2, y=0..1), x=0..1);

$$\int_0^1 \int_0^1 x^2 + y^2 \, dy \, dx$$

> % = value(%);

$$\int_0^1 \int_0^1 x^2 + y^2 \, dy \, dx = \frac{2}{3}$$

> evalf(Integrate(x^2+y^2, [y=0..1, x=0..1]));
0.6666666667
```

If the limits of integration are finite constants and low accuracy (less than 5 digits of accuracy) suffices the Monte Carlo method **d01gbc** from the NAG library can be applied. We recompute the above integral.

```
> evalf(Integrate(x^2+y^2, [y=0..1, x=0..1],
>     method=_MonteCarlo, epsilon=0.5e-4));
0.6666558697
```

Limits of integration do not need to be finite constants.

```
> Integrate(Integrate(x^2+y^2, y=0..sqrt(1-x^2)), x=0..1):
> % = value(%);
```

```


$$\int_0^1 \int_0^{\sqrt{1-x^2}} x^2 + y^2 dy dx = \frac{\pi}{8}$$

> evalf(rhs(%));
0.3926990818
> evalf(Integrate(x^2+y^2, [y=0..sqrt(1-x^2), x=0..1]));
0.3926990817

```

## 10.4 Integral Transforms

In this section, we shall give examples of integral transformations such as Laplace transforms, Fourier transforms, Mellin transforms, Hilbert transforms, and Hankel transforms. In general, the integral transform  $\mathcal{T}(f)$  of a function  $f$  with respect to the kernel  $K$  is defined by

$$\mathcal{T}(f)(s) = \int_a^b f(t) K(s, t) dt,$$

at least when this integral exists. The Fourier, Laplace, and Mellin transforms are the most popular ones, with kernels  $e^{-ist}$ ,  $e^{-st}$ , and  $t^{s-1}$ , respectively. In Table 10.2 we list the integral transforms that are available in Maple via the `inttrans` package.

Transform	Definition	Maple Function
Fourier	$\int_{-\infty}^{\infty} f(t) e^{-ist} dt$	<code>fourier(f(t), t, s)</code>
Fourier-Bessel, Hankel	$\int_0^{\infty} f(t) \sqrt{st} J_n(st) dt$	<code>hankel(f(t), t, s, n)</code>
Fourier-Cosine	$\sqrt{\frac{2}{\pi}} \int_0^{\infty} f(t) \cos(st) dt$	<code>fouriercos(f(t), t, s)</code>
Fourier-Sine	$\sqrt{\frac{2}{\pi}} \int_0^{\infty} f(t) \sin(st) dt$	<code>fouriersin(f(t), t, s)</code>
Hilbert	$\frac{1}{\pi} \int_{-\infty}^{\infty} \frac{f(t)}{t-s} dt$	<code>hilbert(f(t), t, s)</code>
Laplace	$\int_0^{\infty} f(t) e^{-st} dt$	<code>laplace(f(t), t, s)</code>
Mellin	$\int_0^{\infty} f(t) t^{s-1} dt$	<code>mellin(f(t), t, s)</code>

Table 10.2. Integral transforms in Maple

Expressions in the form of sums of rational expressions of polynomials or certain functions like the **Dirac** function, the **Heaviside** function, or Bessel functions can be transformed to their Laplace transform by the procedure **laplace**. The corresponding inverse Laplace transform can be computed with **invlaplace**.

```
> with(inttrans);

[addtable, fourier, fouriercos, fouriersin, hankel, hilbert, invfourier,
invhilbert, invlaplace, invmellin, laplace, mellin, savetable]
> t*BesselJ(0,a*t);
t BesselJ(0, a t)
> laplace(% , t , s);
s
-----
(s^2 + a^2)^(3/2)
> invlaplace(% , s , t);
t BesselJ(0, a t)
> (cosh(3*t)-3*t*sinh(3*t)-1)/t^2;
cosh(3 t) - 3 t sinh(3 t) - 1
-----
t^2
> laplace(% , t , s);
- 1
-- s (2 ln(s) - ln((s - 3) (s + 3)))
2
> invlaplace(% , s , t);
- invlaplace(s ln(s), s, t) + 1
-- invlaplace(s ln(s + 3), s, t)
2
+ 1
-- invlaplace(s ln(s - 3), s, t)
2
```

We were too optimistic; Maple is not able to convert the problem to inverse Laplace transforms that are tabulated internally. We have to assist Maple a bit. First, we rewrite the formula and specify that  $s > 3$ .

```
> interface(showassumed=2): assume(s>3):
> 1/2*collect(2*%, s);
1
-- (-2 ln(s) + ln((s - 3) (s + 3))) s
2
> F := combine(% , ln, integer);
F := 1
-- ln((s - 3) (s + 3)) s
2
```

Now, Maple can still not compute the inverse Laplace transform. But a higher information level will indicate why not.

```

> infolevel[invlaplace] := 5:
> invlaplace(F, s, t);

invlaplace: Entering top-level invlaplace with
1/2*ln((s-3)*(s+3)/s^2)*s   s   t
inttrans/invlaplace/main: Working on equation (invlaplace main)
(ln(tt+3)-2*ln(tt)+ln(tt-3))*tt
inttrans/invlaplace/main: Working on equation (invlaplace main)
tt*ln(tt+3)
inttrans/invlaplace/main: Working on equation (invlaplace main)
ln(tt+3)
inttrans/invlaplace/main: Working on equation (invlaplace main)
1/(tt+3)
inttrans/invlaplace/main: invlaplace main 4   exp(-3*t)
inttrans/invlaplace/main: invlaplace main 4   -exp(-3*t)/t
inttrans/invlaplace/main: invlaplace main 3   FAIL
.....

```

$$\begin{aligned} & \frac{1}{2} \operatorname{invlaplace}(s \ln(s+3), s, t) - \operatorname{invlaplace}(s \ln(s), s, t) \\ & + \frac{1}{2} \operatorname{invlaplace}(s \ln(s-3), s, t) \end{aligned}$$

The problem is that Maple uses internally a copy (*tt*) of the variable *s* with another condition (*tt*>0). If you use a more deeply built-in procedure, then Maple will find the answer.

```

> 'inttrans/invlaplace/main'(F, s, t);

inttrans/invlaplace/main: Working on equation (invlaplace main)
1/2*ln((s-3)*(s+3)/s^2)*s
.....

$$-\frac{3 \sinh(3t)}{t} + \frac{1}{2} \frac{-2 + 2 \cosh(3t)}{t^2}$$


```

We check the answer.

```

> normal(%);


$$\frac{\cosh(3t) - 3t \sinh(3t) - 1}{t^2}$$


```

The next example shows that integral transforms can be used to enlarge the class of integration problems that can be solved through Maple.

```

> integrate(x^5*BesselJ(0,x), x=0..t);

$$t^5 \left( \frac{8}{3} \frac{(-12 + \frac{3t^2}{2}) \operatorname{BesselJ}(0, t)}{t^3} + \frac{16}{3} \frac{(-3t^2 + 12 + \frac{3}{16}t^4) \operatorname{BesselJ}(1, t)}{t^4} \right)$$

> simplify(%);


$$\begin{aligned} & t(-32 \operatorname{BesselJ}(0, t) + 4 \operatorname{BesselJ}(0, t)t^3 - 16 \operatorname{BesselJ}(1, t)t^2 \\ & + 64 \operatorname{BesselJ}(1, t) + \operatorname{BesselJ}(1, t)t^4) \end{aligned}$$


```

```
> collect(%, BesselJ);

$$t(-32t + 4t^3) \text{BesselJ}(0, t) + t(-16t^2 + 64 + t^4) \text{BesselJ}(1, t)$$

```

We check the answer.

```
> expand(diff(%, t));

$$t^5 \text{BesselJ}(0, t)$$

```

It can be proved that

$$\int_0^t x^m J_n(x) dx,$$

where  $m \geq n \geq 0$  can be expressed in terms of Bessel functions and powers of  $t$  in case  $(m + n)$  is odd. Let us assist Maple: we compute the Laplace transform, simplify the intermediate result, apply the inverse Laplace transform, and we determine the integration constant.

```
> with(inttrans):
> laplace(Integrate(x^5*BesselJ(0,x), x=0..t), t, s);

$$\frac{\frac{945 s^5}{(s^2 + 1)^{(11/2)}} - \frac{1050 s^3}{(s^2 + 1)^{(9/2)}} + \frac{225 s}{(s^2 + 1)^{(7/2)}}}{s}$$

> normal(%);

$$\frac{15 (8 s^4 - 40 s^2 + 15)}{(s^2 + 1)^{(11/2)}}$$

> invlaplace(% , s, t);

$$\frac{8}{21} t^3 \text{BesselJ}(3, t) + \frac{16}{21} t^4 \text{BesselJ}(2, t) + \frac{8}{63} t^5 \text{BesselJ}(1, t) - \frac{40}{63} t^4 \text{BesselJ}(4, t) - \frac{40}{63} t^5 \text{BesselJ}(3, t) + \frac{5}{21} t^5 \text{BesselJ}(5, t)$$

> eval(% , t=0);
0
```

Let us check the answer by differentiation.

```
> expand(diff(%%, t));

$$t^5 \text{BesselJ}(0, t)$$

> expand(%%%);

$$64 t \text{BesselJ}(1, t) - 32 t^2 \text{BesselJ}(0, t) - 16 t^3 \text{BesselJ}(1, t) + 4 t^4 \text{BesselJ}(0, t) + t^5 \text{BesselJ}(1, t)$$

> collect(%, BesselJ, factor);

$$4 t^2 (-8 + t^2) \text{BesselJ}(0, t) + t (-8 + t^2)^2 \text{BesselJ}(1, t)$$

> expand(diff(%, t));
```

$$t^5 \text{BesselJ}(0, t)$$

Conclusion:

$$\int x^5 J_0(x) dx = \frac{8}{21} x^3 J_3(x) + \frac{16}{21} x^4 J_2(x) + \frac{8}{63} x^5 J_1(x) - \frac{40}{63} x^4 J_4(x) - \frac{40}{63} x^5 J_3(x) + \frac{5}{21} x^5 J_5(x).$$

and

$$\int x^5 J_0(x) dx = x(-8 + x^2)^2 J_1(x) + 4x^2(-8 + x^2) J_0(x)$$

Like any integral transform, the main application of Laplace transforms is in the field of differential equations and integral equations. Differential equations will be studied extensively in Chapter 17. Here, we give an example of an integral equation.

```
> with(inttrans):
> int_eqn := integrate(exp(a*x)*f(t-x), x=0..t)
>     + b*f(t) = t;
int_eqn :=  $\int_0^t e^{(ax)} f(t-x) dx + b f(t) = t$ 
> laplace(% , t , s);

$$\frac{\text{laplace}(f(t), t, s)}{s - a} + b \text{laplace}(f(t), t, s) = \frac{1}{s^2}$$

> isolate(% , laplace(f(t), t, s));

$$\text{laplace}(f(t), t, s) = \frac{1}{s^2 (\frac{1}{s - a} + b)}$$

> invlaplace(% , s , t);

$$f(t) = \frac{at}{-1 + ba} - \frac{2 \sinh(\frac{(-1 + ba)t}{2b}) e^{(\frac{(-1 + ba)t}{2b})}}{(-1 + ba)^2}$$

> normal(%);

$$f(t) = \frac{-at + a^2 tb - 2 \sinh(\frac{(-1 + ba)t}{2b}) e^{(\frac{(-1 + ba)t}{2b})}}{(-1 + ba)^2}$$

```

Let us check the answer.

```
> f := unapply(rhs(%), t);
f := t →  $\frac{-at + a^2 tb - 2 \sinh(\frac{1}{2} \frac{(-1 + ba)t}{b}) e^{(1/2 \frac{(-1 + ba)t}{b})}}{(-1 + ba)^2}$ 
> testeql(int_eqn);
```

*true*

The procedure **fourier** can transform sums of rational functions of polynomials to their Fourier transforms.

```
> 1/(1+t^3);

$$\frac{1}{1 + t^3}$$

> fourier(% , t , omega);

$$\frac{1}{3} \pi ((e^{(\omega I)} I - e^{(-1/2 I \omega)} e^{(\frac{\sqrt{3} \omega}{2})} I + e^{(-1/2 I \omega)} \sqrt{3} e^{(\frac{\sqrt{3} \omega}{2})}) \text{Heaviside}(-\omega) + \\ (e^{(-1/2 I \omega)} \sqrt{3} e^{(-\frac{\sqrt{3} \omega}{2})} + e^{(-1/2 I \omega)} e^{(-\frac{\sqrt{3} \omega}{2})} I - e^{(\omega I)} I) \text{Heaviside}(\omega))$$

```

We can check the answer by computing the inverse Fourier transform with **invfourier**.

```
> invfourier(% , omega , t);

$$\frac{1}{(t + 1)(t^2 - t + 1)}$$

> normal(% , 'expanded');

$$\frac{1}{1 + t^3}$$

```

Maple can also apply convolution methods, table look-ups, and definite integration to handle special functions such as trigonometric functions, the **Dirac** and **Heaviside** functions, the exponential function, and Bessel functions.

```
> fourier(BesselJ(0,t) , t , omega);

$$\frac{2 (\text{Heaviside}(1 + \omega) - \text{Heaviside}(\omega - 1))}{\sqrt{1 - \omega^2}}$$

> fourier(BesselJ(0, sqrt(t^2+1)) , t , omega);

$$\frac{2 e^{(\omega I)} \cos(-1 + \omega^2) (\text{Heaviside}(1 + \omega) - \text{Heaviside}(\omega - 1))}{\sqrt{1 - \omega^2}}$$

```

You can define Fourier transforms for your own functions with the procedure **addtable** and save this information to a file with the procedure **savetable**. For example, Maple does not know the Fourier transform of  $x/\sinh x$ . When you give this function a name, say  $F$ , then you can add it to the look-up table of **fourier** in the following way.

```
> addtable(
>   fourier,    # name of integral transform
>   F(t),      # name of user function
>   2/Pi*exp(Pi*w)/(1+exp(Pi*w))^2, # transform
>   t, w       # variables used in transform
> ):
```

```

> fourier(F(x), x, omega);

$$\frac{2\pi e^{(\pi\omega)}(-1 + 4e^{(\pi\omega)} - (e^{(\pi\omega)})^2)}{(1 + e^{(\pi\omega)})^4}$$

> fourier(x^2*F(x), x, omega);

$$\frac{2\pi e^{(\pi\omega)}(-1 + 4e^{(\pi\omega)} - (e^{(\pi\omega)})^2)}{(1 + e^{(\pi\omega)})^4}$$


```

In the same way you can add with **addtable** an item to the look-up table of any integral transform.

**fourier** and **invfourier** are procedures for computing symbolic Fourier transforms and their inverses. The procedures **FFT** and **iFFT** are meant for numerical Fast Fourier Transforms and their inverses. Recall the definition of the Fourier transform  $X = [X_0, X_1, \dots, X_{N-1}]$  for a list of complex numbers  $x = [x_0, x_1, \dots, x_N]$  of length  $N$ .

$$X[k] = \sum_{j=0}^{N-1} x_j e^{-2\pi i j k / N},$$

for  $0 \leq k \leq N-1$ . For  $N$  equal to a power of two, the so-called Fast Fourier Transform Method [62] has been implemented. In the first example below, we take  $N = 2^3$ , and compute the fast Fourier transform of the sequence of real numbers  $[-1, -1, -1, -1, 1, 1, 1, 1]$ .

```

> x := Array([-1,-1,-1,-1,1,1,1,1]): # real parts of data
> y := Array([0,0,0,0,0,0,0,0]): # imaginary parts of data
> FFT(3,x,y) # transform data
> x;          # real parts of transformed data
[0, -2.000000001, 0., -1.99999999, 0, -1.99999999, 0., -2.000000001]
> y; # imaginary parts of transformed data
[0, 4.828427122, 0., 0.828427124, 0, -0.828427124, 0., -4.828427122]

```

The procedure **zip** comes in handy when you want to write the complex numbers in a more conventional way.

```

> # normal complex notation
> zip((a,b)->a+b*I, x, y): convert(% , list);

[0, -2.000000001 + 4.828427122 I, 0. + 0. I, -1.99999999 + 0.828427124 I,
0, -1.99999999 - 0.828427124 I, 0. + 0. I, -2.000000001 - 4.828427122 I]
> iFFT(3,x,y): # check results
> x;
[-1.000000000, -0.9999999990, -0.9999999995, -0.9999999985,
1.000000000, 0.9999999990, 0.9999999995, 0.9999999985]
> y;

```

$$[0., 0.2500000000 \cdot 10^{-9}, 0., -0.2500000000 \cdot 10^{-9}, 0., \\ -0.2500000000 \cdot 10^{-9}, 0., 0.2500000000 \cdot 10^{-9}]$$

The procedure **fnormal** comes in handy when you want to normalize small values to zero.

```
> y := map(fnormal, y);
y := [0., 0., 0., -0., 0., -0., 0., 0.]
```

A common and important application of fast Fourier transforms is doing convolutions in data smoothing. Below is a simple, but artificial example. It is presented without comments, but all data types and procedures used will be dealt with later on in this book.

First, we load the random number generator for the standard normal distribution from the **stats** package.

```
> noise := stats[random, normald]:
```

Next, we generate data; in particular, we generate the real and imaginary parts of the data separately.

```
> re_data := Array([seq(sin(0.0625*k)+0.1*noise(), k=1..2^8)]):
> im_data := Array([seq(0, k=1..2^8)]):
```

To get an idea of what data we have in hand, we plot the data points. By construction it will look like the sine function as can be seen in Figure 10.2.

```
> xcoords := Array([seq(0.0625*k, k=1..2^8)]):
> plotdata := convert(zip((a,b)->[a,b], xcoords,
> re_data), list):
> plot(plotdata, style=POINT);
```

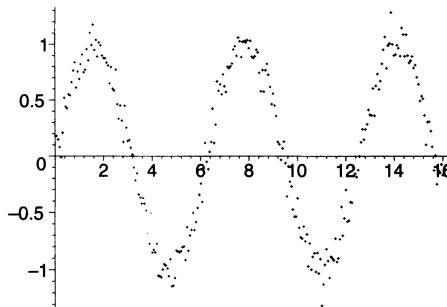


Figure 10.2. Sine function with noise.

We shall use the kernel function  $t \rightarrow \exp(-\frac{100}{256}t^2)$  for smoothing the data.

```
> re_kernel := Array([seq(exp(-100.0*(k/2^8)^2),
> k=1..2^8)]):
> im_kernel := Array([seq(0, k=1..2^8)]):
```

We compute the fast Fourier transforms of the data and kernel and write them as arrays of complex numbers.

```
> FFT(8, re_data, im_data):
> FFT(8, re_kernel, im_kernel):
> data := zip((a,b)->(a+b*I), re_data, im_data):
> kernel := zip((a,b)->(a+b*I), re_kernel, im_kernel):
> newdata := zip((a,b)->a*b, data, kernel):
> new_re_data := map(Re, newdata):
> new_im_data := map(Im, newdata):
```

Finally, we compute the inverse Fast Fourier Transform of the product of Fourier transforms of data and kernel. Up to a scalar, this is a smooth version of the original data. Figure 10.3 shows this best.

```
> iFFT(8, new_re_data, new_im_data):
> plotdata := convert(zip((a,b)->[a,b], xcoords,
> new_re_data), list):
> plot(plotdata, style=POINT);
```

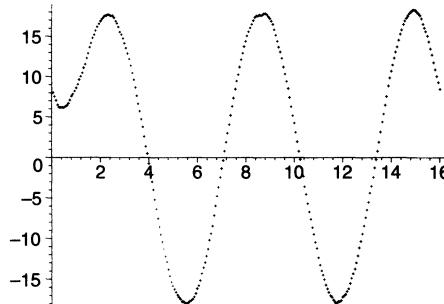


Figure 10.3. Data smoothing.

We end this section on integral transforms with some examples of the other integral transforms available in Maple.

- Hankel transform

```
> restart: with(inttrans):
> hankel(sqrt(t), t, s, k) assuming k::posint;
```

$$\frac{k}{s^{(3/2)}}$$

- Fourier-Sine and Fourier-Cosine transform

```
> fouriercos(Heaviside(a-t), t, s) assuming a>0;
```

$$\frac{\sqrt{2} \sin(as)}{\sqrt{\pi}s}$$

```
> fouriercos(% , s, t);
```

$$\text{Heaviside}(a-t)$$

- Hilbert transform

```
> hilbert(Dirac(x)+sin(k*x)/x, x, y)
> assuming k::posint;
```

$$\frac{-1 + \pi \cos(yk) - \pi}{y\pi}$$

- Mellin transform

```
> mellin(1/(t+c), t, s);

$$\pi c^{(s-1)} \csc(\pi s)$$

> convert(%, 'sincos');

$$\frac{\pi c^{(s-1)}}{\sin(\pi s)}$$

> mellin(exp(-t), t, s);

$$\Gamma(s)$$

> invmellin(%, s, t);
```

*Invmellin can transform GAMMA(t) if Re(t) <> 0, Re(t) > -1*

```
> invmellin(%%, s, t, 1..infinity);
```

$$e^{(-t)}$$

## 10.5 Assisting Maple's Integrator

In this section, we shall look at some integration problems where Maple cannot find a solution all by itself, but with some assistance actually can solve the problem. Human assistance is quite often needed when non-elementary functions are involved or when parameters must satisfy certain conditions.

The first integration problem involves a parameter that must satisfy a certain condition in order that the integral can be solved analytically.

```
> Integrate(exp(-c*x^2), x=0..infinity); % = value(%);
```

Definite integration: Can't determine if the integral is convergent  
 Need to know the sign of --> c  
 Will now try indefinite integration and then take limits.

$$\int_0^\infty e^{-cx^2} dx = \lim_{x \rightarrow \infty} \frac{1}{2} \frac{\sqrt{\pi} \operatorname{erf}(\sqrt{c}x)}{\sqrt{c}}$$

A common mistake in thought: *you* may think of  $c$  as a positive real constant, but *Maple* does not start from this assumption! From the above answer it is clear that Maple at least knows the indefinite integral. If you want to get further with the computation of the definite integral you must

assume that  $c$  is a positive real constant; for non-positive values the integral diverges. This can be done with the **assuming** operator.

```
> Integrate(exp(-c*x^2), x=0..infinity):
> % = value(%) assuming c>0;
```

$$\int_0^\infty e^{-cx^2} dx = \frac{\sqrt{\pi}}{2\sqrt{c}}$$

Below is another example of an integral where Maple does not work perfectly because of lack of information.

```
> infolevel[integrate] := 2:
> Integrate(1/sqrt((1-a^2*sin(t)^2)/(1-a*sin(t)^2)),
> t=0..Pi/2): % = value(%);
```

```
int/ellalg/trxstandard/signum: NoName
Tried to determine the sign of --> 2*Re(1/a)
int/indef1: first-stage indefinite integration
int/indef2: second-stage indefinite integration
int/trigon: case of integrand containing trigs
int/indef1: first-stage indefinite integration
int/algebraic2/algebraic: algebraic integration
int/algebraic2/algebraic: algebraic integration
```

$$\int_0^{1/2\pi} \frac{1}{\sqrt{1 - a^2 \sin(t)^2} (1 - a \sin(t)^2)} dt = \text{EllipticPi}(a, \text{csgn}(a) a)$$

We made Maple more communicative by raising the **infolevel[integrate]** to 2. The messages that appear then on the screen inform us about what Maple is doing; one of the first messages is telling

"Tried to determine the sign of --> 2\*Re(1/a)"

This gives us the clue to add the assumption  $a > 0$ , which we took for granted, but *Maple* not.

```
> Integrate(1/sqrt((1-a^2*sin(t)^2)/(1-a*sin(t)^2)),
> t=0..Pi/2): % = value(%) assuming a>0;
```

$$\int_0^{1/2\pi} \frac{1}{\sqrt{1 - a^2 \sin(t)^2} (1 - a \sin(t)^2)} dt = \frac{\text{EllipticPi}\left(a, \frac{1}{a}, \frac{1}{a}\right)}{a}$$

The result is much simpler when we add the condition  $a < 1$ .

```
> Integrate(1/sqrt((1-a^2*sin(t)^2)/(1-a*sin(t)^2)),
> t=0..Pi/2): % = value(%) assuming 0<a, a<1;
```

$$\int_0^{1/2\pi} \frac{1}{\sqrt{1 - a^2 \sin(t)^2} (1 - a \sin(t)^2)} dt = \text{EllipticPi}(a, a)$$

Choosing the right assumptions that make Maple do an integration job as far as possible is tricky business, and you often have to use all your mathematical knowledge or computational skills. In [179] is described how you can redefine the built-in **signum** function so that Maple asks for sign

information during a computation. A drawback of this method is that Maple can ask you information where it is unclear what it has to do with your problem.

Sometimes you must delineate a method, e.g., a change of variables or integration by parts. This is conveniently done through the **Student** package. We shall use the **Calculus1** subpackage.

```
> with(Student[Calculus1] # load the package
> Integrate(sqrt(x+1+sqrt(x+1)), x);
```

$$\int \sqrt{x + 1 + \sqrt{x + 1}} \, dx$$

You can ask for a hint:

```
> Hint(%);
```

$$[change, x + 1 = u^2, u]$$

Let us apply this substitution rule, simplify the intermediate result, compute the transformed integral, and finally do backsubstitution to get the requested result.

```
> Rule[%](%);
```

$$\int \sqrt{x + 1 + \sqrt{x + 1}} \, dx = \int 2 u^{(3/2)} \sqrt{u + 1} \, du$$

```
> combine(%) assuming positive;
```

$$\int \sqrt{x + 1 + \sqrt{x + 1}} \, dx = \int 2 \sqrt{u(u + 1)} \, u \, du$$

```
> lhs(%) = value(rhs(%));
```

$$\begin{aligned} & \int \sqrt{x + 1 + \sqrt{x + 1}} \, dx = \\ & \frac{2(u^2 + u)^{(3/2)}}{3} - \frac{(2u + 1)\sqrt{u^2 + u}}{4} + \frac{1}{8} \ln\left(\frac{1}{2} + u + \sqrt{u^2 + u}\right) \end{aligned}$$

```
> subs(u=sqrt(x+1), %);
```

$$\begin{aligned} & \int \sqrt{x + 1 + \sqrt{x + 1}} \, dx = \frac{2(x + 1 + \sqrt{x + 1})^{(3/2)}}{3} \\ & - \frac{(2\sqrt{x + 1} + 1)\sqrt{x + 1 + \sqrt{x + 1}}}{4} \\ & + \frac{1}{8} \ln\left(\frac{1}{2} + \sqrt{x + 1} + \sqrt{x + 1 + \sqrt{x + 1}}\right) \end{aligned}$$

```
> radnormal(diff(rhs(%), x)); # check the answer
```

$$\sqrt{x + 1 + \sqrt{x + 1}}$$

Another example, where Maple does not give any clue, but where we ourselves are able to apply integration by parts.

```
> Integrate(x*exp(-a^2*x^2)*erf(b*x), x);

$$\int x e^{(-a^2 x^2)} \operatorname{erf}(b x) dx$$

> Hint(%);
[ ]
> # find antiderivative of first part
> integrate(x*exp(-a^2*x^2), x);

$$-\frac{1}{2} \frac{e^{(-a^2 x^2)}}{a^2}$$

> # apply integration by parts
> Rule[parts, erf(b*x), %](%%%);

$$\int x e^{(-a^2 x^2)} \operatorname{erf}(b x) dx = -\frac{1}{2} \frac{\operatorname{erf}(b x) e^{(-a^2 x^2)}}{a^2} - \int -\frac{e^{(-a^2 x^2)} e^{(-b^2 x^2)} b}{a^2 \sqrt{\pi}} dx$$

> lhs(%) = value(rhs(%));

$$\int x e^{(-a^2 x^2)} \operatorname{erf}(b x) dx = -\frac{1}{2} \frac{\operatorname{erf}(b x) e^{(-a^2 x^2)}}{a^2} + \frac{1}{2} \frac{b \operatorname{erf}(\sqrt{a^2 + b^2} x)}{a^2 \sqrt{a^2 + b^2}}$$

```

Sometimes you have to apply the well-known method of contour integration and use Maple as a computational tool. Below, we shall compute in this way

$$\int_{-\infty}^{\infty} \frac{e^{ax}}{1 + e^x} dx, \quad 0 < a < 1.$$

First, we replace the real variable by the complex variable  $z$  and integrate around the contour shown in Figure 10.4.

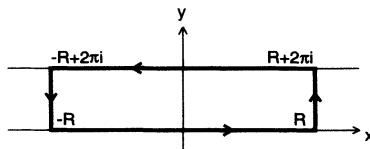


Figure 10.4. Contour of integration.

If we take the limit  $R \rightarrow \infty$ , the horizontal sections of the contour lead to our integral and the vertical sections vanish because of the assumptions on the parameter  $a$ :

$$\oint_C \frac{e^{az}}{1 + e^z} dz = (1 - e^{2\pi a}) \lim_{R \rightarrow \infty} \int_{-\infty}^{\infty} \frac{e^{ax}}{1 + e^x} dx.$$

The answer to the contour integral is  $2\pi i \sum residues \text{ inside strip}$ . So, we search for the roots of the denominator that lie in the strip and compute their residues.

```

> exp(a*z)/(1+exp(z));

$$\frac{e^{(a z)}}{1 + e^z}$$

> solve(denom(%), z); # the solution in the strip

$$\pi I$$

> residue(%%, z=%);

$$-e^{(a \pi I)}$$

> (1-exp(2*Pi*a*I))*integral = 2*Pi*I*%;

$$(1 - e^{(2 I \pi a)}) \text{ integral} = -2 I \pi e^{(a \pi I)}$$

> (lhs/rhs)(%) = 1;

$$\frac{\frac{1}{2} I (1 - e^{(2 I \pi a)}) \text{ integral}}{\pi e^{(a \pi I)}} = 1$$

> simplify(%);

$$\frac{\text{integral} \sin(a \pi)}{\pi} = 1$$

> Integrate(exp(a*x)/(1+exp(x)),
> x=-infinity..infinity) = solve(% , integral);

$$\int_{-\infty}^{\infty} \frac{e^{(a x)}}{1 + e^x} dx = \frac{\pi}{\sin(a \pi)}$$


```

Usual tricks such as differentiation with respect to parameters prior to integration are performed easily in Maple. An example:

```

> Integrate(tanh(a*x/2)/(x*cosh(a*x)) , x=0..infinity);

$$\int_0^{\infty} \frac{\tanh(\frac{1}{2} a x)}{x \cosh(a x)} dx$$

> diff(%, a);

$$\int_0^{\infty} \frac{1 - \tanh(\frac{1}{2} a x)^2}{2 \cosh(a x)} - \frac{\tanh(\frac{1}{2} a x) \sinh(a x)}{\cosh(a x)^2} dx$$

> value(%) assuming a>0;

$$0$$


```

So, the definite integral does not depend on the positive parameter  $a$ .

Last but not least, some good advice: use your brains when using Maple and think twice before computing. An example: Maple consumes a lot of computing time and computer resources to determine

$$\int_{-\pi}^{\pi} \frac{\sin t}{1 + \sin^{40} t} dt.$$

```

> restart; setttime := time():
> integrate(sin(t)/(1+sin(t)^40), t=-Pi..Pi);
0
> cpu_time=(time()-settime)*seconds; # computing time
cpu_time := 322.253 seconds

```

But there is no need to use Maple in this case. You can almost immediately conclude that the integrand is odd and that therefore the integral must be equal to zero.

## 10.6 Summation

Finite sums of numbers can be easily computed with the procedure **add**.

```

> Sum(k^7, k=1..20) = add(k^7, k=1..20);

$$\sum_{k=1}^{20} k^7 = 3877286700$$

> primes := [11,31,41,71,101,131,181,191,211,241]:
> Sum(i, i=primes) = add(i, i=primes);


$$\sum_{i=\%1} i = 1210$$

%1 := [11, 31, 41, 71, 101, 131, 181, 191, 211, 241]
> poly := add(a[i]*x^i, i=0..5);

$$poly := a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5$$


```

“Symbolic summation” is also possible. Let us first consider indefinite summation: given a sequence

$$a_0, a_1, a_2, a_3, \dots$$

or more precisely, an expression generating this sequence, we want to find an expression  $s_k$ , in which no summation sign occurs and such that

$$a_k = s_k - s_{k-1}.$$

This is the discrete analogue of indefinite integration. Maple uses the following methods to find  $s_k$ .

- Polynomials are summed using a formula based on Bernoulli polynomials:

$$\sum_{k=0}^{n-1} k^m = \frac{1}{m+1} \sum_{k=0}^m \binom{m+1}{k} B_k n^{m+1-k},$$

where the Bernoulli numbers are defined by the implicit recurrence relation

$$B_0 = 1 \quad \text{and} \quad \sum_{k=0}^m \binom{m+1}{k} B_k = 0, \quad \text{for all } m \geq 0.$$

- Abramov's method is used for summing rational functions of the summation index. The result is a rational function plus a sum of terms involving the Polygamma function Psi and its derivatives.
- Gosper's decision procedure [106] is the discrete analogue of the Risch algorithm. It deals with the case that  $s_k$  is a hypergeometric term, therefore  $s_k/s_{k-1} \in \mathbb{Q}(k)$ .
- The extended Gosper algorithm [151] deals with the question: given a nonnegative integer  $m$ , find the sequence  $s_k$  for given  $a_k$  satisfying  $a_k = s_k - s_{k-m}$  in the particular case that  $s_k$  is an  $m$ -fold hypergeometric term.

In Maple are also implemented the (extended) Wilf-Zeilberger method for definite summation of hypergeometric terms [235, 236] together with the use of hypergeometric identities. The interested reader is referred to [151, 217] and references therein. The packages **sumtools** and **SumTools** are collections of routines for handling indefinite and definite sums explicitly. In most cases, the procedure **sum** already does the job automatically.

A few examples:

- Method of Bernoulli polynomials

```
> Sum(k^7, k=1..n); % = value(%);
```

$$\sum_{k=1}^n k^7 = \frac{(n+1)^8}{8} - \frac{(n+1)^7}{2} + \frac{7(n+1)^6}{12} - \frac{7(n+1)^4}{24} + \frac{(n+1)^2}{12}$$

```
> factor(%);
```

$$\sum_{k=1}^n k^7 = \frac{n^2 (3n^4 + 6n^3 - n^2 - 4n + 2)(n+1)^2}{24}$$

- Abramov's method

```
> Sum(1/(k^2+k)^3, k=1..infinity); % = value(%);
```

$$\sum_{k=1}^{\infty} \frac{1}{(k^2+k)^3} = 10 - \pi^2$$

```

> Sum(1/(k^5+2*k+1), k=0..infinity) : % = value(%);


$$\sum_{k=0}^{\infty} \frac{1}{k^5 + 2k + 1} = - \left( \sum_{\alpha=\%1} \left( \frac{4096}{11317} + \frac{2560}{11317} \alpha^4 - \frac{1600}{11317} \alpha^3 + \frac{1000}{11317} \alpha^2 - \frac{625}{11317} \alpha \right) \Psi(-\alpha) \right)$$

%1 := RootOf(_Z^5 + 2 _Z + 1)
> evalf(%);

```

$$1.282579632 = 1.282579632 - 0. I$$

Note that the numerical summation leads to a floating-point real number and that the numerical determination of the roots of the fifth degree polynomial leads to a numeric complex number with ignorable imaginary part. In essence, both answers are in numerical agreement.

```

> Sum(1/(k^2-4), k=3..infinity) : % = value(%);


$$\sum_{k=3}^{\infty} \frac{1}{k^2 - 4} = \frac{25}{48}$$

> Sum(1/(3*k+1)/(3*k+2)/(3*k+3)/(3*k+4),
> k=0..infinity) : % = value(%);


$$\sum_{k=0}^{\infty} \frac{1}{(3k+1)(3k+2)(3k+3)(3k+4)} = \frac{1}{6} + \frac{\pi\sqrt{3}}{36} - \frac{1}{4}\ln(3)$$


```

- Gosper's method

```

> # see (SIAM Review, 1994, Problem 94-2)
> Sum((-1)^(k+1)*(4*k+1)*(2*k)!/
> (k!*4^k*(2*k-1)*(k+1)!), k=1..n):
> % = value(%);

```

$$\begin{aligned} \sum_{k=1}^n \frac{(-1)^{(k+1)} (4k+1) (2k)!}{k! 4^k (2k-1) (k+1)!} = \\ - \frac{2(n+2)(-1)^{(n+2)} (2n+2)!}{(n+1)! 4^{(n+1)} (2n+1) (n+2)!} + 1 \end{aligned}$$

- Hypergeometric identities

```

> Sum((-1)^(n-k)*binomial(2*n,k)^2, k=0..2*n):
> % = value(%);

```

$$\sum_{k=0}^{2n} (-1)^{(n-k)} \text{binomial}(2n, k)^2 = \frac{(-1)^n \sqrt{\pi} 2^{(2n)}}{\Gamma(n+1) \Gamma(\frac{1}{2} - n)}$$

The fact that **sum** finds a summation answer does not mean that this is the most simple answer. The following example illustrates this, using the **SumTools[Hypergeometric]** package.

```
> s := binomial(m,k)^2*binomial(n+2*m-k,2*m);
      s := binomial(m, k)^2 binomial(n + 2 m - k, 2 m)
> Sum(s, k=0..m) = sum(s, k=0..m);


$$\sum_{k=0}^m \text{binomial}(m, k)^2 \text{binomial}(n + 2 m - k, 2 m) =$$


$$\frac{\text{binomial}(n + 2 m, 2 m) \Gamma(2 m + 1) \Gamma(n + 1 + m)^2}{\Gamma(m + 1)^2 \Gamma(n + 1) \Gamma(n + 2 m + 1)} -$$


$$\text{binomial}(m, m + 1)^2 \text{binomial}(n + m - 1, 2 m)$$


$$\text{hypergeom}([1, 1, 1, m + 1 - n], [m + 2, m + 2, -n - m + 1], 1)$$

> with(SumTools[Hypergeometric]):
> Sum(s, k=0..m) = DefiniteSum(s, m, k, 0..m);


$$\sum_{k=0}^m \text{binomial}(m, k)^2 \text{binomial}(n + 2 m - k, 2 m) = \frac{\Gamma(n + 1 + m)^2}{\Gamma(m + 1)^2 \Gamma(n + 1)^2}$$

> convert(% , binomial);


$$\sum_{k=0}^m \text{binomial}(m, k)^2 \text{binomial}(n + 2 m - k, 2 m) = \text{binomial}(n + m, m)^2$$

```

For computing a numerical estimate of an infinite sum, Maple uses Levin's *u*-transform [233] to accelerate the convergence of a sum or even to give values to divergent sums. You better always check the quality of the answer.

```
> Sum(1/k^2, k=1..infinity);

$$\sum_{k=1}^{\infty} \frac{1}{k^2}$$

> evalf(%); # the numeric answer
      1.644934067
> value(%); # the exact answer
       $\frac{\pi^2}{6}$ 
> evalf(%-%); # comparison
      0.1 10^{-8}
> Sum(3^(-k), k=1..infinity);
```

```


$$\sum_{k=1}^{\infty} 3^{(-k)}$$

> evalf(%);
0.5000000000
> value(%);

$$\frac{1}{2}$$

> Sum(3^k, k=1..infinity);

$$\sum_{k=1}^{\infty} 3^k$$

> evalf(%);
-1.500000000
> value(%);

$$\infty$$

> Sum(1/k^3, k=1..infinity);

$$\sum_{k=1}^{\infty} \frac{1}{k^3}$$

> evalf(%);
1.202056903
> value(%);

$$\zeta(3)$$

> evalf(%);
1.202056903
> Sum(1/k^(1/3), k=1..infinity);

$$\sum_{k=1}^{\infty} \frac{1}{k^{(1/3)}}$$

> evalf(%);
-0.9733602484
> value(%);

$$\infty$$


```

In the examples

$$\sum_{k=1}^{\infty} 3^k$$

and

$$\sum_{k=1}^{\infty} \frac{1}{\sqrt[3]{k}}$$

you better check whether the numerical answer matches your definition of summation for the divergent series. The answers of Maple may look strange (negative results for positive summands) but are not really weird. For example, the last result can be justified via analytic continuation of the Riemann  $\zeta$ -function (approximate  $\zeta(1/3)$  in Maple to see this).

Another example of a summation of a divergent series, where Maple can use the so-called Cesaro summation to get a definite answer, is the following:

```
> Sum((-1)^k, k=0..infinity);

$$\sum_{k=0}^{\infty} (-1)^k$$

> evalf(%);
0.5000000000
> value(%);

$$undefined$$

> _EnvFormal := true: # enable formal summation
> sum((-1)^k, k=0..infinity);

$$\frac{1}{2}$$

```

## 10.7 Exercises

1. Compute the following indefinite integrals and check the answers by differentiation and simplification.

- (a)  $\int \sqrt{e^x - 1} dx$
- (b)  $\int \frac{x}{(2ax - x^2)^{3/2}} dx$
- (c)  $\int \sqrt{x^2 - a^2} dx$
- (d)  $\int \frac{1}{x\sqrt{1+x^2}} dx$
- (e)  $\int \sec^3 x dx$
- (f)  $\int \frac{1}{1 + \sin x + \cos x} dx$
- (g)  $\int \frac{\ln x}{x(x^2+1)^2} dx$

2. Compute  $\int x^n e^x dx$  for general integer  $n$  and check the result for distinct values of  $n$ .
3. (a) Compute  $\int_0^1 \left( \int_0^1 \frac{x-y}{(x+y)^3} dy \right) dx$ .  
 (b) Compute  $\int_0^1 \left( \int_0^1 \frac{x-y}{(x+y)^3} dx \right) dy$ .  
 (c) Compare the results of (a) and (b). Does Maple make a mistake or is there something else going on?
4. Compute the following definite integrals.
- (a)  $\int_1^{10} \frac{4x^4 + 4x^3 - 2x^2 - 10x + 6}{x^5 + 7x^4 + 16x^3 + 10x^2} dx$   
 (b)  $\int_0^{\pi/2} x^4 \sin x \cos x dx$   
 (c)  $\int_{1/7}^{1/5} \frac{1}{x\sqrt{5x^2 - 6x + 1}} dx$   
 (d)  $\int_{-2}^{-1} \frac{1}{x} dx$
5. Compute the following definite integrals:
- (a)  $\int_0^1 \frac{1}{\sqrt{1-x^2}} dx$   
 (b)  $\int_0^1 x \arctan x dx$   
 (c)  $\int_0^\infty e^{-ax} \cos^2(bx) dx$ , for a positive real number  $a$   
 (d)  $\int_0^\infty \frac{\sin x}{x} dx$   
 (e)  $\int_0^\infty e^{-x} \ln x dx$   
 (f)  $\int_0^\infty \frac{e^{-ax} \ln x}{\sqrt{x}} dx$ , for a positive real number  $a$   
 (g)  $\int_0^\infty \frac{e^{-\sqrt{t}}}{t^{1/4}(1-e^{-\sqrt{t}})} dt$   
 (h)  $\int_1^\infty \frac{1}{\sqrt{x^4-1}} dx$   
 (i)  $\int_0^{\pi/2} \sqrt{\cos x} dx$   
 (j)  $\int_1^3 \frac{1}{\sqrt{x^4+4x^2+3}} dx$   
 (k)  $\int_0^{\pi/2} \sqrt{\tan x} dx$   
 (l)  $\int_0^\infty \frac{\sin^2 ax \sin bx}{x} dx$   
 (m)  $\int_0^\infty \frac{1}{\cosh ax} dx$ , for a positive real number  $a$

6. Let  $F$  be the function defined by  $F(T) := \int_1^T \frac{\exp(-u^2 T)}{u} du$ .
- Define the corresponding Maple function  $F$  and determine a numerical approximation of  $F(2)$ .
  - Compute the derivative  $F'$  (by **D**) and compute  $F'(2)$ .
7. Let  $A$  be the area  $\{(x, y) \in \mathbb{R}^2 | 1/2 \leq xy \leq 2, 1 \leq x \leq 3\}$ . Compute  $\iint_A \frac{\exp(\frac{1}{xy})}{y^2(x+1)^2} dx dy$ .
8. In this exercise, we shall keep track of the Risch algorithm and prove that the integral
- $$\int \frac{\ln^2(x-1)}{x} dx$$
- does not exist in the class of elementary functions. In the first place, Liouville's principle implies the following representation of the integral, when it would exist as an elementary function:
- $$B_3(x) \ln^3(x-1) + B_2(x) \ln^2(x-1) + B_1(x) \ln(x-1) + B_0(x),$$
- where  $B_3(x)$ ,  $B_2(x)$ , and  $B_1(x)$  are rational functions and only  $B_0(x)$  may contain new logarithmic extensions.
- Determine the differential equations satisfied by  $B_0(x), \dots, B_3(x)$ .
  - Show that  $B_3(x)$  is a rational constant.
  - Prove that the differential equation for  $B_2(x)$  cannot be solved in terms of rational functions. This proves that  $\frac{\ln^2(x-1)}{x}$  cannot be integrated in terms of elementary functions.
9. Show by the method of residues that
- $$\int_0^{2\pi} \frac{1}{a^2 \cos^2 \theta + b^2 \sin^2 \theta} d\theta = \frac{2\pi}{ab},$$
- for  $a, b$  real and nonzero, and  $\left| \frac{b-a}{b+a} \right| < 1$ . Compare your answer with Maple's answer when entering the integral.
10. Solve the following integral equation by Laplace transforms.
- $$\sin t = \int_0^t J_0(t-\theta) f(\theta) d\theta$$
11. Solve the integral equation  $f(t) = 1 + \int_0^t (t-\theta) f(\theta) d\theta$ .
12. Compute  $\int_0^\infty \frac{x}{1+x^2} \sin \omega^2 x dx$ .
13. Compute  $\int_0^{\frac{\pi}{4}} \frac{1}{(1+k^2 \sin^3 t)^3 \sqrt{1+k^2 \sin^2 t}} dt$ .
14. Compute  $\int_0^1 \frac{x^a - 1}{\ln x} dx$ , for nonnegative  $a$ .

15. Compute  $\int_0^\infty \frac{\ln x}{(x+a)(x-1)} dx$ , for positive  $a$ .
16. Compute  $\int \frac{\ln(x^2+1)}{x^2+1} dx$ .
17. Compute  $\int \frac{x^2}{\sqrt{x^6+1}} dx$ .
18. Compute  $\int \tan\left(\frac{1}{3} \arctan(x)\right) dx$ .
19. Compute  $\int \arcsin^2(x/a) dx$ .
20. Compute  $\int \frac{1}{\sqrt{a\sqrt{x}+b}} dx$ .
21. Compute the following infinite sums.
- $\sum_{k=0}^{\infty} \frac{2k+3}{(k+1)(k+2)(k+3)}$
  - $\sum_{k=1}^{\infty} \frac{k^2+k-1}{(k+2)!}$
  - $\sum_{k=2}^{\infty} \frac{k}{(k-1)^2(k+1)^2}$
  - $\frac{1}{4} + \sum_{k=1}^{\infty} \frac{3n+2}{n^3(n+1)(n+2)}$
  - $\sum_{k=0}^{\infty} \frac{k^3+7k^2+4k+6}{k^2(k^2+2)(k^2+2k+3)}$
22. Compute  $\sum_{k=0}^n \binom{2n}{2k} (-3)^k$ .
23. Compute  $\sum_{k=0}^n \binom{2k}{k} \left(\frac{1}{2}\right)^{2k}$ .
24. Compute the product of the first 32 prime numbers.
25. Compute the product  $\prod_{k=2}^{\infty} 1 - 1/k^2$ .
26. Compute  $\sum_{k=1}^n (m k - 1)$  and compare the result with Formula 0.122 of Gradshteyn-Ryzhik [109].
27. Compute  $\sum_{k=1}^{n-1} k^2 x^2$  and compare the result with Formula 0.114 of Gradshteyn-Ryzhik [109].

# 11

## Series, Approximation, and Limits

Four topics from calculus will be examined in this chapter: truncated series, formal power series, approximation theory, and limits. From the examples it will be clear that various truncated series expansions such as Taylor series, Laurent series, Puiseux series, and Chebychev series are available in Maple. Padé and Chebychev-Padé are part of Maple's numerical approximation package, which is called **numapprox**. When appropriate, series expansions are used in computing limits [94].

### 11.1 Truncated Series

Taylor series expansions of univariate functions are easily and quickly computed with Maple. For example, in a very short time you can compute the Taylor series of  $\sin(\tan x) - \tan(\sin x)$  about  $x = 0$  up to order twenty-five.

```
> taylor(sin(tan(x))-tan(sin(x)), x=0, 25);  
  
-  $\frac{1}{30} x^7 - \frac{29}{756} x^9 - \frac{1913}{75600} x^{11} - \frac{95}{7392} x^{13} - \frac{311148869}{54486432000} x^{15}$   
-  $\frac{10193207}{4358914560} x^{17} - \frac{1664108363}{1905468364800} x^{19} - \frac{2097555460001}{7602818775552000} x^{21}$   
-  $\frac{374694625074883}{6690480522485760000} x^{23} + O(x^{25})$ 
```

You only have to specify the function, the variable, the expansion point, and the truncation order in the call to the procedure **taylor**. The order

symbol  $O(x^{25})$  indicates that Maple has a special internal data structure for series expansions.

```
> whattype(%);
series

> dismantle(%);
SERIES(22)
  NAME(4): x
  RATIONAL(3): -1/30
    INTNEG(2): -1
    INTPOS(2): 30
  [7]
  RATIONAL(3): -29/756
    INTNEG(2): -29
    INTPOS(2): 756
  [9]
  RATIONAL(3): -1913/75600
    INTNEG(2): -1913
    INTPOS(2): 75600
  [11]
  .....
  [21]
  RATIONAL(3): -374694625074883/6690480522485760000
    INTNEG(5): -374694625074883
    INTPOS(6): 6690480522485760000
  [23]
  FUNCTION(3)
    NAME(4): 0 #[protected]
    EXPSEQ(2)
      INTPOS(2): 1
  [25]
```

The general format of the **series** data structure is shown in Figure 11.1 below.

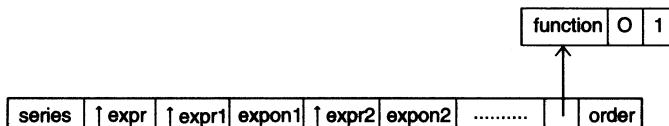


Figure 11.1. Internal representation of a truncated series.

Here, the first expression **expr** has the general form  $x-a$ , where **x** denotes the main variable and **a** denotes the expansion point. The remaining entries are pairs of pointers to coefficients and exponents. The coefficient **O(1)** at the end of the data vector is interpreted by Maple as the “order” term.

The above data type is also the name of a corresponding procedure **series**. This procedure can be used for more general truncated series expansions, e.g., for computing a Laurent series.

```
> series(GAMMA(x), x=0, 2);
```

$$x^{-1} - \gamma + \left( \frac{\pi^2}{12} + \frac{\gamma^2}{2} \right) x + O(x^2)$$

The truncation order to which the Taylor series expansion is computed can be obtained by inspection of the order symbol or by the procedure **order**.

```
> order(%);
2
```

The third argument of the **series** procedure may be left out. Maple then uses the value of the environment variable **Order** to determine the truncation order. Its default value is equal to six.

```
> Order;
6
> Order := 3: series(f(x), x=a);
f(a) + D(f)(a) (x - a) +  $\frac{1}{2}$  (D(2))(f)(a) (x - a)2 + O((x - a)3)
> series(f(x)/(x-a)^2, x=a);
f(a) (x - a)-2 + D(f)(a) (x - a)-1 +  $\frac{1}{2}$  (D(2))(f)(a) + O(x - a)
> series(1/(cos(x)-sec(x)), x=0);
-x-2 + O(x-1)
```

The last example illustrates that the “truncation order of a Laurent series” is only the order used by Maple during the series computation; fewer terms may be present in the final answer. It is like losing precision in numerical arithmetic. The opposite can happen as well in series expansions.

```
> series(1/(1-x^2), x=0, 5);
1 + x2 + x4 + O(x6)
```

Although you only ask for the series expansion up to order 5, Maple decides on its own that the fifth order term is equal to zero, and it informs you about this. In most cases, the reason is that through the **remember option** Maple keeps track of previously computed expansions of larger order. For example, after the command **series(cos(x), x=0, 25)**, the expansion of the cosine function will always be at least up to twenty-five terms.

If you want to restrict yourself to Laurent series, use the procedure **laurent** from the **numapprox** package. Otherwise, Maple may yield more general series such as Puiseux series.

```
> 1/(x*(1+sqrt(x)));
1
-----
x (1 + sqrt(x))
> series(% , x);
```

$$\frac{1}{x} - \frac{1}{\sqrt{x}} + 1 - \sqrt{x} + x - x^{(3/2)} + x^2 - x^{(5/2)} + x^3 - x^{(7/2)} + x^4 \\ - x^{(9/2)} + x^5 - x^{(11/2)} + O(x^6)$$

Sometimes Maple chooses coefficients in a series expansion that depend on the main variable.

```
> series(x^(x^x), x=0, 5);


$$x + \ln(x)^2 x^2 + \left(\frac{1}{2} \ln(x)^3 + \frac{1}{2} \ln(x)^4\right) x^3 \\ + \left(\frac{1}{6} \ln(x)^4 + \frac{1}{2} \ln(x)^5 + \frac{1}{6} \ln(x)^6\right) x^4 + O(x^5)

> series(dilog(x), x=0, 3);


$$\frac{\pi^2}{6} + (\ln(x) - 1) x + \left(\frac{1}{2} \ln(x) - \frac{1}{4}\right) x^2 + O(x^3)$$$$

```

The growth of a coefficient in such *generalized series expansions* [94] must be less than the polynomial in  $x$ .

A nice feature of Maple is that it can compute a truncated series expansion of a function that is only defined by an integral or differential equation. An example:

```
> integrate(ln(1+s*t)^2/(1+t^2), t=0..infinity);


$$\int_0^\infty \frac{\ln(1+s t)^2}{1+t^2} dt

> series(% , s=0, 5);


$$\frac{\pi^2}{3} s + \frac{\pi}{2} s^2 + \left(-\frac{\pi^2}{9} + \pi I + \frac{7}{6} - \ln(s)\right) s^3 - \frac{11\pi}{24} s^4 + O(s^5)$$$$

```

The above series expansion is computed in three steps:

- differentiate the integrand with respect to  $s$  and integrate it,
- compute the series expansion of the intermediate result at  $s = 0$ , and
- integrate the terms of the series expansion and determine the correct integration constant.

Another example: the series expansion of the solution of the differential equation describing a two-dimensional pendulum (see Figure 17.3).

```
> DESol( 1*diff(theta(t),[t$2]) = -g*sin(theta(t)), 
> theta(t), {theta(0)=0, D(theta)(0)=v[0]/l} );

DESol({l (d^2 theta(t))/dt^2 + g sin(theta(t))}, {theta(t)}, {theta(0) = 0, D(theta)(0) = v[0]/l})
```

$$> series(% , t);$$

$$\frac{v_0}{l} t - \frac{1}{6} \frac{g v_0}{l^2} t^3 + \frac{1}{120} \frac{g v_0 (g + \frac{v_0^2}{l})}{l^3} t^5 + O(t^6)$$

The order symbol vanishes in the case of polynomials, at least when the truncation order of the series expansion is high enough.

```
> polynomial := a*x^3+b*x^2+c*x+d;
polynomial := a x3 + b x2 + c x + d
> taylor_series := series(polynomial, x, 4);
taylor_series := d + c x + b x2 + a x3
> series((x+y)^7, x, infinity); # infinite order
y7 + 7 y6 x + 21 y5 x2 + 35 y4 x3 + 35 y3 x4 + 21 y2 x5 + 7 y x6 + x7
```

Although a series expansion looks like a polynomial, certainly in the last examples, the internal data structure is completely different.

```
> whattype(polynomial);
+
> op(polynomial);
a x3, b x2, c x, d
> whattype(taylor_series);
series
> op(taylor_series);
d, 0, c, 1, b, 2, a, 3
> op(0, taylor_series); # the main variable
x
```

Objects of type *series* are dealt with differently in Maple; many operations allowed for polynomials do not work for series expansions.

```
> sin_series := series(sin(x), x=0, 6);
sin_series := x - 1/6 x3 + 1/120 x5 + O(x6)
> subs(x=2, sin_series);

Error, invalid substitution in series
```

On the other hand, evaluation with respect to a side relation does the job.

```
> eval(sin_series, x=2);
14
15
```

But multiplication of Taylor series does not work well.

```
> sin_series * sin_series;
```

```


$$(x - \frac{1}{6}x^3 + \frac{1}{120}x^5 + O(x^6))^2$$

> expand(%);

$$(x - \frac{1}{6}x^3 + \frac{1}{120}x^5 + O(x^6))^2$$

> series(%, x);

$$x^2 - \frac{1}{3}x^4 + O(x^6)$$


```

Some of these difficulties have been removed with the procedure **mtaylor**, fit for multivariate Taylor series expansions. The reason is that the result type of **mtaylor** is not a series but an ordinary polynomial.

```

> F := (3*x*y-2*y^2) * (3*x+y) / (6*x^2-6*y);

$$F := \frac{(3xy - 2y^2)(3x + y)}{6x^2 - 6y}$$

> mtaylor(F, [x=1,y], 5);

$$\frac{3y}{2} + y^2 - \frac{5(x-1)y^2}{2} + \frac{2y^3}{3} + 4y^2(x-1)^2 - \frac{23(x-1)y^3}{6} + \frac{2y^4}{3}$$

> normal(%);

$$\frac{3}{2}y + \frac{15}{2}y^2 - \frac{21}{2}xy^2 + \frac{9}{2}y^3 + 4y^2x^2 - \frac{23}{6}xy^3 + \frac{2}{3}y^4$$

> subs(y=1, %);

$$\frac{85}{6} - \frac{43}{3}x + 4x^2$$


```

Coming back to the series representation of univariate functions, you normalize series expansions with **normal** and you extract coefficients with respect to the main variable with **coeff**.

```

> series(F, x, 4);

$$\frac{y^2}{3} + \frac{y}{2}x - \frac{9y - 2y^2}{6y}x^2 + \frac{1}{2}x^3 + O(x^4)$$

> normal(%);

$$\frac{y^2}{3} + \frac{y}{2}x + (-\frac{3}{2} + \frac{y}{3})x^2 + \frac{1}{2}x^3 + O(x^4)$$

> coeff(%, x^2);

$$-\frac{3}{2} + \frac{y}{3}$$


```

You can only extract coefficients in the main variable.

```

> coeff(%%, y^2);
Error, unable to compute coeff

```

With the procedure **coeftayl**, you can compute a coefficient in a Taylor series expansion without actually computing the series. Instead,

**coeftayl**( $f, x = x_0, k$ )

computes the coefficient of  $(x - x_0)^k$  in the series expansion of  $f$  about  $x = x_0$  as

$$\lim_{x \rightarrow x_0} \frac{\frac{d^k f}{dx^k}(x)}{k!}.$$

```
> coeftayl(F, x=0, 20);
```

$$\frac{1}{3y^8} - \frac{3}{2y^9}$$

**coeftayl** can also be used for multivariate series.

```
> coeftayl(F, [x,y]=[1,0], [1,3]);
```

$$\frac{-23}{6}$$

If you are only interested in the leading term of a univariate series expansion you can explicitly ask for it.

```
> series(1/tan(x), x);
```

$$x^{-1} - \frac{1}{3}x - \frac{1}{45}x^3 + O(x^4)$$

```
> series(leadterm(1/tan(x)), x, infinity);
```

$$x^{-1}$$

You can also differentiate and integrate a series.

```
> sin_series;
```

$$x - \frac{1}{6}x^3 + \frac{1}{120}x^5 + O(x^6)$$

```
> diff(sin_series, x);
```

$$1 - \frac{1}{2}x^2 + \frac{1}{24}x^4 + O(x^5)$$

```
> integrate(sin_series, x);
```

$$\frac{1}{2}x^2 - \frac{1}{24}x^4 + \frac{1}{720}x^6 + O(x^7)$$

Series reversion can be done with the Maple procedure **solve**.

```
> solve(y=sin_series, x);
```

$$y + \frac{1}{6}y^3 + \frac{3}{40}y^5 + O(y^6)$$

Compare with

```
> arcsin_series := series(arcsin(y), y);
```

$$\text{arcsin\_series} := y + \frac{1}{6} y^3 + \frac{3}{40} y^5 + O(y^6)$$

and with

```
> series(RootOf(y=sin(x), x), y);
y +  $\frac{1}{6} y^3 + \frac{3}{40} y^5 + O(y^7)$ 
```

A problem in applied mathematics, which can be solved as a problem of series reversion, is the computation of the series expansion of the solution of Kepler's equation

$$E = u + e \sin E.$$

Here,  $E$  is the moon's eccentric anomaly and we want to express it in terms of the mean anomaly  $u$  and the orbital eccentricity  $e$ , which is regarded as a small quantity (q.v., [11]). We shall also write the coefficients in the expansion as linear combinations of trigonometric functions.

```
> series(E-u-e*sin(E), E=u);
-e sin(u) + (1 - e cos(u)) (E - u) +  $\frac{1}{2} e \sin(u) (E - u)^2$ 
+  $\frac{1}{6} e \cos(u) (E - u)^3 - \frac{1}{24} e \sin(u) (E - u)^4 - \frac{1}{120} e \cos(u) (E - u)^5$ 
+ O((E - u)^6)
> solve(%, E-u);
sin(u) e + cos(u) sin(u) e^2 + (cos(u)^2 sin(u) -  $\frac{1}{2} \sin(u)^3$ ) e^3 +
(cos(u)^3 sin(u) -  $\frac{5}{3} \cos(u) \sin(u)^3$ ) e^4 +
(cos(u)^4 sin(u) -  $\frac{11}{3} \cos(u)^2 \sin(u)^3 + \frac{13}{24} \sin(u)^5$ ) e^5 + O(e^6),
-  $\frac{e}{-1 + e} u - \frac{e}{6 (-1 + e)^4} u^3 - \frac{e (9 e + 1)}{120 (-1 + e)^7} u^5 + O(u^6)$ 
```

We need the first solution and simplify the expansion of  $E$  in  $e$ .

```
> u + map(combine, %[1], 'trig');
u + (sin(u) e +  $\frac{1}{2} \sin(2 u) e^2 + (\frac{3}{8} \sin(3 u) - \frac{1}{8} \sin(u)) e^3$ 
+ ( $\frac{1}{3} \sin(4 u) - \frac{1}{6} \sin(2 u)$ ) e^4
+ ( $\frac{125}{384} \sin(5 u) - \frac{27}{128} \sin(3 u) + \frac{1}{192} \sin(u)$ ) e^5 + O(e^6))
```

In many computations it is convenient to convert series expansions into polynomials. This is a simple matter.

```
> sin_series;

$$x - \frac{1}{6} x^3 + \frac{1}{120} x^5 + O(x^6)$$

> convert(sin_series, polynom);

$$x - \frac{1}{6} x^3 + \frac{1}{120} x^5$$

```

Padé approximations and continued fraction series expansions, i.e., approximations by rational functions, are also available in Maple. You can use the procedure **convert** or the procedures **pade** and **confracform** from the **numapprox** package (if you do not want to compute a series expansion first or if you want a procedure instead of a formula).

```
> convert(sin_series, ratpoly);

$$\frac{-\frac{7}{60} x^3 + x}{1 + \frac{x^2}{20}}$$

```

This is same as the rational expression obtained by the procedure **ratrecon** in the following way.

```
> ratrecon(%%, x^6, x, 3, 2);

$$\frac{-\frac{7}{3} x^3 + 20 x}{20 + x^2}$$

> convert(sin_series, ratpoly, 1, 2);

$$\frac{x}{1 + \frac{x^2}{6}}$$

> ratrecon(x-1/3*x^3, x^4, x, 1, 2);

$$\frac{6x}{6 + x^2}$$

> convert(sin_series, 'confrac');

$$\frac{x}{1 + \frac{x^2}{6 - \frac{7x^2}{10}}}$$

```

These approximations can be used in conjunction with the so-called Chebyshev series expansion.

```
> Digits := 5: # low precision
> numapprox[chebyshev](sin(x), x); # Chebyshev expansion
```

$$0.88010 T(1, x) - 0.039127 T(3, x) + 0.00049952 T(5, x) \\ - 0.30152 \cdot 10^{-5} T(7, x)$$

```
> convert(%, ratpoly);

$$\frac{0.89083 T(1, x) - 0.027887 T(3, x)}{T(0, x) + 0.025529 T(2, x)}$$

> simplify(subs(T=ChebyshevT,%), 'ChebyshevT');

$$\frac{0.97449 x - 0.11155 x^3}{0.97447 + 0.051058 x^2}$$

```

Maple is liberal with respect to conversion to polynomials.

```
> series(sin(x+a)/x^2, x, 4);

$$\sin(a) x^{-2} + \cos(a) x^{-1} - \frac{1}{2} \sin(a) - \frac{1}{6} \cos(a) x + O(x^2)$$

> convert(%, polynom);

$$\frac{\sin(a)}{x^2} + \frac{\cos(a)}{x} - \frac{1}{2} \sin(a) - \frac{1}{6} \cos(a) x$$

> series(sqrt(x*(1-x)), x=0, 3);

$$\sqrt{x} - \frac{x^{(3/2)}}{2} - \frac{x^{(5/2)}}{8} + O(x^{(7/2)})$$

> convert(%, polynom);

$$\sqrt{x} - \frac{x^{(3/2)}}{2} - \frac{x^{(5/2)}}{8}$$

> (a*x^3+b*x^2)/(x^2+1);

$$\frac{a x^3 + b x^2}{x^2 + 1}$$

> series(%, x=infinity, 8); # asymptotic expansion

$$a x + b - \frac{a}{x} - \frac{b}{x^2} + \frac{a}{x^3} + \frac{b}{x^4} - \frac{a}{x^5} - \frac{b}{x^6} + \frac{a}{x^7} + O(\frac{1}{x^8})$$

> convert(%, polynom);

$$a x + b - \frac{a}{x} - \frac{b}{x^2} + \frac{a}{x^3} + \frac{b}{x^4} - \frac{a}{x^5} - \frac{b}{x^6} + \frac{a}{x^7}$$

```

Stirling's formula is an example of an *asymptotic series expansion* of general type.

```
> asympt(ln(x!), x, 4);

$$(\ln(x) - 1) x + \ln(\sqrt{2} \sqrt{\pi}) + \frac{1}{2} \ln(x) + \frac{1}{12 x} + O(\frac{1}{x^3})$$

```

Here, we have used the procedure **asympt**, instead of the **series** command, with **infinity** as expansion point.

```
> simplify(%); # simplification of logarithmic terms

$$(\ln(x) - 1) x + \frac{1}{2} \ln(2) + \frac{1}{2} \ln(\pi) + \frac{1}{2} \ln(x) + \frac{1}{12 x} + O(\frac{1}{x^3})$$

```

In [4] it is stated that one of the difficulties in using a computer algebra system is the fact that a system may not be able to use all the mathematical rules it knows in the right places. This is illustrated with the following example.

$$\lim_{\epsilon \rightarrow 0} \frac{\epsilon}{\sqrt{1+\epsilon} \sin^2 \phi + \sqrt{1-\epsilon} \cos^2 \phi - 1}$$

Earlier versions of Maple and most other systems are not able to use the trigonometric simplification rule  $\sin^2 \phi + \cos^2 \phi = 1$  in the right places and in this way produce incorrect results. The problem lies in most cases in an incorrect series expansion. In the current version, the computation of the above limit already works fine:

```
> expr := epsilon / (sqrt(1+epsilon)*sin(phi)^2 +
> sqrt(1-epsilon)*cos(phi)^2-1);

expr := 
$$\frac{\epsilon}{\sqrt{1+\epsilon} \sin(\phi)^2 + \sqrt{1-\epsilon} \cos(\phi)^2 - 1}$$


> limit(expr, epsilon=0);


$$\frac{2}{\sin(\phi)^2 - \cos(\phi)^2}$$


> combine(%);


$$-\frac{2}{\cos(2\phi)}$$

```

However, the series computation still goes wrong:

```
> series(expr, epsilon, 3);


$$\frac{1}{\sin(\phi)^2 + \cos(\phi)^2 - 1} \epsilon - \frac{\frac{1}{2} \sin(\phi)^2 - \frac{1}{2} \cos(\phi)^2}{(\sin(\phi)^2 + \cos(\phi)^2 - 1)^2} \epsilon^2 + O(\epsilon^3)$$

```

The denominator of the first term simplifies to zero.

Fortunately, you can fine-tune the procedure **series** with respect to the zero recognition of coefficients. All you have to do is to redefine the environment variables **Testzero** and **Normalizer** such that coefficients are handled better in the **series** command.

```
> op(Testzero);

proc(O) evalb(Normalizer(O) = 0) end proc

> op(Normalizer);

proc() option builtin, remember; 220 end proc

> Testzero := proc() evalb(Normalizer(args[1])=0) end proc;
> Normalizer := proc() normal(simplify(args[1])) end proc;
```

After this adaptation of Maple everything works well.

```

> forget(series): # clear the remember table of series
> forget(limit): # clear the remember table of limit
> series(expr, epsilon, 3);


$$\frac{1}{\frac{1}{2} \sin(\phi)^2 - \frac{1}{2} \cos(\phi)^2} + \frac{2 \left(-\frac{1}{8} \sin(\phi)^2 - \frac{1}{8} \cos(\phi)^2\right)}{(-1 + 2 \cos(\phi)^2) \left(\frac{1}{2} \sin(\phi)^2 - \frac{1}{2} \cos(\phi)^2\right)} \varepsilon + O(\varepsilon^2)$$

> map(combine, %, 'trig');


$$-\frac{2}{\cos(2\phi)} + \frac{1}{\cos(4\phi) + 1} \varepsilon + O(\varepsilon^2)$$


```

## 11.2 Approximation of Functions

To evaluate most mathematical functions, you must first produce easily computable approximations to them. In general, it is more efficient with respect to space and time to have an analytical approximation rather than to store a table and use interpolation. There are various forms of approximation functions: Taylor polynomials, Padé approximates, Chebyshev series, and so on. The `numapprox` package contains various procedures for developing numerical approximations. In this section we shall have a look at them when trying to find good approximations of the exponential map on the interval (-2,2). The interested reader is referred to [99] for another example.

Our first approximation is a Taylor polynomial of third degree around zero.

```

> with(numapprox): # load the numapprox package
> interface(displayprecision=4): # set display precision
> taylor(exp(x), x);


$$1 + x + \frac{1}{2} x^2 + \frac{1}{6} x^3 + \frac{1}{24} x^4 + \frac{1}{120} x^5 + O(x^6)$$

> P[5] := convert(%, polynom);


$$P_5 := 1 + x + \frac{1}{2} x^2 + \frac{1}{6} x^3 + \frac{1}{24} x^4 + \frac{1}{120} x^5$$


```

The polynomial can be converted to Horner (nested multiplication) form.

```

> hornerform(P[5], x);


$$1 + (1 + (\frac{1}{2} + (\frac{1}{6} + (\frac{1}{24} + \frac{1}{120} x) x) x) x) x$$

> taylorapprox := unapply(%, x);


$$taylorapprox := x \rightarrow 1 + (1 + (\frac{1}{2} + (\frac{1}{6} + (\frac{1}{24} + \frac{1}{120} x) x) x) x) x$$


```

The Taylor remainder theorem gives an estimation of the error:

$$e^x - P_5(x) = \frac{1}{720} e^\xi x^6,$$

where  $\xi$  is between 0 and  $x$ . To examine the error carefully on an interval, you can use the procedures **maximize** and **minimize**. Alternatively, you can use the procedure **infnorm** from the **numapprox** package to compute the error. This procedure computes an estimate of the infinity norm of a continuous real function on the segment  $[a, b]$ , which is defined as

$$\|f\|_\infty = \max_{a \leq x \leq b} |f(x)|.$$

In practice, this procedure works better than the procedures **maximize** and **minimize**.

```
> infnorm(exp-taylorapprox, -2..2);
```

```
0.1224
```

So, the maximum error is approximately 0.122. The error is not distributed evenly through the interval (-2,2) as can be seen in Figure 11.2.

```
> plot(exp-taylorapprox, -2..2);
```

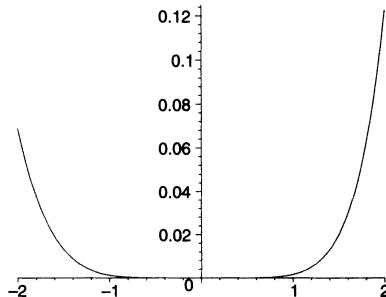


Figure 11.2. Error curve of Taylor approximation of exponential map about 0 of 5th degree.

Next, the Padé rational approximation of degree (3,2) is taken under consideration.

```
> R[3,2] := pade(exp(x), x, [3,2]);
```

$$R_{3,2} := \frac{1 + \frac{3}{5}x + \frac{3}{20}x^2 + \frac{1}{60}x^3}{1 - \frac{2}{5}x + \frac{1}{20}x^2}$$

To minimize multiplications you can convert to continued fraction form.

```
> confracform(%, x);
```

$$\frac{x}{3} + \frac{17}{3} + \frac{152}{3 \left( x - \frac{117}{19} + \frac{3125}{361(x - \frac{35}{19})} \right)}$$

```
> padeapprox := unapply(% , x);
padeapprox := x →  $\frac{1}{3}x + \frac{17}{3} + \frac{152}{3} \frac{1}{x - \frac{117}{19} + \frac{3125}{361} \frac{1}{x - \frac{35}{19}}}$ 
```

The error is smaller than in the Taylor approximation.

```
> infnorm(exp-padeapprox, -2..2);
0.0557
```

So, the maximum error is approximately 0.056. The distribution of the error over the interval (-2,2) is shown in Figure 11.3.

```
> plot(exp-padeapprox, -2..2);
```

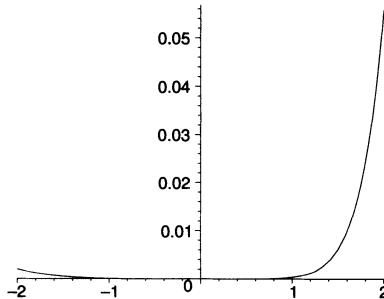


Figure 11.3. Error curve of Padé approximation of exponential map about 0 of degree (3,2).

Instead of expansion in monomials  $x^n$  on  $(-2,2)$ ,  $n \geq 0$ , you can use orthogonal polynomials to obtain better approximations. In the **numapprox** package you can work with Chebyshev polynomials via **chebyshev** and **chebpade**. Utility functions are: **chebmultiplication**, **chebsort**, and **chebdeg**.

Figures 11.4 and 11.5 show the error curves of the Chebyshev and Chebyshev-Padé approximation of the exponential map, respectively.

```
> chebP[5] := chebyshev(exp(x), x, 0.001);
chebP5 := 1.2661 T(0, x) + 1.1303 T(1, x) + 0.2715 T(2, x)
+ 0.0443 T(3, x) + 0.0055 T(4, x) + 0.0005 T(5, x)
> chebP[5] := simplify(subs(T=ChebyshevT, %));
chebP5 := 1.0000 + 1.0000 x + 0.4992 x2 + 0.1665 x3 + 0.0438 x4 + 0.0087 x5
```

```

> hornerform(%, x);
1.0000 + (1.0000 + (0.4992 + (0.1665 + (0.0438 + 0.0087 x) x) x) x) x
> chebapprox := unapply(%, x):
> infnorm(exp-chebapprox, -2..2);
0.0816
> plot(exp-chebapprox, -2..2);

```

Figure 11.4. Error curve of Chebyshev approximation of exponential map about 0 of 5th degree.

```

> chebR[3,2] := chebpade(exp(x), x=-2..2, [3,2]);
chebR3,2 :=

$$\frac{1.2099 T(0, \frac{x}{2}) + 1.2269 T(1, \frac{x}{2}) + 0.2909 T(2, \frac{x}{2}) + 0.0320 T(3, \frac{x}{2})}{T(0, \frac{x}{2}) - 0.7077 T(1, \frac{x}{2}) + 0.0812 T(2, \frac{x}{2})}$$

> chebR[3,2] := simplify(subs(T=ChebyshevT, %),
> 'ChebyshevT');
chebR3,2 := 
$$\frac{0.9190 + 0.5655 x + 0.1454 x^2 + 0.0160 x^3}{0.9188 - 0.3538 x + 0.0406 x^2}$$

> chebpadeapprox := unapply(%, x):
> infnorm(exp-chebpadeapprox, -2..2);
0.0007
> plot(exp-chebpadeapprox, -2..2);

```

The best uniform approximation to the exponential mapping found above has an error curve that oscillates in sign. It can be improved by “leveling” the error curve. This can be done by the procedure **minimax**, which uses the Remez algorithm [201, 202] to find the best minimax rational approximation of certain degree with respect to a given weight function. Specifically, given a continuous real function  $f$  and a positive weight function  $w$  on a segment  $[a, b]$ , **minimax** computes the rational function  $p/q$

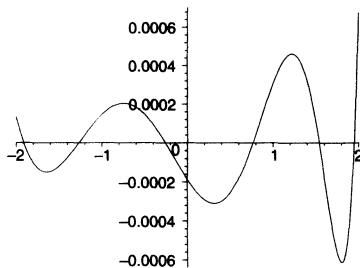


Figure 11.5. Error curve of Chebyshev-Padé approximation of exponential map about 0 of 5th degree.

such that

$$\max_{a \leq x \leq b} w(x) |f(x) - p(x)/q(x)|$$

is minimized over all rational functions  $p/q$  with numerator  $p$  of degree  $\leq m$  and denominator  $q$  of degree  $\leq n$ .

Let us first look at the minimax approximation of the exponential map to a polynomial of fifth degree.

```
> minimaxP[5] := minimax(exp, -2..2, [5,0], 1,
>   'maxerror');
```

$$\begin{aligned} \text{minimaxP}_5 := x &\rightarrow 1.0031 \\ &+ (1.0027 + (0.4860 + (0.1625 + (0.0507 + 0.0101 x) x) x) x) x \end{aligned}$$

The maximum error in the minimax approximation to a 5th degree polynomial is recorded during the Remez algorithm.

```
> maxerror;
0.0033
```

The error curve is shown in Figure 11.6. The best rational approximation of degree (3,2) and its error curve is computed below.

```
> plot(exp-minimaxP[5], -2..2);
```

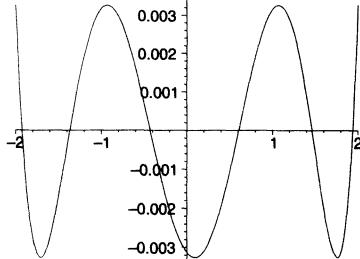


Figure 11.6. Error curve of minimax approximation of exponential map to 5th degree polynomial.

Next, we find the minimax approximation of the exponential map to a rational function of degree (3,2).

```
> minimaxR[3,2] := minimax(exp, -2..2, [3,2], 1,
>   'maxerror');
minimaxR3,2 := x →
0.9223619766 + (0.5751247314 + (0.1517495967 + 0.01744849686 x) x) x
0.9223699871 + (-0.3480389344 + 0.03881500647 x) x
> maxerror;
0.0003
> plot(exp-minimaxR[3,2], -2..2);


```

Figure 11.7. Error curve of minimax approximation of exponential map to a rational function of degree (3,2).

We end this section with the minimax approximation of the exponential map to a rational function of degree (3,2) with respect to the relative error.

```
> minimaxapprox32 := minimax(exp, -2..2, [3,2],
>   x → 1/exp(x), 'maxerror');
minimaxapprox32 := x →
0.9133842467 + (0.5504989591 + (0.1363414286 + 0.01407497728 x) x) x
0.9131465824 + (-0.3625452296 + 0.04342670878 x) x
> maxerror;
0.0003
```

The error curve of this approximation is shown in Figure 11.8, together with the error curve of the previous minimax approximation with respect to absolute error.

```
> plot({1-minimaxapprox32/exp,
>   exp-minimaxapprox32}, -2..2);
```

## 11.3 Power Series

The **powseries** package provides facilities for manipulation of formal power series. As with all Maple library packages it must be loaded first with the command **with**.

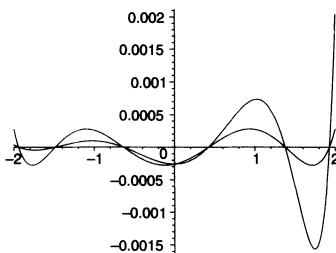


Figure 11.8. Error curve of minimax approximations of exponential map to rational functions of degree (3,2).

```
> with(powseries);
[compose, evalpow, inverse, multconst, multiply, negative, powadd, powcos,
powcreate, powdiff, powexp, powint, powlog, powpoly, powsin,
powsolve, powsqrt, quotient, reversion, subtract, template, tpsform]
```

You learn best about this package by examples. Henceforth we shall consider the power series

$$\exp(ax) = \sum_{n=0}^{\infty} \frac{a^n}{n!} x^n \quad \text{and} \quad \ln(1+x) = \sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n} x^n$$

First, we define the two power series with the procedure **powcreate**.

```
> powcreate(f(n)=a^n/n!):
> powcreate(g(n)=(-1)^(n+1)/n, g(0)=0):
```

We have now specified the rules to compute coefficients in the power series. Let us ask for the first five terms with **tpsform** (truncated power series form).

```
> f_series := tpsform(f, x, 5);
f_series := 1 + a x + \frac{a^2}{2} x^2 + \frac{a^3}{6} x^3 + \frac{a^4}{24} x^4 + O(x^5)
> g_series := tpsform(g, x, 5);
g_series := x - \frac{1}{2} x^2 + \frac{1}{3} x^3 - \frac{1}{4} x^4 + O(x^5)
```

Let us do some manipulations with these power series: addition, multiplication, inversion with respect to multiplication, and composition, respectively.

```
> s := powadd(f, g): tpsform(s, x, 3);
1 + (a + 1) x + (\frac{a^2}{2} - \frac{1}{2}) x^2 + O(x^3)
> p := multiply(f, g): tpsform(p, x, 4);
```

```


$$x + \left(-\frac{1}{2} + a\right)x^2 + \left(\frac{1}{3} - \frac{1}{2}a + \frac{1}{2}a^2\right)x^3 + O(x^4)$$

> i := inverse(f): tpsform(i, x, 5);


$$1 - ax + \frac{a^2}{2}x^2 - \frac{a^3}{6}x^3 + \frac{a^4}{24}x^4 + O(x^5)$$

> p := multiply(i, f): tpsform(p, x, 10);


$$1 + O(x^{10})$$

> c := compose(f, g): tpsform(c, x, 4);


$$1 + ax + \left(-\frac{1}{2}a + \frac{1}{2}a^2\right)x^2 + \left(\frac{1}{3}a - \frac{1}{2}a^2 + \frac{1}{6}a^3\right)x^3 + O(x^4)$$

> a := 1: # special case
> c := compose(f, g): eval(tpsform(c, x, 10));


$$1 + x + O(x^{10})$$


```

The last result could have been obtained much more efficiently by

```

> c := powexp(g): tpsform(c, x, 10);


$$1 + x + O(x^{10})$$


```

You can also apply the logarithmic function to a power series.

```

> a := 'a': # reset a to a free variable
> r := powlog(f): tpsform(r, x, 10);


$$ax + O(x^{10})$$


```

Other computations are differentiation and integration.

```

> d := powdiff(f): tpsform(d, x, 4);


$$a + a^2x + \frac{a^3}{2}x^2 + \frac{a^4}{6}x^3 + O(x^4)$$

> i := powint(f): tpsform(i, x, 4);


$$x + \frac{a}{2}x^2 + \frac{a^2}{6}x^3 + O(x^4)$$


```

Reversion of a power series with respect to composition is also easily done in Maple. We shall use the power series expansion of the function  $\ln(1+x)$  as our example; the reverse power series should be of the series expansion of  $\exp(x) - 1$ .

```

> r := reversion(g): tpsform(r, x, 5);


$$x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + O(x^5)$$

> c := compose(g, r): tpsform(c, x, 5);


$$x + O(x^5)$$

> c := compose(r, g): tpsform(c, x, 5);

```

$$x + O(x^5)$$

We end this section with the popular example of the computation of the “ $f$  and  $g$  series,” which are used in certain expansions of elliptic motion in celestial mechanics (q.v., [82]). The coefficients of the series expansions are defined by the following recurrences.

$$\begin{aligned} f_n &= -\mu g_{n-1} - \sigma(\mu + 2\epsilon) \frac{\partial f_{n-1}}{\partial \epsilon} + (\epsilon - 2\sigma^2) \frac{\partial f_{n-1}}{\partial \sigma} - 3\mu\sigma \frac{\partial f_{n-1}}{\partial \mu}, \quad f_0 = 1, \\ g_n &= f_{n-1} - \sigma(\mu + 2\epsilon) \frac{\partial g_{n-1}}{\partial \epsilon} + (\epsilon - 2\sigma^2) \frac{\partial g_{n-1}}{\partial \sigma} - 3\mu\sigma \frac{\partial g_{n-1}}{\partial \mu}, \quad g_0 = 0. \end{aligned}$$

First we define the coordinate functions.

```
> mu := (mu,sigma,epsilon) -> mu:
> sigma := (mu,sigma,epsilon) -> sigma:
> epsilon := (mu,sigma,epsilon) -> epsilon:
```

Using the **D** operator, we can define the recurrence relations.

```
> with( powseries ):
> powcreate( f(n) = -mu*g(n-1) - sigma*(mu+2*epsilon)
>           *D[3](f(n-1)) + (epsilon-2*sigma^2)*D[2](f(n-1))
>           - 3*mu*sigma*D[1](f(n-1)), f(0) = 1 ):
> powcreate( g(n) = f(n-1) - sigma*(mu+2*epsilon)
>           *D[3](g(n-1)) + (epsilon-2*sigma^2)*D[2](g(n-1))
>           - 3*mu*sigma*D[1](g(n-1)), g(0) = 0 ):
```

We compute the first terms of the series expansion.

```
> tpsform(f, T, 5);

$$1 - \mu T^2 + 3 \mu \sigma T^3 + (\mu^2 + 3(\epsilon - 2\sigma^2)\mu - 9\mu\sigma^2)T^4 + O(T^5)$$

```

There is a drawback. You would probably prefer to expand intermediate results, but when you define the recurrence relations in the **powcreate** command, there is no easy way to specify this. After the computation of the series you can simplify it.

```
> map(factor, %);

$$1 - \mu T^2 + 3 \mu \sigma T^3 - \mu(-\mu - 3\epsilon + 15\sigma^2)T^4 + O(T^5)$$

```

The only trick to expanding intermediate results is based on the implementation details of the **powseries** package: before computing any truncated series expansion with **tpsform** add the following two statements.

```
> f(_k) := 'expand'(f(_k)): g(_k) := 'expand'(g(_k)):
```

Then the intermediate results are indeed expanded.

## 11.4 Limits

Internally, Maple often uses generalized series expansion for the computation of limits [94, 118]. For example, from the asymptotic series expansion

```
> ln(x)-ln(x+exp(-x));

$$\ln(x) - \ln(x + e^{-x})$$

> series(% , x=infinity, 2);

$$-\frac{1}{x e^x} + \frac{O(\frac{1}{x^2})}{(e^x)^2}$$

```

it is clear that

```
> Limit(%, x=infinity): % = value(%);

$$\lim_{x \rightarrow \infty} \ln(x) - \ln(x + e^{-x}) = 0$$

```

Here, we have used the procedure **value** to evaluate the call to the inert procedure **Limit**. What you see is that you don't have to worry much about details but only have to use the procedure **limit** or **Limit**.

You may specify some options to **limit** and **Limit**: **left**, **right**, **real**, and **complex**. An example:

```
> Limit(cos(x)^(1/x^3), x=0): % = value(%);

$$\lim_{x \rightarrow 0} \cos(x)^{\left(\frac{1}{x^3}\right)} = \text{undefined}$$

> Limit(cos(x)^(1/x^3), x=0, 'right'): % = value(%);

$$\lim_{x \rightarrow 0^+} \cos(x)^{\left(\frac{1}{x^3}\right)} = 0$$

> Limit(cos(x)^(1/x^3), x=0, 'left'): % = value(%);

$$\lim_{x \rightarrow 0^-} \cos(x)^{\left(\frac{1}{x^3}\right)} = \infty$$

```

If the direction is not specified in **limit**, the limit is the real bidirectional limit (except in the case where the limit point is  $+\infty$  or  $-\infty$ , in which case the limit is from the left or the right, respectively).

In some cases Maple needs more information. For example:

```
> y := exp(-a*x)*cos(b*x);

$$y := e^{(-a x)} \cos(b x)$$

> limit(y, x=infinity);

$$\lim_{x \rightarrow \infty} e^{(-a x)} \cos(b x)$$

```

Of course Maple did not presume that you had in mind a positive value of  $a$ . But you could also have added the assumption  $a > 0$  to the computation of the limit.

```
> limit(y, x=infinity) assuming a>0;
```

0

In the first section on truncated series expansion you have seen an example of a series expansion in which you had to adapt the zero test function to get correct results. Here, we give another example of the same kind.

```
> expr := x/(sinh(phi)^2-(1+x)*cosh(phi)^2+1);
```

$$\text{expr} := \frac{x}{\sinh(\phi)^2 - (1 + x) \cosh(\phi)^2 + 1}$$

```
> limit(% , x=0);
```

0

In fact, the expression does not depend on  $x$  and it is not equal to 0.

```
> simplify(expr);
```

$$-\frac{1}{\cosh(\phi)^2}$$

The reason for the wrong result lies in the incorrect truncated series expansion about zero.

```
> series(expr, x=0, 3);
```

$$\frac{1}{\sinh(\phi)^2 - \cosh(\phi)^2 + 1} x + \frac{\cosh(\phi)^2}{(\sinh(\phi)^2 - \cosh(\phi)^2 + 1)^2} x^2 + O(x^3)$$

```
> series(leadterm(expr), x);
```

$$\frac{1}{\sinh(\phi)^2 - \cosh(\phi)^2 + 1} x$$

The denominator of the first term simplifies to zero, but this is not automatically recognized by Maple. Actually, Maple uses two procedures for determining the leading term of a series, viz., **Testzero** and **Normalizer**. The environment variable **Testzero** determines whether a leading coefficient is zero for division. The environment variable **Normalizer** is used in **series** to normalize leading terms of divisors. All you have to do to obtain correct results is to redefine these procedures such that coefficients are handled better in the **series** command.

```
> Testzero := proc() evalb(Normalizer(args[1])=0) end proc;
> Normalizer := proc() normal(simplify(args[1])) end proc:
```

After this adaptation of Maple everything works well.

```

> forget(limit); # clear the remember table of limit
> forget(series); # clear the remember table of series
> limit(expr, x=0);

$$-\frac{1}{\cosh(\phi)^2}$$

> series(expr, x=0, 3);

$$-\frac{1}{\cosh(\phi)^2}$$


```

## 11.5 Exercises

1. Study various numerical approximations of the sine function on the segment  $[-\pi, \pi]$ .

2. Study various numerical approximations of the function

$$x \mapsto \frac{1}{x} \int_0^x \frac{\sin(\sin u)}{u} du$$

for  $|x| < \pi$ .

3. Compute the following limits and check the answers.

- $\lim_{x \rightarrow 0} \frac{\sin x}{x}$
- $\lim_{x \rightarrow 0} (\sin x)^{1/x}$
- $\lim_{x \rightarrow 0} \frac{1 - \cos x}{x}$
- $\lim_{x \rightarrow \infty} (1 + \frac{\pi}{x})^x$
- $\lim_{x \rightarrow 0} x^{\sin x}$
- $\lim_{x \rightarrow \infty} (2^x + 3^x)^{1/x}$

4. Compute the following limits.

- $\lim_{x \rightarrow \infty} \frac{\ln x}{x}$
- $\lim_{x \rightarrow \infty} \frac{\ln x}{e^x}$
- $\lim_{x \rightarrow \infty} \frac{x^2 + \sin x}{2x^2 + \cos 4x}$
- $\lim_{x \downarrow 0} \frac{2}{1 + e^{-1/x}}$
- $\lim_{x \rightarrow \infty} \sinh(\tanh x) - \tanh(\sinh x)$

5. In a fresh Maple session or after the **restart** command, compute the Taylor series expansion of  $x^3 - 1 - 4x^2 + 5x$  about  $x = 1$  up to order 10. We do want you to enter the terms in the polynomial in the above ordering. Of what data type is the obtained expression? To what order does Maple think that the series expansion goes?

6. What is the asymptotic expansion of  $\binom{2n}{n}$ ?
7. What is the power series expansion of Lambert's W function?
8. In this exercise the Chebyshev polynomials of the first kind are introduced via their generating function.
  - (a) Compute the Taylor series expansion of  $\frac{1-t^2}{1-2xt+t^2}$  about  $x = 0$  and  $t = 0$  up to degree 8.
  - (b) The Chebyshev polynomials  $T_n(x)$  of the first kind are defined by the generating function

$$\frac{1-t^2}{1-2xt+t^2} = \sum_{n=0}^{\infty} \epsilon_n T_n(x) t^n,$$

where  $\epsilon_0 = 1$  and  $\epsilon_n = 2$  for  $n \geq 1$ . Compute  $T_2(x)$  and  $T_{10}(x)$ . Check your answer with the built-in command **ChebyshevT**.

9. Find the Taylor series expansion up to order 25 of the solution for Kepler's equation  $E = u + e \sin E$  by use of the **RootOf** procedure. Compare the efficiency with the method described in the first section of this chapter.
10. Find the series expansion  $x = a$  up to order 3 of the function
 
$$x \mapsto \frac{x^a - a^x}{x^x - a^a}.$$
11. When you study Josephson's junction circuit you may need the series expansion of the function ptan defined as  $\text{ptan}(s) = p$ , if  $p - \tan p = s$ . Compute the series expansion of this function.
12. Compare the series solution of Kepler's equation  $E = u + e \sin E$ , which was computed in the first section of this chapter, with the following exact solution in terms of Bessel functions:  $E = u + 2 \sum_{n=1}^{\infty} J_n(n e) \sin(n u)/n$ .

# 12

## Composite Data Types

Active knowledge of Maple data types is quite often needed even when using Maple interactively as a symbolic calculator. Recall the elementary data types *polynom* and *ratpoly*, and the simplification and manipulation of such expressions. This chapter will introduce you to *composite data types* like *sequence*, *set*, *list*, *Array*, *(r)table*, *Record*, and *function call*, which are built from elementary data types and used for grouping objects together.

### 12.1 Sequence

You have seen objects of type **exprseq** (**expression sequence**) before. For example, the result of the procedure **op** is in most cases a sequence of objects separated by commas.

```
> polynomial := x^3 - 6*x^2 + 11*x - 6;  
polynomial :=  $x^3 - 6x^2 + 11x - 6$   
> sequence := op(polynomial);  
sequence :=  $x^3, -6x^2, 11x, -6$ 
```

Sequences are frequently used in Maple. Below, not only the result of the command **solve(polynomial, x)** is a sequence, but also the arguments of the function call form a sequence.

```
> arguments := polynomial, x;  
arguments :=  $x^3 - 6x^2 + 11x - 6, x$ 
```

```
> whattype(arguments);
                                         exprseq
> solve(arguments);
                                         1, 2, 3
```

It is possible to assign each solution in one command.

```
> x1, x2, x3 := %:
> x1, x2, x3;
                                         1, 2, 3
```

Inside a procedure definition you can refer to the sequence of actual arguments via the variable `args`; the actual number of arguments is available via the variable `nargs`. The following example illustrates this.

```
> procargs := proc()
>   print( 'number of arguments' = nargs,
>          'actual arguments' = args )
> end proc:
> procargs(x);

                                         number of arguments = 1, actual arguments = (x)

> procargs(x, y, z);

                                         number of arguments = 3, actual arguments = (x, y, z)
```

Note that a sequence is one object: internally it is represented by one data vector, viz.,

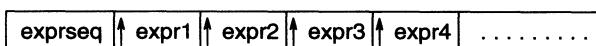


Figure 12.1. Internal data structure of a sequence.

Here, the symbols `↑expr1`, `↑expr2`, ... indicate pointers to the data vectors that correspond to the expressions `expr1`, `expr2`, and so on. The components of an expression sequence are not necessarily of the same type.

```
> seq1 := M, a, p, l, e:
> seq2 := 77, 97, 112, 108, 101: # name in ASCII code
> 'concatenated sequence' := seq1, seq2;

                                         concatenated sequence := M, a, p, l, e, 77, 97, 112, 108, 101
```

Maple uses the special name `NULL` to denote the empty sequence.

```
> 'empty sequence' := NULL;

                                         empty sequence :=
> 1, 2, 'empty sequence', 2, 1;
                                         1, 2, 2, 1
```

Sequences can also be generated by the procedure `seq` (`sequence`).

```
> # generate the first nine odd prime numbers
> seq(ithprime(i), i=2..10);
3, 5, 7, 11, 13, 17, 19, 23, 29
```

A function call of the form `seq( f(i), i = m...n )` generates the sequence  $f(m)$ ,  $f(m+1)$ ,  $f(m+2)$ , ...,  $f(n)$ . Alternatively, Maple provides the sequence operator `$`.

```
> x$4;
x, x, x, x
```

Except for the above use in combination with a function call of `diff`, the sequence operator `$` is of limited use. It may even be dangerous to use.

```
> ('[k,1] $ k=1..2') $ l=3..4;
[1, 3], [2, 3], [1, 4], [2, 4]
```

In this example, the quotes are absolutely necessary to prevent premature evaluation, and more quotes would be needed if the sequence indices `k` and `l` had been assigned values before. In the worst case, you would have had to enter a horrible command like

```
> ('['k', 'l'] $ 'k'=1..2') $ 'l'=3..4;
```

Instead, you would certainly prefer the following command.

```
> seq(seq([i,j], i=1..2), j=3..4);
```

The call `seq(f(i), i = expression)` generates a sequence by applying  $f$  to each operand of the *expression*.

```
> seq(i^2, i={1,2,3,4,5});
1, 4, 9, 16, 25
> seq(i^2, i=x+y+z);
x2, y2, z2
```

You select an element of a sequence by the selection operator `[ ]`.

```
> %[2]; # 2nd element
y2
> %%[-1]; # last element
z2
```

You can also select more than one element of a sequence at the same time.

```
> sequence := v, w, x, y, z: sequence[2..4];
w, x, y
```

You may consider the last command as an abbreviation of

```
> seq(sequence[i], i=2..4);
w, x, y
```

Such abbreviations also occur at other places, e.g., when you want to generate a sequence of names  $p_1, p_2, \dots$  by concatenation via

```
> p||(1..5);
                p1, p2, p3, p4, p5
> seq(p||i, i=1..5);
                p1, p2, p3, p4, p5
```

We end this section with a short remark on the subexpression of the form

*left\_expression .. right\_expression*

It is a regular Maple expression of type *range*. You have already seen such expressions being used as range specifications for integration and summation. You may choose any Maple expression as *left\_expression* and *right\_expression*, but in most cases it should evaluate to a numerical value. By the way, you may use more than two dots as the range operator; but remember that *one dot* is used as the matrix-vector multiplication operator.

## 12.2 Set

You use objects of type *set* most frequently when you try to solve systems of equations with the procedure **solve**. The arguments of this procedure are a set of equations and a set of unknowns, and the solution is in general a sequence of objects of type *set*.

Maple uses the common mathematical notation of a set: a sequence between braces. Data do not occur more than once in a set. As a user you have almost no control on the ordering of the elements of a set, because the system uses internal address ordering.

```
> {1, 3, five, 2, 4};
                {1, 2, 3, 4, five}
> {x, x, x*y, y, x*(x-1), y*x, x^2-x};
                {x y, x (x - 1), x^2 - x, x, y}
> 'empty set' := {};
                empty set := {}
```

Figure 12.2 shows the internal representation of a set.



Figure 12.2. Internal data structure of a set.

The entries in the expression sequence are sorted in increasing address order. In this way, it can be assured that no duplicates are present.

For objects of type *set* the usual operators are present: **union**, **minus**, and **intersect**.

```
> ''{0,1,2,3} union {0,2,4,6}'': % = eval(%);
      {0, 1, 2, 3} ∪ {0, 2, 4, 6} = {0, 1, 2, 3, 4, 6}
> ''{0,1,2,3} minus {0,2,4,6}'': % = eval(%);
      {0, 1, 2, 3} \ {0, 2, 4, 6} = {1, 3}
> ''{0,1,2,3} intersect {0,2,4,6}'': % = eval(%);
      {0, 1, 2, 3} ∩ {0, 2, 4, 6} = {0, 2}
```

You can ask whether a certain Maple object is an element of a set, and if so, find its position in the set.

```
> member(2, {0,1,2,3}, 'position');
      true
> position;
      3
```

You can select an element of a set by the selection operator [] or by the procedure **op**. The latter selection scheme is less efficient as it will evaluate the whole set.

```
> # generate all subsets of {1,2,3}
> collection := combinat[powerset](3);
      collection := {{}, {1, 3}, {1, 2, 3}, {2, 3}, {3}, {1}, {2}, {1, 2}}
> nops(collection); # number of elements in the set
      8
> collection[4]; # 4th element in set
      {2, 3}
> collection[-1]; # last element in set
      {1, 2}
> op(8, collection);
      {1, 2}
> collection[6..8]; # returns a subset
      {{1}, {2}, {1, 2}}
> collection[3..-3]; # omit first and last 2 elements
      {{1, 2, 3}, {2, 3}, {3}, {1}}
> collection[]; # sequence of set elements
```

$$\{\}, \{1, 3\}, \{1, 2, 3\}, \{2, 3\}, \{3\}, \{1\}, \{2\}, \{1, 2\}$$

If you want to select elements of a set that meet some criterion, then the Maple function **select** is helpful. The opposite function **remove** and a combination of both, **selectremove**, also exist. A few examples:

```
> die := rand(-10..10): numberset := {'die()' $ 11};
numberset := {-8, -6, -5, -4, 0, 6, 7, 8, 10}
> select(isprime, numberset); # select prime numbers
{7}
> select(type, numberset, 'nonnegint');
{0, 6, 7, 8, 10}
> remove(x->x<-3, numberset); # remove numbers < -3
{0, 6, 7, 8, 10}
> selectremove(type, numberset, 'nonnegint');
{0, 6, 7, 8, 10}, {-8, -6, -5, -4}
```

The general format of selection and removal is

**select( criterion, set, extra arguments )**

and

**remove( criterion, set, extra arguments )**

The criterion should always return a boolean value *true* or *false*. Extra arguments of the selection/removal criterion are always added after the set from which elements will be selected or removed. The same format is used for the **selectremove** procedure.

## 12.3 List

Whereas sets are sequences enclosed by braces, you can construct an object of type *list* by enclosing a sequence in square brackets. You have already seen such objects being used as arguments in **collect** to specify the ordering of variables in polynomials.

```
> 1+x+y^2+z^3+x^2*y^2+x^2*z^3+y^3*z^3;
1 + x + y2 + z3 + x2 y2 + x2 z3 + y3 z3
> collect(% , [x, y, z]);
(y2 + z3) x2 + x + 1 + y2 + z3 + y3 z3
> collect(% , [z, y, x]);
(x2 + 1 + y3) z3 + (x2 + 1) y2 + x + 1
```

Important differences between objects of type *list* and *set* are that

- in a list the same objects may occur more than once;
- the ordering of the elements of a list is preserved;
- list elements can easily be assigned new values.

The internal structure of a list is however similar to that of a set.



Figure 12.3. Internal data structure of a list.

```

> [x, x, x*y, x*(x-1), y*x, x^2-x];
      [x, x, x y, x (x - 1), x y, x2 - x]
> sort(%, 'address'); # sort by machine address
      [x, x, x y, x y, x (x - 1), x2 - x]
> convert(%, set);
      {x, x y, x (x - 1), x2 - x}
> [x$5]; # list of five x's
      [x, x, x, x, x]
> 'empty list' := [];
      empty list := []
  
```

Below, the most frequently used operations on lists are listed and examples show how they can be carried out in Maple.

- Find the length of a list.

```

> cl := [black, red, green, yellow, blue, white];
> nops(cl); # number of colors
  
```

6

- Search for an element in a list.

```

> member(indigo, cl);
      false
> member(blue, cl, 'position');
      true
> position;
  
```

5

- Select one or more elements.

```
> cl[5];      # efficient selection
               blue
> op(5, cl); # inefficient selection
               blue
```

The efficiency of the selection of a list element by `op` is  $O(n)$ , where  $n$  is the length of the list, because the whole list is evaluated. When the square brackets are used, Maple uses a different evaluation scheme such that the evaluation of the whole list is suppressed and becomes constant time. The following example illustrates this.

```
> L := [x$(2^26-4)]: # the longest x-list
> # access time to the longest list
> time(): L[1]: time(): cpu_time=(%-%%)*second;
               cpu_time = 0.020 second
> time(): op(1,L): time(): cpu_time=(%-%%)*seconds;
               cpu_time = 2.634 seconds
> time(): L[2^26-4]: time():
> cpu_time=(%-%%)*second;
               cpu_time = 0.020 second
> time(): op(2^26-4, L): time():
> cpu_time = (%-%%)*seconds
               cpu_time = 2.643 seconds
```

The selection of list elements is more general. Some examples, involving the preceding list `cl`:

```
> cl[-1];      # first list element from the right
               white
> cl[3..6];    # return a sublist
               [green, yellow, blue, white]
> op(3..6, cl); # return a sequence
               green, yellow, blue, white
> cl[-2..-1];  # the sublist of last two elements
               [blue, white]
> cl[];        # sequence of list elements
               black, red, green, yellow, blue, white
> # special cases: drop the first or the last element
> cl[2..nops(cl)];
               [red, green, yellow, blue, white]
```

```

> cl[2..-1];
      [red, green, yellow, blue, white]
> subsop(1=NULL, cl);
      [red, green, yellow, blue, white]
> cl[1..nops(cl)-1];
      [black, red, green, yellow, blue]
> cl[1..-2];
      [black, red, green, yellow, blue]
> subsop(nops(cl)= NULL, cl);
      [black, red, green, yellow, blue]

```

- Prepend, append, and insert elements.

```

> [crimson, op(cl)];
      [crimson, black, red, green, yellow, blue, white]
> [op(cl), crimson];
      [black, red, green, yellow, blue, white, crimson]
> [op(cl[1..3]), crimson, op(cl[4..nops(cl)])];
      [black, red, green, crimson, yellow, blue, white]

```

Note that for long lists the creation of a sublist followed by conversion to a sequence is more efficient than direct use of **op** with the requested range because in the latter case the whole list is fully evaluated.

- Concatenate lists.

```

> cl2 := [crimson, indigo]: [op(cl), op(c12)];
      [black, red, green, yellow, blue, white, crimson, indigo]

```

- Replace one element in a list.

```

> subsop(5=indigo, cl); # cl unchanged
      [black, red, green, yellow, indigo, white]
> cl[5] := purple; cl; # cl changed
      cl5 := purple
      [black, red, green, yellow, purple, white]

```

The latter assignment is more efficient than the command `cl := subsop(5=indigo, cl)`. However, direct assignment of a list element is restricted to lists with at most 100 entries.

- Replace all specified elements in a list.

```
> subs(red=crimson, cl); # replace every color red
[black, crimson, green, yellow, purple, white]
```

- Rearrange lists.

```
> # sort the list (in lexicographic ordering)
> sort(cl, lexorder); # variable cl is not changed
[black, green, purple, red, white, yellow]
> # reverse the list
> [seq(cl[-i], i=1..nops(cl))];
[white, purple, yellow, green, red, black]
> # rotate the list one position to the left
> [op(2..nops(cl), cl), cl[1]];
[red, green, yellow, purple, white, black]
> [op(cl[2..-1]), cl[1]];
[red, green, yellow, purple, white, black]
> # rotate the list one position to the right
> [cl[nops(cl)], op(1..nops(cl)-1, cl)];
[white, black, red, green, yellow, purple]
> [cl[-1], op(cl[1..-2])];
[white, black, red, green, yellow, purple]
```

Using a few basic Maple functions and constructs you can carry out most operations on lists so easily that the designers of Maple have chosen in the beginning not to waste function names for these operations. If you wish, you can easily transcribe the above examples into Maple procedures. For example, the last two operations could be transcribed into procedures in the following way.

```
> rotateleft := proc(L::list)
>   [op(L[2..-1]), L[1]]
> end proc;
> rotateright := proc(L::list)
>   [L[-1], op(L[1..-2])]
> end proc;
> rotateleft(cl);
[red, green, yellow, purple, white, black]
> (rotateright@@3)(cl); # rotate 3 places to the right
[yellow, purple, white, black, red, green]
> (rotateleft@rotateright)(cl); # do nothing
[black, red, green, yellow, purple, white]
```

But opinions of software designers can change: as of version 7, Maple contains the **ListTools** package with some utilities to manipulate lists. We redo a few of the above examples.

```
> with(ListTools);

Warning, the assigned name Group now has a global binding

[BinaryPlace, BinarySearch, Categorize, DotProduct, FindRepetitions,
Flatten, FlattenOnce, Group, Interleave, Join, JoinSequence,
MakeUnique, Pad, PartialSums, Reverse, Rotate, Sorted, Split,
Transpose]

> 'cl' = cl;
      cl = [black, red, green, yellow, purple, white]
> 'rotated cl' = Rotate(cl, 3);
      rotated cl = [yellow, purple, white, black, red, green]
> 'reversed cl' = Reverse(cl);
      reversed cl = [white, purple, yellow, green, red, black]
```

Maybe, the most important command from the **ListTools** package is the **Flatten** command: it removes brackets in nested list structures and thus provides an easy method to get rid of superfluous list brackets.

```
> Flatten([cl, crimson]);
      [black, red, green, yellow, purple, white, crimson]
> Flatten([[1,2,3], [4, [5,6]]]);
      [1, 2, 3, 4, 5, 6]
> Flatten([[1,2,3], [4, [5,6]]], 1); # flatten at top level
      [1, 2, 3, 4, [5, 6]]
```

Look once again at the example of appending an element to a list:

```
> cl := [op(cl), crimson]:
```

You may wonder why we append an element *elem* to a list *L* via `[op(L), elem]` and not by the assignment `L[nops(L)+1] := elem`. The example below shows that you can assign value to existing list elements, but that you cannot extend a list in this way.

```
> L := [a,b,c];
      L := [a, b, c]
> L[3] := [c,c]; # assignment of existing list element
      L3 := [c, c]
> L;
```

```

[a, b, [c, c]]
> L[3,2] := d; # assignment of existing list element
L3,2 := d
> L;
[a, b, [c, d]]
> L[4] := e;    # no extension of list
Error, out of bound assignment to a list

```

Table 12.1 summarizes the selectors for a composite data structure  $T$  that can be of type *sequence*, *set*, or *list*.

Selection	Returned Data Structure
$T[]$	expression sequence of components
$T[i]$ , ( $i \in \mathbb{N}$ )	$i$ -th component from the left
$T[-i]$ , ( $i \in \mathbb{N}$ )	$i$ -th component from the right
$T[i..j]$ , ( $i, j \in \mathbb{Z}$ )	the range between $i$ and $j$ as sequence, set, or list
$T[i,j,...]$	$T[i][j,...]$
$T[i..j,k,...]$	$\text{seq}((T[n])[k,...], n=i..j)$ as a sequence, set, or list

Table 12.1. Selection in sequence, set, or list.

## 12.4 Arrays

Like conventional programming languages, Maple has a data structure for one-, two-, or multi-dimensional arrays. As a matter of fact, it has two data structures for arrays: **array** and **Array**. Below is are two examples of an object of type *array*.

```

> array(-1..1, 0..1,
>       {(-1,0)=p, (-1,1)=q, (0,0)=r, (0,1)=s});
array( - 1..1, 0..1, [
(-1, 0) = p
(-1, 1) = q
(0, 0) = r
(0, 1) = s
(1, 0) = ?1,0
(1, 1) = ?1,1
])

```

```
> a := array(1..3, 1..2,
>           {(1,1)=p, (1,2)=q, (2,1)=r, (2,2)=s});
```

$$a := \begin{bmatrix} p & q \\ r & s \\ a_{3,1} & a_{3,2} \end{bmatrix}$$

We give the same two examples as objects of type *Array* and then discuss the differences.

```
> Array(-1..1, 0..1,
>        {(-1,0)=p, (-1,1)=q, (0,0)=r, (0,1)=s});
```

*Array*(-1..1, 0..1, {(-1, 0) = *p*, (-1, 1) = *q*, (0, 0) = *r*, (0, 1) = *s*},  
*datatype* = anything, *storage* = rectangular, *order* = Fortran\_order)

```
> A := Array(1..3, 1..2,
>           {(1,1)=p, (1,2)=q, (2,1)=r, (2,2)=s});
```

$$A := \begin{bmatrix} p & q \\ r & s \\ 0 & 0 \end{bmatrix}$$

Firstly, the ranges of the indices of a two-dimensional array are specified. Characteristic of an array is the use of a Cartesian product of segments of contiguous integers as the set of allowed indices. Secondly, some, but not necessarily all, array elements are assigned values. Be careful not to mess up the round and square brackets: this is a little bit confusing, as selection of array elements is done by square brackets. Thirdly, one- and two-dimensional arrays whose index ranges start at one are most frequently used as vectors and matrices, respectively. Hence, the typical vector- and matrix-notation is used.

The data type *array* is meant for sparse, mutual arrays. It is internally based on the data structure *table*, which is frequently used by Maple itself (e.g., remember tables of procedures) and will be discussed in the next section. Unspecified *array* elements are displayed using the name of the array or a question mark.

```
> seq(seq(a[i,j], j=1..2), i=1..3);
p, q, r, s, a3,1, a3,2
```

This is in contrast with the data structure *Array*, which is meant for dense, mutable arrays. Unspecified elements get a default value, here 0.

```
> seq(seq(A[i,j], j=1..2), i=1..3);
p, q, r, s, 0, 0
```

An *Array* is internally based on the data structure *rtable* (rectangular table), which is described in §12.7. This data structure provides many options for describing in detail the shape of the array, the type of array elements, and the way they are stored internally.

At first sight, the data types *array* and *Array* are similar to the data type *list*. This is reflected in the way objects of these types can be constructed from objects of type *list*. Below, one-, two-, and three-dimensional arrays are constructed. It is shown that they only look like lists, vectors, or matrices, but that they are in fact different data structures.

```

> v := array([1+p, 2+q, 3+r]);
          v := [1 + p, 2 + q, 3 + r]
> type(v, list);
          false
> type(v, And(vector, array, table));
          true
> m := array([[1-p, 2-q], [1-r, 2-s]]);
          m := 
$$\begin{bmatrix} 1 - p & 2 - q \\ 1 - r & 2 - s \end{bmatrix}$$

> type(m, matrix);
          true
> V := Array([1+p, 2+q, 3+r]);
          V := [1 + p, 2 + q, 3 + r]
> type(V, {list, vector, array});
          false
> type(V, Vector);
          false
> type(V, And(Array, rtable));
          true

```

The one-dimensional array *V* is not of type *Vector*. For this to become true you must specify this subtype in the *rtable* constructor.

```

> V := rtable([1+p, 2+q, 3+r], subtype=Vector[row]);
          V := [1 + p, 2 + q, 3 + r]
> type(V, Vector);
          true

```

For two-dimensional arrays similar things happen.

```

> M := Array([[1-p, 2-q], [1-r, 2-s]]);
          M := 
$$\begin{bmatrix} 1 - p & 2 - q \\ 1 - r & 2 - s \end{bmatrix}$$

> type(M, {matrix, Matrix});

```

```

          false
> type(V, And(Array, rtable));
          true
> M := rtable([[1-p, 2-q], [1-r, 2-s]], subtype=Matrix);
          M := 
$$\begin{bmatrix} 1-p & 2-q \\ 1-r & 2-s \end{bmatrix}$$

> type(M, Matrix);
          true

```

By entering the array elements through lists (or lists of lists) a lot of typing is avoided. Furthermore, if you leave out the ranges of indices, Maple determines the index ranges from the lengths of the lists, assuming that the ranges start at one. You can see this in the next example of a three-dimensional array, where the index range is explicitly mentioned in the output.

```

> A := Array([[[1,2], [3,4]], [[5,6], [8,9]],
>             [[10,11], [12,13]]]);
A := [1..3 x 1..2 x 1..2 3 - D Array Data Type : anything
      Storage : rectangular Order : Fortran_order ]

```

Here, only the bounds of each dimension, the data type of the array elements, the storage method, and the internal ordering of the **Array** structure are displayed. The procedures **ArrayDims**, **ArrayNumDims**, and **ArrayNumElems** can also be used to inspect these properties.

```

> ArrayDims(A);
          1..3, 1..2, 1..2
> ArrayNumDims(A);
          3
> ArrayNumElems(A);
          12

```

By converting the **Array** structure into an **array**, the elements are displayed. Below we omitted however most of the output.

```

> convert(A, array);
array(1..3, 1..2, 1..2, [
  (1, 1, 1) = 1
  .....
  (3, 2, 2) = 13
])

```

You could also have constructed the matrix **m** and vector **v** in **array** representation by the commands **matrix** and **vector**, respectively.

```
> m := matrix([[1-p, 2-q], [1-r, 2-s]]);  
m := 
$$\begin{bmatrix} 1-p & 2-q \\ 1-r & 2-s \end{bmatrix}$$
  
> v := vector([1+p, 2+q, 3+r]);  
v := [1 + p, 2 + q, 3 + r]  
> lprint(eval(v));  
array(1 .. 3, [(1)=1+p, (2)=2+q, (3)=3+r])
```

You may wonder why we added the call of the procedure **eval** in the above instruction. The reason is that evaluation of variables that point to data of type *array* is different from the usual *full evaluation*, which applies to variables that point to formulae or lists. This special evaluation scheme will also hold for the data structures *table* and *procedure*. In §12.6 evaluation of these data types and the consequences will be explained in full. Here, we only point out this aspect.

```
> v; # no full evaluation of array  
v
```

Evaluation of the variable **v** does not show the **array**. You must force full evaluation by **eval**.

```
> eval(v);  
[1 + p, 2 + q, 3 + r]
```

Use **op** or **lprint** if you want to see the index range too.

```
> op(eval(v));  
1..3, [1 = 1 + p, 2 = 2 + q, 3 = 3 + r]
```

Oppositely, objects of type **Array** are subject to full evaluation.

```
> V; # full evaluation of Array  
V := [1 + p, 2 + q, 3 + r]
```

You may start wondering why Maple has two data structures for arrays. The main reason is that the original linear algebra package **linalg** is **array**-based. The underlying idea of the software developers was that last-name evaluation would make it easier to do matrix computations in an abstract way. However, the disadvantages of this approach have come in the meantime to the surface. This approach is difficult for users because they always have to think about special evaluation rules and about when to use the special evaluation routine **evalm** to get matrix computations actually done with the **&\*** operator.

```
> evalm(m &* m); # product of array-based matrices
```

$$\begin{bmatrix} (1-p)^2 + (2-q)(1-r) & (1-p)(2-q) + (2-q)(2-s) \\ (1-r)(1-p) + (2-s)(1-r) & (2-q)(1-r) + (2-s)^2 \end{bmatrix}$$

This approach is also inefficient for numerical computations with large matrices and it is not flexible when it comes to the use of numerical libraries (e.g., algorithms of the Numerical Algorithms Group, better known as NAG) or external programs (e.g., Matlab). This is why a new linear algebra package, viz. `LinearAlgebra`, has been introduced. Although many of the procedures of this package also work for objects of type *array*, you are strongly advised to use objects of type *Array* only. This is also why we continue this section with a discussion of `Array` data structures only.

So, let us return to one-dimensional objects of type *Array*. Sometimes they behave as vectors, and sometimes not. The following examples show the reason: arithmetic operations are done component-wise. So, addition, subtraction, and scalar multiplication are like vector arithmetic, but multiplication and powers are certainly not vector-like operations (interpreted for instance as inner product).

```
> V := Array([1+p, 1+q, 1+r]);
V := [1 + p, 1 + q, 1 + r]
> W := Array([-p, -q, -r]);
W := [-p, -q, -r]
> V+W;
[1, 1, 1]
> V*W;
[-(1 + p) p, -(1 + q) q, -(1 + r) r]
> V^2;
[(1 + p)^2, (1 + q)^2, (1 + r)^2]
```

Similarly two-dimensional objects of type *Array* resemble matrices. But keep in mind that there are actually different objects.

```
> M := Array([[1-p, 2-q], [1-r, 2-s]]);
M :=  $\begin{bmatrix} 1 - p & 2 - q \\ 1 - r & 2 - s \end{bmatrix}$ 
> M+2*M; # addition & scalar multiplication
 $\begin{bmatrix} (3 - 3p) & (6 - 3q) \\ (3 - 3r) & (6 - 3s) \end{bmatrix}$ 
> M^2;    # component-wise squaring
 $\begin{bmatrix} (1 - p)^2 & (2 - q)^2 \\ (1 - r)^2 & (2 - s)^2 \end{bmatrix}$ 
```

Although many of the procedures in the **LinearAlgebra** package work for native **Arrays**, you better use the **Vector** and **Matrix** procedures when you want to introduce objects that really behave like vectors and matrices in the meaning of linear algebra.

```
> M := Matrix([[1-p, 2-q], [1-r, 2-s]]);  
M := 
$$\begin{bmatrix} 1-p & 2-q \\ 1-r & 2-s \end{bmatrix}$$
  
> M^2; # square of a matrix  

$$\begin{bmatrix} (1-p)^2 + (2-q)(1-r) & (1-p)(2-q) + (2-q)(2-s) \\ (1-r)(1-p) + (2-s)(1-r) & (2-q)(1-r) + (2-s)^2 \end{bmatrix}$$

```

Let us linger a bit more on the subtle differences between objects created by the procedures **Array** and **Matrix**, respectively.

```
> M := Array([[1, 2, x], [3, 4, y]]);  
M := 
$$\begin{bmatrix} 1 & 2 & x \\ 3 & 4 & y \end{bmatrix}$$
  
> type(M, Matrix);  
false  
> type(M, rtable);  
false
```

In §12.7 you will learn about the data structure **rtable**, but the following assignments and extractions of an **Array** object may not surprise you much.

First, we assign a subarray.

```
> M[1..2,1..2] := Array([[1-p, 2-q], [3-r, 4-s]]);  
M1..2, 1..2 := 
$$\begin{bmatrix} 1-p & 2-q \\ 3-r & 4-s \end{bmatrix}$$
  
> M; # full evaluation  

$$\begin{bmatrix} 1-p & 2-q & x \\ 3-r & 4-s & y \end{bmatrix}  
> M[1..2,1..2]; # subArray  

$$\begin{bmatrix} 1-p & 2-q \\ 3-r & 4-s \end{bmatrix}$$$$

```

The first component of the array can be selected; in this case, it will give the first row.

```
> M[1];  
[1-p, 2-q, x]
```

The last column can be selected in the following way.

```
> M[1..2,3];
[x, y]
```

Use the **Matrix** procedure to create matrices that are based upon rectangular tables. Default value of an object of type *Matrix* is 0, but you can specify another default value via the *fill* option.

```
> M := Matrix(2, 3, [[1-p, 2-q], [3-r, 4-s]], fill=u);
M := 
$$\begin{bmatrix} 1-p & 2-q & u \\ 3-r & 4-s & u \end{bmatrix}$$

> M[1..2,1..2]; # selection of subMatrix

$$\begin{bmatrix} 1-p & 2-q \\ 3-r & 4-s \end{bmatrix}$$

> # assignment of a subMatrix
> M[1..2,1..2] := Matrix([[alpha, beta], [gamma, delta]]);
M[1..2,1..2] := 
$$\begin{bmatrix} \alpha & \beta \\ \gamma & \delta \end{bmatrix}$$

> M; # full evaluation of Matrix
M := 
$$\begin{bmatrix} \alpha & \beta & u \\ \gamma & \delta & u \end{bmatrix}$$

```

Because the **Matrix** procedure really creates matrices in the mathematical sense, there are some differences between object of type *Matrix*, *Array*, and *rtable*. The following examples illustrate this.

```
> M[1];
Error, Matrix requires two indices
> M[1,1..3]; # first row of Matrix
[alpha, beta, u]
> M[1..2,3]; # third column of Matrix

$$\begin{bmatrix} u \\ u \end{bmatrix}$$

> M := M[1..2,1..2]; # remove third row and column
M := 
$$\begin{bmatrix} \alpha & \beta \\ \gamma & \delta \end{bmatrix}$$

```

The dot operator and the power symbol denote matrix multiplication and computing of matrix powers, respectively.

```
> M.M; # Matrix product

$$\begin{bmatrix} \alpha^2 + \beta\gamma & \alpha\beta + \beta\delta \\ \gamma\alpha + \delta\gamma & \beta\gamma + \delta^2 \end{bmatrix}$$

```

```
> M^2; # Matrix power
```

$$\begin{bmatrix} \alpha^2 + \beta\gamma & \alpha\beta + \beta\delta \\ \gamma\alpha + \delta\gamma & \beta\gamma + \delta^2 \end{bmatrix}$$

Sometimes you just want to declare a matrix without specifying the elements. This is no problem in Maple. As you have seen before, the default value of matrix elements will be 0 and can be specified in the **Matrix** command via the **fill** option.

```
> M := Matrix(100,100): M[13,17];
0
> M := Matrix(100,100, fill=1): M[13,17];
1
```

Index functions are convenient for filling large matrices. Although such matrices are not displayed when the dimensions are too large, you can still get an idea of the structure of the data by the so-called Matrix Browser. You enable it by right-clicking the Matrix output and selecting the **Browse...** item. Below, in Figure 12.4, you see a screen dump of the matrix browser for this particular example. Enter **?structuredview** for more details about this structured data viewer.

```
> Matrix(15, 15, (i,j)->igcd(i,j)-1);

[15 x 15 Matrix Data Type: anything
Storage: rectangular Order: Fortran_order]
```

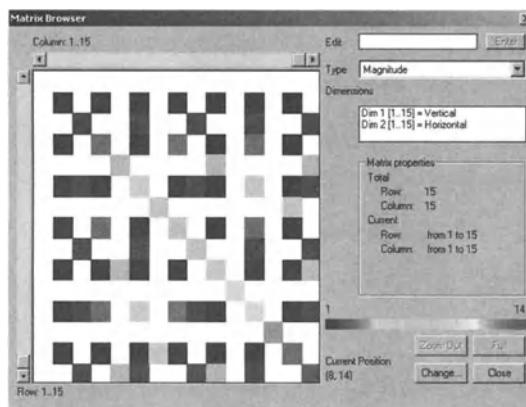


Figure 12.4. The Matrix Browser in use.

An index function can also be defined outside the **Matrix** instruction. For example, a  $4 \times 4$  Hilbert matrix can be made by

```
> h := (i,j) -> 1/(i+j-x): # index function
> Matrix(3, 3, h);
```

$$\begin{bmatrix} \frac{1}{2-x} & \frac{1}{3-x} & \frac{1}{4-x} \\ \frac{1}{3-x} & \frac{1}{4-x} & \frac{1}{5-x} \\ \frac{1}{4-x} & \frac{1}{5-x} & \frac{1}{6-x} \end{bmatrix}$$

In this case, you could also have used the built-in function **HilbertMatrix** from the **LinearAlgebra** package. output of following instruction is the same as above and is omitted

```
> LinearAlgebra[HilbertMatrix](3, 3, x);
```

Another example is the construction of a zero matrix. In the following function call, 0 means the zero function.

```
> Matrix(3, 3, 0);
```

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

A symmetric matrix can be introduced in the following way.

```
> M := Matrix(2, 2, shape=symmetric): M[1,2] := 1: M;
```

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

An alternative is to specify the special indexing in the **Array** procedure.

```
> Array(symmetric, 1..2, 1..2, (1,2)=1);
```

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

The role of the indexing functions is to indicate how matrix entries can be computed so that actual values do not have to be stored in the internal data structure. When more than one option has been specified, they are handled from left to right. A few more examples to illustrate this:

```
> M := Matrix(3, 3, shape=triangular[lower, unit], fill=2);
```

$$M := \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 2 & 2 & 1 \end{bmatrix}$$

```
> M[1,2] := y;
```

```
Error, attempt to assign to upper triangle of lower triangular
rtable
```

```
> M[1,2] := 0; # assignment in agreement with indexing
```

$$M_{1,2} := 0$$

```
> Matrix(3, 3, (i,j)->i, shape=diagonal);
```

```


$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$$

> Matrix(3, 3, Vector([3,2,1]), shape=diagonal);

$$\begin{bmatrix} 3 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$


```

Immutable matrices can be created without difficulty; just add the option `readonly=true`.

```

> Id := Matrix(2, 2, shape=identity, readonly=true);
Id :=  $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ 
> Id[1,1] := x;
Error, cannot assign to a read-only Matrix

```

Table 12.2 lists currently built-in indexing functions.

Name	Creation of
<code>antihermitean</code>	an antihermitian matrix
<code>antisymmetric</code>	an antisymmetric matrix
<code>band[b<sub>1</sub>, b<sub>2</sub>]</code>	a banded matrix
<code>constant[n]</code>	a vector or matrix with all entries equal to $n$
<code>diagonal</code>	a diagonal matrix
<code>hermitian</code>	a hermitian matrix
<code>Hessenberg[upper]</code>	a triangular matrix with an extra subdiagonal
<code>Hessenberg[lower]</code>	a triangular matrix with an extra superdiagonal
<code>identity</code>	an identity matrix
<code>scalar[n]</code>	a matrix with value $n$ on the main diagonal and 0 elsewhere
<code>scalar[j, n]</code>	a vector with value $n$ at position $j$ and 0 elsewhere
<code>symmetric</code>	a symmetric matrix
<code>triangular[lower]</code>	a lower triangular matrix
<code>              unit]</code>	on the main diagonal
<code>triangular[lower, upper]</code>	a lower triangular matrix with value 1
<code>triangular[upper]</code>	an upper triangular matrix
<code>triangular[upper, unit]</code>	an upper triangular matrix with value 1 on the main diagonal
<code>unit[j]</code>	a vector with 1 at position $j$ and 0 elsewhere
<code>zero</code>	a vector or matrix with all entries equal to 0

Table 12.2. Built-in special indexing functions.

You can create your own indexing functions. As an example, we shall define the special indexing function **tridiagonal**, or actually the procedure **index/tridiagonal**. When **index/tridiagonal** is called with two arguments it will retrieve the value of an entry; with three arguments it assigns an entry.

Format	Purpose
`index/tridiagonal`(`idx, rtb)`	retrieval of value
`index/tridiagonal`(`idx, rtb, val)`	assignment

Table 12.3. Formats of call of special indexing function.

Here, **idx** is a list containing the indices, **rtb** is the object of type *rtable* being indexed (indexing functions exist for this data type that includes *Array* and *Matrix*), and **val** is a list with the value to be assigned. The first argument of **rtb** is a name sequence, of which the first name of this sequence is perhaps **tridiagonal**. The first name decides which special indexing function to use, and is stripped for this call. Hence, the first argument of the table consists of the remaining names, which could be **symmetric** or **antisymmetric** among others.

When **index/tridiagonal** is called with two arguments, it returns the value of the entry being selected. When **index/tridiagonal** is called with three arguments, it is expected that the function will produce the assignment to an array entry. The value returned is used as the value resulting from the assignment. For this type of call, the indexing function will normally have a side-effect, typically altering **rtb**.

The actual code of **index/tridiagonal** can be as follows.

```
> 'index/tridiagonal' :=  
> proc(idx::list(posint), rtb::rtable, val::list)  
>   local idx1, init_val;  
>   idx1 := op(idx); # get the sequence of indices  
>   if nargs=2 then  
>     # retrieval  
>     if outofband(idx1) then  
>       # indexing function determines value, i.e.,  
>       # location is not mutable  
>       0;  
>     else  
>       # get value from storage  
>       rtb[idx1]  
>     end if;  
>   else  
>     # storage  
>     if outofband(idx1) then  
>       # indexing function determines value, i.e.,  
>       # location is not mutable
```

```

>      if op(val) <> 0 then
>          # invalid value
>          error "attempt to assign outside the band";
>      else
>          # be sure to do explicit write, but catch
>          # this if the storage fails, as storage
>          # for non-mutable locations need not be
>          # allocated
>          try
>              rtb[idx1] := op(val);
>              catch "unable to store":
>                  # couldn't write, return value (as if
>                  # the write succeeded)
>                  op(val);
>          end try;
>      end if;
>      else
>          # general value
>          rtb[idx1] := op(val);
>      end if;
>  end if;
> end proc:
```

Here, the following routine tests whether an index is inside or outside the band of width 1.

```

> outofband := proc()
>     local indset, pairs, p, d:
>     indset := {args};
>     if nops(indset) <= 1 then return false end if;
>     pairs := combinat[choose](indset, 2);
>     for p in pairs do
>         d := p[1]-p[2];
>         if type(d, integer) and abs(d) > 1
>             then return true # band width 1 is exceeded
>         end if;
>     end do:
>     return false
> end proc:
```

A few examples:

```

> A := Array(tridiagonal,1..4, 1..4, fill=UNKNOWN):
> A[1,2] := x;
```

$$A_{1,2} := x$$

```
> A[1,4] := y;
```

```
Error, (in index/tridiagonal) attempt to assign outside the band
```

```
> A;
```

```


$$\begin{bmatrix} ? & x & 0 & 0 \\ ? & ? & ? & 0 \\ 0 & ? & ? & ? \\ 0 & 0 & ? & ? \end{bmatrix}$$

> A := Array(tridiagonal,1..4, 1..4, (i,j)->i);

A := 
$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 2 & 2 & 2 & 0 \\ 0 & 3 & 3 & 3 \\ 0 & 0 & 4 & 4 \end{bmatrix}$$


> A := Matrix(1..4, 1..4, shape=[tridiagonal,
> antisymmetric], fill=y);

A := 
$$\begin{bmatrix} 0 & y & 0 & 0 \\ -y & 0 & y & 0 \\ 0 & -y & 0 & y \\ 0 & 0 & -y & 0 \end{bmatrix}$$


> A[1,2] := x; A[4,3] := z;
A_{1,2} := x
A_{4,3} := z
> A;

$$\begin{bmatrix} 0 & x & 0 & 0 \\ -x & 0 & y & 0 \\ 0 & -y & 0 & -z \\ 0 & 0 & z & 0 \end{bmatrix}$$


```

Here, the indexing functions `tridiagonal` and `antisymmetric` are both used. When using an entry, the following steps are gone through:

1. the first name `tridiagonal` is used, and `index/tridiagonal` is called first;
2. if the index is outside the band, then the value 0 is returned. Otherwise `index/tridiagonal` will access an entry of an array that is only antisymmetric (the `tridiagonal` word is dropped);
3. `index/antisymmetric` will access a table entry with the proper indexing order and a value will be returned;
4. `index/tridiagonal` returns this value.

We have defined the indexing function in such a way that it also works for higher-dimensional arrays and when indices contain unknowns.

```
> A := Array(tridiagonal, 1..3, 1..3, 1..3, fill=X);
```

```

A := [ 1..3 x 1..3 x 1..3 3 - D Array Data Type: anything
      Storage: rectangular Shape: tridiagonal Order: Fortran_order ]
> convert(A,array);

array(1..3, 1..3, 1..3, [
  (1, 1, 1) = X
  (1, 1, 2) = X
  (1, 1, 3) = 0
  .....
])
> for i from 1 to 3 do
>   'A'[1,2,i] = A[1,2,i]
> end do;

A1, 2, 1 = X
A1, 2, 2 = X
A1, 2, 3 = 0

```

Where matrices come into play, vectors are also present. Use the **Vector** procedure to create a vector that is internally based upon a rectangular table. Keep in mind that this makes it different from an object of type *vector*; it is of type *Vector*. By default, vectors are displayed in column format, with unspecified entries equal to 0.

```

> Vector(2);
          ⎡ 0 ⎤
          ⎣ 0 ⎦
> type(% , vector);
          false
> type(% , Vector);
          true

```

Of course you can change the default filling of entries.

```

> Vector(2, fill=1);
          ⎡ 1 ⎤
          ⎣ 1 ⎦

```

The dot operator is used for computing the hermitian product of two vectors.

```

> v := Vector([p,q]);
          v := ⎡ p ⎤
                  ⎣ q ⎦

```

```
> w := Vector([r,s]);
w := 
$$\begin{bmatrix} r \\ s \end{bmatrix}$$

> v.w;

$$\bar{p}r + \bar{q}s$$

```

You can choose any desired orientation of the display of an object of type *Vector* and you can specify a particular shape of such an object.

```
> Vector[row](2, shape=constant[1]);
```

```
[1, 1]
```

Use of user-defined index functions comes in handy, too.

```
> Vector[row](10, i->i-thprime(i));
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
```

As expected, large vectors are not shown; only a summary of the object is displayed. An easy way to get the vector on the screen in full detail is to convert it into an object of type *vector*.

```
> Vector[row](11, i->i-thprime(i));
```

```
[11 Element Row Vector Data Type: anything
Storage: rectangular Order: Fortran_order ]
```

```
> convert(%, vector);
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31]
```

Table 12.4 summarizes the main differences between the **table** based **array** and **matrix** on the one hand, and the **rtable** based **Array** and **Matrix** on the other hand.

<b>topic</b>	<b>array</b>	<b>matrix</b>	<b>Array</b>	<b>Matrix</b>
storage	sparse	sparse	dense	dense
special storage forms	no	no	yes	yes
indexing functions	yes	no	yes	yes
start of index ranges	any	1	any	1
default entries	symbolic	symbolic	0 (can be set)	0 (can be set)
mutable substructures	no	no	yes	yes
import/export via	<b>readdata</b>	<b>readdata</b>	<b>ImportMatrix</b>	<b>ImportMatrix</b>
	<b>writedata</b>	<b>writedata</b>	<b>ExportMatrix</b>	<b>ExportMatrix</b>
output display	full	full	full or short	full or short
multiplication via	<b>&amp;*&amp; + evalm</b>	<b>&amp;*&amp; + evalm</b>	.	.
kind of evaluation	last name	last name	full	full

Table 12.4. Comparison of arrays, matrices, Arrays, and Matrices.

## 12.5 Table: `table`

The Maple data structure `table` is similar to the record data structure in other programming languages like Pascal, Algol68, or C. Below is an example of an object of type *table*.

```
> color := table([red=RGB(1,0,0),
>                 turquoise=RGB(0.7,0.9,0.9), white=RGB(1,1,1)]);
color := table([white = RGB(1, 1, 1), red = RGB(1, 0, 0),
                turquoise = RGB(.7, .9, .9)])
```

The entries of a table are stored internally in a hash table and are as such not directly available to you; you have no control in what order they are stored internally. The use of hash tables allows constant-time access for all elements instead of access efficiency that depends on the position of an element in a table. But there is no need for internal access to components because you can easily retrieve the indices and entries of a table.

```
> indices(color);
[white], [red], [turquoise]
> entries(color);
[RGB(1, 1, 1)], [RGB(1, 0, 0)], [RGB(.7, .9, .9)]
```

You select or change an element of a table by the selection operator `[ ]`.

```
> color[red];
RGB(1, 0, 0)
> color[red] := HUE(0);
colorred := HUE(0)
> print(color);
table([white = RGB(1, 1, 1), red = HUE(0),
                turquoise = RGB(.7, .9, .9)])
```

The assignment to an indexed name is also the easiest way to build up a new table structure in Maple. In the following example we create a table called `Laplace` with only one entry, viz., the Laplace transform of  $\text{erf}(\sqrt{t})$ .

```
> Laplace[erf(sqrt(t))] := 1/(s*sqrt(s+1));
Laplaceerf( $\sqrt{t}$ ) :=  $\frac{1}{s\sqrt{s+1}}$ 
> print(Laplace);
table([erf( $\sqrt{t}$ ) =  $\frac{1}{s\sqrt{s+1}}$ ])
```

The procedure **index/newtable**, if defined, is used when an assignment of an indexed name is made where the table or array is not defined yet. By default it will then create a table, but it can also be adapted to your own wishes. For example, you cannot assign to a set element or a sequence by assignment of type  $S[x] := y$ .

```
> S := {a,b,c}: Q := a,b,c: S[3] := U;
```

```
Error, cannot assign to a set
```

```
> Q[3] := C;
```

```
Error, cannot assign to an expression sequence
```

However, the following procedure definition will allow this.

```
> 'index/newtable' :=
> proc(idx::list, tab::table, val::list)
>   if not assigned(tab)
>     then tab := table(): tab[op(idx)] := op(val)
>   elif nops([eval(tab)]) > 1 # expression sequence
>     and type(idx, [integer]) and idx[1] >= 1 and
>       idx[1] <= nops([eval(tab)])
>     then tab := op(subsop(idx[1]=op(val), [eval(tab)]))
>   elif type(eval(tab), set) and
>     type(idx, [integer]) and idx[1] >= 1 and
>       idx[1] <= nops(eval(tab))
>     then tab := subsop(idx[1]=op(val), eval(tab))
>   else error "not a table/array/set/sequence"
>   end if
> end proc:
```

```
> S; S[3] := U; S;
```

```
{a, b, c}
```

```
S3 := U
```

```
{a, b, U}
```

```
> Q; Q[3] := C; Q;
```

```
a, b, c
```

```
Q3 := C
```

```
a, b, C
```

Let us continue our discussion of tables and explain why we used **print** in previous examples to display tables and why we used so many **eval**'s in the source code of **index/newtable**. The reason is that the evaluation of variables that point to data of type *table* is different from the usual *full evaluation*. In §12.6 evaluation of an object of type *table* will be explained in full detail. Here, we only point out this aspect.

```
> Laplace; # evaluation to the table name
```

```
Laplace
```

You can use **op** or **eval** if you want to display the data structure.

```
> op(Laplace);

$$\text{table}([\operatorname{erf}(\sqrt{t}) = \frac{1}{s\sqrt{s+1}}])$$

> eval(Laplace);

$$\text{table}([\operatorname{erf}(\sqrt{t}) = \frac{1}{s\sqrt{s+1}}])$$

```

Removal of a table entry is done by unassignment, either by apostrophes or by **evaln**.

```
> color[red] := evaln(color[red]);

$$\text{color}_{\text{red}} := \text{color}_{\text{red}}$$

> print(color);

$$\text{table}([\text{white} = \text{RGB}(1, 1, 1), \text{turquoise} = \text{RGB}(.7, .9, .9)])$$

> for c in indices(color) do
>   color[op(c)] := evaln(color[op(c)])
> end do:
> print(color);

$$\text{table}([])$$

```

Now you also know how to define an empty table: just enter

```
> table();

$$\text{table}([])$$

```

Or even shorter: **table()**.

Like **Array** and **Matrix**, the procedure **table** has options available for structures of special shape.

```
> with(Units[Natural]): # warnings omitted
> distance := table(symmetric);

$$\text{distance} := \text{table}(\text{symmetric}, [])$$

> distance[Amsterdam, Brussels] := 157*km:
> distance[Amsterdam, 'Rio de Janeiro'] := 9512*km:
> print(distance);


$$\text{table}(\text{symmetric}, [(Brussels, Amsterdam) = 157 [\text{km}],$$


$$(Amsterdam, Rio de Janeiro) = 9512 [\text{km}]$$


$$> distance[Brussels, Amsterdam];

$$157 [\text{km}]$$$$

```

You can define your own special indexing function for tables in the way we explained in the previous section.

By the way, we have used the **Units** package in the above example so that you can see how unit can be dealt with in Maple. In the ‘natural’ setting, you can add the unit symbol to the quantity and the Maple system takes care of computations with units. With the **Units** package it is also easy to change the unit system.

```
> convert(%, units, miles);

$$\frac{2453125}{25146} [mi]$$

> distance[Amsterdam, 'Rio de Janeiro'];
> - distance[Brussels, Amsterdam];

$$9355 [km]$$

```

For further details about the **Units** package we refer to the online help system.

For completeness, we show in Figure 12.5 how an array or table is internally represented.

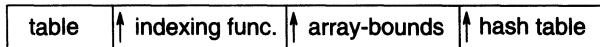


Figure 12.5. Internal data structure of a table.

## 12.6 Last Name Evaluation

Variables that point to data of type *array*, *table*, or *procedure* evaluate differently from the usual *full evaluation*. We shall illustrate this on the following example of a rotation matrix. The examples will also make clear why we gave the advice to use the *Array*, *Matrix*, *rtable* data types, and the advice to use the **LinearAlgebra** package: then objects are subjected to the less cumbersome full evaluation.

```
> R := array([[cos(alpha), -sin(alpha)],
   >           [sin(alpha), cos(alpha)]]);

$$R := \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{bmatrix}$$

> R;

$$R$$

> whattype(R);

$$symbol$$

> type(R, array);

$$true$$

```

Evaluation of the variable `R` does not show the array. You must force full evaluation by `eval`.

```
> eval(R);
```

$$\begin{bmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{bmatrix}$$

Actually, the individual array entries are not evaluated in the last instruction, as can be seen from the following continuation of the session.

```
> alpha := 1: eval(R);
```

$$\begin{bmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{bmatrix}$$

```
> R[1,2];
```

$$-\sin(1)$$

```
> map(eval, R);
```

$$\begin{bmatrix} \cos(1) & -\sin(1) \\ \sin(1) & \cos(1) \end{bmatrix}$$

The evaluation of arrays, tables, and procedures is called *last name evaluation*. This means that the result of evaluation is the last name in the chain of values just before the final object of type *array*, *table*, or *procedure* is reached. So, in the above example, Maple evaluates `R` just to its own name of type *symbol*. Full evaluation can be forced by `eval`, or even further by mapping `eval`.

The effect of *last name evaluation* of arrays can of course be shown better by an example in which a variable does not evaluate itself to an array but to a name of an array.

```
> T := S;
```

$$T := S$$

```
> S := R;
```

$$S := R$$

```
> eval(T, 1); # value of T
```

$$S$$

```
> eval(T, 2); # value of S
```

$$R$$

```
> eval(T, 3); # value of R
```

$$\begin{bmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{bmatrix}$$

```
> map(eval, T); # evaluation of array entries
```

$$\begin{bmatrix} \cos(1) & -\sin(1) \\ \sin(1) & \cos(1) \end{bmatrix}$$

```
> T; # evaluation of T to last name
```

*R*

The internal data structure looks as Figure 12.6.



Figure 12.6. Internal data structure showing evaluation of arrays.

It is clear that *R*, *S*, and *T* all point to the same array. When you refer to the array element *R*[1, 2], you get the same result as when you had chosen *S*[1, 2] or *T*[1, 2].

```
> alpha := 'alpha': # reset alpha to its name
> R[1,2], S[1,2], T[1,2];
          -sin(alpha), -sin(alpha), -sin(alpha)
```

When you change the array element *S*[1, 2], all three arrays are simultaneously changed in the same way. The last name evaluation makes it possible to use names as synonyms.

```
> eval(R);
          [ cos(alpha) -sin(alpha)
            sin(alpha)  cos(alpha) ]
> S[2,1] := 0:
> eval(R), eval(S), eval(T);
          %1, %1, %1
          %1 := [ cos(alpha) -sin(alpha)
                    0           cos(alpha) ]
```

If you want to make a copy of the array and then change this copy without disturbing the original array, you should use the procedure **copy**.

```
> S := copy(R): S[1,2] := 0:
> eval(R), eval(S);
          [ cos(alpha) -sin(alpha) ], [ cos(alpha) 0
            0           cos(alpha) ]
```

## 12.7 Rectangular Table: **rtable**

Arrays and tables of type *array* and *table*, respectively, are very flexible data structures in Maple, but there is a price to pay: a special evaluation scheme is used, viz., last name evaluation in order to make abstract algebra possible with arrays. However, this is more difficult to understand and less

easy to remember than full evaluation. Also, Maple uses a sparse implementation of arrays and table, but for large data structures this is not efficient. For example, it is almost impossible or at least very time-consuming to work in this style with matrices of dimension greater than 500. As we said before: enough reasons for the developers of Maple to introduce similar, but different data structures, namely **rtable** and **Array**.

The procedure **rtable** creates as its name suggest a rectangular **table**, i.e., a table whose entries are within given integer ranges. For example, to create a two-dimensional table where row- and column-indices start at 0, enter

```
> A := rtable(0..1, 0..1);
```

$$A := \text{Array}(0..1, 0..1, \{\}, \text{datatype} = \text{anything}, \text{storage} = \text{rectangular}, \\ \text{order} = \text{Fortran\_order})$$

By default, rectangular table entries will be zero.

```
> A[0,0], A[0,1], A[1,0], A[1,1];\\ 0, 0, 0, 0
```

You can add a third argument to specify the initial values of the rectangular table. Via the **datatype** option you can specify the data type of the **rtable** elements. Henceforth, all required type conversions will be carried out automatically

```
> A := rtable(1..3, 1..3, 1, datatype=float[8]);
```

$$A := \begin{bmatrix} 1. & 1. & 1. \\ 1. & 1. & 1. \\ 1. & 1. & 1. \end{bmatrix}$$

You see that if the rectangular table is two-dimensional and with integer ranges starting at 1, i.e., if the rectangular table is in fact a two-dimensional array, then the data structure is represented as a matrix. If you ask Maple about its data type, the answer will be **Array**. Note the upper case first character: it indicates that the data structure is internally based on the **rtable** data structure. Do not forget: the lower case procedure **array** is based on the **table** structure.

```
> whattype(A);
```

*Array*

Assignment of rectangular table entries is very flexible: you can assign individual entries or replace complete subtables.

```
> A[3,3] := 2; A;
```

$$\begin{bmatrix} 1. & 1. & 1. \\ 1. & 1. & 1. \\ 1. & 1. & 2. \end{bmatrix}$$

```
> A[1..3, 1..2] := rtable(1..3,1..2): A;

$$\begin{bmatrix} 0. & 0. & 1. \\ 0. & 0. & 1. \\ 0. & 0. & 2. \end{bmatrix}$$

```

Also note that full evaluation applies to rectangular tables: there is no need for the `eval` command. The internal ordering of a rectangular table is `Fortran_order`, i.e., storage by columns. With the optional argument `order=C_order`, you specify storage by rows. However, this is only important when you are importing or exporting a rectangular table as the following example of extracting rows and columns of a two-dimensional rectangular table illustrates.

```
> A[1]; # first row
[0., 0., 1.]
> A[1..3,1]; # first column
[0., 0., 0.]
```

As we said in the introduction, rectangular tables have been introduced for efficiency reasons, too. If you introduce a large rectangular table, Maple will not show the data structure in detail, but instead it will give a summary. When you use the worksheet interface, you can call the interactive data structure viewer via the context-sensitive menu, which is invoked by right-clicking the Maple result.

```
> A := rtable(1..11, 1..11);
A := [1..11 x 1..11 2 - D Array Data Type: anything
      Storage: rectangular Order: Fortran_order ]
```

The interface variable `rtablesize` determines the upper bound of the dimension range for which rectangular tables are still displayed inline. If you give it the value `infinity`, then Maple will never hide the data structure behind a placeholder. Its default value can be found as follows.

```
> interface(rtablesize);
10
```

With the optional argument `readonly=true` you specify an immutable rectangular table, i.e., a table whose entries cannot be modified anymore.

```
> v := rtable(1..3, fill=1, readonly=true);
v := [1, 1, 1]
> v[1] := 0;
Error, cannot assign to a read-only Array
```

Above, we have used the option `fill=1` to specify that 1 is the default value that unspecified rectangular table elements are filled in with. Note

that the one-dimensional array is of type **Array** and not of type **Vector**. For this to become true you must specify this subtype. At the same time, you can specify a column display of the vector.

```
> whattype(v);
                                         Array
> type(v, Vector);
                                         false
> v := rtable(1..3, subtype=Vector[column]);
                                         v := 
$$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

> type(v, Vector);
                                         true
```

Finally we would like spend a few words on special indexing functions and special storage forms. Three examples, viz., the **identity** and **uppertriangular** indexing function, and the upper-triangular storage form will suffice.

```
> A := rtable(identity, 1..3, 1..3);
                                         A := 
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

```

Note the difference between the upper-triangular indexing function and the upper-triangular storage form.

```
> A := Array(triangular[upper], 1..3, 1..3, fill=1);
                                         A := 
$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

> A := rtable(1..3, 1..3, storage=triangular[upper], fill=1);
                                         A := 
$$\begin{bmatrix} 1 & 1 & 1 \\ - & 1 & 1 \\ - & - & 1 \end{bmatrix}$$

> A[2,1];
                                         Error, unable to lookup index (2, 1) -- not in storage
```

Nevertheless, you can define your own special storage form in exactly the same way as we introduced our own indexing functions in §12.4 Enter **?rtable** for an overview of built-in special storage forms.

There exist of course procedures to find out the bounds of each dimension, the number of dimensions, and the number of elements stored in a rectangular table.

```
> rtable_dims(A);
1..4, 1..4
> rtable_num_dims(A);
2
> rtable_num_elems(A, Stored);
10
```

## 12.8 Record Data Structure

Pascal-style records, which are composite structures that have some number of named “fields”, can be constructed in Maple via the **Record** procedure. As an example, we shall implement complex numbers in polar form in this way.

```
> c1 := Record('radius'=4, 'argument'=Pi/2);
c1 := module() export radius, argument; option record; end module
```

As you see, Maple records are implemented by using so-called modules. We refer to the Advanced Programming Guide [181] for further details. Here, all you need to know is how to access fields of a record data structure: you can use the member selection operator `:-` for this purpose.

```
> c1:-radius; # get the radius of c1
4
> c1:-argument; # get the argument of c1
 $\frac{1}{2}\pi$ 
```

We define a procedure to test whether an object is a complex number in polar form that is implemented in the above record data structure.

```
> 'type/Polar' := 'record(radius, argument)':
> type(c1, Polar);
true
```

We use this type-checking function to define a procedure for multiplication of two complex numbers.

```
> PolarMultiply := (c1::Polar, c2::Polar) ->
>     Record( 'radius' = c1:-radius * c2:-radius,
>             'argument' = c1:-argument + c2:-argument):
```

We multiply the following complex number with itself and verify that the result is equal to the above complex number  $c1$ .

```
> c2 := Record('radius'=2, 'argument'=Pi/4):
> c3 := PolarMultiply(c2, c2):
> verify(c1, c3, record);
true
```

We end this section with a procedure to convert the record data structure that represents a complex number in polar form into a complex number in Cartesian form. The **use ... end use** construction allows us to use short names for record fields in the procedure definition

```
> Polar2Cartesian := proc(c::Polar)
>   use r=c:-radius, phi=c:-argument in
>     r*cos(phi) + r*sin(phi)*I
>   end use
> end proc:
> Polar2Cartesian(c1);
4 I
> Polar2Cartesian(c2);
 $\sqrt{2} + I\sqrt{2}$ 
```

## 12.9 Function Call

You can also use a function call as a record data structure. For example, a complex number in polar coordinates is represented in Maple as a function call of type **polar**(*absolute value, argument*).

```
> polar(3, Pi/2);
polar(3,  $\frac{1}{2}\pi$ )
```

Here, **polar**( $\dots$ ) works as a placeholder so that other procedures can act upon the data structure.

```
> evalc(%);
3 I
```

Other examples of the use of function calls as record data structures are **RootOf**, **DESol**, and **PIECEWISE**.

One of the nice aspects of function calls is that you yourself can define how to print the function call. For our complex number in polar coordinates you could define the following printing routine.

```
> 'print/polar' := proc(r,a) r*e^(I*a) end proc:
> polar(5, Pi/3); # pretty display
5 e(1/3 I  $\pi$ )
```

```

> whattype(%);    # still a function call
                           function
> lprint(%%);      # internal representation
polar(5,1/3*Pi)
> evalc(%%);
                            $\frac{5}{2} + \frac{5}{2} I \sqrt{3}$ 
> convert(%, 'polar');
                            $5 e^{(1/3 I \pi)}$ 

```

Selection of a component of a function call is done by **op**.

```

> op(2, %);
                            $\frac{1}{3} \pi$ 

```

With the **macro** facility you can make sensible names for selection operations.

```

> macro(radius=1, argument=2):
> op(radius, %%), op(argument, %%);
                            $5, \frac{1}{3} \pi$ 

```

Of course, you can also define procedures to select components.

```

> macro(radius=radius):
> radius := proc(z)
>   if type(z, polar(anything, anything))
>   then op(1, z)
>   else 'procname(args)'
>   end if
> end proc:
> polar(5,Pi/3);
                            $5 e^{(1/3 I \pi)}$ 
> radius(%);
                           5

```

You can change a component of a function call by **subsop**.

```

> subsop(argument=Pi/6, %%);
                            $5 e^{(1/6 I \pi)}$ 

```

For function calls you can easily extend library routines such as **evalf**, **simplify**, **convert**, and so on. For example, you can extend the evaluation of a complex number in floating-point arithmetic as follows.

```

> 'evalf/polar' := proc()
>   local z;
>   z := polar(op(evalf([args])));
>   if type(z, polar(float, float)) then
>     op(1,z) * exp(op(2,z)*I)
>   end if
> end proc:
> polar(5,Pi/3);


$$5 e^{(1/3 I \pi)}$$


> evalf(%);

$$2.500000001 + 4.330127018 I$$


```

When **evalf** is applied to an expression that contains function calls **polar**( $\dots$ ), Maple searches for the definition of the procedure **evalf/polar** and, if necessary, loads it from the Maple library or from your private library. In the above session a definition of **evalf/polar** is available and can be applied.

Another example of extending Maple's facilities: conversion from polar to Cartesian form.

```

> 'convert/cartesian' := proc(z)
>   local r,a;
>   if type(z, polar(anything, anything))
>   then r := op(1,z); a := op(2,z); r*cos(a) + r*I*sin(a)
>   else 'procname(args)'
>   end if
> end proc:
> polar(5,Pi/3);


$$5 e^{(1/3 I \pi)}$$


> convert(% , cartesian);

$$\frac{5}{2} + \frac{5}{2} I \sqrt{3}$$

> polar(R,phi): % = convert(% , cartesian);

$$R e^{(I \phi)} = R \cos(\phi) + I R \sin(\phi)$$


```

Whenever **convert**( $\dots$ , *type*) is called, Maple searches for the definition of the procedure **convert/type** and, if necessary, loads it from the Maple library or from your private library. In the above session, a definition of **convert/cartesian** is available and can be applied to the expression.

## 12.10 Conversion between Composite Data Types

You have already seen how to convert a sequence into a set or list: simply surround the object by braces or square brackets, respectively. Objects of type *set* or *list* can be converted through the function **op**. Maple also

provides some explicit conversion routines for sets and lists. Below are a few examples that summarize these conversions.

```
> mysequence := a, b, b, q;
      mysequence := a, b, b, q
> mylist := [mysequence]; # sequence --> list
      mylist := [a, b, b, q]
> myset := {mysequence}; # sequence --> set
      myset := {a, b, q}
> op(mylist); # list --> sequence
      a, b, b, q
> mylist[];    # list --> sequence
      a, b, b, q
> op(myset);  # set --> sequence
      a, b, q
> myset[];    # set --> sequence
      a, b, q
> convert(mylist, set);      # list --> set
      {a, b, q}
> convert(myset, list);      # set --> list
      [a, b, q]
```

Such conversion routines can also be used for conversions between sets and lists on the one hand, and arrays and function calls on the other hand.

```
> myarray := array(mylist); # list --> array
      myarray := [a, b, b, q]
> convert(mylist, array);   # list --> array
      [a, b, b, q]
> convert(myarray, set);   # array --> set
      {a, b, q}
> myArray := Array(mylist); # list --> Array
      myArray := [a, b, b, q]
```

Curiously, **convert** does not change a list into an object of type *Array*.

```
> convert(mylist, Array);
Error, unrecognized conversion
```

```

> convert(myArray, list);      # Array --> list
                                [a, b, b, q]
> myVector := convert(myArray, Vector[column]);
> # Array -> Vector
                                
$$myVector := \begin{bmatrix} a \\ b \\ b \\ q \end{bmatrix}$$

> Vector(mylist);            # list -> Vector
                                
$$myVector := \begin{bmatrix} a \\ b \\ b \\ q \end{bmatrix}$$

> convert(myVector, vector); # Vector -> vector
                                [a, b, b, q]
> mycall := F(mysequence);   # sequence --> function call
                                mycall := F(a, b, b, q)
> convert(mycall, list);    # function --> list
                                [a, b, b, q]
> convert(mycall, set);     # function --> set
                                {a, b, q}
> myarray2D := array([mylist, mylist]);
                                myarray2D := 
$$\begin{bmatrix} a & b & b & q \\ a & b & b & q \end{bmatrix}$$

> convert(myarray2D, listlist); # array --> list of lists
                                [[a, b, b, q], [a, b, b, q]]
> convert(%, array); # list of lists --> array
                                
$$\begin{bmatrix} a & b & b & q \\ a & b & b & q \end{bmatrix}$$


```

These conversions also work for higher-dimensional arrays. You can use combinations of above conversions to change the shape of lists.

```

> L := [1, 2, 3, 4, 5, 6, 7, 8];
> matrix(2, 4, L); # array
                                
$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{bmatrix}$$

> convert(%, listlist); # array --> list

```

```

[[1, 2, 3, 4[, [5, 6, 7, 8]]]
> convert(matrix(4, 2, L), listlist); # array --> list
[[1, 2], [3, 4], [5, 6], [7, 8]]
> setattr(attribute(%), matrix);

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{bmatrix}$$

> op(%);
[1, 2], [3, 4], [5, 6], [7, 8], [9, 10], [11, 12]
> map(op, %);
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

```

In Table 12.5 we summarize then main conversions between the composite data types sequence, set, list, array and function call.

$S \setminus T$	seq.	set	list	array	call
seq.		{%}	[%]	array([%])	f(%)
set	op(%)		[op(%)]	array([op(%)])	f(op(%))
	%[]		convert(%[],list)		f(%[])
list	op(%)	{op(%)}		array(%)	f(op(%))
	%[]	convert(%[],set)		convert(%[],array)	f(%[])
array			convert(%[],set)	convert(%[],list)	
call	op(%)	convert(%[],set)	convert(%[],list)		

Table 12.5. Main conversions from composite data type  $S$  to  $T$ .

## 12.11 Exercises

1. Let  $U$  be the set of the first ten prime numbers. Let  $V$  be the set of the first twenty natural numbers of the form  $2^n - 1$ .
  - (a) Generate the sets  $U$  and  $V$  in Maple.
  - (b) Compute  $U \cup V$  and  $U \cap V$ .
2.
  - (a) Generate a list of 100 integers between  $-10$  and  $10$ .
  - (b) Remove all duplicate entries from this list.
  - (c) Select from the list obtained in (b) all numbers that can be divided by 2 or 3.
  - (d) Pick out from the list obtained in (b) all numbers greater than 5.
  - (e) For each number in the list obtained in (d) compute the number of occurrences in the original list of random numbers.

3. You know of course how many solutions the equation  $x^{50} - x^{20} = 0$  has over the complex numbers. When you use the procedure **solve** to solve equations, so many equations scroll over the terminal screen that it is difficult to count the number of solutions found by Maple. Use the procedures **nops** and **map** for the above mentioned equation to count the number of solutions and to check the answers by back substitution. (Advice: in order to be able to use **map** in combination with **subs** you had better consider the equation as a system of one equation in one variable.)
4. How would you double the elements in the list  $[a, b, c]$  into  $[a, a, b, b, c, c]$ ? How can you hereafter transform the list  $[a, a, b, b, c, c]$  into  $[[a, a], [b, b], [c, c]]$ ? Can you apply your methods also to the list of the first ten thousand natural numbers and get the result in reasonable time?
5. Write a procedure called **maketable** that, given a list containing indices and a list containing entries, constructs a corresponding table. For example, **maketable([red,green], [0,1/3])** should create the table associations **red → 0** and **green → 1/3**.
6. Create the  $3 \times 3$  matrix  $M$  defined by
- $$M_{ij} = x^{\gcd(i+1,j+1)} + 1,$$
- and write it as a matrix with factored entries.
7. Create the  $5 \times 5$  matrix  $M$  defined by
- $$M_{ij} = 10^{-(i^2+j^2)},$$
- and use the procedure **fnormal** to normalize small matrix entries into zero.
8. Create the  $7 \times 7$  lower-triangular matrix  $M$  with all nonzero entries equal to one. Convert it into a list of integers using the row major order. How would you do these tasks for a  $100 \times 100$  matrix?
9. Generalize the special indexing function **tridiagonal** from §12.4 such that it can be used for banded matrices of band width greater than 1.
10. Define a special indexing function, say **polynomial**, which restricts entries of such arrays to polynomial type of total degree less than 5.
11. Represent the algebraic number of the form  $a + b\sqrt{2}$ , with  $a$  and  $b$  rational numbers, as an ordered pair.
- (a) Implement this number in Maple as the list structure  $[a,b]$ . Write procedures **ADD**, **MUL**, and **DIV** that add, multiply, and divide two algebraic numbers of the given type, respectively, and that deliver their result in canonical form.
  - (b) Implement the above algebraic number as the function call **algnm(a,b)**. Write a procedure **print/algnm** that prints the number in the common mathematical notation  $a + b\sqrt{2}$ .
  - (c) In the implementation of part (b), rewrite the procedures **ADD**, **MUL**, and **DIV**.

# 13

## The Assume Facility

The **assume** facility allows you to specify properties of variables and expressions. This chapter is an introduction to this facility: the reasons behind it, the model, and the Maple implementation. See also [64] for a leisurely introduction to **assume**, and [9, 229, 230, 231] for more thorough discussions of the model on which the facility is based. Throughout this chapter we shall assume that a variable for which assumptions have been made is distinguished from ordinary variables by a trailing tilde in its name. This can be accomplished by the following interface setting.

```
> interface(showassumed=1):
```

### 13.1 The Need for an Assume Facility

The main motivation for having an **assume** facility in Maple is the following: it allows the incorporation of knowledge about the domain of a variable, hence it allows simplifications to be carried out as far as possible without sacrificing the validity of the transformations. Let us illustrate this with the example of the square root function.

```
> expr := sqrt(x^2);
```

$$expr := \sqrt{x^2}$$

Maple does not simplify this expression automatically to  $x$  or  $\text{abs}(x)$ . Both answers are incorrect in general. **simplify** considers the square root function as a multiple-valued complex function and this has some effect.

```
> simplify(expr);
csgn(x) x
```

The expression is transformed into one that contains the complex sign of  $x$ . When you are more specific on the domain of computation or on the value of  $x$ , then Maple will be more specific as well. This can be done in various ways. You can do the simplification within a real context. The **RealDomain** package provides such an environment in which Maple's default behavior of assuming that the basic underlying number system is the complex field is replaced by the assumption that Maple is working in a 'real context', i.e., a context in which the basic underlying number system is the field of real numbers.

```
> use RealDomain in simplify(expr) end use;
|x|
```

The **RealDomain** package is not compatible with the use of the **assume** facility, which is actually the topic of this chapter. This facility allows you to add assumptions in the **simplify** command, for example that all variables are real.

```
> simplify(expr, assume=real);
|x|
```

You can also carry out the simplification under some conditions by using the **assuming** operator.

```
> simplify(expr) assuming x<0;
-x
```

And last but not least, by making assumptions on individual variables.

```
> assume(x, real):
```

Henceforth Maple will use this assumption in its computations.

```
> simplify(expr);
|x~|
```

Or if you prefer the absolute value function.

```
> convert(%, signum);
signum(x~) x~

> assume(x>0): simplify(expr);
x~

> assume(x<0): simplify(expr);
-x~
```

The tilde appended to the variable  $x$  indicates that an assumption has been made about it.

The following two examples show that Maple can compute results better when it has more information.

```
> integral := integrate(sin(a*x)/x, x=0..infinity);
```

$$\text{integral} := \frac{1}{2} \operatorname{csgn}(a) \pi$$

```
> integral assuming a<0;
```

$$-\frac{\pi}{2}$$

```
> integral := integrate(exp(-s*t), t=0..infinity);
```

**Definite integration:** Can't determine if the integral is convergent.  
Need to know the sign of  $\rightarrow s$   
Will now try indefinite integration and then take limits.

$$\text{integral} := \lim_{t \rightarrow \infty} -\frac{e^{(-s)t} - 1}{s}$$

The partial result means that Maple considers the computation of a limit as an easier problem than integration. If you are more specific about  $s$ , Maple will be able to compute the limit.

```
> integral assuming s>0;
```

$$\frac{1}{s}$$

```
> integral assuming s<=0;
```

$$\infty$$

So, indeed you find with Maple that

$$\int_0^\infty e^{-st} dt = \begin{cases} \frac{1}{s} & \text{if } s > 0 \\ \infty & \text{otherwise} \end{cases}$$

Maple actually does more than just checking the sign of a variable during computations. When, for example, the real variable  $c$  is introduced and  $s$  is assigned the square of  $c$ , then Maple can also figure out which branch of the above integral to choose.

```
> assume(c, real): s := c^2;
```

$$s := c^2$$

Look carefully: it is  $c^2$  raised to the power 2 and not  $c$  raised to the power -2. Another point: Maple forgets by this assignment all previous assumptions about  $s$ .

```
> integral;
```

$$\frac{1}{c^2}$$

The above examples also illustrate that you need to make assumptions when Maple returns:

- an unevaluated **limit** for an integral;
- an answer that has not been simplified as expected or requested;
- an unevaluated **csgn**, **signum**, or **abs** function call;
- an unevaluated **Re** or **Im** function call.

A good piece of advice:

*Specify assumptions as early as possible in a computation.*

In this way you avoid that Maple first solves a generic problem and afterwards specializes under your assumptions to a simpler but erroneous result. An example to support this point of view: in [4] the question is raised whether the following complex function  $V$  in cylindrical coordinates

$$V = \frac{1}{8\pi\epsilon_0} \left( \frac{1}{\sqrt{r^2 + (z - I)^2}} + \frac{1}{\sqrt{r^2 + (z + I)^2}} \right)$$

could be considered as an electrostatic potential of a charge density of physical meaning. The charge density  $\sigma$  in the plane  $z = 0$  is computed by using

$$\sigma = \epsilon_0 \left( \lim_{x \downarrow 0} \frac{\partial V}{\partial z} - \lim_{x \uparrow 0} \frac{\partial V}{\partial z} \right)$$

It is reported that Maple and other computer algebra systems give the wrong result  $\sigma = 0$ . The session below illustrates how Maple can indeed find the correct answer by adding assumptions on  $r$ .

```
> V := 1/(8*Pi*epsilon[0]) * (1/sqrt(r^2+(z-I)^2) +
> 1/sqrt(r^2+(z+I)^2));

$$V := \frac{1}{8} \frac{1}{\sqrt{r^2 + (z - I)^2}} + \frac{1}{\sqrt{r^2 + (z + I)^2}}$$

> E[z] := - diff(V, z);

$$E_z := -\frac{1}{8} \frac{\frac{1}{2} \frac{2 z - 2 I}{(r^2 + (z - I)^2)^{(3/2)}} - \frac{1}{2} \frac{2 z + 2 I}{(r^2 + (z + I)^2)^{(3/2)}}}{\pi \epsilon_0}$$

> sigma := epsilon[0]*(
> limit(E[z], z=0, right) - limit(E[z], z=0, left));

$$\sigma := 0$$

> assume(r>=1); sigma := epsilon[0]*(
> limit(E[z], z=0, right) - limit(E[z], z=0, left));

$$\sigma := 0$$

```

```

> assume(0<r, r<1); sigma := epsilon[0]*(
>     limit(E[z], z=0, right) - limit(E[z], z=0, left));

$$\sigma := -\frac{1}{2\pi(1-r^2)^{(3/2)}}$$

> sigma := subs(r='r', sigma);

$$\sigma := -\frac{1}{2\pi(1-r^2)^{(3/2)}}$$

> r := 1/2: simplify(sigma);

$$-\frac{4\sqrt{3}}{9\pi}$$


```

Why did we first do the substitution and not immediately assign `r` the value 1/2 before evaluation of `sigma`? Let us restore the situation and see what would happen.

```

> r := 'r': assume(0<r, r<1):
> sigma := epsilon[0]*(
>     limit(E[z], z=0, right) - limit(E[z], z=0, left));

$$\sigma := -\frac{1}{2\pi(1-r^2)^{(3/2)}}$$

> r := 1/2: sigma;

$$-\frac{1}{2\pi(1-r^2)^{(3/2)}}$$


```

The answer still contains `r~`? The reason is as follows: when you entered `assume(0<r, r<1)`, you actually assigned the variable `r` to be a new variable `r~` with the property that its range is (0,1). At that point you could refer to the variable `r~` via `r`. Next, you computed the charge density in terms of this new variable. But then you assigned `r` the value 1/2 instead of assigning `r~` this value. The charge density however still uses `r~`. With the assignment `r:=1/2` you also destroyed the easy way of referencing to `r~`. In §3.6 we have explained this side effect of Maple's implementation of `assume` more thoroughly and we have described how to assign a value to a variable that has associated properties. The substitution trick that we used earlier on in this Maple session is the simplest way out of assignment trouble.

The above difficulty of assigning a variable for which assumptions hold is one of the reasons why the Maple developers introduced the **assuming** operator. You can use it to evaluate an expression under the condition that all variables share some property or under assumptions on named variables. The above computation can be done in this working style as follows.

```

> r := 'r': hasassumptions(r);
false

```

```

> sigma := epsilon[0]*(limit(E[z], z=0, right)
> - limit(E[z], z=0, left));

$$\sigma := 0$$

> sigma := epsilon[0]*(limit(E[z], z=0, right)
> - limit(E[z], z=0, left)) assuming 0 < r, r < 1;

$$\sigma := -\frac{1}{2\pi(1-r^2)^{(3/2)}}$$


```

The **assuming** operator identifies the names that are assumed to have special properties, then replaces them with equivalent names that have these properties. The evaluation uses these new names in the computation, then restores the original names before returning the output. This process ensures that no assumptions are made to the original names in the expression. Therefore, the computations invoked by **assuming** have no effect on computations performed before or after calling **assuming**. For example, you can assign a variable without any unexpected problem.

```
> r := 1/2: sigma;
```

$$-\frac{2\sqrt{3}\sqrt{4}}{9\pi}$$

Thus, **assuming** permits, in an easy way, experiments concerning the value of an expression under different assumptions. However, if you must carry out a computation in many steps and assumptions on variables are needed, then it is cumbersome to restate the assumptions via **assuming** a couple of times. Instead you better use once the procedure **assume**.

## 13.2 Basics of **assume**

The **assume** facility contains seven basic commands and one command to extend Maple's knowledge with a new property; they are listed in Table 13.1 below.

Command	Meaning
<b>assume</b>	specify assumption(s)
<b>additionally</b>	specify additional assumption(s)
<b>hasassumptions</b>	check whether assumptions were made
<b>about</b>	display current assumption(s)
<b>is</b>	query a property
<b>coulditbe</b>	determine the possibility of a property
<b>addproperty</b>	add a property to Maple's knowledge base

Table 13.1. Commands of assume facility

With the procedure **assume** you tell Maple properties about variables and relationships between variables. There are basically two formats:

**assume(** < *inequality* > )

and

**assume(** <variable>, <property> ).

When you make assumptions about an expression, thereafter the variables that are involved in the assumption print with an appended tilde to indicate that they carry assumptions. The procedure **about** displays current assumptions about variables and expressions.

An example: you can specify that *r* is a real constant with absolute value less than 1 in various ways.

```
> assume(abs(r)<1): hasassumptions(r);

                                         true

> about(r);

Originally r, renamed r~:
Involved in the following expressions with properties
abs(r) assumed RealRange(-infinity,Open(1))
also used in the following assumed objects
[abs(r)] assumed RealRange(-infinity,Open(1))

> assume(r, RealRange(Open(-1), Open(1))): about(r);

Originally r, renamed r~:
is assumed to be: RealRange(Open(-1),Open(1))

> assume(-1<r, r<1): about(r);

Originally r, renamed r~:
is assumed to be: RealRange(Open(-1),Open(1))
```

The results of **about** are included so that you see that the various assumptions are not always treated in the same way.

The second last command shows that you can combine more than one assumption in one call of **assume**. Note the difference with

```
> assume(-1<r): assume(r<1): about(r);

Originally r, renamed r~:
is assumed to be: RealRange(-infinity,Open(1))
```

Whenever you make an assumption with **assume**, all previous assumptions on variables that are involved in the new assumption are discarded. At least, when the environment variable **\_Envadditionally** has its default value **false**.

```
> _Envadditionally := true:
> assume(-1<r):
> about(r);

Originally r, renamed r~:
is assumed to be: RealRange(Open(-1),Open(1))
```

Instead of setting the value of `_Envadditionally` to `true`, you can also use the procedure `additionally`.

```
> additionally(arcsin(r),
>      RealRange(Open(-Pi/2), Open(Pi/2))):  

>      about(r);  
  
Originally r, renamed r~:  
Involved in the following expressions with properties  
arcsin(r) assumed RealRange(Open(-1/2*Pi),Open(1/2*Pi))  
is assumed to be: RealRange(Open(-1),Open(1))  
also used in the following assumed objects  
[arcsin(r)] assumed RealRange(Open(-1/2*Pi),Open(1/2*Pi))
```

The following library functions know about assumptions that have been made: `csgn`, `signum`, `abs`, `Re`, `Im`, `frac`, `trunc`, `round`, `ceil`, `floor`. For example, with the above assumption about `r`, Maple can compute all by itself

```
> abs(r-1), signum(r^5-1);  
1 - r~, -1
```

You can also ask via the command `coulditbe` whether a property is in conflict with the assumptions already made or not.

```
> coulditbe(r>-3); # no conflict  
true  
> coulditbe(r>1/3); # also no conflict  
true
```

The query on a property from the known properties is done in the functions listed above via the procedure `is`. You can also use this procedure to test properties yourself. Again with the previously set assumptions about `r`

```
> is(r>1/3); # not necessarily true  
false  
> is((1-r^2)/(1+r^2)<=1); # no conflict  
true
```

The result of `is` is either `true`, `false`, or `FAIL`. When `is` returns `FAIL` it means that it could not determine whether the desired property was true or false. This could be caused by not having sufficient information, by not being able to compute the logical derivation, or by not spending enough computing resources to determine whether the property is true, false, or undecidable. When `is` returns `true` it means that all the possible values of the expression have the desired property. When `is` returns `false` it means that at least one possible value of the expression does not have the desired property.

After the discussion of the algebra of properties and its implementation in Maple in the next two sections you will have an idea of the power of Maple

with regards to properties. Here, we only note that the **is** procedure works in two modes, determined by the environment variable `_EnvTry: normal` mode and `hard` mode.

```
> _EnvTry := normal: # the default value
> assume(x>0, y>0, x>=y, x*y<=y^2):
> is(x=y); # successful equality test
                                         true
> assume(x>=y, y>=z, x>0, y>0, z>0, x*y<=y^2, x+y<=2*z):
> is(y=z); # test failure
                                         false
> forget(is): # forget results of "is"
> _EnvTry := hard: # let Maple do its very best
> is(y=z); # successful equality test
                                         true
```

The above example deserves a discussion of two other technical points.

- **assume** does not work in the normal way for indexed names, for this would among other things cause problems in commands where indices are substituted or when indices are equal to running loop indices. Instead, when you make an assumption about an indexed name, then the property is attributed to the corresponding table structure.

```
> assume(indexedname[ idx]>0):
> about(indexedname[ idx]);
```

```
Originally indexedname[ idx], renamed indexedname~[ idx]:
is assumed to be: RealRange(Open(0),infinity)
```

```
> about(indexedname);
```

```
Originally indexedname, renamed indexedname~:
indexedname[ idx] is assumed to be: RealRange(Open(0),infinity)
```

- **is** is a procedure with option `remember`. This means that once the result of a test has been FAIL or whatever value, it will henceforth keep that value, no matter whether we set the value of the environment variable `_EnvTry` to `hard` or not. In order to see the effect of this variable in the above Maple session, we explicitly forgot the results of **is** before questioning the equality  $y = z$  for the second time. You now know why.

The use of the **forget** procedure to avoid problems with earlier computed results works in general for any library function with option `remember`.

*Whenever Maple looks up a previously computed result, whereas you want it to recompute, you can explicitly forget the previous results stored in a remember table via the procedure **forget**.*

### 13.3 An Algebra of Properties

An example of the property problem is

Given :  $m$  and  $n$  are odd integers  
 Query : is  $m^2 + n$  an odd integer?

or more generally

Given :  $obj_1 \in Prop_1, \dots, obj_n \in Prop_n$   
 Query : is  $f(obj_1, \dots, obj_n) \in Prop_0$ ?

Such queries can be handled by manipulating the properties of the objects. Take the above example: given odd integers  $m$  and  $n$ , query “is  $m^2 + n$  odd?”. The problem can be split into two parts:

1. Transform the properties along with the objects: For example, if  $m$  is an odd integer, then  $m^2$  is also an odd integer. So, with the square function is associated a mapping on properties that maps the “odd integer” property into the same property. When we go on to our example  $(m, n) \mapsto m^2 + n$ , where  $m$  and  $n$  are odd integers, we get the mappings

$$(m, n) \rightarrow add \circ (sqr, id)(m, n)$$

$$(\mathcal{P}, \mathcal{P}) \rightarrow \overline{add \circ (sqr, id)}(\mathcal{P}, \mathcal{P})$$

Here,  $add$ ,  $sqr$ , and  $id$  stand for addition, the square function, and the identity function, respectively, and  $\mathcal{P}$  denotes the property “odd integer.”

2. Reduce the query to the problem

$$\overline{add \circ (sqr, id)}(\mathcal{P}, \mathcal{P}) = \overline{add \circ (sqr, id)}(\mathcal{P}, \mathcal{P}) \subseteq \mathcal{P} ?$$

More generally, the two steps are:

1. Transform the properties along with the objects:

$$(obj_1, \dots, obj_n) \rightarrow f(obj_1, \dots, obj_n)$$

$$(Prop_1, \dots, Prop_n) \rightarrow \overline{f}(Prop_1, \dots, Prop_n)$$

2. Reduce the query to the problem

$$\overline{f}(Prop_1, \dots, Prop_n) \sqsubseteq Prop_0 ?$$

where  $\sqsubseteq$  is some relation on properties such that

$$obj \in Prop \Rightarrow obj \in Prop' \text{ iff } Prop \sqsubseteq Prop'.$$

Because we shall consider a property as a set of objects that satisfy that property,  $\sqsubseteq$  will be set inclusion.

Some examples of properties in Maple:

<code>integer</code>	set of integers
<code>odd</code>	set of odd integers
<code>RealRange(0, Open(1))</code>	$[0, 1) = \{x \in \mathbb{R}   0 \leq x < 1\}$
<code>monotonic</code>	set of monotonic real functions
<code>InfinitelyDifferentiable</code>	set of $C^\infty$ functions
<code>Non(singular)</code>	set of nonsingular matrices

Properties form hierarchies under set inclusion. For example,

$$\text{posint} \subset \text{natural} \subset \text{integer}.$$

Properties like `integer`, `fraction`, or constants 0, 1 are basic properties. They can be used to build new properties with the induced lattice operators  $\neg$ ,  $\wedge$ ,  $\vee$ :

- $\neg$  “not”, as in “not zero” (in Maple denoted as `Non(0)`).
- $\wedge$  “and”, as in “integer and greater than 2” (in Maple denoted as `AndProp(integer, RealRange(2, infinity))` ).
- $\vee$  “or”, as in “unary or binary” (in Maple denoted as `OrProp(unary, binary)` ).

Furthermore, the top element  $\top$  is introduced to represent the property “existent”, i.e., there is no restriction on the object. In Maple, this property is called `TopProp`. Symmetrically, the lattice structure is completed with the bottom element  $\perp$  that represents the property “nonexisting”, i.e., an object with this property does not exist. In Maple, the bottom element is called `BottomProp`.

Queries on composed functions are handled in the following setting. With a given object function  $f$  corresponds a property function  $\bar{f}$ . For example, the addition operator  $+$  induces the property operator  $\bar{+}$  for which holds

$$\begin{aligned}\bar{+}(\text{odd}, \text{odd}) &= \text{even} \\ \bar{+}(\text{odd}, \text{even}) &= \text{odd} \\ \bar{+}(\text{even}, \text{odd}) &= \text{odd} \\ \bar{+}(\text{even}, \text{even}) &= \text{even} \\ \bar{+}(\text{prime}, \text{composite}) &= \text{integer} \\ \bar{+}(\text{fraction}, \text{fraction}) &= \text{rational} \\ \bar{+}(\text{irrational}, \text{irrational}) &= \text{real}\end{aligned}$$

And so on. Of course  $\bar{f}$  need not be defined for all properties. We add some basic property functions such as  $\bar{+}$ ,  $\bar{*}$ ,  $\bar{\ln}$ ,  $\bar{\exp}$  and their inverses to our model and arrive at an algebra of properties that satisfies the classical lattice axioms plus some extra axioms that allow an effective calculus on properties. In fact, properties have the structure of a boolean algebra.

## 13.4 Implementation of **assume**

Currently acceptable properties in Maple are

- (a) *a property name*. They are categorized as follows:
  - alias e.g., `realcons` instead of `AndProp(real, constant)` and `nonneg` instead of `RealRange(0, infinity)`.
  - numerical e.g., `imaginary` and `prime`.
  - matricial e.g., `SquareMatrix` and `PositiveDefinite`.
  - functional e.g., `unary` and `OddMap`.
  - other e.g., `TopProp` and `MutuallyExclusive`.
- (b) *most types*. e.g., `integer`, `fraction`, and `rational`. This also includes constants such as 0 and 1.
- (c) *a numerical range*. e.g., `RealRange(-infinity, Open(0))`.
- (d) *an “and” of properties*. e.g., `AndProp(integer, positive)`.
- (e) *an “or” of properties*. e.g., `OrProp(positive, negative)`.
- (f) *a property range*. e.g., `PropRange(prime, integer)`. If `P = PropRange(Prop1, Prop2)` then all possible objects in `Prop1` have property `P`, and all possible objects in `P` have property `Prop2`.
- (g) *a linear property*. e.g., `LinearProp(3, integer, 0)` represents the set of integers divisible by 3.
- (h) *a parametric property*. e.g., `Non(0)` and `Non(singular)`.

An exhaustive list of properties that are known to Maple can be obtained via the on-line help system: enter the command `?property`. The help page reflects the principle that the basic properties (of class (a) and (b) above) are arranged in a directed acyclic graph. Figures 13.1, 13.2, and 13.3 at the end of this chapter show the graph of the predefined numerical, matricial, and functional properties, respectively.

The specification of a property includes all parents and children in the ``property/ParentTable`` and ``property/ChildTable``, respectively. You can ask about the place of a basic property in the directed acyclic graph with `about`. For example,

```
> about(integer);

integer:
  a known property having {rational, GaussianInteger} as
  immediate parents and {1, composite, prime} as immediate
  children. mutually exclusive with {fraction, irrational}
```

The procedure `addproperty` can be used to install a new property in the hierarchy of properties. For example,

```
> alias( nonnegative = RealRange(0,infinity),
>       nonnegbutlessthan1 = RealRange(0,Open(1)) );
> addproperty(nonnegbutlessthan1, nonnegative, 0);
```

The lattice of numerical properties is updated.

```
> 'property/ParentTable'[0];
{composite, nonnegbutlessthan1}
> 'property/ParentTable'[nonnegbutlessthan1];
{nonnegative}
> 'property/ChildTable'[nonnegative];
{nonnegbutlessthan1, RealRange(Open(0), infinity)}
> is(1/2, nonnegbutlessthan1);
true
> is(Pi, nonnegbutlessthan1);
false
```

Properties themselves are Maple objects that can satisfy other properties. Their hierarchy is shown in Figure 13.4 at the end of this chapter. So, you can ask questions like the following.

```
> is(prime, property);
true
```

For an object function  $f$  the corresponding property function  $\overline{f}$  is implemented as the procedure `property/f'. A lot of the information on the property function is stored in the remember table of the procedure. Let us use the exponential mapping as our example. By setting the interface variable **verboseproc** the value 3, Maple will also print the remember table of a procedure.

```
> interface(verboseproc=3): print('property/exp');

proc(a)
local b, c;
option remember, 'Copyright (c) 1992 Gaston Gonnet, Wissen\
schaftliches Rechnen, ETH Zurich. All rights reserved.'
if a::RealRange then
    if op(1, a)::{0, Open(0), identical( $-\infty$ )} and
        op(2, a)::{0, identical( $\infty$ ), Open(0)} then
            RealRange(procname(op(1, a)), procname(op(2, a)))
        else
            b := procname(op(1, a)); c := procname(op(2, a));
            RealRange(b, c)
        end if
```

```

elif a::EvalfableProp then
  PropRange(BottomProp, RealRange('property/Shake‘(exp(a))))
elif a::Open(EvalfableProp) then PropRange(BottomProp,
  RealRange('property/Shake‘(exp(op(1, a)))))
else error FAIL
end if
end proc
# (0) = 1
# (-∞) = Open(0)
# (real) = RealRange(Open(0), ∞)
# (complex) = complex
# (∞) = ∞
# (Open(0)) = Open(1)

```

The property function is used to compute with properties, but there is more. For example, what conclusion can Maple make for the exponential mapping when applied to an integer? Maple works as follows. First, it notices that the property function  $\overline{\exp}$  does not give any information. So, Maple decides to investigate the direct ancestors of the property **integer**. But alas, the property function is not defined for **GaussianInteger** and **rational** either. So, the system continues looking at the grandparents, and with success.

$$\begin{aligned}\overline{\exp}(\text{complex}) &= \text{complex} \\ \overline{\exp}(\text{real}) &= \text{RealRange}(\text{Open}(0), \text{infinity})\end{aligned}$$

Maple comes to the conclusion that

$$\begin{aligned}\overline{\exp}(\text{integer}) &\subseteq \text{AndProp}(\overline{\exp}(\text{complex}), \overline{\exp}(\text{real})) \\ &= \text{AndProp}(\text{complex}, \text{RealRange}(\text{Open}(0), \text{infinity})) \\ &= \text{RealRange}(\text{Open}(0), \text{infinity})\end{aligned}$$

By inspection of children and grandchildren Maple finds

$$\begin{aligned}\overline{\exp}(\text{integer}) &\supseteq \text{OrProp}(\overline{\exp}(0), \overline{\exp}(1)) \\ &= \text{OrProp}(1, \text{PropRange}(\text{BottomProp}, \\ &\quad \text{RealRange}(2.71828182574, 2.71828183118))) \\ &\supseteq \{1\}\end{aligned}$$

As far as Maple is concerned, it knows for sure that

$$\{1\} \subseteq \overline{\exp}(\text{integer}) \subseteq (0, \infty).$$

The main functions in the assume facility are **assume** and **is**. With the first function you provide Maple with information about an object, i.e., you declare the properties of an object. For example,

```
> assume(x>0);
```

silently assigns a local variable  $x^{\sim}$  to  $x$  and generates or updates the entry `'property/object'[x~]`. Let us inspect this table entry and see what happens to it when we add another assumption.

```
> 'property/object'[x];
RealRange(Open(0), infinity)
> additionally(x, integer):
> 'property/object'[x];
AndProp(integer, RealRange(1, infinity))
```

So, all properties about an object are stored in the corresponding entry in the table `'property/object'`.

To retrieve information about objects you use the procedure **is**. It applies the rules of the algebra of properties as described in the previous section to verify a property of an expression. For example, with the above assumptions about  $x$ , Maple can conclude that  $e^x > 1$ .

```
> is(exp(x)>1);
true
```

This is done in four steps:

1. Reformulate the query in terms of properties, i.e., `is(exp(x~), RealRange(Open(1), infinity))`.
2. Express the object in terms of the known objects you start with, i.e., `object = exp(x~)`.
3. Look up in the `'property/object'` table which properties about  $x^{\sim}$  exist and compute `exp('property/object'[x~])`.
4. Determine whether `exp('property/object'[x~])` is included in the property `RealRange(Open(1), infinity)`. The answer is trivially “yes” in this case.

All queries with the procedure **is** are treated in this way:

1. Reformulate the query as `is(object, Property)`.
2. Express the object in terms of the known objects:  
`object = f(obj_1, obj_2, ..., obj_n)`.
3. Look up in the `'property/object'` table which properties about `obj_1, obj_2, ..., obj_n` exist and compute  
`f(Property_1, Property_2, ..., Property_n)`.
4. Determine whether `f(Property_1, Property_2, ..., Property_n)` is included in `Property`.

We end this section with a peculiar consequence of the Maple implementation of the **assume** facility. Since the assumptions about an object are not associated (e.g., in the form of an attribute) to the object itself but are stored in the `property/object` table, all information is lost when you save an object in a file, leave Maple, restart the system, and finally restore the object from the file. The following Maple sessions illustrate how careful you must be with storing and retrieving variables with assumptions.

```

> assume(x>0):
> 'property/object'[x]; # check the property about x
                                RealRange(Open(0), infinity)
> save x, "foo":
> restart;
> 'property/object'[x]; # no assumptions about x
                                property/object_x
> read "foo";
                                x := x
> 'property/object'[x]; # still no assumptions about x!
                                property/object_x

```

Apparently the assumption about  $x$  was not stored in the file.

```

> assume(x>0):
> 'property/object'[x];
                                RealRange(Open(0), infinity)

```

Now, we also store the property table and restart.

```

> save x, 'property/object', "foo":
> restart:
> 'property/object'[x]; # no assumptions about x
                                property/object_x
> read "foo": # read the file
> 'property/object'[x]; # assumptions are also read!
                                RealRange(Open(0), infinity)

```

All is well at this point.

A warning: when you store all variables into a file in internal Maple format, be sure that all information about variables is stored. Otherwise you run into difficulties as the following session illustrate this.

```

> restart;
> assume(x>0):
> 'property/object'[x]; # check the property of x
                                RealRange(Open(0), infinity)
> about(x);

```

```
Originally x, renamed x~:
  is assumed to be: RealRange(Open(0),infinity)
> save x, "foo.m": # store in internal format
> restart;
> 'property/object'[x]; # check the property of x
                                property/objectx
> read "foo.m"; # read the file
> 'property/object'[x];
                                property/objectx~
```

Note the trailing tilde in the index! Although nothing is known about **x**, it will become clear below that it has the value  $x^{\sim}$ .

```
> about(x);
x:
  nothing known about this object
> hasassumptions(x);
```

*false*

```
> x; # value of x
      x~
```

As a matter of fact, the assignment  $x := x^{\sim}$  has been stored in the file in internal Maple format. Once again, all goes well when you store all information about variables.

```
> restart;
> assume(x>0):
> 'property/object'[x];
                                RealRange(Open(0), infinity)
> save x, 'property/object', "foo.m":
> restart;
> 'property/object'[x];
                                property/objectx
> read "foo":
> 'property/object'[x];
                                RealRange(Open(0), infinity)
> about(x);
x:
  is assumed to be: RealRange(Open(0),infinity)
```

## 13.5 Exercises

1. Compute  $\cos(n\pi)$  and  $\sin(n\frac{\pi}{2})$  under the assumptions that  $n$  is
  - an integer.
  - an odd integer.
  - an odd and positive integer less than 3.
2. Add the interval  $(-1,1)$  to the hierarchy of numerical properties.
3. A natural number is called perfect when it is one more than the sum of its nontrivial divisors (e.g.,  $6 = 2 + 3 + 1$  and  $28 = 2 + 4 + 7 + 14 + 1$  are perfect). Add the perfect numbers to the hierarchy of numerical properties.
4. An  $n \times n$  matrix  $A$  is called skewdiagonal iff  $A_{ij} = 0$  unless  $i + j = n + 1$ . Add the property `skewdiagonal` to Maple's knowledge base.

## 13.6 Hierarchy of Properties

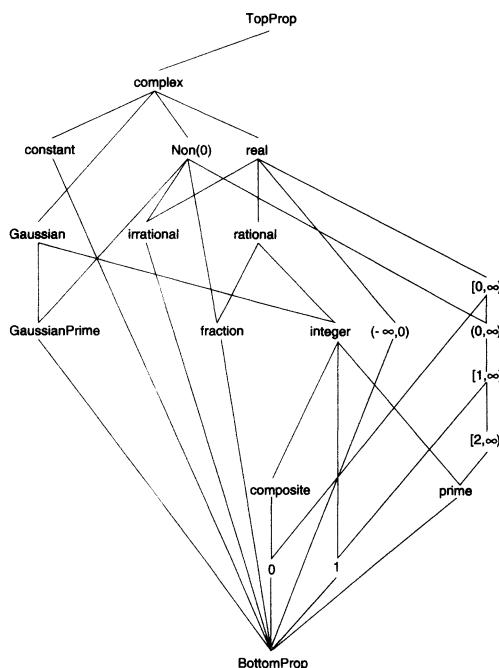


Figure 13.1. Hierarchy of numerical properties.

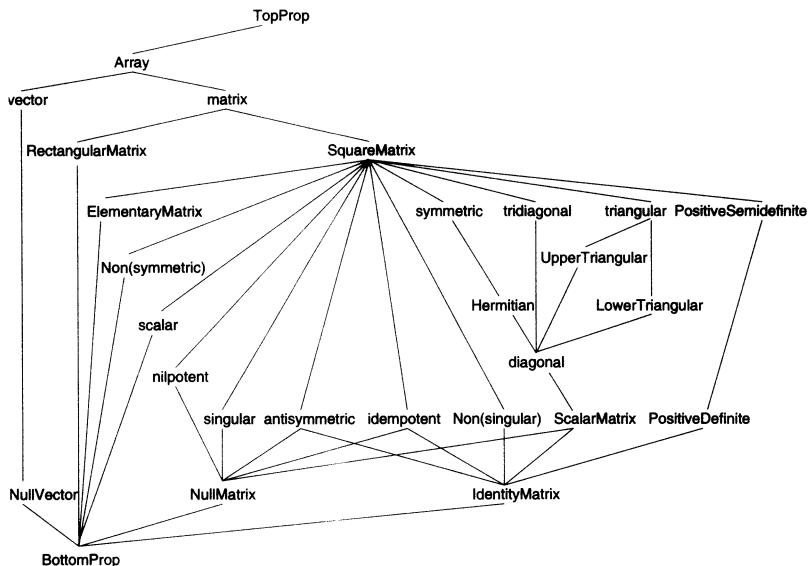


Figure 13.2. Hierarchy of properties on matrices.

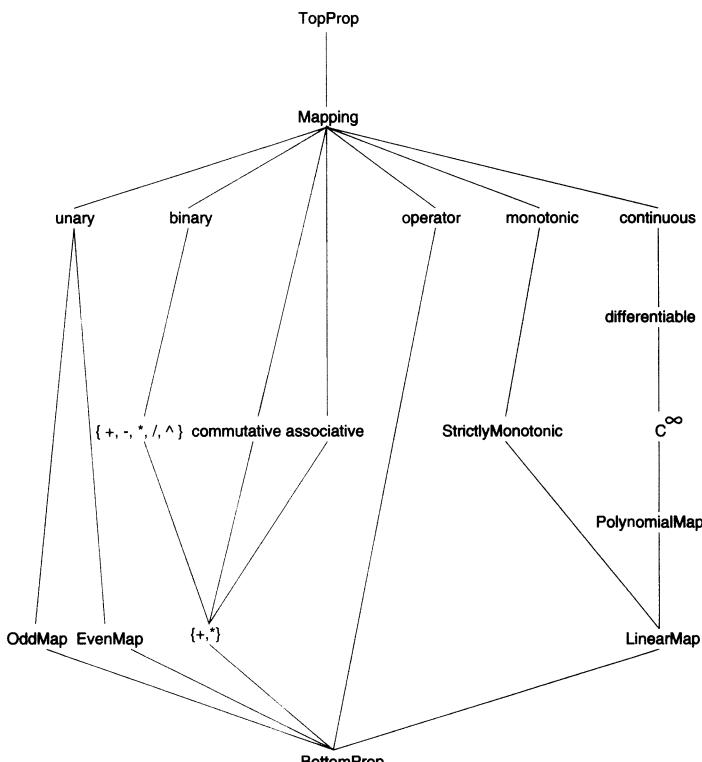


Figure 13.3. Hierarchy of functional properties.

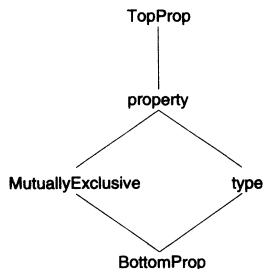


Figure 13.4. Hierarchy of miscellaneous properties.

# 14

## Simplification

In Chapter 7 we discussed manipulation of polynomials and rational functions. Manipulations such as normalization, collection, sorting, factorization, and expansion were discussed. Characteristic of these manipulations is that they are carried out on expressions as a whole. However, in computations that involve mathematical functions, you frequently want to apply simplification rules that are known for these mathematical functions. For example, in computations that involve trigonometric functions you often want to apply the equality  $\sin^2 + \cos^2 = 1$ . This chapter describes how simplifications of expressions containing mathematical functions can be performed, how valid the simplifications offered by Maple are, how simplification can be controlled, and how you can define your own simplification routines or overrule existing Maple routines.

After a short introduction to automatic simplification, the procedures **expand**, **combine**, **simplify**, and **convert** will be described. One of Maple's specialities is that the system itself finds out which mathematical functions are involved and which transformation rules are available. This is another example of Maple's *hybrid algorithmic structure*.

For each simplification procedure we shall look at its effect on trigonometric functions, exponential function and logarithms, powers, radical expressions, and other functions, respectively. In three separate sections, extra attention will be paid to trigonometric simplification, to simplification of expressions with respect to side relations, and to control of the simplification process. We shall also show how to define your own simplification routines and how to overrule Maple's simplification. At the end of this chapter an overview of possible simplifications is given in tabular form.

## 14.1 Automatic Simplification

Let us first look at some automatic simplifications in Maple.

```
> arcsin(1/2), Zeta(2), GAMMA(1/2), Psi(1/2);

$$\frac{\pi}{6}, \frac{\pi^2}{6}, \sqrt{\pi}, -\gamma - 2 \ln(2)$$

> tan(arctan(x)), arctan(tan(x));

$$x, \arctan(\tan(x))$$

> sqrt(Pi^2*x^2);

$$\pi \sqrt{x^2}$$

> 'abs(abs(x))' = abs(abs(x));

$$||x|| = |x|$$

> 'abs(-Pi*x)' = abs(-Pi*x);

$$|-\pi x| = \pi |x|$$

> 'sin(-x)' = sin(-x), 'cos(Pi/2+x)' = cos(Pi/2+x);

$$\sin(-x) = -\sin(x), \cos(\frac{1}{2}\pi + x) = -\sin(x)$$

> 'exp(3*ln(x))' = exp(3*ln(x));

$$e^{(3 \ln(x))} = x^3$$

> 'ln(exp(Pi*I))' = ln(exp(Pi*I)),
> 'ln(exp(3*Pi*I))' = ln(exp(3*Pi*I));

$$\ln(e^{(\pi I)}) = \pi I, \ln(e^{(3 I \pi)}) = \pi I$$

> ln(exp(x)), exp(ln(x));

$$\ln(e^x), x$$

```

Adding assumptions or using the `RealDomain` package may bring you further.

```
> ln(exp(x)) assuming x::real;

$$x$$

> use RealDomain in ln(exp(x)) end use;

$$x$$

```

In §1.3 you have already been warned that not all automatic simplifications are valid in full generality; sometimes they only work for the generic case. For example, the automatic simplifications  $0 \cdot x \rightarrow 0$ ,  $(x - x) \rightarrow 0$ , and  $x/x \rightarrow 1$  are desirable (if only for efficiency reasons) but incorrect when  $x$  is undefined, a random number, or infinity. They have the following side effects in Maple.

```
> limit(exp(1/x^2), x=0);
```

```

 $\infty$ 
> % * 0; # automatic simplification to 0
0
> infinity * 0;
undefined
> rand() - rand(); # automatic simplification to 0
0

```

Compare the last result with the following.

```

> rand(): rand(): %-%%;
-106308975811

```

The automatic simplification of  $0^k \rightarrow 0$ , for all  $k$ , has the following side effect in Maple.

```

> sum(a[k]*x^k, k=0..n);

$$\sum_{k=0}^n a_k x^k$$

> eval(% , x=0);
0

```

The automatic simplification of  $\exp(\ln(x)) \rightarrow x$  is incorrect for  $x \neq 0$  but nevertheless carried out in Maple without scruples. This special case is simply ignored in favor of the more often applicable generic case.

The above examples show a trade-off between usability and efficiency on the one hand and mathematical correctness on the other hand. Be aware that Maple like most, if not all, computer algebra systems performs automatic simplifications that are not 100% safe.

Automatic simplifications of multiple-valued functions represented by principal values are another source of problems for many systems. The developers have put much effort in correct handling of complex multiple-valued functions in Maple. Automatic simplifications are limited to those that are provable correct (with the above exception of  $\exp(\ln(x)) \rightarrow x$ ). For example,

```

> sqrt(Pi^2*x^2);
 $\pi \sqrt{x^2}$ 

```

is not further simplified to

```

> sqrt(Pi^2*x^2, symbolic);
 $\pi x$ 

```

For this, you have to make further assumptions or, as we did, use the keyword **symbolic** to do whatever simplification without justification. Make sure that you have not assigned **symbolic** some value, because putting this keyword between quotes will not prevent malfunction.

Another example:

```
> ln(exp(x)); # no automatic simplification
          ln( $e^x$ )
> simplify(%); # still no simplification
          ln( $e^x$ )
> simplify(%) assuming real; # enforced simplification
          x
> use RealDomain in simplify(%%) end use;
          x
```

It is also possible to make assumptions about individual variables.

```
> assume(x, real):
> ln(exp(x)); # simplification under assumptions
          x~
```

But you must keep alert and understand that simplification of multi-valued functions is still not optimal. For example, the simplification  $\exp(\ln(x)) \rightarrow x$  is actually correct when the argument of the complex number  $x$  is between  $-\pi$  and  $\pi$  (inclusive). But Maple does not use this fact.

```
> assume(-Pi < Im(X), Im(X) <= Pi): simplify(ln(exp(X)));
          ln( $e^{X\wedge}$ )
```

If you are still not convinced that caution in the use of mathematical software is needed in case of multi-valued functions, you should look at [73, 141, 215], where similar surprising results are described.

## 14.2 **expand**

In general, the procedure **expand** does credit to its name and works out an expression. For mathematical functions, this often means that sum rules are applied.

- **Trigonometric and hyperbolic functions**

```
> cos(2*x): % = expand(%);
          cos(2 x) = 2 cos(x)2 - 1
> cos(5*x): % = expand(%);
```

```

 $\cos(5x) = 16\cos(x)^5 - 20\cos(x)^3 + 5\cos(x)$ 
> cosh(5*x): % = expand(%);
 $\cosh(5x) = 16\cosh(x)^5 - 20\cosh(x)^3 + 5\cosh(x)$ 
> tan(2*x): % = expand(%);
 $\tan(2x) = \frac{2\tan(x)}{1 - \tan(x)^2}$ 
> tanh(2*x): % = expand(%);
 $\tanh(2x) = \frac{2\sinh(x)\cosh(x)}{2\cosh(x)^2 - 1}$ 
> cos(x+y): % = expand(%);
 $\cos(x + y) = \cos(x)\cos(y) - \sin(x)\sin(y)$ 
> cos(x+2*y): % = expand(%);
 $\cos(x + 2y) = 2\cos(x)\cos(y)^2 - \cos(x) - 2\sin(x)\sin(y)\cos(y)$ 
> cos(x*(y+z)): % = expand(%);
 $\cos(x(y + z)) = \cos(xy)\cos(xz) - \sin(xy)\sin(xz)$ 

```

Two aspects catch the eye in the above example:

(1) The procedure **expand** works out a trigonometric or hyperbolic function over a sum or a multiple. The above examples illustrate again that expansion of trigonometric functions is done as far as possible. A partial expansion like  $\cos 5x \rightarrow \cos x \cos 4x - \sin x \sin 4x$  cannot be done so easily with **expand**. You will have to make do with tricks like

```

> cos(5*x);
 $\cos(5x)$ 
> subs(5*x=x+y, %);
 $\cos(x + y)$ 
> expand(%);
 $\cos(x)\cos(y) - \sin(x)\sin(y)$ 
> subs(y=4*x, %);
 $\cos(4x)\cos(x) - \sin(x)\sin(4x)$ 

```

Or you must, on the basis of the following source code of **expand/cos**, reprogram the expansion procedure such that the recursive step is taken out of the code.

```

> interface(verboseproc=3): # make Maple more verbose
> print('expand/cos'); # print source code

```

```

proc(x)
local n, y;
option'Copyright (c) 1991 by the University of Waterloo.';
y := expand(x);
if type(y, '+') then
    n := op(1, y); y := y - n; expand(cos(n) * cos(y) - sin(n) * sin(y))
    .....
end proc

```

We omitted most of the code because we only want to change the last statement shown.

```
> expand(cos(5*x)); # an example
```

$$16 \cos(x)^5 - 20 \cos(x)^3 + 5 \cos(x)$$

The new code could be as follows:

```

> 'expand/cos' := proc(x)
>   local n,y;
>   y := expand(x);
>   if type(y, '+') then
>     n := op(1,y);
>     y := y-n;
>     cos(n)*cos(y)-sin(n)*sin(y)
>   elif type(y, '*') then
>     n := op(1,y);
>     if type(n, numeric) and n < 0 then expand(cos(-y))
>     elif type(n, integer) and 0 < n and n < 100 then
>       y := y/n;
>       2*cos((n-1)*y)*cos(y)-cos((n-2)*y)
>     else cos(y)
>     end if
>   else cos(y)
>   end if
> end proc:

```

It will not work immediately as **expand** remembers its previous results, and we already expanded  $\cos(5x)$  as  $16 \cos^5 x - 20 \cos^3 x + 5 \cos x$ . Use **forget** with the option **reinitialize=false** to clear the remember table and to suppress its reinitialization.

```

> expand(cos(5*x));

$$16 \cos(x)^5 - 20 \cos(x)^3 + 5 \cos(x)$$

> forget(expand, reinitialize=false):
> expand(cos(5*x));

$$2 \cos(4x) \cos(x) - \cos(3x)$$


```

By the way, the remember option of **expand** explains why in the next step in the recursion

```
> expand(%);
```

$$16 \cos(x)^5 - 16 \cos(x)^3 + 3 \cos(x) - 2 \cos(2x) \cos(x)$$

no terms  $\cos(3x)$  occur anymore. In §14.9 we shall discuss how you can best overrule existing Maple routines.

(2) A tangent (and also a cotangent) is expanded into a rational expression consisting of tangents, but the hyperbolic tangent (and hyperbolic cotangent) is expanded in terms of hyperbolic sines and cosines.

```
> tan(x+y): % = expand(%);
```

$$\tan(x+y) = \frac{\tan(y) + \tan(x)}{1 - \tan(y) \tan(x)}$$

```
> tanh(x+y): % = expand(%);
```

$$\tanh(x+y) = \frac{\sinh(x) \cosh(y) + \cosh(x) \sinh(y)}{\cosh(x) \cosh(y) + \sinh(x) \sinh(y)}$$

In §14.9 we shall show how you can redefine the expansion routine of the hyperbolic tangent so that consistency with the corresponding trigonometric function is established.

### • **exp, ln**

```
> exp(x+y): % = expand(%);
```

$$e^{(x+y)} = e^x e^y$$

```
> ln(2*y): % = expand(%);
```

$$\ln(2y) = \ln(2) + \ln(y)$$

```
> ln(x*y): % = expand(%);
```

$$\ln(xy) = \ln(x) + \ln(y)$$

Maple did not do a simplification in the last command because the transformation  $\ln(xy) \rightarrow \ln x + \ln y$  is in general not valid (see what happens when you substitute  $x = y = -1$ ). With the procedure **simplify** and the keyword **symbolic** it will do simplification, but the responsibility for whether the transformation is valid or not is shifted to the user. In other words, Maple is always willing to do the transformation when forced to.

```
> simplify(ln(x*y), symbolic);
```

$$\ln(x) + \ln(y)$$

The expansion  $\ln(xy) \rightarrow \ln x + \ln y$  is only valid if for these complex numbers holds the equality  $\arg(xy) = \arg(x) + \arg(y)$ . In case of real, positive numbers this condition is certainly satisfied.

```
> ln(x*y): % = expand(%) assuming positive;
```

$$\ln(xy) = \ln(x) + \ln(y)$$

The expansion  $\ln(x^y) \rightarrow y \ln x$  is only valid if  $\pi < \Im(y \ln x) \leq \pi$ . Thus

```
> ln(x^y): % = expand(%);
          ln(xy) = ln(xy)
> simplify(ln(xy), symbolic);
          y ln(x)
```

Another example:  $\ln(x/y) \rightarrow \ln x - \ln y$ .

```
> ln(x/2): % = expand(%);
          ln(1/2 x) = -ln(2) + ln(x)
> ln(x/y): % = expand(%);
          ln(x/y) = ln(x) - ln(y)
> simplify(ln(x/y), symbolic);
          ln(x) - ln(y)
> ln(x/y): % = expand(%) assuming positive;
          ln(x/y) = ln(x) - ln(y)
```

### • Powers and radicals

```
> x^(y+z): % = expand(%);
          x(y+z) = xy xz
> (x^(1/2))^(y/2): % = expand(%);
          (sqrt(x))(y/2) = x(y/4)
```

The expansion  $(xy)^z \rightarrow x^z y^z$  is only valid if  $z$  is an integer or if  $\arg(xy) = \arg(x) + \arg(y)$ . This explains the following.

```
> (x*y)^z: % = expand(%);
          (x y)z = (x y)z
> simplify((x*y)^z, symbolic);
          xz yz
> (x*y)^z: % = expand(%);
          (x y)z = xz yz
> (-x)^y: % = expand(%);
          (-x)y = (-x)y
> (-x)^y: % = expand(%) assuming x>0;
          (-x)y = xy (-1)y
```

```

> (x/y)^z: % = expand(%);

$$\left(\frac{x}{y}\right)^z = \left(\frac{x}{y}\right)^z$$

> (x/y)^z: % = expand(%) assuming x>0;

$$\left(\frac{x}{y}\right)^z = x^z \left(\frac{1}{y}\right)^z$$

> x^(y/3): % = expand(%);

$$x^{(\frac{y}{3})} = x^{(\frac{y}{3})}$$

> simplify(x^(y/3), symbolic);

$$x^{(\frac{y}{3})}$$


```

As you see in the previous result, the keyword **symbolic** is not always the magic word to get things done. Here you have to help Maple more to get the transformation  $x^{(y/3)} \rightarrow (x^y)^{1/3}$  be applied.

```

> %^3;

$$(x^{(\frac{y}{3})})^3$$

> simplify(% , symbolic);

$$x^y$$

> %^(1/3);

$$(x^y)^{(\frac{1}{3})}$$


```

We repeat: not all transformations done by **expand** in request are 100% safe. For example, the transformation  $(xy)^z \rightarrow x^z y^z$  is not valid in general; substitution of  $x = y = -1, z = 1/2$  is a convincing example.

```

> (x*y)^z: % = simplify(% , symbolic);

$$(xy)^z = x^z y^z$$

> eval(%, {x=-1, y=-1, z=1/2});

$$1 = -1$$


```

It is the responsibility of the user to verify whether the transformation is valid or not in a particular case. He or she should pay extra attention when a symbolic computation with simplification is carried out and afterwards concrete values are substituted in the result; all transformations carried out during the computation may not be necessarily valid for the specified values. Such mistakes occur quite often and the phenomenon is known as the *problem of specialization*.

### • Other expansions

Other simplifications with **expand** are expansion of factorized natural numbers, expansion of factorials, binomial coefficients, the Gamma function, and other special functions. Some examples:

```

> (n+1)!: % = expand(%);

$$(n + 1)! = n! (n + 1)$$

> binomial(n+1,k+1): % = expand(%);

$$\text{binomial}(n + 1, k + 1) = \frac{(n + 1) \text{binomial}(n, k)}{k + 1}$$

> binomial(n-1,k-1) + binomial(n-1,k);

$$\text{binomial}(n - 1, k - 1) + \text{binomial}(n - 1, k)$$

> expand(%);

$$\frac{k \text{binomial}(n, k)}{n} + \frac{(n - k) \text{binomial}(n, k)}{n}$$

> normal(%);

$$\text{binomial}(n, k)$$

> ifactor(123456789): % = expand(%);

$$(3)^2 (3803) (3607) = 123456789$$

> BesselJ(5,t): % = expand(%);

$$\text{BesselJ}(5, t) = \frac{384 \text{BesselJ}(1, t)}{t^4} - \frac{192 \text{BesselJ}(0, t)}{t^3}$$


$$- \frac{72 \text{BesselJ}(1, t)}{t^2} + \frac{12 \text{BesselJ}(0, t)}{t} + \text{BesselJ}(1, t)$$

> collect(%, BesselJ); # group terms

$$\text{BesselJ}(5, t) =$$


$$\left(\frac{384}{t^4} - \frac{72}{t^2} + 1\right) \text{BesselJ}(1, t) + \left(-\frac{192}{t^3} + \frac{12}{t}\right) \text{BesselJ}(0, t)$$

> Zeta(50): % = expand(%);

$$\zeta(50) = \frac{39604576419286371856998202}{285258771457546764463363635252374414183254365234375} \pi^{50}$$

> dilog(1/x): % = expand(%);

$$\text{dilog}\left(\frac{1}{x}\right) = -\text{dilog}(x) - \frac{1}{2} \ln(x)^2$$


```

What remains to be told is how you can prevent expansions of particular mathematical functions in an expression. This is important because more than one mathematical function may be part of a symbolic expression, but you may not want all functions expanded. For example, suppose that you only want to expand the exponential function in the expression  $\sin(x + y) + \exp(x + y)$ . When you apply **expand** with the expression as the sole parameter, the effect will be that the expression is fully expanded.

Only when you add the keyword **sin** as an extra argument in the function call, this trigonometric function is left intact.

```
> expression := sin(x+y) + exp(x+y);
expression := sin(x + y) + e^(x+y)
> expand(expression);
sin(x) cos(y) + cos(x) sin(y) + e^x e^y
> expand(expression, sin);
sin(x + y) + e^x e^y
```

It is the same mechanism described in §7.1: the extra arguments in **expand** specify what should be left untouched.

```
> expand(sin(x^sin(y+z)+w), x^sin(y+z));
sin(x^sin(y+z)) cos(w) + cos(x^sin(y+z)) sin(w)
> expand((cos(2*x)+sin(2*y))^2, cos, sin);
cos(2 x)^2 + 2 cos(2 x) sin(2 y) + sin(2 y)^2
```

If you want to avoid expansion of *all* mathematical functions, then you can use the Maple procedure **frontend**.

```
> frontend(expand, [expression^2]);
sin(x + y)^2 + 2 sin(x + y) e^(x+y) + (e^(x+y))^2
```

When you know beforehand that you will not need expansion of certain mathematical functions for some time, then you can inform Maple with the procedure **expandoff**.

```
> expand(expandoff()): # enable library function
> expandoff(sin): # turn off expansion of sin
> expression := sin(p+q) + exp(p+q);
expression := sin(p + q) + e^(p+q)
> expand(expression);
sin(p + q) + e^p e^q
```

Perhaps you (still) wonder why we have rewritten the previous expression in **x** and **y** in new unknowns **p** and **q**. Well, if we had not done this, Maple would have remembered that the command **expand(expression)** had been entered before, and consequently would have picked up the previous result from the remember table of **expand** instead of recomputing it. See for yourself, using the opposite **expandon** of **expandoff**.

```
> expandon(sin): # turn on expansion of sin
> expand(expression);
sin(p + q) + e^p e^q
```

No success! But, as discussed before in this section, we could have used the library function **forget** to clear the remember table of **expand**.

```
> forget(expand):
> expand(expression);
 $\sin(p) \cos(q) + \cos(p) \sin(q) + e^p e^q$ 
```

In §14.8 we shall investigate in more detail how to control the simplification process.

## 14.3 combine

When a computer algebra system provides facilities to expand expressions you may expect the existence of a procedure to do the opposite, i.e., to combine expressions. Be aware of the fact that Maple often leaves it up to the user to verify validity of a combination.

In the example below we shall often add a keyword like **radical**, **trig**, **ln** to the procedure **combine** when it is not really required to get the simplification done. The main reason is that we want to show how you can be specific about which combination of expressions should be done and which combination should be left untouched.

```
> sqrt(x)*sqrt(y); % = combine(%);
 $\sqrt{x} \sqrt{y} = \sqrt{x} \sqrt{y}$ 
```

We can force Maple to do the simplification by adding the keyword **symbolic**.

```
> sqrt(x)*sqrt(y); % = combine(% , symbolic);
 $\sqrt{x} \sqrt{y} = \sqrt{xy}$ 
```

The reason that Maple did not carry out the combination of square roots is because the equality does not hold for all values of  $x$  and  $y$ .

```
> sqrt(x)*sqrt(y); % = combine(%) assuming positive;
 $\sqrt{x} \sqrt{y} = \sqrt{xy}$ 
> eval(%, {x=-1,y=-1});
 $-1 = 1$ 
```

Another example: combination of divergent sums.

```
> Sum(1/(2*k), k=1..infinity) -
> Sum(1/(2*k-1), k=1..infinity);


$$\left( \sum_{k=1}^{\infty} \frac{1}{2k} \right) - \left( \sum_{k=1}^{\infty} \frac{1}{2k-1} \right)$$

> value(%); # correct answer
```

*undefined*

```
> combine(%); # combine sums without checking conditions
```

$$\sum_{k=1}^{\infty} \left( \frac{1}{2k} - \frac{1}{2k-1} \right)$$

```
> value(%);
```

$$-\ln(2)$$

### • Trigonometric and hyperbolic functions

**combine** transforms a polynomial in sines and cosines into the finite Fourier form by successive application of the following rules.

$$\begin{aligned}\sin x \sin y &\longrightarrow \frac{1}{2} \cos(x-y) - \frac{1}{2} \cos(x+y), \\ \sin x \cos y &\longrightarrow \frac{1}{2} \sin(x-y) + \frac{1}{2} \sin(x+y), \\ \cos x \cos y &\longrightarrow \frac{1}{2} \cos(x-y) + \frac{1}{2} \cos(x+y).\end{aligned}$$

```
> 2*sin(x)*cos(x): % = combine(%);
```

$$2 \sin(x) \cos(x) = \sin(2x)$$

```
> sin(x)^3: % = combine(%);
```

$$\sin(x)^3 = -\frac{1}{4} \sin(3x) + \frac{3}{4} \sin(x)$$

Similar transformations hold for hyperbolic functions:

$$\begin{aligned}\sinh x \sinh y &\longrightarrow \frac{1}{2} \cosh(x+y) - \frac{1}{2} \cosh(x-y), \\ \sinh x \cosh y &\longrightarrow \frac{1}{2} \sinh(x+y) + \frac{1}{2} \sinh(x-y), \\ \cosh x \cosh y &\longrightarrow \frac{1}{2} \cosh(x+y) + \frac{1}{2} \cosh(x-y).\end{aligned}$$

```
> cosh(x)^5: % = combine(%);
```

$$\cosh(x)^5 = \frac{1}{16} \cosh(5x) + \frac{5}{16} \cosh(3x) + \frac{5}{8} \cosh(x)$$

When you want to use the Maple procedure **combine**, exclusively for combination of trigonometric functions, then you must say so via the option **trig**. Compare the following statements.

```
> f := exp(cos(x)^2) * exp(sin(x)^2):  
> f = combine(f);
```

$$e^{(\cos(x)^2)} e^{(\sin(x)^2)} = e$$

```
> f = combine(f, trig);
```

$$e^{(\cos(x)^2)} e^{(\sin(x)^2)} = e^{(1/2 \cos(2x) + 1/2)} e^{(1/2 - 1/2 \cos(2x))}$$

```
> f = combine(f, exp);
```

$$e^{(\cos(x)^2)} e^{(\sin(x)^2)} = e^{(\cos(x)^2 + \sin(x)^2)}$$

Strangely enough, you also must use the keyword **trig** for hyperbolic functions instead of the keyword **trigh**, as you might expect.

```
> f := sqrt(cosh(x)^2 - 1) * sqrt(cosh(x)^2 + 1):  
> f = combine(f, trig);
```

$$\sqrt{\cosh(x)^2 - 1} \sqrt{\cosh(x)^2 + 1} = \frac{1}{4} \sqrt{2 \cosh(2x) - 2} \sqrt{2 \cosh(2x) + 6}$$

```
> f = combine(f, symbolic);
```

$$\sqrt{\cosh(x)^2 - 1} \sqrt{\cosh(x)^2 + 1} = \frac{1}{2} \sqrt{(\cosh(2x) - 1)(\cosh(2x) + 3)}$$

```
> f = combine(f, radical, symbolic);
```

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

### • exp, ln

**combine** combines expressions in which the exponential function and logarithm occur according to the following rules.

$$\begin{aligned} \exp x \exp y &\longrightarrow \exp(x + y), \\ \exp(x + n \ln y) &\longrightarrow y^n \exp(x), \text{ for } n \in \mathbb{Z}, \\ (\exp x)^n &\longrightarrow \exp(n x), \text{ for } n \in \mathbb{Z}, \\ y \ln x &\longrightarrow \ln(x^y), \text{ for } y \in \mathbb{Q}, \text{ and } y \arg(x) = \arg(x^y), \\ \ln x + \ln y &\longrightarrow \ln(xy), \text{ provided } \arg(x) + \arg(y) = \arg(xy). \end{aligned}$$

In the logarithmic case, Maple only carries out those combinations that are provable correct.

```
> exp(x)*exp(y)^2: % = combine(%);
```

$$e^x (e^y)^2 = e^{(x+2y)}$$

```
> x*ln(2) + 3*ln(x)+4*ln(5): % = combine(%);
```

$$x \ln(2) + 3 \ln(x) + 4 \ln(5) = x \ln(2) + 3 \ln(x) + \ln(625)$$

```
> ln(x)+ln(2)+ln(y)-ln(3): % = combine(%);
```

$$\ln(x) + \ln(2) + \ln(y) - \ln(3) = \ln(2x) + \ln\left(\frac{y}{3}\right)$$

```
> exp(x-2*ln(y)): % = combine(%);
```

$$e^{(x-2 \ln(y))} = \frac{e^x}{y^2}$$

Add the keywords **exp** or **ln** if you want to restrict combination of expressions to those involving the exponential or logarithmic function. Compare the following statements:

```

> f := exp(2)*exp(3)+ln(2)+ln(3):
> f = combine(f); # full combination

$$e^2 e^3 + \ln(2) + \ln(3) = e^5 + \ln(6)$$

> f = combine(f, exp); # only exponentials

$$e^2 e^3 + \ln(2) + \ln(3) = e^5 + \ln(2) + \ln(3)$$

> f = combine(f, ln); # only logarithms

$$e^2 e^3 + \ln(2) + \ln(3) = e^2 e^3 + \ln(6)$$


```

Suppose that you really want the combination  $\ln x + \ln y \longrightarrow \ln(xy)$  to be carried out by Maple. Then you can enforce this by adding the keyword **symbolic** to the function specific call of **combine** or by adding assumptions.

```

> f := ln(x)+1/2*ln(y)-ln(z);

$$f := \ln(x) + \frac{1}{2} \ln(y) - \ln(z)$$

> combine(f); # not much done

$$\ln(x) + \ln(\sqrt{y}) - \ln(z)$$

> f = combine(f, ln, symbolic);

$$\ln(x) + \frac{1}{2} \ln(y) - \ln(z) = \ln\left(\frac{x \sqrt{y}}{z}\right)$$

> f = combine(f) assuming positive;

$$\ln(x) + \frac{1}{2} \ln(y) - \ln(z) = \ln\left(\frac{x \sqrt{y}}{z}\right)$$


```

Actually, you have more control of the simplification process. For example, you may restrict combination to logarithmic terms with integer coefficients.

```

> f = combine(f, ln, integer, symbolic);

$$\ln(x) + \frac{1}{2} \ln(y) - \ln(z) = \frac{1}{2} \ln(y) + \ln\left(\frac{x}{z}\right)$$

> f = combine(f, ln, 'positive', symbolic);

$$\ln(x) + \frac{1}{2} \ln(y) - \ln(z) = -\ln(z) + \ln(x \sqrt{y})$$


```

The types **symbol** and **anything** are useful in the generic case.

```

> f := m*ln(x)+n*ln(y)+3*ln(z);

$$f := m \ln(x) + n \ln(y) + 3 \ln(z)$$

> f = combine(f, ln, symbolic);

$$m \ln(x) + n \ln(y) + 3 \ln(z) = m \ln(x) + n \ln(y) + \ln(z^3)$$

> f = combine(f, ln, symbol, symbolic);

```

```

 $m \ln(x) + n \ln(y) + 3 \ln(z) = 3 \ln(z) + \ln(x^m y^n)$ 
> f = combine(f, ln, anything, symbolic);
 $m \ln(x) + n \ln(y) + 3 \ln(z) = \ln(x^m y^n z^3)$ 

```

Possible ways of having the transformations

$$n \ln y \longrightarrow \ln(y^n)$$

and

$$\exp(x + n \ln y) \longrightarrow y^n \exp(x)$$

carried out via **combine** regardless of the type of  $n$  are the following:

```

> y*ln(x): % = combine(% , ln, anything, symbolic);
 $y \ln(x) = \ln(x^y)$ 
> f := exp(x+n*ln(y));
 $f := e^{(x+n \ln(y))}$ 
> f = combine(f, ln, anything, symbolic);
 $e^{(x+n \ln(y))} = e^{(x+\ln(y^n))}$ 
> combine(%);
 $e^{(x+n \ln(y))} = y^n e^x$ 

```

### • Powers and radicals

The two most important **combine** rules for powers are

$$\begin{aligned} x^y x^z &\longrightarrow x^{y+z}, \\ (x^y)^z &\longrightarrow x^{yz}. \end{aligned}$$

```

> x^y*x^z: % = combine(% , 'power');
 $x^y x^z = x^{(y+z)}$ 

```

By adding the keyword **symbolic** or by adding assumptions you can enforce transformations that are not always valid.

```

> combine((x^y)^z, 'power');
 $(x^y)^z$ 
> % = combine(% , 'power' , symbolic);
 $(x^y)^z = x^{(y z)}$ 

```

Another option in this class is for powers with rational exponents: **radical**. It takes care mainly of the transformation  $x^{(m/d)} y^{(n/d)} \longrightarrow (x^m y^n)^{(1/d)}$ , where  $x$  and  $y$  are both positive,  $m, n$ , and  $d$  are integers such that  $|m| < d$ ,  $|n| < d$ , and  $d > 1$ . By omitting the keyword, **combine** sometimes does too much radical simplification.

```

> f := x^(1/4) * y^(1/4);

$$f := x^{(1/4)} y^{(1/4)}$$

> combine(f, radical); # no simplification

$$x^{(1/4)} y^{(1/4)}$$

> f = combine(f, radical, symbolic); # simplification!

$$x^{(1/4)} y^{(1/4)} = (xy)^{(1/4)}$$

> f = combine(f, radical) assuming positive;

$$x^{(1/4)} y^{(1/4)} = (xy)^{(1/4)}$$


```

Let us look at some more examples of simplification of powers and radicals.

```

> (1/2)^m * (1/2)^n: % = combine(%);

$$\left(\frac{1}{2}\right)^m \left(\frac{1}{2}\right)^n = 2^{(-m-n)}$$

> x^y / x^(2/3): % = combine(%);

$$\frac{x^y}{x^{(2/3)}} = x^{(y-2/3)}$$

> 2^(1/3) * (x+1)^(1/3): % = combine(% , radical);

$$2^{(1/3)} (x + 1)^{(1/3)} = (2x + 2)^{(1/3)}$$


```

## • Other combinations

Other applications of **combine** are combinations of expressions involving arctangents, and polylogarithms, and rearrangements of expressions to get rid of minus signs.

```

> arctan(x) + arctan(1/x): % = combine(% , arctan);

$$\arctan(x) + \arctan\left(\frac{1}{x}\right) = \frac{1}{2} \operatorname{signum}(x) \pi$$

> arctan(1/2) + arctan(1/3): % = combine(%);

$$\arctan\left(\frac{1}{2}\right) + \arctan\left(\frac{1}{3}\right) = \frac{1}{4} \pi$$

> combine(arctan(x)+arctan(y));

$$\arctan(x) + \arctan(y)$$

> % = combine(% , arctan, symbolic);

$$\arctan(x) + \arctan(y) = \arctan\left(\frac{x+y}{1-xy}\right)$$

> polylog(2,z) + polylog(2,1-z): % = combine(%);

$$\operatorname{polylog}(2, z) + \operatorname{polylog}(2, 1 - z) = \frac{\pi^2}{6} - \ln(z) \ln(1 - z)$$


```

```
>  $-\frac{(-x+1)/x}{}$ : % = combine(%);

$$-\frac{-x+1}{x} = \frac{x-1}{x}$$

```

## 14.4 simplify

**simplify** is Maple's general purpose simplification routine.

- **Trigonometric and hyperbolic functions**

Rational expressions in which trigonometric functions and hyperbolic functions occur are normalized by **simplify** according to the following rules.

$$\begin{aligned}\sin^2 x &\longrightarrow 1 - \cos^2 x, \\ \sinh^2 x &\longrightarrow \cosh^2(x) - 1, \\ \tan x &\longrightarrow \frac{\sin x}{\cos x}, \\ \tanh x &\longrightarrow \frac{\sinh x}{\cosh x}.\end{aligned}$$

More precisely, powers of sines and hyperbolic sines with exponent greater than one are simplified by the above rules as much as possible. The tangent and hyperbolic tangent rule is only applied in the presence of other trigonometric functions.

```
>  $\cosh(x)^2 - \sinh(x)^2$ : % = simplify(%);

$$\cosh(x)^2 - \sinh(x)^2 = 1$$

>  $\sinh(x)^3$ : % = simplify(%);

$$\sinh(x)^3 = -\sinh(x) + \sinh(x)\cosh(x)^2$$

>  $2*\sin(x)/(1+\tan(x)^2)$ : % = simplify(%);

$$\frac{2\sin(x)}{1+\tan(x)^2} = 2\sin(x)\cos(x)^2$$

```

If you prefer the rule  $\cos^2 x \rightarrow 1 - \sin^2 x$  to the rule  $\sin^2 x \rightarrow 1 - \cos^2 x$ , then you should simplify with respect to side relations. We shall discuss simplification with respect to side relations in detail in §14.7.

```
>  $\sin(x)^3 + \cos(x)^3$ ;

$$\sin(x)^3 + \cos(x)^3$$

> simplify(%);

$$\cos(x)^3 + \sin(x) - \sin(x)\cos(x)^2$$

> simplify(%%, {cos(x)^2+sin(x)^2=1}, [sin(x), cos(x)]);

$$\cos(x)^3 + \sin(x) - \sin(x)\cos(x)^2$$

```

```
> simplify(%%, {cos(x)^2+sin(x)^2=1}, [cos(x),sin(x)]);
sin(x)^3 - cos(x) sin(x)^2 + cos(x)
```

If you want to specify trigonometric simplification only, add the keyword **trig**.

```
> 4^(1/2) - sin(x)^2 - 1;

$$\sqrt{4} - \sin(x)^2 - 1$$

> simplify(% , trig);

$$\sqrt{4} - 2 + \cos(x)^2$$

> simplify(%);

$$\cos(x)^2$$

```

Also note that normalization is done before trigonometric simplification. The following lines show the consequences.

```
> (1-sin(x)^3) / (1-sin(x)^2);

$$\frac{1 - \sin(x)^3}{1 - \sin(x)^2}$$

> simplify(% , trig);

$$\frac{\sin(x) + 2 - \cos(x)^2}{\sin(x) + 1}$$

> normal(%);

$$\frac{\sin(x)^2 + \sin(x) + 1}{\sin(x) + 1}$$

> map(simplify, %% , trig);

$$\frac{1 - \sin(x) + \sin(x) \cos(x)^2}{\cos(x)^2}$$

```

### • Inverse trigonometric and hyperbolic functions

**simplify** takes care of the transformation  $\text{arctrig}(\text{trig}(x)) \rightarrow x$ , where  $\text{trig} = \sin, \cos, \tan, \sinh, \cosh, \tanh$ , etc., when this transformation is valid. Furthermore, **simplify** “knows” some transformations for inverse tangents like **combine** did.

```
> simplify(arcsin(sin(x)));
arcsin(sin(x))
> simplify(% , assume=RealRange(-Pi/2,Pi/2));

$$x$$

> simplify(% , 'arctrig' , symbolic);

$$x$$

```

```
> arctan(x) + arctan(1/x):
> use RealDomain in % = simplify(%) end use;
```

$$\arctan(x) + \arctan\left(\frac{1}{x}\right) = \frac{1}{2} \operatorname{signum}(x) \pi$$

• **exp, ln**

For the exponential function, **simplify** does the same as **expand** with two exceptions, viz.,  $\exp x \exp y \rightarrow \exp(x + y)$  and  $\frac{1}{\exp x} \rightarrow \exp(-x)$ .

```
> exp(x)*exp(y): % = simplify(%);
```

$$e^x e^y = e^{(x+y)}$$

```
> expand(rhs(%));
```

$$e^x e^y$$

```
> exp(x)^2: % = simplify(%);
```

$$(e^x)^2 = e^{(2x)}$$

```
> 1/exp(x): % = simplify(%);
```

$$\frac{1}{e^x} = e^{(-x)}$$

For the natural logarithm, **simplify** factors the argument and then applies the following transformation rules.

- $\ln(x^y) \rightarrow y \ln(x)$  for positive  $x$  and real  $y$ ,
- $\ln(x^y) \rightarrow y \ln(-x)$  for negative  $x$  and even integer  $y$ ,
- $\ln(x^y) \rightarrow y \ln(x)$  for negative  $x$  and odd integer  $y$ ,
- $\ln(x^y) \rightarrow y \ln(x)$  for real  $x$  and odd integer  $y$ ,
- $\ln(x^y) \rightarrow \frac{y}{2} \ln(x^2)$  for real  $x$  and even integer  $y$ ,
- $\ln(xy) \rightarrow \ln(x) + \ln(y)$  for positive  $x$ ,
- $\ln(xy) \rightarrow \ln(-x) + \ln(-y)$  for negative  $x$ ,
- $\ln(\exp(x)) \rightarrow x$  for real  $x$ ,
- $\ln(\text{LambertW}(x)) \rightarrow \ln(x) + \text{LambertW}(x)$  for positive  $x$ .

The simplifications  $\ln(xy) \rightarrow \ln(x) + \ln(y)$  and  $\ln(x^y) \rightarrow y \ln(x)$  can always be forced by using the keyword **ln** together with the keyword **symbolic** or an extra assumption.

```
> f := simplify(ln(x^2), ln);
```

$$f := \ln(x^2)$$

```
> f = simplify(f, ln, symbolic);
```

$$\ln(x^2) = 2 \ln(x)$$

```
> f = simplify(f, ln, assume=positive);
```

```

 $\ln(x^2) = 2 \ln(x)$ 
> f = simplify(f, ln, assume=negative);
 $\ln(x^2) = 2 \ln(-x)$ 

```

### • Powers and radicals

**simplify** does the same as **expand** for most powers, with the important exception  $x^y x^z \rightarrow x^{y+z}$  and, as you will see below, simplification of powers with fractions as exponents. Like **expand**, the procedure **simplify** takes care of the validity of a transformation as you will notice in the examples below.

```

> x^y*x^z: % = simplify(%);
 $x^y x^z = x^{(y+z)}$ 
> expand(rhs(%));
 $x^y x^z$ 
> simplify((x^y)^z);
 $(x^y)^z$ 
> % = simplify(% , 'power' , symbolic);
 $(x^y)^z = x^{(yz)}$ 
> simplify((x/y)^z);
 $(\frac{x}{y})^z$ 
> % = simplify(% , 'power' , symbolic);
 $(\frac{x}{y})^z = x^z y^{(-z)}$ 
> simplify((-x)^y);
 $(-x)^y$ 
> % = simplify(% , 'power' , symbolic);
 $(-x)^y = (-1)^y x^y$ 
> (x^(1/2))^(y/2): % = simplify(%);
 $(\sqrt{x})^{(\frac{y}{2})} = x^{(\frac{y}{4})}$ 
> simplify((x*y)^z);
 $(xy)^z$ 
> % = simplify(% , 'power' , symbolic);
 $(xy)^z = x^z y^z$ 

```

For powers with fractional exponents, the procedure **simplify** differs a lot from **expand**. In this case, **simplify** relies in fact on the procedure **radsimp** (radical simplification). The procedure **radsimp** is especially designed to simplify expressions in which square roots and powers with other fractional exponents occur. Such simplifications are often difficult and time-consuming. When you mention the keyword **radical** in a call of **simplify** then Maple knows that you want simplification only of this type and nothing else.

```
> r := (-2/27)^(1/3): r = simplify(r);

$$\frac{1}{27} (-2)^{(1/3)} 27^{(2/3)} = \frac{1}{6} (1 + I\sqrt{3}) 2^{(1/3)}$$

> use RealDomain in r = simplify(r) end use;

$$\frac{(-2)^{(1/3)} 27^{(2/3)}}{27} = \text{undefined}$$

```

Note the difference between the following:

```
> r = simplify(r, 'power');

$$\frac{1}{27} (-2)^{(1/3)} 27^{(2/3)} = \frac{1}{3} (-1)^{(1/3)} 2^{(1/3)}$$

> use RealDomain in r = simplify(r, 'power') end use;

$$\frac{(-2)^{(1/3)} 27^{(2/3)}}{27} = -\frac{2^{(1/3)}}{3}$$

> r = simplify(r, radical);

$$\frac{1}{27} (-2)^{(1/3)} 27^{(2/3)} = \frac{1}{6} (1 + I\sqrt{3}) 2^{(1/3)}$$

> (1-y^2)^(3/2) - (1-y^2)^(1/2): % = simplify(%);

$$(1 - y^2)^{(3/2)} - \sqrt{1 - y^2} = -\sqrt{1 - y^2} y^2$$

> (1-sin(x)^2)^(3/2) - (1-sin(x)^2)^(1/2):
> % = simplify(%), radical;

$$(1 - \sin(x)^2)^{(3/2)} - \sqrt{1 - \sin(x)^2} = -\sqrt{1 - \sin(x)^2} \sin(x)^2$$

> use RealDomain in %% = simplify(%%) end use;

$$(1 - \sin(x)^2)^{(3/2)} - \sqrt{1 - \sin(x)^2} = |\cos(x)| (-1 + \cos(x)^2)$$

```

Radical simplification is carried out by Maple with much care.

```
> r := (x^4)^(5/4): r = simplify(r);

$$(x^4)^{(5/4)} = x^4 (x^4)^{(1/4)}$$

> r = simplify(r) assuming positive;

$$(x^4)^{(5/4)} = x^5$$

```

- Other simplifications

Maple knows many rules for functions like the gamma function, the Riemann zeta function, the hypergeometric function, and a lot more. Three examples:

```
> GAMMA(n+1/2)/GAMMA(n-1/2): % = simplify(%);
```

$$\frac{\Gamma(n + \frac{1}{2})}{\Gamma(n - \frac{1}{2})} = n - \frac{1}{2}$$

```
> hypergeom([-1,-3/2],[1/2],z^2/t^2): % = simplify(%);
```

$$\text{hypergeom}\left([-1, -\frac{3}{2}], [\frac{1}{2}], \frac{z^2}{t^2}\right) = \frac{t^2 + 3z^2}{t^2}$$

```
> LaguerreL(3,x): % = simplify(%,'LaguerreL');
```

$$\text{LaguerreL}(3, x) = 1 - 3x + \frac{3}{2}x^2 - \frac{1}{6}x^3$$

If you do not want to apply all possible simplifications provided for, then you must explicitly mention, in the call to **simplify**, those mathematical functions for which simplification should be carried out (as in the above example where the keyword **radical** was added). Note that this is the opposite to the way in which expansion of functions is suppressed.

```
> exp(x)*exp(y) + cos(x)^2 + sin(x)^2;
```

$$e^x e^y + \cos(x)^2 + \sin(x)^2$$

```
> simplify(%);
```

$$e^{(x+y)} + 1$$

```
> simplify(%%, exp);
```

$$e^{(x+y)} + \cos(x)^2 + \sin(x)^2$$

```
> simplify(%%%, trig);
```

$$e^x e^y + 1$$

## 14.5 convert

Expressions in which (hyperbolic) trigonometric functions and their inverses occur can be explicitly transformed into different forms with **convert**. Some examples:

- Conversion of (hyperbolic) trigonometric functions into exponential form and the reverse conversion.

```
> cos(x): % = convert(% , exp);
```

```

cos(x) =  $\frac{1}{2} e^{(xI)} + \frac{1}{2} \frac{1}{e^{(xI)}}$ 
> convert(% , trig);
cos(x) =  $\frac{1}{2} \cos(x) + \frac{1}{2} I \sin(x) + \frac{1}{2} \frac{1}{\cos(x) + \sin(x) I}$ 
> cosh(x): % = convert(% , exp);
cosh(x) =  $\frac{1}{2} e^x + \frac{1}{2} \frac{1}{e^x}$ 
> convert(% , trig);
cosh(x) =  $\frac{1}{2} \cosh(x) + \frac{1}{2} \sinh(x) + \frac{1}{2} \frac{1}{\cosh(x) + \sinh(x)}$ 

```

You can be more specific in your wishes.

```

> expr := cos(x)*sin(y): % = convert(% , exp, x);
> # convert only function having x
cos(x) cos(y) = ( $\frac{1}{2} e^{(xI)} + \frac{1}{2} \frac{1}{e^{(xI)}}$ ) sin(y)

```

The following two commands would give the same result (output omitted).

```

> expr = convert(expr, exp, exclude=sin);
> expr = convert(expr, exp, only=cos);

```

Also note the difference between the following two examples.

```

> exp(x+I*y): % = convert(% , trig);
e(x+I y) = (cosh(x) + sinh(x)) (cos(y) + I sin(y))
> exp(x+I*y): % = evalc(%);
e(x+I y) = ex cos(y) + I ex sin(y)

```

- Conversion of inverse (hyperbolic) trigonometric functions into logarithmic expressions.

```

> arcsin(x): % = convert(% , ln);
arcsin(x) = -I ln( $\sqrt{1-x^2} + I x$ )
> arcsinh(x): % = convert(% , ln);
arcsinh(x) = ln( $x + \sqrt{x^2 + 1}$ )

```

The above conversions of (hyperbolic) trigonometric functions and their inverses can be combined into one conversion by the keyword **expln**.

- Conversion of trigonometric functions into expressions with only tangents.

```
> sin(x): % = convert(% , tan);
```

$$\sin(x) = \frac{2 \tan(\frac{x}{2})}{1 + \tan(\frac{x}{2})^2}$$

$$\cos(x) = \frac{1 - \tan(\frac{x}{2})^2}{1 + \tan(\frac{x}{2})^2}$$

$$\sin(x) \sin(y) = \frac{2 \tan(\frac{x}{2}) \sin(y)}{1 + \tan(\frac{x}{2})^2}$$

$$\frac{\sin(x)}{\cos(x)} = \tan(x)$$

$$\frac{\sin(x) \sin(y)}{\cos(x) \cos(y)} = \frac{\sin(y) \tan(x)}{\cos(y)}$$

- Conversion of (hyperbolic) trigonometric functions into expressions with only (hyperbolic) sines and cosines

$$\tan(x) : \% = \text{convert}(\%, \text{'sincos'});$$

$$\tan(x) = \frac{\sin(x)}{\cos(x)}$$

$$\tanh(x) : \% = \text{convert}(\%, \text{'sincos'});$$

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)}$$

- Conversion of trigonometric functions into expressions with only sines and cosines, and conversion of hyperbolic functions into expressions with only exponential functions.

$$\tan(x) : \% = \text{convert}(\%, \text{'expsincos'});$$

$$\tan(x) = \frac{\sin(x)}{\cos(x)}$$

$$\tanh(x) : \% = \text{convert}(\%, \text{'expsincos'});$$

$$\tanh(x) = \frac{(e^x)^2 - 1}{(e^x)^2 + 1}$$

- Conversion of trigonometric functions of a rational multiple of  $\pi$  into radical forms.

```
> sin(Pi/15): % = convert(%, radical);

$$\sin\left(\frac{\pi}{15}\right) = \left(\frac{1}{8} - \frac{1}{8}\sqrt{5}\right)\sqrt{3} + \frac{1}{8}\sqrt{2}\sqrt{5+\sqrt{5}}$$

```

- Conversion of special functions into 'standard' functions.

```
> hypergeom([1,2], [1/2], z):
> % = convert(%, 'StandardFunctions');

$$\text{hypergeom}([1, 2], [\frac{1}{2}], z) = \frac{z+2}{2(z-1)^2} - \frac{3}{2} \frac{\sqrt{z}\sqrt{1-z}\arcsin(\sqrt{z})}{(z-1)^3}$$

> MeijerG([], [], [[1/2], [0]], 1/4*z):
> % = simplify(convert(%, 'StandardFunctions'));

$$\text{MeijerG}([], [], [[\frac{1}{2}], [0]], \frac{z}{4}) = \frac{\sin(\sqrt{z})}{\sqrt{\pi}}$$

> %% = convert(%, hypergeom);

$$\text{MeijerG}([], [], [[\frac{1}{2}], [0]], \frac{z}{4}) = \frac{1}{2} \frac{\sqrt{4}\sqrt{z}\text{hypergeom}([], [\frac{3}{2}], -\frac{z}{4})}{\sqrt{\pi}}$$

```

- Furthermore, Maple provides conversions of factorials and binomial coefficients into gamma functions and vice versa.

```
> n!: % = convert(%, GAMMA);

$$n! = \Gamma(n+1)$$

> rhs(%) = convert(rhs(%), 'factorial');

$$\Gamma(n+1) = \frac{(n+1)!}{n+1}$$

> lhs(%) = expand(rhs(%));

$$\Gamma(n+1) = n!$$

> binomial(n,k): % = convert(%, GAMMA);

$$\text{binomial}(n, k) = \frac{\Gamma(n+1)}{\Gamma(k+1)\Gamma(n-k+1)}$$

> binomial(n,k): % = convert(%, factorial);

$$\text{binomial}(n, k) = \frac{n!}{k!(n-k)!}$$

> rhs(%) = convert(rhs(%), binomial);

$$\frac{n!}{k!(n-k)!} = \text{binomial}(n, k)$$

> multinomial(n,a,b,c,d): % = convert(%, GAMMA);

$$\text{multinomial}(n, a, b, c, d) = \frac{\Gamma(n+1)}{\Gamma(a+1)\Gamma(b+1)\Gamma(c+1)\Gamma(d+1)}$$

```

## 14.6 Trigonometric Simplification

Substitution of trigonometric expressions plays a special role in Maple. The computer algebra system provides an extra facility: for simple expressions it can suggest equivalent ones. The Maple procedure is called **trigsubs**. A few examples:

```
> trigsubs(sin(2*x));
[sin(2 x), -sin(-2 x), 2 sin(x) cos(x),  $\frac{1}{\csc(2 x)}$ ,  $-\frac{1}{\csc(-2 x)}$ ,  $\frac{2 \tan(x)}{1 + \tan(x)^2}$ ,
 $\frac{-1}{2} I(e^{(2 I x)} - e^{(-2 I x)})]$ 
> convert(trigsubs(tan(x)^2), set);
{ $\tan(x)^2$ ,  $\frac{4 \tan(\frac{x}{2})^2}{(1 - \tan(\frac{x}{2})^2)^2}$ ,  $\frac{1}{\cot(x)^2}$ ,  $\frac{4 \cot(\frac{x}{2})^2}{(\cot(\frac{x}{2})^2 - 1)^2}$ ,  $\frac{4}{(\cot(\frac{x}{2}) - \tan(\frac{x}{2}))^2}$ ,
 $-\frac{(e^{(x I)} - e^{(-I x)})^2}{(e^{(x I)} + e^{(-I x)})^2}$ ,  $(\csc(2 x) - \cot(2 x))^2$ ,  $\sec(x)^2 - 1$ ,  $\frac{\sin(x)^2}{\cos(x)^2}$ ,
 $\frac{\sin(2 x)^2}{(1 + \cos(2 x))^2}$ ,  $\frac{(1 - \cos(2 x))^2}{\sin(2 x)^2}$ }
```

> trigsubs(sin(x)+sin(y));

$$[2 \sin(\frac{x}{2} + \frac{y}{2}) \cos(\frac{x}{2} - \frac{y}{2})]$$

Do not expect miracles; many trigonometric equivalences are not recognized by Maple via **trigsubs**. In the case where trigonometric substitution is desired, **trigsubs** can also be used as an alternative for the substitution procedure **subs**, with the extra feature that the trigonometric equality in the first argument is verified via the knowledge base.

```
> trigsubs(cos(2*x)=cos(x)^2-sin(x)^2,
>           cos(2*x)+sin(x)^2+1);
cos(x)^2 + 1
```

Trigonometric simplification can be cumbersome, as you may have to convert between different representations during a computation. A simple example: let us start with the expansion of  $\tan(3x)$ .

```
> expand(tan(3*x));

$$\frac{3 \tan(x) - \tan(x)^3}{1 - 3 \tan(x)^2}$$

```

Try to recover from this result the original expression. You may expect that the following command does the job.

```
> combine(%);
```

$$\frac{-3 \tan(x) + \tan(x)^3}{-1 + 3 \tan(x)^2}$$

Alas. You first have to write the expression in terms of sines and cosines.

```
> convert(%, 'sincos');
```

$$\frac{-3 \frac{\sin(x)}{\cos(x)} + \frac{\sin(x)^3}{\cos(x)^3}}{-1 + \frac{3 \sin(x)^2}{\cos(x)^2}}$$

Do not be too optimistic and immediately apply **combine**.

```
> combine(%);
```

$$\frac{\sin(6x) + 3 \sin(2x) + 3 \sin(4x)}{3 \cos(4x) + 1 + 3 \cos(2x) + \cos(6x)}$$

You first have to simplify the rational expression of sines and cosines before you can combine trigonometric terms.

```
> normal(%);
```

$$\frac{\sin(x)(-3 \cos(x)^2 + \sin(x)^2)}{\cos(x)(-\cos(x)^2 + 3 \sin(x)^2)}$$

```
> combine(%);
```

$$\frac{\sin(3x)}{\cos(3x)}$$

Finally, convert the expression into a tangent.

```
> convert(%, tan);
```

$$\tan(3x)$$

Maple takes into account knowledge about properties of variables when doing trigonometric simplification. An example:

```
> cos(x+n*Pi);
```

$$\cos(x + n\pi)$$

```
> expand(%);
```

$$\cos(x)\cos(n\pi) - \sin(x)\sin(n\pi)$$

```
> % assuming n::integer;
```

$$\cos(x)(-1)^n$$

We end this section with an example of trigonometric simplification that shows how cumbersome it can be in practical cases. The task is to compute

```
> Sum(cos(omega*t-alpha-n*beta), n=-N..N);
```

$$\sum_{n=-N}^N \cos(\omega t - \alpha - n\beta)$$

```
> value(%):
```

The lengthy result is not shown; we first simplify it.

```
> simplify(%):
```

This result is not much shorter than before and therefore also not displayed. But what we gained is that, in its present form, combination of the trigonometric terms will improve the simplicity of the result.

```
> combine(%);
```

$$\frac{1}{2}(\sin(-\alpha + N\beta + \omega t) - \sin(-\alpha - N\beta + \omega t) - \sin(-N\beta - \beta + \omega t - \alpha) \\ + \sin(N\beta + \beta + \omega t - \alpha)) / \sin(\beta)$$

$\omega t - \alpha$  appears in all terms; let us take this term apart.

```
> algsubs(omega*t-alpha = zeta, %);
```

$$\frac{1}{2} \frac{\sin(N\beta + \zeta) + \sin(N\beta - \zeta) + \sin(N\beta + \beta - \zeta) + \sin(N\beta + \beta + \zeta)}{\sin(\beta)}$$

We expand the expression and simplify the result again.

```
> simplify(expand(%));
```

$$\frac{\cos(\zeta)(\sin(N\beta) + \sin(N\beta)\cos(\beta) + \cos(N\beta)\sin(\beta))}{\sin(\beta)}$$

We combine the trigonometric terms again but leave  $\cos \zeta$  intact. One way to do this here is the following:

```
> map(combine, %);
```

$$\frac{\cos(\zeta)(\sin(N\beta) + \sin(N\beta + \beta))}{\sin(\beta)}$$

This formula is not ugly, but let us see whether the formula for sum of sines brings us further. First, we temporarily forget about the term  $\cos \zeta$ .

```
> coeff(% , cos(zeta));
```

$$\frac{\sin(N\beta) + \sin(N\beta + \beta)}{\sin(\beta)}$$

Let us ask **trigsubs** for help.

```
> subs( s=beta+N*beta, trigsubs( subs(N*beta+beta=s, numer(%)) ) );
```

$$[2 \sin(N\beta + \frac{1}{2}\beta) \cos(-\frac{\beta}{2})]$$

```
> trigsubs(denom(%%%));
```

$$\left[ \sin(\beta), -\sin(-\beta), 2\sin\left(\frac{\beta}{2}\right)\cos\left(\frac{\beta}{2}\right), \frac{1}{\csc(\beta)}, -\frac{1}{\csc(-\beta)}, \frac{2\tan\left(\frac{\beta}{2}\right)}{1 + \tan^2\left(\frac{\beta}{2}\right)}, \right. \\ \left. \frac{-1}{2} I(e^{(\beta I)} - e^{(-I \beta)}) \right]$$

The third substitution option is most promising as it will lead to most cancellations.

```
> op(1,%)/op(3,%);
```

$$\frac{\sin(N\beta + \frac{1}{2}\beta)}{\sin(\frac{\beta}{2})}$$

So, the sum is equal to

```
> subs(zeta=omega*t-alpha, cos(zeta)*%);
```

$$\frac{\cos(\omega t - \alpha) \sin(N\beta + \frac{1}{2}\beta)}{\sin(\frac{\beta}{2})}$$

A nice result, but obtained via a rather lengthy derivation.

## 14.7 Simplification w.r.t. Side Relations

Let us consider the following problem from the Dutch Mathematics Olympiad of September 6, 1991.

Let  $a, b, c$  be real numbers such that

$$a + b + c = 3, \quad a^2 + b^2 + c^2 = 9, \quad a^3 + b^3 + c^3 = 24.$$

Compute  $a^4 + b^4 + c^4$ .

Maple's solution is the following:

```
> siderels := {a+b+c=3, a^2+b^2+c^2=9, a^3+b^3+c^3=24};
```

$$siderels := \{a + b + c = 3, a^2 + b^2 + c^2 = 9, a^3 + b^3 + c^3 = 24\}$$

```
> simplify(a^4+b^4+c^4, siderels);
```

To understand how Maple computes this result you must have some notion of what a *Gröbner basis* is and how it is used. In this section, only the idea

behind a Gröbner basis is described; for a more mathematically oriented introduction the interested reader is referred to textbooks [3, 16, 37, 66, 85, 91, 98, 156] and to overview articles [5, 35, 36, 105, 127, 175].

First, Maple considers the set of polynomials that specify the side relations as pure polynomials rather than equations.

```
> polys := map(lhs-rhs, siderels);
polys := {a + b + c - 3, a2 + b2 + c2 - 9, a3 + b3 + c3 - 24}
```

Next, Maple computes the minimal, monic Gröbner basis with respect to the pure lexicographic ordering. Roughly speaking, a Gröbner basis is a set of polynomials that generates the same ideal as the original set of polynomials, with some extra properties imposed on the basis. The Maple package for computing this basis is called **Groebner**. The command in this package that does the work is **gbasis**.

```
> with(Groebner): # load the Groebner basis package
> G := gbasis(polys, plex(a,b,c));
G := [c3 + 1 - 3c2, b2 + c2 + bc - 3b - 3c, a + b + c - 3]
```

The Gröbner basis depends on the ordering of monomials in  $a, b$ , and  $c$ . Here, the pure lexicographic ordering with  $a > b > c$  is used (q.v., §5.2).

**Characterization of Gröbner bases.** *A finite set  $G$  of polynomials is a Gröbner basis if each element of the ideal generated by  $G$  reduces to zero by application of “reductions” with respect to the ordering  $\succ$ .*

What is meant by “reductions” can best be explained via the example. From the first polynomial comes the reduction with respect to  $a$ ,

$$a \longrightarrow 3 - b - c.$$

From the second polynomial the “easiest” reduction with respect to pure lexicographic ordering is that the “largest” monomial is eliminated via

$$b^2 \longrightarrow -bc + 3b - c^2 + 3c.$$

( $bc$  is the “largest” monomial occurring on the right-hand side.) From the third polynomial the “easiest” reduction is to eliminate the highest-degree term

$$c^3 \longrightarrow 3c^2 - 1.$$

When these reductions are applied to a polynomial until they can be applied no longer, then a so-called *normal form* of the given polynomial is reached. A Gröbner basis  $G$  can also be characterized by the property that zero is the unique normal form of every element of the ideal generated by  $G$ . In other words, in a Gröbner basis  $G$  it is guaranteed that the result of successive applications of these reductions on a polynomial in the ideal generated by  $G$  will be zero.

A Gröbner basis  $G$  is a minimal, monic basis when each element  $g$  of  $G$  has a leading coefficient equal to 1 and is in normal form with respect to  $G \setminus \{g\}$ . For a minimal, monic Gröbner basis the normal form of any polynomial is unique; it is a “canonical form” in the sense that two polynomials are equivalent when their normal forms are the exact same polynomial.

Maple provides the procedure **normalf** for computing the normal form. Let us determine the normal form of  $a^4 + b^4 + c^4$  with respect to the computed Gröbner basis.

```
> normalf(a^4+b^4+c^4, G, plex(a,b,c));
```

69

Hence, simplification with respect to polynomial side relations is nothing but the computation of the normal form of a polynomial with respect to the minimal, reduced Gröbner basis of the ideal generated by the side relations in some term ordering. On a rational expression, simplification with respect to polynomial side relations is applied separately to the numerator and denominator of the normalized quotient. If no variables are specified or if indeterminates are specified as a set, then total degree ordering is chosen. If variables are specified as a list, then induced pure lexicographic ordering is chosen. (For the definition of these term orderings we refer to §5.2.) To control the outcome of simplification with respect to side relations, it is often prudent to specify variables in the ordering needed.

```
> simplify(x^3+y^3, {x^2+y^2=1}, [x,y]);
y^3 - x y^2 + x
> simplify(x^3+y^3, {x^2+y^2=1}, [y,x]);
x^3 - y x^2 + y
> simplify((x^3-y^3)/(x^3+y^3), {x^2+y^2=1});
-y^3 - x y^2 + x
y^3 - x y^2 + x
```

Compare this simplification with the following one.

```
> siderel := {cos(x)^2+sin(x)^2=1};
siderel := {cos(x)^2 + sin(x)^2 = 1}
> f := cos(x)^3+sin(x)^3;
f := cos(x)^3 + sin(x)^3
> simplify(f, siderel, [cos(x),sin(x)]);
sin(x)^3 - cos(x) sin(x)^2 + cos(x)
> simplify(f, siderel, [sin(x),cos(x)]);
cos(x)^3 - sin(x) cos(x)^2 + sin(x)
```

From this it should be clear how simplification with side relations works for generalized rational expressions.

Simplification with respect to side relations is a powerful tool. Consider, for example, the simplification of the polynomial  $f$  defined as

```
> f := sort(expand(subs(u=(x^2*y-y+x+4), u^3-4*u^2+10*u)));
f := y^3 x^6 - 3 y^3 x^4 + 3 y^2 x^5 + 8 y^2 x^4 + 3 y^3 x^2 - 6 y^2 x^3 + 3 y x^4
- 16 y^2 x^2 + 16 y x^3 - y^3 + 3 y^2 x + 23 y x^2 + x^3 + 8 y^2 - 16 y x
+ 8 x^2 - 26 y + 26 x + 40
> simplify(f, {u=x^2*y-y+x+4}, {x,y});
u^3 - 4 u^2 + 10 u
```

By pencil and paper such compositions of polynomials are difficult to find and to verify.

In many cases, the procedure **match** may be a good alternative because it uses a polynomial time algorithm of algebraic pattern matching, whereas the time and memory requirements for Gröbner basis calculations can be immense.

```
> guess := a*u^3 + b*u^2 + c*u + d;
> u := x^2*y - y + x + 4;
> match(f=guess, x, 'parms');
true
> parms;
{a = 1, d = 0, c = 10, b = -4, y = y}
> subs(parms, eval(guess,1));
u^3 - 4 u^2 + 10 u
```

By the way, Maple provides the procedure **compoly** for finding a composition of polynomials. For the above function  $f$ , **compoly** finds the composition

```
> compoly(f);
40 + 26 x + 8 x^2 + x^3, x = x^2 y - y + x
```

This must be interpreted as  $f = 40 + 26v + 8v^2 + v^3$ , where  $v = yx^2 - y + x$ . As you see, compositions of polynomials are not unique.

As a second example, we consider the univariate polynomial

```
> f := x^6 + 6*x^4 + x^3 + 9*x^2 + 3*x - 5;
f := x^6 + 6 x^4 + x^3 + 9 x^2 + 3 x - 5
> compoly(f);
x^2 - 5 + x, x = 3 x + x^3
```

So,  $f$  composes as  $g \circ h$ , where  $g = x^2 + x - 5$  and  $h = x^3 + 3x$ . Let Maple check its answer.

```
> subs(%[2], %[1]);
          (3 x + x3)2 - 5 + 3 x + x3
> expand(%-f);
          0
```

Note that the composition is not unique:  $f$  also composes as  $g \circ h$ , where  $g = x^2 - \frac{21}{4}$  and  $h = x^3 + 3x + \frac{1}{2}$ .

The algorithm used for univariate polynomials is described in [13, 119]. The more general case of composing a rational function in one variable into a composition of rational functions is discussed in [120, 121, 244].

## 14.8 Control Over Simplification

There are basically five ways to control the simplification process:

- adding assumptions;
- using the **RealDomain** package to restrict calculations to a real context
- omitting validity checks;
- restricting the transformations to a class of functions;
- using the expression size as the most important simplification criterion.

Throughout this chapter you have already seen examples of such controls. In this section, we summarize them by giving more examples.

### • simplify with assumptions

The command **simplify( expression, assume = property )** will simplify the *expression* assuming that all variables have the given *property*. In this way, you can carry out a simplification that would otherwise have no effect or only a partial effect. An example:

```
> expr := sqrt((x-1)^2);
          expr :=  $\sqrt{(x - 1)^2}$ 
> simplify(expr);
          csgn(x - 1) (x - 1)
> simplify(expr, assume=real);
          |x - 1|
```

Alternatively, you can use the **assuming** operator. This offers even more opportunities such as assumptions about distinct variables.

```
> simplify(expr) assuming x::RealRange(1,infinity);
x - 1
> simplify(expr) assuming x::RealRange(-infinity,1);
1 - x
```

Note that the assumption **real** does *not* mean that the domain of computation is the set of real numbers. It *only* means that variables are assumed to be real. This explains why

```
> simplify((-1)^(1/3)) assuming real;
1/2 + 1/2 I √3
```

still returns a complex value and not the answer -1, which you might have expected. Use the **RealDomain** package to get this result.

```
> use RealDomain in simplify((-1)^(1/3)) end use;
-1
```

When assumptions are explicitly made about variables via **assuming** or **assume**, then Maple will (try to) use this information in the simplification process.

```
> ln(exp(x));
ln(ex)
> ln(exp(x)) assuming real;
x
> (x^3*y)^(1/3): % = simplify(%) assuming x>0;
(x3 y)(1/3) = x y(1/3)
> assume(y, RealRange(-Pi/2,Pi/2)):
> arcsin(sin(y)): % = simplify(%);

arcsin(sin(y)) = y
with assumptions on y
```

### • Using the **RealDomain** package

The **RealDomain** package provides an environment in which Maple's default behavior of assuming that the basic underlying number system is the complex field is replaced by the assumption that Maple is working in a 'real context', i.e., a context in which the basic underlying number system is the field of real numbers. You can use this package within the scope of a **use** statement, or after issuing the command **with(RealDomain):**. Some functions are then shadowed by package functions. The global (default)

procedures assigned to those exported by the `RealDomain` package can still be accessed by using the `:-` prefix. For instance, the default `sin` procedure can be accessed using `:-sin`. A few example will do:

```
> use RealDomain in sqrt(x^2): simplify()% end use;
 $\sqrt{x^2}$ 
|x|
> with(RealDomain):
Warning, these protected names have been redefined and unprotected:
Im, Re, ^, arccos, arccosh, arccot, arccoth, arccsc, arccsch, arcsec,
arcsech, arcsin, arcsinh, arctan, arctanh, cos, cosh, cot, coth, csc,
csch, eval, exp, expand, limit, ln, log, sec, sech, signum, simplify,
sin, sinh, solve, sqrt, surd, tan, tanh
> eval(arcsin(cosh(x)), x=Pi); # using RealDomain
 $\text{undefined}$ 
> :-eval(arcsin(cosh(x)), x=Pi); # global evaluator
 $\frac{\pi}{2} - \pi I$ 
> eval(:-arcsin(:-cosh(x)), x=Pi);
> # RealDomain evaluator, global symbols
 $\text{undefined}$ 
```

- **omitting validity checks**

The keyword `symbolic` in a call of a simplification procedure indicates that although the validity of a transformation cannot be proven, it is nevertheless carried out. The keyword `delay` has the opposite effect; if you cannot prove validity of a transformation, then return the expression unevaluated. In this case, there is a subtle difference between returning an expression unevaluated and encoding an answer using sign functions. The following example reveals this.

```
> expr := (x^2)^(1/2); # no automatic simplification
expr :=  $\sqrt{x^2}$ 
> simplify(expr, symbolic); # simplify regardless
 $x$ 
> simplify(expr, 'delay'); # valid simplify
 $\sqrt{x^2}$ 
> simplify(expr); # simplify with sign functions
csgn(x) x
```

- **restricting simplifications**

A Maple teaser:

```
> (4^x-1)/(2^x-1);

$$\frac{4^x - 1}{2^x - 1}$$

> {simplify(%), normal(%), combine(%, 'power')};


$$\left\{ \frac{4^x - 1}{2^x - 1} \right\}$$

```

No simplification procedure seems to recognize  $4^x$  as  $(2^x)^2$  so that the expression can be simplified into  $2^x + 1$ . For this, you first have to convert the expression from powers to exponential mappings and logarithms, and then follow a specific simplification track to get the job done.

```
> subs(4^x=exp(x*ln(4)), 2^x=exp(x*ln(2)), %);

$$\frac{e^{(x \ln(4))} - 1}{e^{(x \ln(2))} - 1}$$

> simplify(%, ln);

$$\frac{e^{(2 x \ln(2))} - 1}{e^{(x \ln(2))} - 1}$$

> simplify(%, exp);

$$\frac{2^{(2 x)} - 1}{2^x - 1}$$

> normal(%, 'expanded');

$$2^x + 1$$

```

In the procedures **combine**, **simplify**, and **convert**, you can add a second argument to restrict the type of simplification to a particular mathematical function or class of expressions. For the procedure **expand** it is the opposite: via extra arguments, you just inform Maple what mathematical functions or expressions should be left untouched during simplification.

```
> expr := ln(2*x)+sin(2*x);
expr := ln(2 x) + sin(2 x)
> expand(expr, ln);
ln(2 x) + 2 sin(x) cos(x)
> expand(expr, sin);
ln(2) + ln(x) + sin(2 x)
> expr := expr^2: % = expand(%, ln, sin);
(ln(2 x) + sin(2 x))^2 = ln(2 x)^2 + 2 ln(2 x) sin(2 x) + sin(2 x)^2
```

Use **frontend** when it is easier to specify which functions may be expanded than to inform Maple which functions must stay intact.

```
> frontend(expand, [expr]);
 $\ln(2x)^2 + 2\ln(2x)\sin(2x) + \sin(2x)^2$ 
> frontend(expand, [expr], [{}, {sin(2*x)}]);
 $\ln(2x)^2 + 4\sin(x)\cos(x)\ln(2x) + 4\sin(x)^2\cos(x)^2$ 
> frontend(expand, [expr], [{'+', '*'}, {trig}, {}]);
 $\ln(2x)^2 + 4\sin(x)\cos(x)\ln(2x) + 4\sin(x)^2\cos(x)^2$ 
```

**frontend** has an optional third argument of a list of two sets: first, a set of type names not to be frozen; second, a set of expressions not to be frozen (default is `[{`+`, `*`}, {}]`). So, in the last command we have specified to expand the expression while keeping all functions intact except trigonometric functions. The type names may include types that you yourself define in Maple. For example,

```
> # sine of 2x
> 'type/t' := z -> evalb(z=sin(2*x)):
> expr := sin(2*x)+sin(2*y)+sin(4*x);
 $expr := \sin(2x) + \sin(2y) + \sin(4x)$ 
> frontend(expand, [expr], [{'+', '*', t}, {}]);
 $2\sin(x)\cos(x) + \sin(2y) + \sin(4x)$ 
> # trigonometric function applied to a sum
> 'type/t' := trig('+'):
> expr := sin(2*x)+sin(3*x)+sin(x+y)+sin(u-v);
 $expr := \sin(2x) + \sin(3x) + \sin(x + y) - \sin(-u + v)$ 
> frontend(expand, [expr], [{'+', t}, {sin}]);
 $\sin(2x) + \sin(3x) + \sin(x)\cos(y) + \cos(x)\sin(y) - \sin(-u)\cos(v) - \cos(-u)\sin(v)$ 
> # trigonometric function applied to a product
> 'type/t' := trig('*'):
> frontend(expand, [expr], [{'*', t}, {sin}]);
 $2\sin(x)\cos(x) + 4\sin(x)\cos(x)^2 - \sin(x) + \sin(x + y) - \sin(-u + v)$ 
```

### • using expression size as main criterion of simplification

You can also ask Maple to simplify an expression with respect to its size; simplify ad the keyword **size** to the procedure **simplify**. The following example illustrates the difference with the ‘usual’ simplification.

```
> 1/(\sin(x)^2-1) + 1/(\sin(x)+1);
```

$$\frac{1}{\sin(x)^2 - 1} + \frac{1}{\sin(x) + 1}$$

```

> simplify(% , size);

$$\frac{\sin(x)}{\sin(x)^2 - 1}$$

> simplify(%%);

$$-\frac{\sin(x)}{\cos(x)^2}$$

> normal(%%%);

$$\frac{\sin(x)}{(\sin(x) + 1)(\sin(x) - 1)}$$


```

## 14.9 Defining Your Own Simplification Routines

In the example of partial trigonometric expansion in §14.2 we already touched upon the subject of programming your own expansion rules for functions. For a function, say **F**, you only have to define the procedure **expand/F**. Henceforth, when you apply **expand** to an expression containing **F**, Maple applies to each function call of **F** the expansion as defined in the procedure **expand/F**. An example of an additive function **F**:

```

> 'expand/F' := proc(x) local y,i:
> y := expand(x):
> if type(y,'+') then sum(F(op(i,y)), i=1..nops(y)) end if
> end proc:
> f(p+q) + F(r+s): % = expand(%);


$$f(p + q) + F(r + s) = f(p + q) + F(r) + F(s)$$

> F(p*(q+r)): % = expand(%);


$$F(p(q + r)) = F(pq) + F(pr)$$


```

As promised in the section about **expand** we shall illustrate how you can redefine the expansion routine of the hyperbolic tangent so that consistency with the corresponding trigonometric function is established, i.e., we are going to redefine the procedure **expand/tanh** so that it resembles **expand/tan**, and so that the built-in expansion routine is always overruled. It is a prototype of how to add your own procedures in a private library and have them carried out before attempting to use a built-in library routine.

For the implementation of **expand/tanh** we seek inspiration from the source code of the comparable procedure **expand/tan**.

```
> interface(verboseproc=3): print('expand/tan');
```

```

proc(y)
local x, n, t, i, S, c, N, D, T;
option'Copyright (c) 1995 Waterloo Maple Inc. All rights reserved.';
x := expand(y);
if x  $\neq$  y then return expand(tan(x)) end if;
if type(x, '+' ) then
    n := nops(x);
    t := [op(x)];
    try t := [seq(tan(i), i = t)]
    catch "singularity encountered" : return tan(convert(t, '+'))
    end try;
    t := expand(t);
    for i from 0 to n do
        c := combinat,choose(t, i);
        Si := convert(map(convert, c, '*'), '+');
        if 1 < irem(i, 4) then Si := -Si end if
    end do;
    N := convert([seq(S2*i-1, i = 1..iquo(n + 1, 2))], '+');
    D := convert([seq(S2*i, i = 0..iquo(n, 2))], '+');
    N / D
elif type(x, '*' ) and type(op(1, x), integer) then
    n := abs(op(1, x));
    t := tan(x/n);
    for i from 0 to n do
        Si := binomial(n, i) * ti; if 1 < irem(i, 4) then Si := -Si end if
    end do;
    N := convert([seq(S2*i-1, i = 1..iquo(n + 1, 2))], '+');
    D := convert([seq(S2*i, i = 0..iquo(n, 2))], '+');
    N / D
elif type(x, nonreal) then
    (I * tanh(Im(x)) + tan(Re(x))) / (1 - I * tanh(Im(x)) * tan(Re(x)))
elif type(x, '*' ) and type(op(1, x), nonreal) then
    expand(tan(Re(op(1, x)) * subsop(1 = 1, x)
    + I * Im(op(1, x)) * subsop(1 = 1, x)))
elif type(x, 'specfunc(anything, ln)') then
    - I * (op(1, x)(2*I) - 1) / (op(1, x)(2*I) + 1)
else tan(x)
end if
end proc

```

Probably this is a bit overwhelming. By inspection of a few expansions and careful reading of the source code it becomes clear that Maple applies the following rule:

$$\tan(x_1 + x_2 + \dots + x_n) = \frac{S_1 - S_3 + S_5 - S_7 + \dots + (-1)^{\lfloor \frac{n-1}{2} \rfloor} S_{2\lfloor \frac{n-1}{2} \rfloor + 1}}{1 - S_2 + S_4 - S_6 - \dots + (-1)^{\lfloor \frac{n}{2} \rfloor} S_{2\lfloor \frac{n}{2} \rfloor}},$$

where the  $S_i$  are the elementary symmetric polynomials in  $\tan(x_j)$ , which are given by

$$\prod_{i=1}^n (t - \tan(x_i)) = \sum_{i=0}^n (-1)^i S_i t^{(n-i)}.$$

It is not very difficult to verify that a similar formula holds for the hyperbolic tangent, viz.,

$$\tanh(x_1 + x_2 + \dots + x_n) = \frac{S_1 + S_3 + S_5 + S_7 + \dots + S_{2\lfloor\frac{n-1}{2}\rfloor+1}}{1 + S_2 + S_4 + S_6 + \dots + S_{2\lfloor\frac{n}{2}\rfloor}},$$

where the  $S_i$  are the elementary symmetric polynomials in  $\tanh(x_j)$ ; So, these are given by

$$\prod_{i=1}^n (t - \tanh(x_i)) = \sum_{i=0}^n (-1)^i S_i t^{(n-i)}.$$

We implement the above expansion formula for the hyperbolic cotangent in the same style as the procedure `expand/tan` and place the code in a Maple file, say `tanh`. The full contents of the file is as follows:

```

#
# where the S[i] are the symmetric polynomials in tanh(a[j]). 
#
# The code essentially writes down the formula (*) as a
# rational expression in expand( tanh(a[i]) ). 
# The result is not further expanded or normalized.
#
# > expand(tanh(x+y+z));
#
#      tanh(y) + tanh(z) + tanh(x) + tanh(y) tanh(z) tanh(x)
# -----
#      1 + tanh(y) tanh(z) + tanh(y) tanh(x) + tanh(z) tanh(x)
#
# > expand(tanh(2*x+2*y));
#
#      tanh(x)          tanh(y)
# 2 ----- + 2 -----
#           2           2
#      1 + tanh(x)      1 + tanh(y)
#
# -----
#      tanh(x) tanh(y)
# 1 + 4 -----
#           2           2
#      (1 + tanh(x)) (1 + tanh(y))
#
# Author: Andre Heck
# Date:   Feb/02

'expand/tanh' := proc(y)
local x, n, t, i, S, c, N, D, T;
x := expand(y);
if x <> y then return expand(tanh(x)) end if;
if type(x, '+') then
  n := nops(x);
  t := [op(x)];
  t := expand([seq(tanh(i), i=t)]);
  for i from 0 to n do
    c := combinat['choose'](t, i);
    S[i] := convert(map(convert, c, '*'), '+');
  end do;
  N := convert([seq(S[2*i-1], i=1..iquo(n+1,2))], '+');
  D := convert([seq(S[2*i], i=0..iquo(n,2))], '+');
  N/D # Don't expand the result
elif type(x, '*') and type(op(1,x), integer) then
  n := abs(op(1,x));
end if;
end proc;

```

```

t := tanh(x/n);
for i from 0 to n do
  S[i] := binomial(n,i)*t^i;
end do;
N := convert([seq(S[2*i-1], i=1..iquo(n+1,2))], '+');
D := convert([seq(S[2*i], i=0..iquo(n,2))], '+');
N/D # Don't expand the result
else tanh(x)
end if;
end proc:
```

But where are we going to store the file? For this, we mimic Maple's library organization. The main Maple library is archived into one file, **maple.lib**, in the library directory, which on a PC platform usually is

```

> libname;
"C:\Program Files\Maple 8/lib"
```

If you inspect this directory, you will actually see a Maple repository, i.e., a collection of three files named **maple.lib**, **maple.ind**, and **maple.hdb** in which all Maple code and help pages are stored. Let us create our own Maple repository, say in the directory **C:/mylib**, in which we shall place the Maple code and help page for the procedure **expand/tanh**. Actually we shall assume in the session below that this has already been done and that the file **tanh** with the above source code resides in a subdirectory **expand/src**. We also assume the existence of a text file called **expandtrigh** in the same subdirectory that contains the comments in the above source code. This will become the help page of the procedure **expand/tanh**. For the rest, we carry out the steps that are described in the summary at the end of the help page about managing Maple repositories (obtained by entering **?repository,management**).

First, we make the directory **C:/mylib** the current working directory (in Unix-style denoted by a dot) and we define an auxiliary function to list the contents of a repository

```

> currentdir("C:/mylib");
> listrep := r -> map2(op, 1, march('list', r));
```

Next, we create a Maple repository that can contain enough objects, say 100, and check that it is empty.

```

> march('create', ".", 100);
> listrep(".");
```

[]

We make sure that the procedure **savelib** will store contents in the private library. For this purpose, we assign the variable **savelibname** the name of our current directory and prepend it to **libname**.

```
> savelibname := ".";
> libname := savelibname, libname:
```

We are ready to read the Maple source code and verify that the procedure **expand/tanh** is placed in the repository.

```
> read "expand/src/tanh":
> savelib(''expand/tanh''): # save in internal Maple format
> listrep(".");
[“expand/tanh.m”]
```

Let us finally create the `maple.hdb` part of the repository, containing the help page of **expand/tanh**.

```
> makehelp('expand/tanh',
> convert(".") || "/expand/src/expandtrigh.txt", name),
> convert(".", name) );
```

Now everything is ready for use in a fresh Maple session, at least when our private library is first in the list of Maple repositories that the system will search through in an attempt to find appropriate code when required. The proof of the pudding is, of course, in the eating.

```
> restart;
> libname := "C:/mylib", libname:
> tanh(5*x); % = expand(%);
```

$$\tanh(5x) = \frac{5\tanh(x) + 10\tanh(x)^3 + \tanh(x)^5}{1 + 10\tanh(x)^2 + 5\tanh(x)^4}$$

The following command will display our help page for the procedure **expand/tanh** inside the Maple help browser. Of course, a Maple worksheet instead of an ordinary text file would have been better.

```
> ?expand,tanh
```

The above method of creating a Maple repository for personal Maple code works fine for one or two procedures, but it becomes cumbersome when you have a large collection of procedures that are regularly updated. For this purpose, Maple provides the `LibraryTools` package. The instructions below show how we could already have used this package to create the private library for our **expand/tanh** procedure. In this case, the only advantage is that the specific creation of a repository is not needed anymore; the procedure of the `LibraryTools` package takes care.

```
> restart;
> with(LibraryTools);

[AddFromDirectory, BuildFromDirectory, DeleteFromLibrary,
SaveToLibrary, UpdateFromDirectory]

> listrep := r -> map2(op, 1, march('list', r)):
> currentdir("C:/mylib"): savelibname := ".";
> read "expand/src/tanh":
> SaveToLibrary(''expand/tanh''): # save in internal format
```

```

> makehelp('expand/tanh',
>   convert("." || "/expand/src/expandtrigh.txt", name),
>   convert(".", name) ):
> listrep(".");

```

[“expand/tanh.m”]

## 14.10 Exercises

1. Show that  $\sum_{k=1}^{\infty} \frac{1}{k^2 + 1} = \frac{\pi}{2} \coth(\pi) - \frac{1}{2}$ .
2. Expand the expression  $(\cos(2x) + 1)^2$  into  $\cos^2(2x) + 2\cos(2x) + 1$ , i.e., expand the expression as if it were a polynomial in  $\cos(2x)$  and do not expand the trigonometric function.
3. Check how the following pairs of symbolic expressions can be transformed into each other by Maple.
  - $x + y + \frac{1}{x + y}$  and  $\frac{(x + y)^2 + 1}{x + y}$
  - $\exp(x + y)$  and  $\exp(x)\exp(y)$
  - $\ln(x/y)$  and  $\ln(x) - \ln(y)$
  - $x^{(y+z)}$  and  $x^y x^z$
  - $\sqrt{x^2 - 1}$  and  $\sqrt{x - 1}\sqrt{x + 1}$
4. Simplify the following symbolic expressions.
  - $\frac{e^x + x}{e^{2x} + 2xe^x + x^2}$
  - $\sqrt[3]{x^5 + 40x^4 + 595x^3 + 3905x^2 + 9680x + 1331}$
  - $\frac{(x - 2)^{3/2}}{(x^2 - 4x + 4)^{1/4}}$
  - $\frac{\sqrt{x} - y}{x - y^2}$
  - $\frac{1}{2 + 5^{1/3}}$
  - $\cos(x + y) + \sin x \sin y + 2^{x+y}$
  - $2\cos^2 x - \cos 2x$
5. Solve the following zero-equivalence problem with Maple.
  - $(2^{1/3} + 4^{1/3})^3 - 6(2^{1/3} + 4^{1/3}) - 6 = 0$
  - $\ln \tan(\frac{1}{2}x + \frac{1}{4}\pi) - \operatorname{arcsinh} \tan x = 0$
6. Use Maple to check the following trigonometric identities.
  - $\sin x + \sin y = 2 \sin \frac{1}{2}(x + y) \cos \frac{1}{2}(x - y)$
  - $\sin 5x = 5 \sin x - 20 \sin^3 x + 16 \sin^5 x$
  - $\cot^2 x + 1 = \csc^2 x$
  - $\tan x + \tan y = \frac{\sin(x + y)}{\cos x \cos y}$
  - $\cos^6 x + \sin^6 x = 1 - 3 \sin^2 x \cos^2 x$

$$(f) \sinh 2x = 2 \frac{\tanh x}{1 - \tanh^2 x}$$

$$(g) \frac{\sin 2x + \sin 2y}{\cos 2x + \cos 2y} = \tan(x + y)$$

7. Verify with Maple the equality  $\pi/4 = 4 \arctan(1/5) - \arctan(1/239)$ .

8. Compute the following indefinite integrals and check the answers through differentiation and simplification.

$$(a) \int \frac{2}{\sqrt{4+x^2}} dx$$

$$(b) \int \sqrt{(1-cx^2)^3} dx$$

$$(c) \int \frac{1}{x^4-1} dx$$

$$(d) \int \frac{1}{x^4-4} dx$$

$$(e) \int \sin 3x \cos 2x dx$$

9. Integrate the real function  $x \mapsto \frac{1}{(ax+b)^2(cx+d)^2}$  and bring the result into the following form.

$$\frac{2ac \ln \left( \frac{cx+d}{ax+b} \right)}{(ad-bc)^3} - \frac{2acx + ad + bc}{(ad-bc)^2(ax+b)(cx+d)}$$

10. Implement the expansion routine **expand/coth** such that it resembles **expand/cot**, i.e., **expand** should expand hyperbolic cotangents in terms of hyperbolic cotangents instead of hyperbolic sines and cosines. Set things up so that your expansion routine overrules the built-in procedure.

## 14.11 Simplification Chart

Generic Simplification Chart			
procedure	trigonometric functions	exp and log	powers
expand	$\cos(x+y) \rightarrow \cos x \cos y - \sin x \sin y$	$\exp(x+y) \rightarrow \exp x \exp y$	$x^{(y+z)} \rightarrow x^y x^z$
	$\cos 2x \rightarrow 2 \cos^2 x - 1$	$\ln(2x) \rightarrow \ln 2 + \ln x$	$(2x)^y \rightarrow 2^y x^y$
	$\cosh 3x \rightarrow 4 \cosh^3 x - 3 \cosh x$	$\ln(x/2) \rightarrow \ln x - \ln 2$	$(x/2)^y \rightarrow x^y (1/2)^y$
combine	$\cos x \cos y - \sin x \sin y \rightarrow \cos(x+y)$	$\exp x \exp y \rightarrow \exp(x+y)$	$x^y x^z \rightarrow x^{(y+z)}$
	$2 \sinh x \cosh x \rightarrow \sinh 2x$	$\ln 2 + \ln x \rightarrow \ln(2x)$	$(x^y)^z \rightarrow x^{yz}$
	$4 \cos^3 x \rightarrow \cos 3x + 3 \cos x$	$2 \ln 3 \rightarrow \ln(9)$	$\sqrt{x+1} \sqrt{x} \rightarrow \sqrt{x^2+x}$
simplify	$\cos^2 x + \sin^2 x \rightarrow 1$	$\exp x \exp y \rightarrow \exp(x+y)$	$x^y x^z \rightarrow x^{(y+z)}$
	$\cosh^2 x - \sinh^2 x \rightarrow 1$	$\ln(2x) \rightarrow \ln 2 + \ln x$	$(x/2)^y \rightarrow x^y 2^{-y}$
	$\tan x \rightarrow \sin x / \cos x$	$\ln(\pi^3) \rightarrow 3 \ln \pi$	$\sqrt{\pi^2 + 2\pi + 1} \rightarrow \pi + 1$
convert	$\cos x \rightarrow (e^{ix} + e^{-ix})/2$	$e^{ix} \leftrightarrow \cos x + i \sin x$	$\sqrt{a} \leftrightarrow \text{RootOf}(-Z^2 - a)$
	$\text{arcsinh } x \rightarrow \ln(x + \sqrt{x^2 + 1})$		$\binom{n}{k} \rightarrow \frac{n!}{k!(n-k)!}$
	$\sin 2x \rightarrow 2 \tan x / (1 + \tan^2 x)$		and other type conversions

# 15

## Graphics

Two-dimensional graphics include

- curves defined by functions of a single real variable,
- curves defined by parametric equations,
- implicit curves defined by an equation,
- contour plots and density plots of functions in two real variables and of data,
- plots of vector fields and gradient fields,
- data plots and statistical plots,
- plots of regions defined by linear inequalities,
- geometrical plots (such as polygons, circles, ellipses, etc.), and
- animation of two-dimensional graphics objects.

Maple provides three-dimensional graphics facilities to

- generate surfaces defined by functions of two real variables,
- generate space curves, tubes, and surfaces defined by parametric equations,
- generate implicit surfaces defined by an equation,
- generate surfaces from lists of three-dimensional data points,
- draw geometrical objects such as regular polytopes, cylinders, spheres, and tori, and
- show an animation of three-dimensional graphics objects.

When producing a two- or three-dimensional plot, Maple makes decisions about the number of sample points, positions of axes and tick marks, ranges of values to display, shading or coloring of the graph, and so on. You can modify graphs by using various options, such as choosing another coordinate system (polar, elliptic, logarithmic, spherical, cylindrical, paraboloidal, etc.), or changing the grid size of a surface.

Procedures like **plot** and **plot3d** for plotting functions and surfaces, respectively, can be called in a straightforward way. More graphics capabilities are available via the **plots** package.

*We shall assume that the plots package has been loaded in all sample sessions of this chapter.*

This is done as follows:

```
> with(plots); # load graphics package
```

**Warning, the name changecoords has been redefined**

```
[animate, animate3d, animatecurve, arrow, changecoords, complexplot,
complexplot3d, conformal, conformal3d, contourplot, contourplot3d,
coordplot, coordplot3d, cylinderplot, densityplot, display, display3d,
fieldplot, fieldplot3d, gradplot, gradplot3d, implicitplot,
implicitplot3d, inequal, listcontplot, listcontplot3d, listdensityplot,
listplot, listplot3d, loglogplot, logplot, matrixplot, odeplot, pareto,
pointplot, pointplot3d, polarplot, polygonplot, polygonplot3d,
polyhedra_supported, polyhedraplot, replot, rootlocus, semilogplot,
setoptions, setoptions3d, spacecurve, sparsematrixplot, sphereplot,
surfdata, textplot, textplot3d, tubeplot]
```

This list of names may give you an idea of what special two-dimensional and three-dimensional graphics have been made readily available in Maple. The most important command from the **plots** package is **display**. It not only allows you to redraw a plot under different options, but also makes it possible to combine plots. For novice users, the **interactive** procedure is the second most important command: it provides a special GUI for setting plot options.

In this chapter, we shall describe the graphics facilities of Maple 8 on a PC with the worksheet interface. However, most of the graphics examples are also possible when using a different user interface. We shall not treat every possible graphical routine and display option, but put stress on understanding the graphical structure underneath the plotting routines.

*In the sample sessions we shall quite often place color information in the commands, but the pictures will appear in the text book in black-and-white.*

You will have to redo the examples yourself to see the color rendering. Anyway, you are invited to experiment with the graphics facilities of Maple. But please take into account the following:

*Some examples in this chapter require quite some computer resources and are time-consuming.*

## 15.1 Some Basic Two-Dimensional Plots

Maple provides the procedure **plot** for graphing a function in one variable. Maple must of course know what kind of plotting or output device you are using. If you run Maple on a Macintosh, on a PC, or on a Unix type computer with the worksheet user interface, then the system selects the display device and display driver automatically. Otherwise, you must inform Maple and, if necessary, allow it to send special character sequences for switching from text mode to graphics mode.

For example, if you want to use **plot** to create a 24-bit JPEG rendering of the sine graph stored in a file called `sine.jpg`, then you must enter

```
> plotsetup(jpeg, plotoutput="sine.jpg"):  
> plot(sin(x), x=0..2*Pi);
```

The file can be used for a webpage. To return to the default display device under the user interface being used, enter the following command.

```
> plotsetup(default);
```

Henceforth, we shall assume that Maple runs on a PC using the worksheet interface because this display driver will also meet the demands for producing three-dimensional plots.

Consider the formula  $f$  given by  $e^{-x^2} \sin(\pi x^3)$  on the interval  $(-2, 2)$ .

```
> f := exp(-x^2) * sin(Pi*x^3);
```

$$f := e^{-x^2} \sin(\pi x^3)$$

The plot of this formula on the given domain is invoked by the command

```
> plot(f, x=-2..2);
```

If the plot device has been set to **window** using the **plotsetup** command, then the plot will appear in a separate window within the Maple application window; the menu allows you to display and manipulate the Maple plot data structure. As shown in the screen dump (Figure 15.1), the default behavior is that the picture is embedded in the worksheet. In the same screen dump you see the special GUI invoked by the procedure **interactive** from the **plots** package.

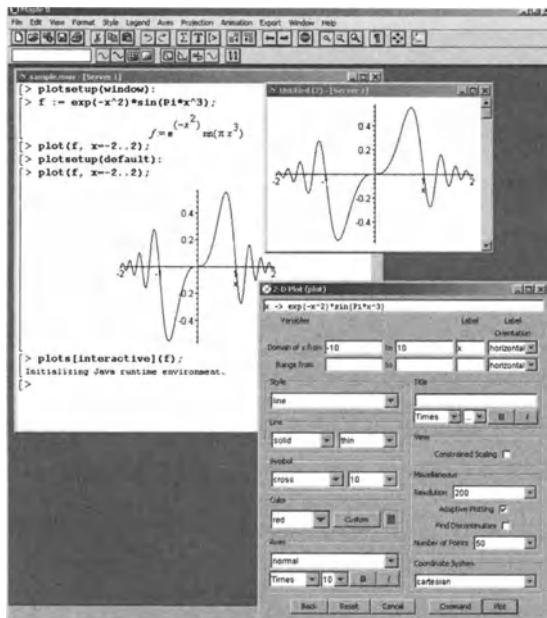


Figure 15.1. Screen dump of worksheet, plot window, and special GUI.

The above command is an example of the following general format for drawing a *formula*  $f$  in  $x$  over the interval  $(a, b)$

$\text{plot}(f, x = a .. b, \text{options});$

where  $a .. b$  is the horizontal range (of  $x$ ), and where *options* describes zero or more options, as will be discussed in the next section.

An alternative way to obtain the graph of a *function*  $f$  over a the interval  $(a, b)$  is

$\text{plot}(f, a .. b, \text{options});$

where *options* describes zero or more options. Note the difference in specifying the horizontal range when plotting a *function* and a *formula* — a function only needs a *range*, whereas a formula needs an equation of the form *variable* = *range*.

So, to plot the formula  $e^{-x^2} \sin(\pi x^3)$  for  $x$  ranging from  $-2$  to  $2$ , you can also enter

```
> f := x -> exp(-x^2)*sin(Pi*x^3);
> plot(f, -2..2);
```

The graph differs from Figure 15.1 only in the absence of the domain variable  $x$  near the  $x$ -axis.

You can also look at the graph of the function over the entire real line or over a half-line. In Figure 15.2 we display the graph of  $f$  over  $(0, \infty)$ .

```
> plot(f, 0..infinity);
```

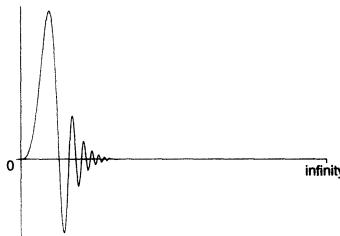


Figure 15.2. Infinity plot.

In this case, Maple transforms the entire real line into the interval  $(-1, 1)$  by a function that approximates  $x \mapsto \frac{2}{\pi} \arctan(\frac{x}{2\pi})$ .

In Figure 15.3 we plot more than one formula at the same time, and on a color display Maple will choose different colors for the graphics objects.

```
> plot({f(x), exp(-x^2), -exp(-x^2)}, x=-2..2);
```

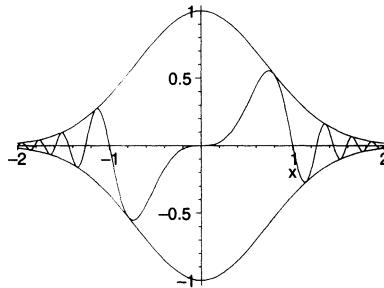


Figure 15.3. Graph of  $e^{-x^2} \sin(\pi x^3)$ ,  $e^{-x^2}$ , and  $-e^{-x^2}$ .

If you want to store the plot in Encapsulated PostScript format in some file, place the cursor in the image, right click, and choose from the **Export As** menu the EPS item. A dialog window will allow you to specify the name and location of the file. If you know beforehand what you want, you can also enter the following command.

```
> plotsetup(PostScript, plotoutput="plotfile.eps",
> plotoptions="portrait,noborder,height=3in,width=4.5in");
```

Henceforth, Maple directs the Encapsulated PostScript code that describes the picture to the file `plotfile.eps` in portrait orientation, without a plot border, and of size  $4.5 \times 3$  (dimensions of width and height are in inches). You can reset the display device to the default setting for the user interface being used by

```
> plotsetup(default);
```

You can simply print the file `plotfile.eps` on a PostScript compatible printer or embed it in a text made with a typesetting system such as `LATeX`

or Framemaker. Without the above **plotoptions**, the graph is printed in landscape mode, with a box around the plot, and the picture is scaled so that it fits maximally on a page of A4 format. Other common plot devices supported by Maple are **wmf** (for Windows Metafile), **bmp** (for Windows BMP), and **gif** (GIF format). Enter **?plot,device** for more details.

The general format for drawing a two-dimensional plane curve defined in the Cartesian coordinate system by *parametric formulas*  $x = f(t)$ ,  $y = g(t)$  is

```
plot([f(t), g(t), t = a .. b], options);
```

where  $a .. b$  is the range of the independent variable  $t$ , and where *options* describes zero or more options. A simple example is shown in Figure 15.4.

```
> plot([sin(t), t, t=0..2*Pi], scaling=constrained);
```

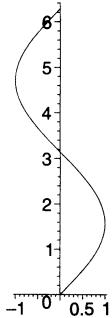


Figure 15.4. Parametric plot of curve  $t \mapsto (\sin t, t)$ .

When a parametric curve is defined by *functions*, the independent variable must be omitted in the **plot** command. A funny example of a function-defined parametric curve is shown in Figure 15.5: a look-alike of a Maple leaf. It is produced as a polar plot.

```
> S := t -> 100/(100+(t-Pi/2)^8): # for scaling
> R := t -> S(t)*(2-sin(7*t)-cos(30*t)/2):
> plot([R, t->t, -Pi/2..3/2*Pi], coords=polar, axes=none,
> color=green, thickness=3, numpoints=500);
```

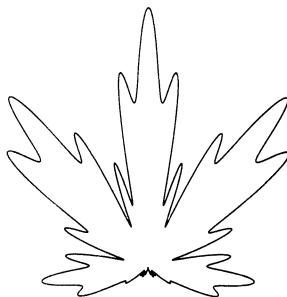


Figure 15.5. Maple leaf created as parametric plot.

## 15.2 Options of **plot**

When Maple plots a graph, it makes many choices. For example, it chooses the depicted range of the graph, sample points to make a smooth curve, which tick marks to show, and so on. Maple takes care of choosing values for these options that are most convenient in daily use. However, you can customize it to your needs.

### Vertical Range

For example, to restrict the vertical range of a graph, you can add the range information to the **plot** command as a third argument. With the following command you plot the graph of the function defined by the formula  $\frac{\sin^2 x}{x^2}$  over the interval  $(-6, 6)$ , where parts of the graph outside the vertical range  $[0, 1]$  are not shown (Figure 15.6).

```
> plot(sin(x^2)/x^2, x=-6..6, 0..1);
```

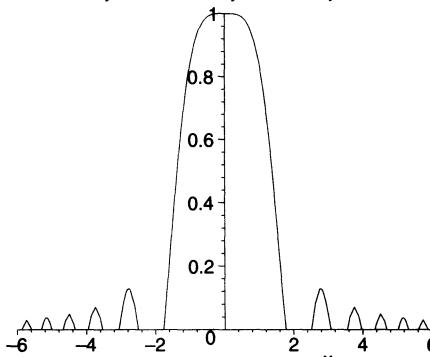


Figure 15.6. Restricted plot of  $\frac{\sin(x^2)}{x^2}$ .

### Scaling

Note that Maple chooses the vertical scale that allows the largest display of the plot in A4 size. When you want the same horizontal as vertical scale you can change this interactively or add the option **scaling = constrained** to the **plot** command (Figure 15.7).

```
> plot(sin(x^2)/x^2, x=-6..6, scaling=constrained);
```

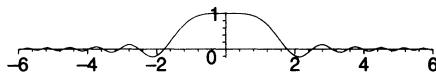


Figure 15.7. Restricted plot of  $\frac{\sin(x^2)}{x^2}$  with equal scaling.

### View

Sometimes, you must restrict the vertical range to get a good picture. In the graph in Figure 15.8, you see that a few function values dominate over

all the others so that the behavior of the tangent function over the given domain is not displayed very well. Often the only way to avoid (spurious) spikes in a graph is to specify a “reasonable” vertical range or to increase the number of sample points.

```
> plot(tan, -Pi..Pi);
```

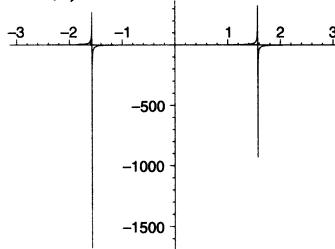


Figure 15.8. Bad plot of tangent.

With the following command you replot the graph of the tangent over the interval  $(-\pi, \pi)$ , where parts of the graph outside the vertical range  $[-10, 10]$  are not shown, and the horizontal view has been extended to  $[-4, 4]$ . However, the graph itself is not computed again; only the rendering is redone (Figure 15.9).

```
> display(% , view=[-4..4,-10..10]);
```

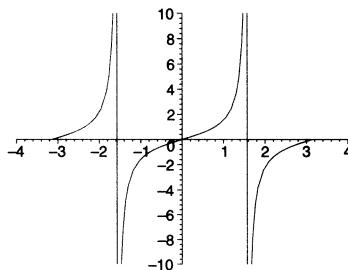


Figure 15.9. Improved plot of tangent.

## Discontinuities

The vertical asymptotic lines at  $x = -\frac{\pi}{2}$  and  $x = \frac{\pi}{2}$  are actually present because Maple does not take care of the discontinuities and draws a line between a very high and a very low sample point. You may let Maple try to overcome the discontinuity by adding the option `discont=true`. But in this case you must work with an expression as Maple needs a variable for locating the discontinuities. Figure 15.10 shows the improved graph of the tangent. Note that we use the symbol font for the tick marks at the axes to get Greek symbols.

```
> plot(tan(x), x=-Pi..Pi, -10..10, discont=true,
>       xtickmarks=['-3.14="p", -1.57="-p/2",
>                   1.57="p/2", 3.14="p"], axesfont=[SYMBOL,12]);
```

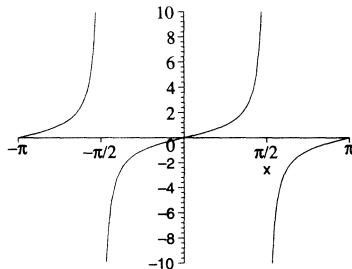
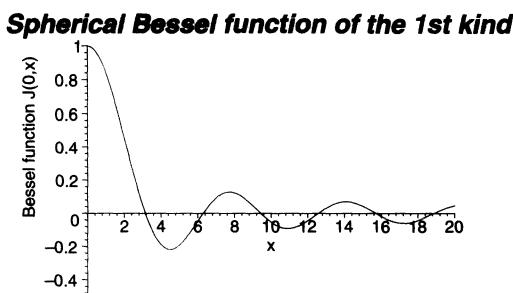


Figure 15.10. Best plot of tangent.

### Labels, Tick Marks, and a Title

In the above plot, we also gave the horizontal axes more meaningful labels via the option `xtickmarks`. Of course `ytickmarks` exists as well. Instead of explicit labeling, you can just specify the number of tick marks in each direction. You may label a graph, choose the direction in which labels are printed along the axes, and specify the font that you want to use (see Figure 15.11).

```
> J := (n,x) -> sqrt(Pi/(2*x)) * BesselJ(n+1/2,x):
> plot(J(0,x), x=0..20, 'Bessel function J(0,x)'=-0.5..1,
>       xtickmarks=8, ytickmarks=4, title=
>       "Spherical Bessel function of the 1st kind",
>       titlefont=[HELVETICA,BOLDOblique,16],
>       labeldirections=[horizontal, vertical]);
```

Figure 15.11. Plot of spherical Bessel function  $J_0(x)$  of the first kind.

When labeling is done in the above way, you cannot do much about the placement of the labels on the axes but make specific changes in the PostScript code of the plot itself. Only the direction in which labels are printed along the axes can be specified. There is an alternative: you may use the commands `textplot` and `display` from the `plots` package to draw one plot containing text on a specific position, to draw one plot containing the curve and axes, and to display these two plots in one picture (Figure 15.12).

```
> curve := plot(x->J(0,x), 0..20, -0.5..1,
>               xtickmarks=8, ytickmarks=4, title=
>               "Spherical Bessel function of the 1st kind",
```

```

> titlefont=[HELVETICA,BOLDOblique,16],
> axesfont=[HELVETICA,BOLD,14]):
> text := plots[textplot]({[2,0.75,"J(0,x)"],
> [14,-0.1,"x"]}, align={ABOVE,RIGHT},
> font=[HELVETICA,BOLD,14]):
> display({curve,text});

```

**Spherical Bessel function of the 1st kind**

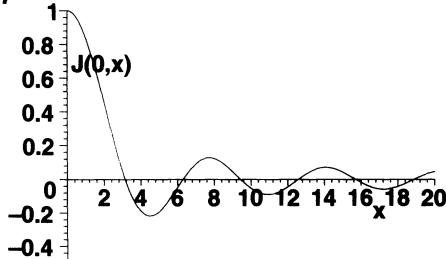


Figure 15.12. Another positioning and style of text in a graph of  $J_0(x)$ .

The fonts may be chosen from a small set of families, styles, and sizes.

### Sampling

When graphing a function, Maple first computes a number of points of the graph — the number of sample points can be set via the option `numpoints` or the option `sample` — and, by default, connects these points by straight lines. As you will see in the next section, Maple will use so-called *adaptive plotting* to make the graphs as smooth as possible; this means that the system increases automatically the number of sample points if necessary to create smooth curves.

### Plot Style

Instead of using the default plot style `patch`, you may select other ones like `point` or `line` (only affecting drawings of polygons). If you select the `point` style, Maple will draw crosses centered around the computed points on a curve and around computed vertices of polygons. You can choose another `symbol` and `symbolsize` (Figure 15.13).

```

> plot(sin, 0..Pi, scaling=constrained, style=point,
>       symbol=circle, symbolsize=20);

```

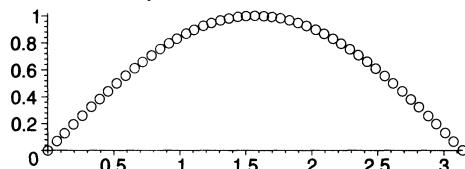


Figure 15.13. Point plot of sine function with large circles.

Data plots can easily be produced with point plot style; you put the data points in a list and plot the points with the `point` style. For example, to plot the first nine prime numbers (Figure 15.14) you can do the following.

```
> plotpoints := [seq([i,ithprime(i)], i=1..9)]:
> plot(plotpoints, style=point, symbol=box);
```

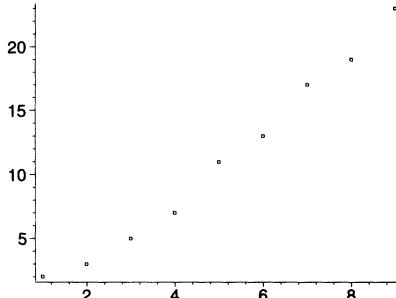


Figure 15.14. Data plot of first 9 prime numbers.

In §15.13 we shall look at various built-in procedures for data plotting that extend the above basic plotting.

As an example of the default `line` style we consider a graphical trace of 20 iterations of the cosine function with starting value 1.2. First, we have to generate the list of points.

```
[ [1.2, 1.2], [1.2, cos(1.2)], [cos(1.2), cos(1.2)], [cos(1.2), cos(cos(1.2))],
  [cos(cos(1.2)), cos(cos(1.2))], [cos(cos(1.2)), cos(cos(cos(1.2)))], ... ]
```

We produce the plot points as pairs of consecutive points starting with

```
> p0 := [[1.2, 1.2], [1.2, cos(1.2)]]; # starting pair
```

```
p0 := [[1.2, 1.2], [1.2, 0.3623577545]]
```

The iteration function to create new pairs of points is

```
> f := p -> [map(cos, p[1]),
   map(cos, p[2])]; # iteration
```

```
f := p -> [map(cos, p1), map(cos, p2)]
```

To see the effect,

```
> f(p0); # first new pair
```

```
[[0.3623577545, 0.3623577545], [0.3623577545, 0.9350636470]]
```

which is the numerical evaluation of the pair

$$\left[ [\cos(1.2), \cos(1.2)], [\cos(1.2), \cos(\cos(1.2))] \right].$$

Now, we can generate all points.

```
> points := map(op, [seq((f@@i)(p0), i=0..10)]):
```

Finally, we generate plots of this list of points connected by straight lines, of the identity function, and of the cosine function on the interval  $(0, \pi/2)$  and with vertical range  $[0, 1]$ .

```

> curveplot := plot(points, x=0..Pi/2, y=0..1,
>   style=line, linestyle=2, color=blue):
> identityplot := plot(x, x=0..Pi/2, y=0..1,
>   linestyle=7, color=gray):
> cosineplot := plot(cos(x), x=0..Pi/2, y=0..1,
>   thickness=4, color=red):

```

We show the plots in one picture with the command **display** from the **plots** package. In the above commands, we chose different colors, line styles (dashing), and drew the cosine function thicker so that the components of the picture can easily be distinguished (Figure 15.15).

```

> display({curveplot, identityplot, cosineplot},
>   title="cobweb-model of iterated cosine");

```

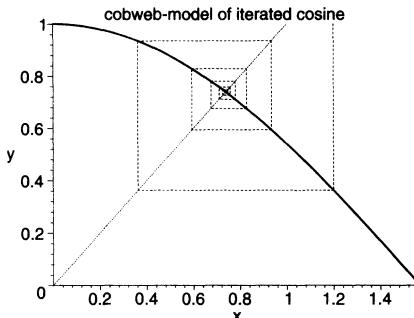


Figure 15.15. Cobweb-model of iterated cosine.

### Style and Thickness of Lines

Let us investigate what line styles and thicknesses are available. At present, the options **linestyle** and **thickness** can have nonnegative integer values in the ranges [0,7] and [0,15], respectively (modular arithmetic forces the values into the proper range; see Figures 15.16 and 15.17)

```

> for i from 0 to 15 do
>   thick[i] := plot(i, x=0..1, thickness=i)
> end do:
> display(convert(thick, set),
>   axes=box, tickmarks=[0..15],
>   title="thickness option: [0,1,...,15]");

```

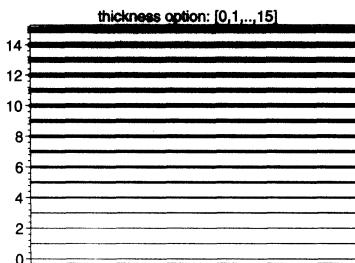


Figure 15.16. Thickness of lines in a plot.

```
> for i from 0 to 7 do
>   line[i] := plot(i, x=0..1, linestyle=i)
> end do:
> display(convert(line, set), axes=box,
>   tickmarks=[0..7], title="linestyle option: [0,1,...,7]");
linestyle option: [0,1,...,7]
```

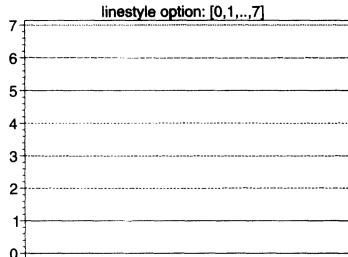


Figure 15.17. Line styles in a plot.

Instead of the numbers 1, 2, 3, and 4 you can also use the texts **SOLID**, **DOT**, **DASH**, and **DASHDOT**, respectively.

### Colors

Maple uses two color models: **RGB** and **HSV**. In **RGB** mode, the *red*, *green*, and *blue* components of a color are set. With the **polygonplot** routine from the **plots** package you can easily create color charts.

```
> P := seq(seq(polygonplot(
>   [[i,j], [i+1,j], [i+1,j+1], [i,j+1]],
>   color=COLOR(RGB, 0.1*i, 0.1*j, 0)),
>   i=0..10), j=0..10):
> display({P}, scaling=constrained,
>   labels=[R,G], title="RGB color (R, G, 0)",
>   tickmarks = [[seq(i+0.5=".||i, i=0..9), 10.5="1"],
>   [seq(j+0.5=".||j, j=0..9), 10.5="1"]]);
```

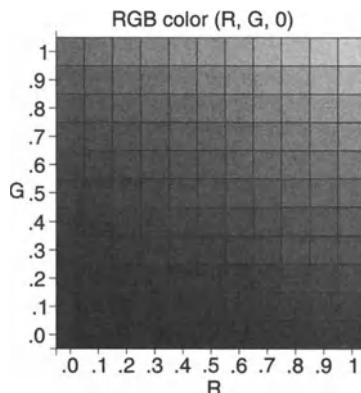


Figure 15.18. Projection of a RGB color chart on a gray chart.

Unfortunately, Figure 15.18 only shows the projection of the colors on gray shadings. However, on the computer screen or on a color printer you

should get what you asked for. Figure 15.19 is another picture showing the RGB color model in three dimensions along the edges of a cube.

```
> with(plottools,cuboid):
> e1 := seq(seq(display(
>   cuboid([i,j,k],[i+1,j+1,k+1]),
>   color=COLOR(RGB, 0.1*i, 0.1*j, 0.1*k), style=patch),
>   i=0..10), j=[0..10]), k=[0..10]):
> e2 := seq(seq(display(
>   cuboid([i,j,k],[i+1,j+1,k+1]),
>   color=COLOR(RGB, 0.1*i, 0.1*j, 0.1*k ), style=patch),
>   i=[0..10], j=0..10), k=[0..10]):
> e3 := seq(display(
>   cuboid([i,j,k],[i+1,j+1,k+1]),
>   color=COLOR(RGB, 0.1*i, 0.1*j, 0.1*k ), style=patch),
>   i=[0..10], j=[0..10]), k=0..10):
> display({e1,e2,e3});
```

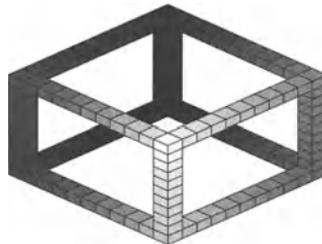


Figure 15.19. RGB colors along the edges of a cube.

In HSV mode, the *hue* (or color), *saturation*, and *value* components of a color are set; the last two components indicate the amount of white and black color to add to the color to obtain different shades, tints, and tones. HSV is circular in its argument and provides a cylindrical color space. In Maple, HUE(*h*) can be used instead of COLOR(HSV, *h*, 0.9, 1.0). In Figures 15.20 and 15.21 the HUE color chart is displayed in two ways, viz., as a circular band and as a color disk.

```
> plot3d(1, t=0..2*Pi, z=-2..2, coords=cylindrical,
>        style=patchnogrid, color=t/2/Pi, axes=box,
>        orientation=[45,35], tickmarks=[3,3,0]);
```

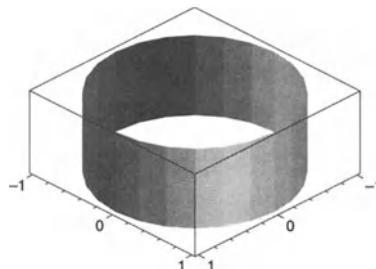


Figure 15.20. HUE colors on a circular band.

```
> P := seq(display(plottools[pieslice](
> [0,0], 5, Pi*i/10..Pi*(i+1)/10,
> color=COLOR(HUE,evalf(i/20))),
> scaling=constrained), i=0..20):
> display({P}, axes=none);
```

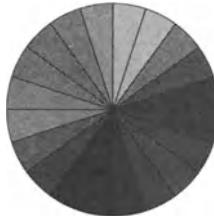


Figure 15.21. HUE color disk.

## Coordinate Systems

Until now, we have used the Cartesian coordinate system in most plots. But you have in fact a large choice of coordinate systems in Maple. An example of a parametric curve in polar coordinates (Figure 15.22):

```
> plot([sin(t), t, t=0..2*Pi], coords=polar,
> scaling=constrained);
```

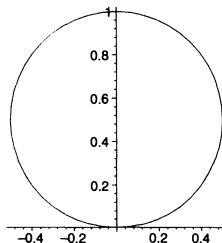


Figure 15.22. Parametric curve in polar coordinates.

You can use the **changecoords** routine from the **plots** package to alter an existing plot structure, which is considered in Cartesian coordinates, to a new coordinate system (Figure 15.23).

```
> P := %: # give the previous plot a name
> changecoords(P, elliptic);
```

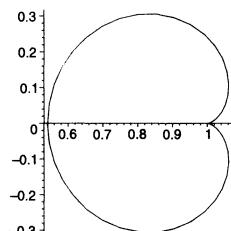


Figure 15.23. Plot of the curve after a change of coordinates.

Notice that this is not the same as

```
> plot([sin(t), t, t=0..2*Pi], coords=elliptic,
>       scaling=constrained);
```

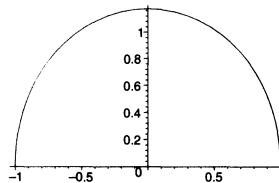


Figure 15.24. Plot of the curve in elliptic coordinates.

**changecoords** considers an existing plot as one created in the default coordinate system (usually Cartesian) and transforms the 2D-plot data structure to the new coordinate system. Thus, **changecoords** does not take into account that the 2D-plot data structure may have been created by using a different coordinate system (as in Figure 15.24).

### Default Options

If you want to inspect the default setting of an plot option you can use the **setoptions** command from the **plots** package.

```
> restart; with(plots):
> setoptions(numpoints);
```

49

Its main purpose is however to change the default value of an option for two-dimensional plotting. For example, if you always want to use a **frame** instead of **normal** axes for 2D-plots, and if you always want the same horizontal and vertical scaling, then you enter

```
> setoptions(axes=frame, scaling=constrained):
```

or add this line into your initialization file. You can always overrule the default by adding the option in a 2D-plot command as the next example illustrates (Figures 15.25, 15.26, and 15.27).

```
> plot(ln(x), x=1/2..2);
```

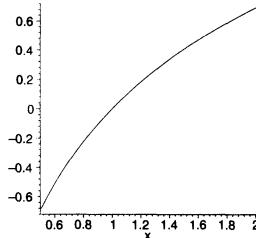


Figure 15.25. Plot of natural logarithm on segment  $(\frac{1}{2}, 2)$ .

```
> display(%); # replot with normal axes
```

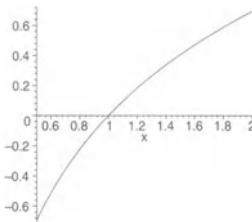


Figure 15.26. Plot of logarithm with axes style changed.

```
> # replot with new display area and equal scaling
> display(%%, view=[0..2,-2..2], scaling=unconstrained);
```

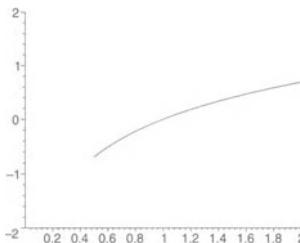


Figure 15.27. Plot of logarithm with view frame and scaling changed.

Note that this overruling of options in the **display** command *only* works for those options that do not need recomputation of the graphics object. For example, you cannot expect to change the number of sample points via **numpoints** without recomputing the graphics object.

Many options of two-dimensional plotting routines can be found in the on-line help system via the command **?plot,options**. At the end of this chapter we shall list all **plot** options that are available. Some of the options in this list have not been discussed yet, but they will be used in examples later in this chapter. The options **style**, **linestyle**, **thickness**, **symbol**, **symbolsize**, **axes**, **legend**, and **projection** can be changed interactively in the worksheet interface. Others must be adjusted in a plot command.

### Plotting More Than One Function Using Different Options

If you want to plot several functions in one picture using options like color or line style to distinguish them, then you can either build the picture from separate plots or you can place the functions as well as the options in a list. In the latter case, the first list element of a plot option specifies the choice for the first function, the second list element of a plot option specifies the choice for the second function, and so on.

An example says more than words. In Figure 15.28 we plot in one picture the sine graph as a red curve and the cosine graph as a thick blue curve. The legend indicates which is which in the plot.

```
> plot([sin(x), cos(x)], x=0..2*Pi,
>       color=[red,blue], thickness=[1,3],
>       legend=["sine","cosine"]);
```

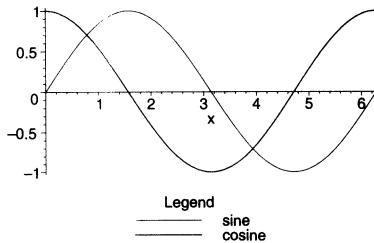


Figure 15.28. Two curves in one picture using different options in one command.

### 15.3 The Structure of Two-Dimensional Graphics

To know how reliable the plotting facilities of Maple are, it is important to have a good idea of how plotting is done. The making of a plot proceeds in two phases. In the first phase, the plot points are computed and put in a **PLOT** object. In the second phase, this object is rendered on the screen. In this section, we shall concentrate on the first phase of plotting.

A two-dimensional graphics object in Maple is a call to the function **PLOT** with arguments describing the axes, function, computed plot points, plot style, and so on. The following object describes a triangle with vertices (1,1), (2,2), and (3,1) (Figure 15.29).

```
> PLOT(CURVES([[1,1], [2,2], [3,1], [1,1]]),
>       AXESSTYLE(NONE), SCALING(CONSTRAINED));
```

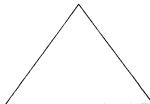


Figure 15.29. A triangle built up from graphics primitives.

The arguments **CURVES(...)**, **AXESSTYLE**, and **SCALING** are parts of the arguments of the graphics object obtained by

```
> P := plot([[1,1], [2,2], [3,1], [1,1]],
>           axes=none, scaling=constrained):
> lprint(P); # print the plot data structure

PLOT(CURVES([[1., 1.], [2., 2.], [3., 1.], [1., 1.]]),
      AXESSTYLE(NONE), COLOUR(RGB,1.0,0,0), AXESLABELS("", ""),
      SCALING(CONSTRAINED), VIEW(DEFAULT,DEFAULT))
```

Figure 15.30 is obtained by evaluation of the graphics object.

```
> P; # plot the graphics object
```

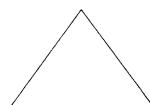


Figure 15.30. A triangle drawn via **plot**.

A more general example of a graphics object behind a plot is the plot structure for the graph of the function

$$x \mapsto \sqrt{2} - \sqrt{2\sqrt{x}},$$

which is shown in Figure 15.31.

```
> f := x -> sqrt(2)-sqrt(2*sqrt(x));
> P := plot(f(x), x=0..1, y=0..sqrt(2),
> title="graph of sqrt(2) - sqrt(2*sqrt(x))":
> lprint(P); # print plot data structure

PLOT(CURVES([[0., 1.41421356237309515],
[.681161067708333304e-3, 1.18574447219701806],
[.136232213541666662e-2, 1.14251649477750616],
[.978272101875000022, .774536984891516234e-2],
[1., 0.]]), COLOUR(RGB,0,0,0),
TITLE("graph of sqrt(2) - sqrt(2*sqrt(x))"),
AXESLABELS("x","y"),VIEW(0. ... 1.,0. ... 1.414213562))
```

Maple associates a graphics object with a function in one variable on an interval. This object, which is a regular Maple data structure, will be used for straight line interpolation between computed plot points.

```
> P; # display the graph
```

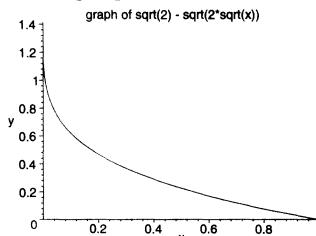


Figure 15.31. Graph of  $2^{1/2}(1 - x^{1/4})$ .

First, the argument in the **plot** call that describes the function is evaluated. Hereafter, the function values at the sample points are computed numerically. To increase the speed of plotting, this computing of function values is usually done in hardware floating-point arithmetic.

To obtain nice smooth plots Maple has an algorithm for refining the number of sample points where necessary. First, the function values of 49 equidistant points that span the interval are computed. Jitter is introduced to the sample points so that they are not exactly equidistant anymore. Next, Maple looks at these points as being connected by line segments. If the *kink angle* between two adjacent line segments is too large, then Maple will sample more points around this kink. Maple will not go on forever with subdividing the plot interval: the maximum resolution is controlled by the option **resolution**. The default value of the display resolution is 200. So, by default, the number of sample points is between 49 and 200.

You can zoom in or out by the **display** routine of the **plots** package.

```
> display(P, view=[0..0.01, 1..sqrt(2)], title=
> "zoomed-in graph of sqrt(2) - sqrt(2*sqrt(x))");
```

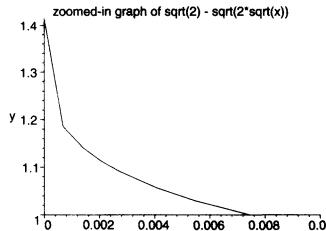


Figure 15.32. Zoomed-in graph of  $2^{1/2}(1 - x^{1/4})$ .

In the above example, Maple already uses the adaptive sampling scheme for small values of  $x$ . You are informed about this when you set a higher value to **infolevel[plot]**.

```
> infolevel[plot] := 2:
> plot(f(x), x=0..1, y=0..sqrt(2)):

plot/adaptive: evalhf succeeded
plot/adaptive: produced 59. output segments
plot/adaptive: using 59. function evaluations
```

For comparison, we set the option **adaptive** to **false** below.

```
> plot(f(x), x=0..1, y=0..sqrt(2), adaptive=false):

plot/adaptive: evalhf succeeded
plot/adaptive: produced 49. output segments
plot/adaptive: using 49. function evaluations
```

Maple now uses the default number of sample points, i.e., 49. The effect of adaptive sampling becomes more evident when we specify explicitly the initial set of sample points through the **plot** option **sample**.

```
> infolevel[plot]:=1: # reset information level
> plot(x^2, x=0..1, sample=[0,1/2,1], adaptive=true);
```

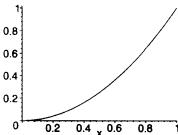


Figure 15.33. Graph of  $x \mapsto x^2$  with few initial sample points.

```
> plot(x^2, x=0..1, sample=[0,1/2,1], adaptive=false);
```

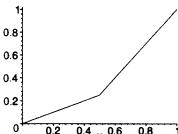


Figure 15.34. Graph of  $x \mapsto x^2$  with no adaptive plotting.

Whenever you notice some irregularities in a graph, such as in Figure 15.35 displaying the function defined by  $\frac{x}{1-\cos 5x}$  over the interval  $(-5, 5)$ , you can try to improve the picture and get a nicer, smoother plot by increasing the number of sample points. Just set a higher value to the option **numpoints** (Figure 15.36).

```
> plot(x/(1-cos(5*x)), x=-5..5, -5..5);
```

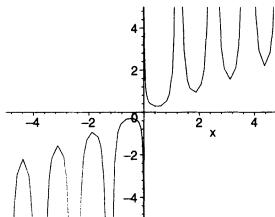


Figure 15.35. Sketch of  $x \mapsto \frac{x}{1-\cos 5x}$ .

```
> plot(x/(1-cos(5*x)), x=-5..5, -5..5, numpoints=200);
```

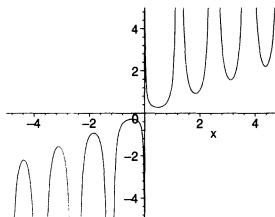


Figure 15.36. Smooth graph of  $x \mapsto \frac{x}{1-\cos 5x}$ .

A plot is a data structure like any other in Maple. You can manipulate it, save it to a file, and read it. An example of a step function:

```
> step := plot(2*Heaviside(x-1)-1, x=-1..2,
>   discont=true, color=black, thickness=15):
> step; # display the graph
```

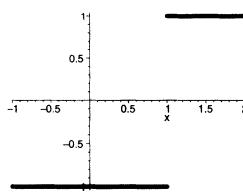


Figure 15.37. Graph of the step function  $2 \text{Heaviside}(x - 1) - 1$ .

Now we save the variable **step** in the file **graph.m**.

```
> save step, "graph.m":
```

We replace the variable **step** by the empty statement to show that after reading the file **graph.m**, the variable has its old value again.

```
> step := NULL: # reassign the variable
> read "graph.m": # load the graph
> step; # display the graph
```

You can convince yourself that the Figure 15.37 is rendered again. An easy manipulation of the data structure is substitution of the thickness component. Because the thickness information of the curve is inside the CURVES(...) component of the plot data structure and because this component is left untouched by the **plots[display]** command, you have to use substitution for quick changes of overall thickness or rebuild the plot data structure via the **plot** command (Figure 15.38).

```
> subs(THICKNESS(15)=THICKNESS(3), step);
```

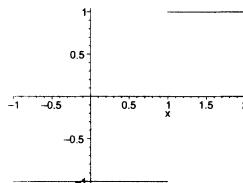


Figure 15.38. Replot of the step function  $2 \text{Heaviside}(x - 1) - 1$  with different thickness.

## 15.4 The **plottools** Package

Let us continue with the description of a two-dimensional graphics object. In the previous section you learned about the low-level graphics primitive CURVES(...); a particular example was

```
CURVES( [ [1., 1.], [2., 2.], [3., 1.], [1., 1.] ], COLOUR(RGB, 1., 0, 0)).
```

Like the entire plot, graphics primitives are implemented in Maple as a function call. The general format is

$$\text{ObjectName}(\text{ObjectInformation}, \text{LocalInformation}).$$

In the above example, CURVES is the object name, the list of points forms the object information, and the color information is local information that specifies the color of the curve. Other object names for 2D-plotting are POINT, POLYGONS, and TEXT. Local information is also implemented as function calls. Typical names are AXESTYLE, AXESTICKS, COLOUR, FONT, THICKNESS, and VIEW, among others. Enter **?plot,structure** to see all names for local information. These low-level graphics primitives are always in upper-case characters; American spelling is also allowed. They can be used to build up a graphics object. As an example, we construct a blue square without border lines, without axes, and with equal scaling in every direction (Figure 15.39).

```
> PLOT(POLYGONS([[0,0], [1,0], [1,1], [0,1]],  
> COLOR(RGB,0,0,1)), AXESSTYLE(NONE),  
> STYLE(PATCHNOGRID), SCALING(CONSTRAINED));
```

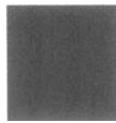


Figure 15.39. Square without border lines and axes in user-defined color.

Another example is an equilateral triangle with thick, dashed, red border lines with the blue text “equilateral triangle” in Helvetica, 24 point size, oblique font inside the triangle (Figure 15.40).

```
> PLOT(CURVES([[0,0],[1,0],[1/2,1/2*sqrt(3)],[0,0]],  
> COLOR(RGB,1,0,0), THICKNESS(15), LINESTYLE(1)),  
> TEXT([1/2,1/6*sqrt(3)], "equilateral triangle",  
> COLOR(RGB,0,0,1), FONT([HELVETICA,BOLDOblique,  
> 24])), AXESSTYLE(NONE), SCALING(CONSTRAINED));
```

**Plotting error, non-numeric vertex definition**

We deliberately made a mistake: the above error message explains that the graphics primitives may only contain numerical data, i.e., a floating-point number 0.289 instead of the algebraic number  $\frac{1}{6}\sqrt{3}$ . The correct input is as follows:

```
> PLOT(CURVES([[0,0],[1,0],[1/2,0.866],[0,0]],  
> COLOR(RGB,1,0,0), THICKNESS(15), LINESTYLE(1)),  
> TEXT([1/2,0.289], "equilateral triangle",  
> COLOR(RGB,0,0,1), FONT(HELVETICA,BOLDOblique,9)),  
> AXESSTYLE(NONE), SCALING(CONSTRAINED));
```

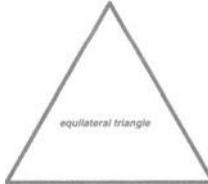


Figure 15.40. Equilateral triangle with text inside.

These examples illustrate how cumbersome it would be if you had to build up graphics objects from low-level graphics primitives all the time. A collection of commonly used building blocks would very much come in hand, and Maple provides one, viz., the **plottools** package. In combination with the **plots** package for specialized graphics, you can create complicated graphics objects quite easily. For example, the above pictures can also be obtained in the following way (pictures are omitted because they are as before).

```
> with(plots): with(plottools): # load packages  
> # blue square  
> display(rectangle([0,0], [1,1], color=blue),  
> axes=none, style=patchnogrid, scaling=constrained);
```

```

> # equilateral triangle with text inside
> lines := curve([[0,0],[1,0],[1/2,1/2*sqrt(3)], [0,0]],
>   color=red, thickness=15, linestyle=1);
> text := textplot([1/2,1/6*sqrt(3)],
>   "equilateral triangle"], color=blue,
>   font=[HELVETICA,BOLDOBLIQUE,9]):
> display({lines, text}, axes=none, scaling=constrained);

```

If the above examples do not impress you of the usefulness of the **plottools** package, then try to build the happy face of Figure 15.41 from low-level graphics primitives.

```

> head := ellipse([0,0.5], 0.7, 0.9, color=black):
> eyes := disk([0.4,0.4],0.1, color=blue),
>   disk([-0.4,0.4],0.1, color=blue):
> mouth := arc([0,0.1], 0.35, 5/4*Pi..7/4*Pi,
>   color=red, thickness=7):
> nose := line([0,0.35], [0,-0.1], color=black,
>   thickness=5):
> display({head, eyes, nose, mouth},
>   scaling=constrained, axes=none); # happy face

```



Figure 15.41. Happy face.

In the **plottools** package are also present functions that alter an existing graphics object. We list them in Table 15.1. Together with the procedure **changecoords** of the **plots** package, they form the Maple procedures that map one graphics data structure into another.

Procedure	Purpose
<b>rotate</b>	rotate counter-clockwise
<b>scale</b>	scale a plot
<b>stellate</b>	create stellated polygons
<b>transform</b>	generate a function to transform a plot
<b>translate</b>	translate a graphics object

Table 15.1. Plottools functions to alter graphics objects.

Let us use these procedures to create a spectrum of arrows in different colors, with different sizes, and translated positions so that the arrow points always hit the unit circle. The spectrum is shown in Figure 15.44. We start with a horizontal, thick arrow (Figure 15.42).

```

> a[0] := display(arrow([0,0], [1,0], 0.1, 0.4, 0.2),
>   color=COLOR(HUE,0), axes=none, scaling=constrained):

```

```
> a[0]; # display the horizontal arrow
```



Figure 15.42. Horizontal arrow.

As an example of an alteration of this arrow we rotate it clockwise over  $-\frac{\pi}{2}$ , scale it with the factor  $\frac{1}{4}$  in all directions, and translate it over a distance  $\frac{3}{4}$  along the vertical axis in negative direction (Figure 15.43).

```
> display(a[0], translate(scale(rotate(a[0], -Pi/2),
> 1/4, 1/4), 0, -3/4));
```



Figure 15.43. Original arrow and arrow after rotation, scaling, and translation.

Because the **rotate**, **scale**, **translate**, and **display** procedures do not allow you to change the color of the arrow via an option, we achieve this in the creation of the arrows spectrum below by substitution.

```
> for i to 15 do
>   a[i] := subs(COLOR(HUE,0)=COLOR(HUE,i/16),
>   translate(scale(rotate(a[0],i*Pi/8), (16-i)/16,
>   (16-i)/16), i/16*cos(Pi*i/8), i/16*sin(Pi*i/8)))
> end do:
> display([seq(a[i], i=0..15)], scaling=constrained);
```



Figure 15.44. Spectrum of arrows in different HUE colors.

The **transform** routine can be used for coordinate transformations (Figure 15.45). As an example, we transform a rectangular grid to a polar grid.

```
> hpoints := [seq([seq([i/16, j*Pi/8], i=0..16)],
> j=0..16)]:
> hlines := map(curve, hpoints):
> vpoints := [seq([seq([i/16, j*Pi/8], j=0..16)], i=0..16)]:
> vlines := map(curve, vpoints):
> grid := display(hlines, vlines, axes=frame):
> f := transform( (r,phi) -> [r*cos(phi), r*sin(phi)] ):
> display(array([grid, f(grid)]), scaling=constrained,
> tickmarks=[3,3]);
```

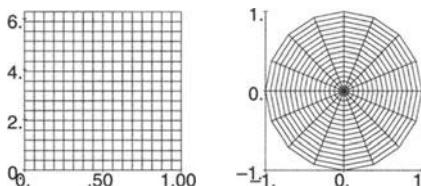


Figure 15.45. Transformation of Cartesian to polar coordinates applied to a rectangular grid

Here, we have made use of the ability of the **display** command from the **plots** package to draw plots in a so-called *graphics array*. The grid on the right can be drawn more nicely with the **coordplot** routine from the **plots** package.

## 15.5 Special Two-Dimensional Plots

Previous examples already illustrated that there is more to graphics than plotting a function in one variable. In this section, we shall list a few more two-dimensional graphics capabilities that are available via the **plots** package. Recall that we assume in this chapter that this package has been loaded in all sample sessions via the command **with(plots)**.

### Combining Plots

As an example of combining plots, we draw the sine and cosine function in one graph and color the area between the two functions with red assuming that the background color is white (Figure 15.46).

```
> sine := plot(sin, 0..4*Pi, color=black, thickness=3):
> s := plot(sin, 0..4*Pi, filled=true, color=red):
> cosine := plot(cos, 0..4*Pi, color=black, thickness=3):
> c := plot(cos, 0..4*Pi, filled=true, color=red):
> f := x -> if cos(x)>0 and sin(x)>0 then
>           min(cos(x),sin(x))
>       elif cos(x)<0 and sin(x)<0 then
>           max(cos(x),sin(x))
>       else 0
>       end if:
> b := plot(f, 0..4*Pi, filled=true, color=white):
> display([sine, cosine, b, s, c ], scaling=constrained);
```

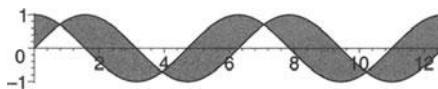


Figure 15.46. Combining plots.

In the above **display** command, the ordering of the graphs in the list is important: graphs are placed behind each other from left to right. So, the background coloring **b** comes in front of the shading between the sine function and the horizontal axis. Convince yourself of this by using three different shading colors in the above construction.

The procedure **display** can also be used to draw several plots side by side or in a rectangular array in one picture (Figures 15.47 and 15.48). For example,

```
> ticks := [[0="0", 3.14="p", 6.28="2 p"],
>           [-0.99="-1", 0="0", 0.99="1"]]:
> sine := plot(sin(x), x=0..2*Pi, tickmarks=ticks,
>               axesfont=[SYMBOL,12], scaling=constrained):
> cosine := plot(cos(x), x=0..2*Pi, tickmarks=ticks,
>                 axesfont=[SYMBOL,12], scaling=constrained):
> display(array([sine,cosine])); # row of plots
```

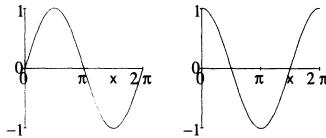


Figure 15.47. Graphics array: row vector of plots.

```
> display(array(1..2,1..1,[[sine],[cosine]]));
> # column of plots
```

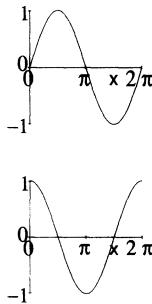


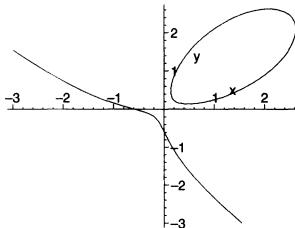
Figure 15.48. Graphics array: column vector of plots.

The somewhat strange introduction of tick marks has been done to assure nice tick marks when the plots are combined into one array.

### Plots of Plane Algebraic Curves

In the **plots** package resides the procedure **implicitplot** to draw a two-dimensional plane curve defined by an equation (Figure 15.49). An example:

```
> implicitplot(x^3+y^3-5*x*y+1/5=0,
>               x=-3..3, y=-3..3, grid=[50,50], color=black);
```

Figure 15.49. Implicit plot of  $x^3 + y^3 - 5xy + 1/5 = 0$ .

In essence, the method used is to consider the equation as a function in three-dimensional space and generate a contour of the equation cutting through the  $z$ -plane. Drawbacks of this method are that it generates rough graphs (q.v., the graph obtained without the **grid** option used), and that it does not guarantee correct drawings near singularities and intersections of the curve. Algebraically, this means that Maple may have problems with any point  $(x, y)$  on the curve  $f(x, y) = 0$  where the partial derivatives  $\frac{\partial f}{\partial x}(x, y)$  and  $\frac{\partial f}{\partial y}(x, y)$  are both zero. A typical example is the curve

$$2x^4 + y^4 - 3x^2y - 2y^3 + y^2 = 0,$$

which has two singular points,  $(0,0)$  and  $(0,1)$ , where the branches intersect [237]. Even when a very fine grid is used, Maple has difficulty in drawing the graph correctly with **implicitplot**. For example, the computation of this implicitly defined curve with a  $500 \times 500$  grid takes up much computing resources and computing time, and still does not provide a satisfactory plot around the origin (we omit the picture). In this case, it is better to compute a parameterization of the curve in polar coordinates and graph it with the **polarplot** command (Figure 15.50).

```

> subs( x=r*cos(phi), y=r*sin(phi),
>       2*x^4 + y^4 - 3*x^2*y - 2*y^3 + y^2 ):
> factor(%);
r^2 (2 r^2 cos(phi)^4 + r^2 sin(phi)^4 - 3 r cos(phi)^2 sin(phi) - 2 r sin(phi)^3 + sin(phi)^2)
> eqn := op(2,%):
> sols := map(combine, {solve(eqn,r)});

sols := { 
$$\frac{9 \sin(\phi) + \sin(3\phi) - \sin(\phi) \sqrt{30 + 8 \cos(2\phi) - 22 \cos(4\phi)}}{9 + 3 \cos(4\phi) + 4 \cos(2\phi)},$$


$$\frac{9 \sin(\phi) + \sin(3\phi) + \sin(\phi) \sqrt{30 + 8 \cos(2\phi) - 22 \cos(4\phi)}}{9 + 3 \cos(4\phi) + 4 \cos(2\phi)}$$
 }

> sols := map(unapply, sols, phi):
> polarplot(sols, 0..2*Pi, view=[-5/2..5/2,0..9/4],
>           scaling=constrained, color=black);

```

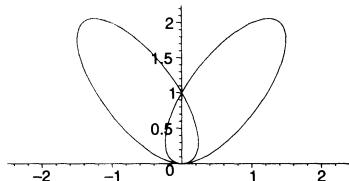


Figure 15.50. Parametric plot in polar coordinates.

You can also compute a parameterization of the curve with the procedure **parametrization** of the **algcurves** package. With this parameterization it is easy to plot the graph (Figure 15.51).

```
> v := algcurves[parametrization](
>   2*x^4 + y^4 - 3*x^2*y - 2*y^3 + y^2, x, y, t);
v := [ 
$$\frac{-8470 - 6883t - 1548t^2 - 15t^3 + 18t^4}{18t^4 + 336t^3 + 5056t^2 + 26696t + 43257},$$


$$\frac{4900 + 6020t + 2689t^2 + 516t^3 + 36t^4}{18t^4 + 336t^3 + 5056t^2 + 26696t + 43257} ]$$

> plot([op(v), t=-infinity..infinity], scaling=constrained);
```

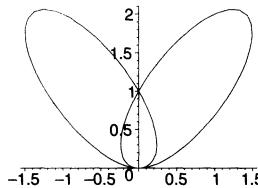


Figure 15.51. Parametric plot in polar coordinates.

### Logarithmic Plots

Logarithmic plots (Figure 15.52), semi-logarithmic plots (i.e., the horizontal axis is in logarithmic scale), and double-logarithmic plots (Figure 15.53) are available in Maple through the procedures **logplot**, **semilogplot**, and **loglogplot** in the **plots** package, respectively.

```
> # use a random number generator for uniform distribution
> noise := stats[random, uniform[0,0.1]]:
> plotpoints := [seq([0.2*i, exp(0.2*i)+noise()], i=0..20)]:
> logplot(plotpoints, style=point, symbol=circle,
>         symbolsize=14);
```

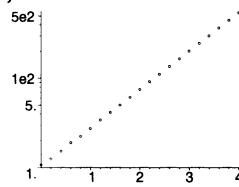


Figure 15.52. Logarithmic plot

```
> loglogplot(x^3 + exp(-x), x=1/10..50, numpoints=200,
> tickmarks=[3,4], labels=["x","x^3 + exp(-x)" ],
> scaling=constrained, axes=frame);
```

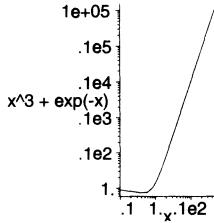


Figure 15.53. Double-logarithmic plot.

The semi-logarithmic plot that is commonly used in electrical engineering is the Bode plot. Suppose that an electronic circuit has the following response voltage  $V_{out}$  in the frequency domain:

```
> interface(imaginaryunit=J):
> Vout := 1/(-12*J*omega^3-2*omega^2+7*J*omega+1);
```

$$V_{out} := \frac{1}{-12I\omega^3 - 2\omega^2 + 7I\omega + 1}$$

In the Bode plot the magnitude of the output voltage is taken in Decibel units (defined as  $20^{10}\log x$ ) and plotted in the semi-logarithmic scaling (Figure 15.54).

```
> magnitude := 20*log[10](evalc(abs(Vout)));
magnitude := 20 \frac{\ln(\frac{1}{\sqrt{-164\omega^4 + 45\omega^2 + 1 + 144\omega^6}})}{\ln(10)}
> M := semilogplot(magnitude, omega=0.01..10,
> labels=[["omega", "magnitude"]]):
> M; # display magnitude
```

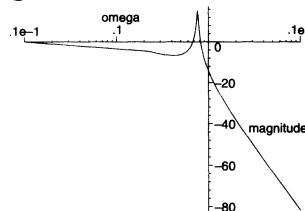


Figure 15.54. Magnitude part of Bode plot.

In a Bode plot, the phase or argument of the output voltage is drawn too (Figure 15.55).

```
> phase := evalc(argument(Vout));
phase := \arctan(12\omega^3 - 7\omega, -2\omega^2 + 1)
> P := semilogplot(180/Pi*phase, omega=0.01..10,
> labels=[["omega", "phase"]]):
> P; # display phase
```

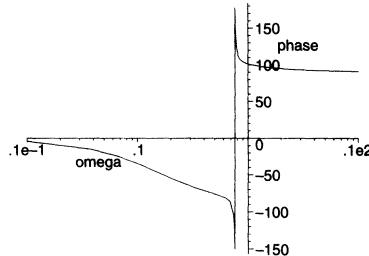


Figure 15.55. Phase part of Bode plot.

The Bode plot actually is the combination of the above two plots in column layout (Figure 15.56).

```
> display(array(1..2,1..1,[[M],[P]]));
```

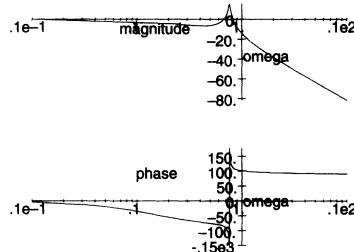


Figure 15.56. Bode plot.

You can group the above commands into a procedure. Below, we have defined a procedure **Bodeplot** in such a way that plotting options also are taken care of (Figure 15.57).

```

Bodeplot := proc(voltage, domain)
>     local M, P, l, magnitude, omega, opts,
>           phase, r, toption;
>     if not typematch( domain,
>           omega::name = l::algebraic..r::algebraic ) then
>       error "invalid input"
>     elif not type( voltage, ratpoly(anything,omega)) then
>       error "input must be a rational function in", omega
>     end if;
>     opts := [args[3..nargs]];
>     if not hasoption(opts, 'thickness', toption, 'opts')
>       then toption:=2 # default thickness
>     end if;
>     l := evalf(l); r := evalf(r);
>     M := plots['semilogplot'](
>       20*log[10](evalc(abs(voltage))),
>       omega=l..r, 'thickness'=toption,
>       labels=[omega,magnitude], op(opts));
>     P := plots['semilogplot'](
>       180/Pi*evalc(argument(voltage)),
>       omega=l..r, 'thickness'=toption,
>       labels=[omega,phase], op(opts));

```

```
> plots['display']( array(1..2,1..1,[ [M],[P]] ) );
> end proc;
> Bodeplot(Vout, omega=0.01..10, thickness=5,
> color=blue, numpoints=200); # greater thickness
```

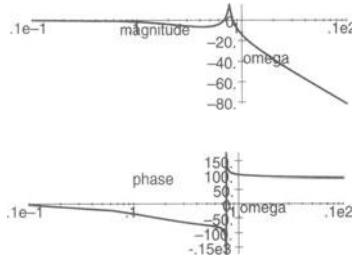


Figure 15.57. Bode plot.

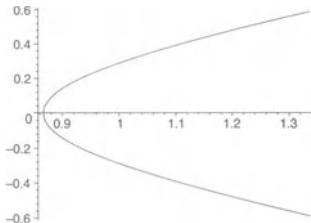
A few remarks on this small piece of graphics programming: In the first conditional statement we check whether `input` is valid or not. Next, we store optional arguments in the variable `opts` and check whether the option `thickness` has been specified. We do this via the procedure `hasoption`. If the plot option `thickness` is present, then `hasoption` takes care of handling it in an appropriate way. If the option has not been specified, then the default value 2 is taken. This is how you pass options and defaults through Maple procedures.

```
> interface(imaginaryunit=I); # I=sqrt(-1) again
```

### Complex Curves

A complex curve, i.e., a function from  $\mathbb{R}$  to  $\mathbb{C}$ , can be plotted by the procedure `complexplot`. One example will do (recall that  $I=\sqrt{-1}$ ) (Figure 15.58).

```
> complexplot(sin(Pi/3+t*I), t=-1..1);
```

Figure 15.58. Complex curve of  $t \mapsto \sin(\frac{\pi}{3} + ti)$ .

### Plots of Conformal Mappings

One example of the procedure `conformal` in the `plots` package will suffice. The complex function  $z \mapsto 1/z$  maps the rectangular grid in Figure 15.59 in the complex plane

```
> conformal(z, z=-1-I..1+I, grid=[25,25],
> axes=frame, scaling=constrained);
```

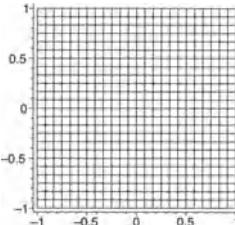
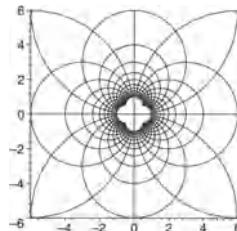


Figure 15.59. Square grid in complex plane.

to the grid

```
> conformal(1/z, z=-1-I..1+I, grid=[25,25],
> numxy=[100,100], axes=frame,
> scaling=constrained, view=[-6..6,-6..6]);
```

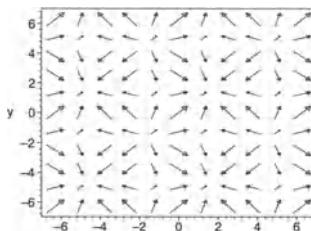
Figure 15.60. Conformal plot of complex function  $z \mapsto \frac{1}{z}$ .

The picture in Figure 15.60 is related to the electric field for the two-dimensional analogue of an electric dipole, i.e., two parallel line charges with opposite polarities, very close together.

### Field Plots

Use the procedure **fieldplot** to plot a two-dimensional vector field. An example (Figure 15.61):

```
> fieldplot([cos(x),cos(y)], x=-2*Pi..2*Pi, y=-2*Pi..2*Pi,
> arrows=SLIM, grid=[11,11], axes=box);
```

Figure 15.61. Field plot of  $[\cos x, \cos y]$  on  $[-2\pi, 2\pi] \times [-2\pi, 2\pi]$ .

As an application of gradient field plotting, we draw the electric field due to two parallel uniform line charges of one unit charge per unit length, of opposite polarities, and at a distance of two unit lengths from each other. Actually, we take line charges infinitely long in the direction perpendicular to the  $xy$ -plane, with the negative one going through  $(-1,0)$ , and the positive line charge going through  $(1,0)$ . In general, the strength of the electrostatic

potential  $\phi$  at position  $\vec{P}$  when line charges  $l_i$  go through positions  $\vec{P}_i$  in the  $xy$ -plane is given by

$$\phi = -\frac{1}{2\pi\epsilon_0} \sum_i l_i \ln(\text{distance}(\vec{P}, \vec{P}_i)).$$

The electric field of the charge distribution is then defined as  $\vec{E} = -\nabla\phi$ . In our case, when we set the scalar factor  $\frac{1}{2\pi\epsilon_0}$  equal to 1:

```
> phi := ln(sqrt((x+1)^2+y^2)) - ln(sqrt((x-1)^2+y^2));
```

$$\phi := \ln(\sqrt{x^2 + 2x + 1 + y^2}) - \ln(\sqrt{x^2 - 2x + 1 + y^2})$$

The two-dimensional gradient vector field can be computed and drawn as follows (Figure 15.62):

```
> gradplot(-phi, x=-2..2, y=-1..1,
> arrows=THICK, grid=[11,11], axes=box);
```

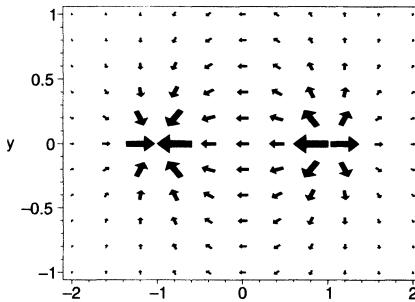


Figure 15.62. Electric field of a line charge distribution.

In Chapter 17, when we discuss the **DEtools** package, we shall come back to this example and compute the field lines. The equipotentials, i.e., the lines of constant electrostatic potential, can be plotted as a contour plot.

### Contour Plots and Density Plots

Let us continue the example of an electric field of line charges and draw a contour plot of the electrostatic potential (Figure 15.63)

```
> contourplot(phi, x=-2..2, y=-1..1, color=black,
> numpoints=500, axes=box, contours=10);
```

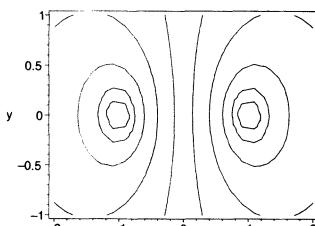


Figure 15.63. Equipotentials of an electrostatic potential.

You can also specify explicitly what contour values you want to have in your picture and fill the contour levels with specified coloring; below we choose gray levels (Figure 15.64).

```
> contourplot(phi, x=-2..2, y=-1..1,
>   numpoints=500, axes=box, filled=true,
>   contours=[seq(i/4,i=-6..6)], coloring=[white,black]);
```

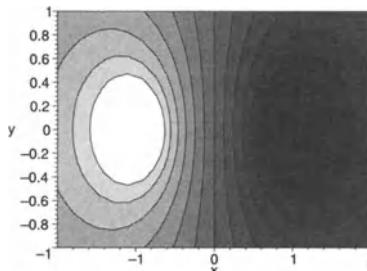


Figure 15.64. Equipotentials of an electrostatic potential at specified contour levels.

With some extra work you can create a legend that explains the meaning of the gray levels. Below we show what can be done with modest means (Figure 15.65).

```
> CP := %: # store contour plot
> legendcolors := seq(
>   plottools[rectangle]( [0, i/4], [1,(i+1)/4],
>   color=COLOR( RGB, 1-(i+7)/13, 1-(i+7)/13, 1-(i+7)/13),
>   i=-7..6):
> legendtext := seq(
>   textplot([1.5, i/2, convert(evalf(i/2,2),name)],
>   font=[HELVETICA,BOLD,10]), i=-3..3),
>   textplot([1.5, 2, "Heights"], font=[HELVETICA,BOLD,10]),
>   textplot([3.5, 2, " "]):
> LP := display({legendcolors, legendtext},
>   axes=none, scaling=constrained): # store legend
> display(array([CP,LP])): # display in vector layout
```

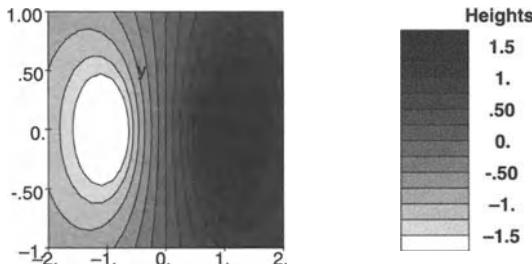


Figure 15.65. Contourplot of an electrostatic potential with a plot legend

A density plot of the electrostatic potential  $\phi$  can be generated with **densityplot** (Figure 15.66).

```
> densityplot(phi, x=-2..2, y=-1..1,
> grid=[100,100], axes=box, style=patchnogrid,
> colorstyle=HUE);
```

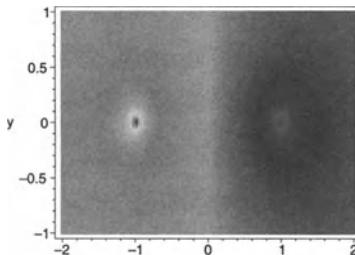


Figure 15.66. Density plot of an electrostatic potential with a fine grid.

## 15.6 Two-Dimensional Geometry

The **geometry** package provides tools for studying two-dimensional Euclidean geometry. Two examples will do to give you an idea of geometrical and graphical possibilities of the package.

**The Circle Theorem of Apollonius.** *In any right triangle the circle passing through the midpoints of the edges contains also the feet of the three altitudes.*

Let us see how we can “prove” and graphically illustrate the theorem by means of the **geometry** package. First, we load the package and define a triangle  $OAB$ .

```
> with(geometry): # load geometry package
> interface(showassumed=2): # no tilde, but a message
> assume(x>0, y>0): # assume positive lengths
> triangle(T, [point(0,0,0), point(A,x,0), point(B,0,y)]):
```

Next, we define the midpoints of the triangle and the circle that passes through these points, i.e., the medial circle.

```
> midpoint(D, 0, A): midpoint(E, A, B):
> midpoint(F, 0, B): circle(C, [D, E, F]):
```

Let  $G$  be the foot of the altitude  $L$  from  $O$  to the edge  $AB$ .

```
> projection(G, 0, line('dummy',[A,B])):
> line(L, [0,G]): # altitude
```

The power of the **geometry** package is that you can ask geometrical questions like “Is a point on a line?” and “Are two lines perpendicular?” So, you can verify the theorem by asking Maple whether  $G$  is on  $C$ .

```
> IsOnCircle(G, C);
true
```

Now, let us choose a specific triangle  $T$  and draw the picture.

```
> assign(x,2): assign(y,1):
> draw([C(printtext=false), T, D, E, F, G, L],
> axes=none, thickness=3, font=[HELVETICA,BOLD,10],
> color=blue, scaling=constrained, printtext=true);
```

As you see in the above command, the procedure **draw** provides the graphical visualization of objects supported in the **geometry** package. Both local options for particular objects and global plot options can be applied. In the second example we shall heavily use this feature to improve the geometrical picture; here, we only inform Maple that the center of the Apollonius circle does not need to be drawn (Figure 15.67).

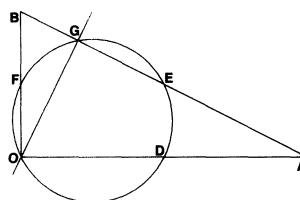


Figure 15.67. Illustration of the circle theorem of Apollonius.

For a completely algebraic treatment of the above theorem we refer to example 3 of section 6.4 in [66].

**Simson's Theorem.** *The pedal points of the perpendiculars drawn from an arbitrary point on a triangle's circumscribed circle to the three edges are collinear.*

Let us see how we can “prove” and graphically illustrate this theorem. First, we define a triangle  $OAB$  with its circumscribed circle  $C$ .

```
> triangle(T, [point(0,0,0), point(A,3,0), point(B,2,2)]):
> circumcircle(C, T):
```

Next, we define a point  $D$  on the circle. For this purpose, we must find a condition for a generic point  $(u, v)$  to be on the circle  $C$ . In our case:

```
> point(D, u, v):
> IsOnCircle(D, C, 'condition');
IsOnCircle: "hint: unable to determine if  $u^2+v^2-3*u-v$  is zero"
FAIL
> condition;
```

$$u^2 + v^2 - 3u - v = 0$$

So, we assume that this condition is satisfied.

```
> assume(condition):
```

We can compute the pedal point  $E, F, G$  of  $D$  on the edges  $OA, OB$ , and  $AB$ , respectively, in the following way.

```
> projection(E, D, line('dummy', [0,A])):
> projection(F, D, line('dummy', [0,B])):
> projection(G, D, line('dummy', [A,B])):
```

We verify Simson's theorem.

```
> AreCollinear(E, F, G);
```

*true*

So, for the given triangle  $T$ , Maple can prove Simson's theorem! Now, let us choose a specific point  $D$  on the circle.

```
> solve(condition);
```

$$\{u = \frac{3}{2} + \frac{\sqrt{9 - 4v^2 + 4v}}{2}, v = v\}, \{u = \frac{3}{2} - \frac{\sqrt{9 - 4v^2 + 4v}}{2}, v = v\}$$

with assumptions on  $u$  and  $v$

From this solution, it can easily be seen that  $(u, v) = (\frac{3}{2} - \frac{1}{2}\sqrt{6}, \frac{3}{2})$  is a valid point. We choose it and we define the lines from  $D$  to the pedal points on the edges, the line  $L$  through the pedal points, and an auxiliary line to improve the picture.

```
> assign(u, 3/2-1/2*sqrt(6)): assign(v, 3/2):
> line(DE, [D,E]): line(DF, [D,F]): line(DG, [D,G]):
> line(L, [E,G]): line(H, [B,G]):
```

Everything is ready for drawing the picture that illustrates Simson's theorem (Figure 15.68).

```
> draw(
>     [C(filled=false,linestyle=1,printtext=false),
>      T(color=black,filled=false,thickness=5,linestyle=1),
>      DE, DF, DG, H, D, L(color=red,linestyle=1), E, F, G],
>      axes=none, linestyle=2, thickness=3, color=blue,
>      scaling=constrained, font=[HELVETICA,BOLD,10],
>      printtext=true);
```

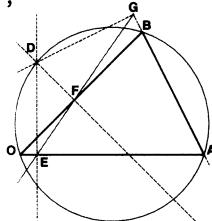


Figure 15.68. Illustration of Simson's Theorem.

We leave it as an exercise to prove Simson's theorem, with the help of the `geometry` package, for a general triangle. A completely algebraic treatment of Simson's theorem can be found in [150].

## 15.7 Plot Aliasing

Adaptive two-dimensional plotting produces, in most cases, reasonable graphs with a minimum of sample points. But you should keep alert for

insufficient sampling. As a rather extreme case of plot aliasing, we plot the function  $x \mapsto \frac{1}{10}(x - 25)^2 + \cos(2\pi x)$  on the interval  $(0, 49)$  (Figure 15.69).

```
> plot((x-25)^2/10 + cos(2*Pi*x), x=0..49);
```

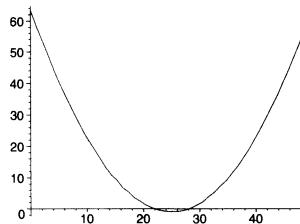


Figure 15.69. Plot aliasing for mapping  $x \mapsto \frac{1}{10}(x - 25)^2 + \cos(2\pi x)$ .

This graph is pretty close to the parabola  $y = \frac{1}{10}(x - 25)^2$ . This is wrong! The reason is that because of the default choice of 49 almost equidistant sample points, the computed function values tend to be very close to the parabola. Maple has no reason to believe that there is something wrong and it has no reason to sample more points, or to sample other points. You yourself will have to decide to sample more points (by the option `numpoints`) or to choose another plot range (Figure 15.70).

```
> plot((x-25)^2/10+cos(2*Pi*x), x=0..49, numpoints=2000);
```

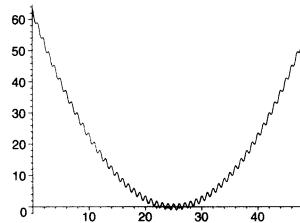


Figure 15.70. Improved graph of  $x \mapsto \frac{1}{10}(x - 25)^2 + \cos(2\pi x)$ .

## 15.8 A Common Mistake

When you ask Maple to plot a function, say  $f$ , as a function of  $x$  by entering a command of the form `plot( f(x), x = xmin..xmax )`, Maple first evaluates the  $f(x)$ , presumably getting a symbolic expression in terms of  $x$ , and subsequently evaluates this expression numerically for the sample points. This may cause you some problems when you try to plot a piecewise defined function or a numerical function. Below we present one example and four ways to overcome these problems. Suppose that you have defined the function  $f$  by

```
> f := t -> if t>0 then exp(-1/t^2) else 0 end if;
```

and want to plot it.

```
> plot(f(t), t=-1..4);
Error, (in f) cannot determine if this expression is
true or false: -t < 0
```

As you see, you get an error message back instead of a graph. The reason is that Maple evaluates  $f(t)$  and the system gets into trouble because it cannot pass the test  $t > 0$ . The trick of the trade is to avoid premature evaluation by using apostrophes around the first argument (Figure 15.71).

```
> plot('f(t)', t=-1..4);
```

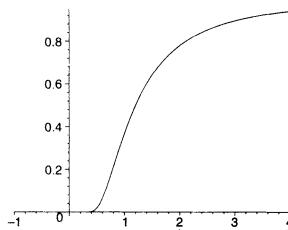


Figure 15.71. Graph of piecewise defined function.

Another option is to use the functional notation. For example, the function  $f$  and its derivative  $f'$  can be plotted in one picture (Figure 15.72) with

```
> plot({f,D(f)}, -1..4);
```

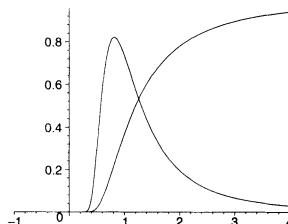


Figure 15.72. Graph of piecewise defined function and its derivative.

Two other solutions to the problem are to change the way you define the piecewise defined function. First, Maple has built-in facilities for piecewise defined functions that do not fail, viz., the procedure **piecewise**. If they are present, why not use them?

```
> f := t -> piecewise(t>0, exp(-1/t^2), 0):
> f(t); # no problem
```

$$\begin{cases} e^{(-\frac{1}{t^2})} & 0 < t \\ 0 & \text{otherwise} \end{cases}$$

Second, as shown in §8.3, you can define the function such that non-numeric arguments are accepted without any further effect. One way of doing this would be the following.

```

> f := t -> if not type(t, numeric) then
>           'procname'(t)
>       elif t>0 then
>           exp(-1/t^2)
>       else 0
>       end if:
> f(t); # no problem

```

$$f(t)$$

## 15.9 Some Basic Three-Dimensional Plots

Plotting a function of two variables is as easy as plotting a function in one variable. Simply use the Maple procedure **plot3d** and specify the ranges of both variables. As an example, let us plot the surface defined by  $z = \cos(xy)$ , where  $x$  and  $y$  range from  $-3$  to  $3$  (Figure 15.73).

```
> plot3d(cos(x*y), x=-3..3, y=-3..3);
```

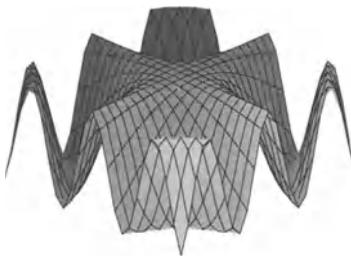


Figure 15.73. Surface plot of  $z = \cos(xy)$ .

The above command is an example of the following general format for drawing a surface defined by the formula  $f(x, y)$  of two variables  $x$  and  $y$ :

```
plot3d( $f(x, y)$ ,  $x = a .. b$ ,  $y = c .. d$ , options);
```

where  $a .. b$  and  $c .. d$  define the range of  $x$  and  $y$ , and where *options* describes zero or more options, as will be discussed in the next section.

An alternative way of obtaining the graph of a function  $f$  in two variables is

```
plot3d( $f$ ,  $a .. b$ ,  $c .. d$ , options);
```

where  $a .. b$  and  $c .. d$  define the two horizontal ranges, and where *options* describes zero or more options.

Below we draw again the graph of the function  $(x, y) \mapsto \cos(xy)$ , but now we choose the functional notation and change some of the options. This can be done interactively by selecting proper menu items from the 3D-plot window. But to ensure that you can reproduce the graph, we have added all options in the **plot3d** command (Figure 15.74).

```
> plot3d(f, -3..3, -3..3, grid=[50,50], axes=box,
> scaling=constrained, style=patchcontour,
> shading=zgrayscale);
```

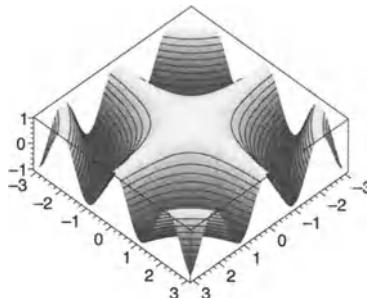


Figure 15.74. Graph of  $(x, y) \mapsto \cos(xy)$  with gray shading and contour levels.

## 15.10 Options of **plot3d**

Just as in the two-dimensional case, there are many options to customize a plot. Under the Worksheet interface you can change many options by selecting menu items and/or by mouse actions. We shall discuss some of the options and refer for others to the table of three-dimensional plotting options at the end of this chapter. In a Maple session, you can always consult the on-line help system by entering **?plot3d,option**.

### Style

A surface can be rendered in several ways with the optional argument **style = displaystyle**. The simplest style is **point**, which draws only the computed plot points. Plot points are connected with line segments when the style option **line** is chosen. By the way, the sample points of a surface are in general the grid points of a rectangular uniformly spaced grid. When adjacent plot points are connected by line segments, the result is a picture of style **wireframe** that approximates the shape of the surface. In the first plot of the previous section, the line segments are hidden because the surface is opaque. Without the surface coloring, this is Maple's style **hidden**. But the default style **patch** displays the surface in full color or gray shading. If you do not want the grid displayed, choose the option **patchnogrid**. With the style option **contour** only the contour lines of the surface are drawn. A combination of two options is the style option **patchcontour**, illustrated in the last picture of the previous section.

### Shading

In Maple, a surface can be colored (or gray shaded) in three ways: **xyz**, **xy**, and **z** shading. When you choose the option **shading = z**, the color/gray shading is according to the **z** value of the points on the surface. In the **xy** or

`xyz` shading schemes, each of the axes has its own range of colors and these colors are added together for a point on the surface. The options `zhue` and `zgrayscale` produce graphs with color hues and gray levels depending on the  $z$ -values. With `shading = none`, the picture is in one uniform color.

## Axes

With the option `axes` you specify how the axes are to be drawn. There are four choices: `none`, `normal`, `box`, and `frame`. The names speak for themselves; we only remark that framed axes are drawn along the edges of the surface so that they do not pass through the surface.

## Orientation and Projection

Rendering of a three-dimensional graphics object in two dimensions on a screen or on paper is commonly done by projection. The center of projection (or view point) is described in spherical coordinates with respect to the local Cartesian coordinate system at the center of the bounding box (Figure 15.75).

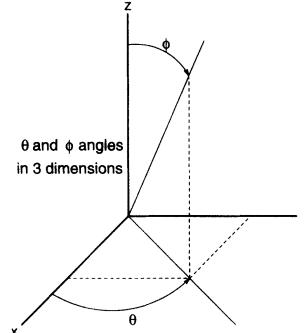


Figure 15.75. Orientation in three dimensions:  $\theta$  and  $\phi$  angles.

The angles  $\theta$  and  $\phi$  can be set with the option `orientation = [θ,φ]`. Increasing the value of rotation angle  $\theta$  rotates the plot counterclockwise along the  $z$ -axis. When this rotation angle is set to zero, you look down on the front and top side of the surface.

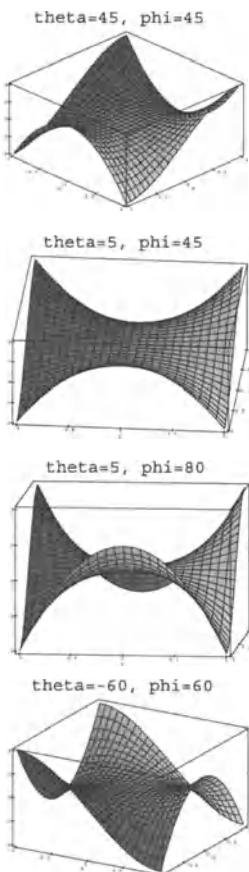
$\phi$  controls the vertical view angle; the value zero means that you look straight down on the surface, ninety degrees means that you look edge on at the  $xy$  plane, and more than ninety degrees means that you look up from beneath the surface.

The option `projection = p`, where  $p$  is a real constant in the range  $[0, 1]$ , specifies the distance from the view point to the surface. Value 0 (`fisheye`) means touching; 1 (`orthogonal`) means infinitely far away; `normal` value is  $\frac{1}{2}$ .

Pictures tell you more than words. Below, plots generated initially by

```
> plot3d(x^3-3*x*y^2, x=-1..1, y=-1..1,
>         style=patch, axes=box);
```

are shown from different points of view.



You can create a 3D-graphics array of the above plots, but it is quite some work, especially if you want a decent labeling of the plots like in Figure 15.76. (By trial and error you can find text locations that project nicely in the 3D-graphics array. The obscure way of labeling the plots below is to overcome the programming bug that the 3D-graphics conversion assumes at least one tick mark per axes; the default boxes for tick marks in 3D-graphics arrays is replaced by points, which you do not see on the axes.)

```
> with(plots, [display, textplot3d]):
> angles := [[45,45], [5,45], [5, 80], [-60,60]]:
> ticks := [[0=" "], [0=" "], [0=" "]]: # blank tick marks
> position :=
>     [[-1,0.25,2], [0,0.1,3], [0,0.1,1.2], [0.8,0.5,2]]:
> sf := [SYMBOL,10]: # symbol font at point size 10
> for i to 4 do
>   P := plot3d(x^3-3*x*y^2, x=-1..1, y=-1..1,
>             style=patch, tickmarks=ticks, orientation=angles[i]):
```

```

> T := textplot3d([op(position[i]),
> cat("[q,f] = ", convert(angles[i],name)),
> font=sf, color=blue);
> F[i] := display({P,T}, axes=frame):
> end do:
> plotarray := matrix(2, 2, [seq( F[i], i=1..4)], array):
> subs(SYMBOL(BOX)=SYMBOL(POINT),
> display(plotarray)); # draw graphics array

```

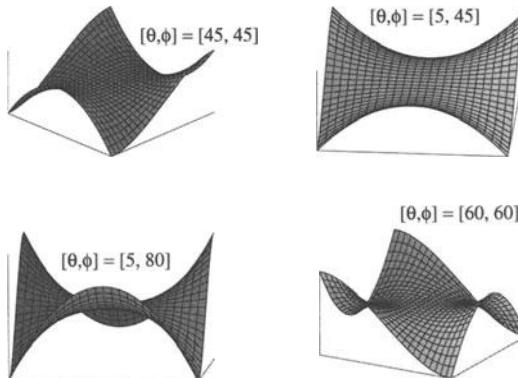


Figure 15.76. Plots of  $x^3 - 3xy^2$  taken from different view points.

Note that changing the shading or changing the view point does not imply recomputing the graphics object. It is only a matter of rendering the picture on the screen. This does not hold for the next option.

### Grid Size

In the three-dimensional case, Maple has no algorithm to refine the grid of sample points. In other words, unlike two-dimensional plotting, **plot3d** uses no adaptive sampling scheme. A default grid with 25 equidistant points in both directions is used. This can be changed by the option **grid** =  $[m,n]$ , if you want  $m$  points in the  $x$  direction and  $n$  points in the  $y$  direction. So, you must be aware of nasty three-dimensional plot aliasing effects like in Figure 15.77.

```

> plot3d(sin(2*Pi*x), x=0..25, y=0..1,
> axes=frame, style=patch, shading=zgrayscale);

```

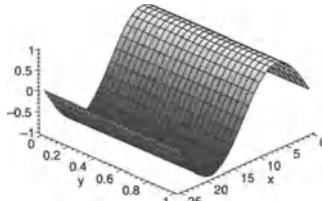


Figure 15.77. Extreme case of three-dimensional plot aliasing in plot of  $\sin(2\pi x)$ .

## View Frame

Like in two-dimensional plotting, you may want to restrict the vertical range, or even all three ranges of the surface. This is possible with the option `view`. There are two formats:

$$\text{view} = z_{\min} \dots z_{\max}$$

and

$$\text{view} = [x_{\min} \dots x_{\max}, y_{\min} \dots y_{\max}, z_{\min} \dots z_{\max}].$$

## Lighting

Maple can simulate lighting for illumination via two options:

- `light` =  $[\phi, \theta, r, g, b]$  simulates a light source with given red, green, and blue intensities in the RGB color model shining on the graphics object from the direction specified in the spherical coordinates  $[\phi, \theta]$ .
- `ambientlight` =  $[r, g, b]$  gives a nondirectional shading with red, green, and blue intensities in the RGB color model.

In the example below, we give the code to look from the top on an octahedron with three light sources (with primary colors) shining on the faces of the polytope.

```
> display(plottools[octahedron]([0,0,0],1),
> style=patch, axes=box, ambientlight=[0.2,0.2,0.2],
> light=[45,90,1,0,0], light=[135,90,0,1,0],
> light=[225,90,0,0,1], orientation=[-90,0]);
```

We do not display the picture because the gray shading in this text would not really reveal the effects of the lighting. You have to do the example yourself.

The following example of planes with gray light sources shining on them from directions perpendicular to either one of the planes clearly shows the effect of lighting, too (Figures 15.78 and 15.79).

```
> plot3d({x,1-x}, x=0..1, y=0..1, style=patchnogrid,
> axes=frame, shading=none, orientation=[-45,25],
> light=[0,0,0.9,0.9,0.9]);
```

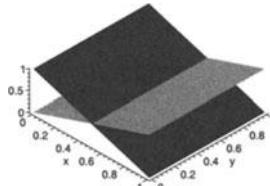


Figure 15.78. Effect of lighting on surfaces: light source from direction [0,0].

```
> plot3d({x,1-x}, x=0..1,y=0..1, style=patchnogrid,
> axes=frame, shading=none, orientation=[-45,25],
> light=[135,0,0.9,0.9,0.9]);
```

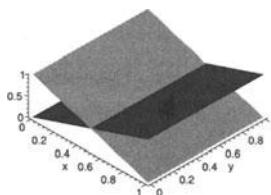


Figure 15.79. Effect of lighting on surfaces: light source from direction [135,0].

## Contours

When you use the 3D-graphics styles **contour** and **patchcontour**, the contour lines of the surface are drawn. The default number of contour lines is 10, but can be reset by the option **contours** (Figure 15.80).

```
> plot3d(5.5-y, x=0..1, y=0..5, axes=frame,
>         shading=zgrayscale, thickness=5, style=contour,
>         contours=5, grid=[2,10], orientation=[50,55]);
```

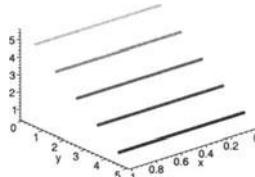


Figure 15.80. Contour lines on a surface.

You can also specify explicitly what contour values you want to have in your picture (Figure 15.81).

```
> plot3d(5.5-y, x=0..1, y=0..5, axes=frame,
>         shading=zgrayscale, thickness=5, style=contour,
>         contours=[1,2,4], grid=[2,20], orientation=[50,55]);
```

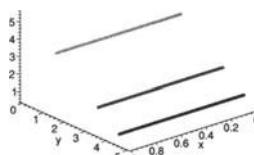


Figure 15.81. Contour lines at specified levels on a surface.

You can inspect or reset default options for all subsequent calls of **plot3d** by the procedure **setoptions3d** from the **plots** package.

At the end of this chapter you can find a list of all three-dimensional plot options that are available. Some of them have not been discussed yet, but they will be used in examples later in this chapter. The options **style**, **linestyle**, **thickness**, **symbol**, **gridstyle**, **shading**, **light**, **axes**, and **projection** can be changed interactively in the worksheet interface. Others must be adjusted in a plot command.

## 15.11 The Structure of Three-Dimensional Graphics

The making of a three-dimensional plot proceeds in two phases, similar to the way two-dimensional plots are generated. In the first phase, the plot points are computed and put in a **PLOT3D** object. In the second phase, this object is rendered on the screen. In this section, we shall present two examples of three-dimensional graphics objects, which can serve as prototypes for others.

A three-dimensional graphics object in Maple is a call to the function **PLOT3D** with arguments describing the axes, function, computed plot points, plot style, grid size, coloring, and so on. The following object describes the edges of the tetrahedron with vertices  $(1, 1, 1)$ ,  $(-1, -1, 1)$ ,  $(-1, 1, -1)$ , and  $(1, -1, -1)$  (Figure 15.82).

```
> PLOT3D(POLYGONS([[1,1,1], [-1,-1,1], [-1,1,-1]],
> [[1,1,1], [-1,-1,1], [1,-1,-1]],
> [[-1,1,-1], [1,-1,-1], [-1,-1,1]],
> [[-1,1,-1], [1,-1,-1], [1,1,1]]),
> STYLE(LINE), COLOR(RGB,0,0,0), AXESSTYLE(BOX),
> ORIENTATION(30,60), SCALING(CONSTRAINED));
```

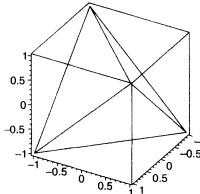


Figure 15.82. Tetrahedron built up from graphics primitives.

The arguments **POLYGONS**(...), **STYLE**, **COLOR**, **AXESSTYLE**, **ORIENTATION**, and **SCALING** are parts of the graphics object obtained by

```
> P := polygonplot3d([
> [[1,1,1], [-1,-1,1], [-1,1,-1]],
> [[1,1,1], [-1,-1,1], [1,-1,-1]],
> [[-1,1,-1], [1,-1,-1], [-1,-1,1]],
> [[-1,1,-1], [1,-1,-1], [1,1,1]]],
> style=line, color=black, axes=box,
> orientation=[30,60], scaling=constrained):
> lprint(P); # print 3D plot data structure

PLOT3D(POLYGONS([[1., 1., 1.], [-1., -1., 1.], [-1., 1., -1.]],
[[1., 1., 1.], [-1., -1., 1.], [1., -1., -1.]], [[-1., 1., -1.],
[1., -1., -1.], [-1., -1., 1.]], [[-1., 1., -1.], [1., -1., -1.],
[1., 1., 1.]]), ORIENTATION(30., 60.), STYLE(LINE),
AXESSTYLE(BOX), SCALING(CONSTRAINED), COLOUR(RGB,0,0,0))

> P; # display the graphics object
```

The picture is not shown here because it is the same as Figure 15.82.

A more general example of a graphics object behind a plot is the plot structure for the graph of the function  $(x, y) \mapsto x y^2$  (Figure 15.83).

```

> P := plot3d(x*y^2, x=-1..1, y=-1..1, grid=[5,5],
> axes=box, orientation=[30,30], style=line,
> color=black, title="graph of z=xy^2"):
> lprint(P); # print the 3D plot data structure

PLOT3D(GRID(-1. . . 1.,-1. . . 1.,Array(1 .. 5,1 .. 5,
{(1, 1) = -1., (1,2) = -.25000000000000000000,
.....,
(5, 4) = .25000000000000000000, (5, 5) = 1.},
datatype = float[8], storage = rectangular, order =
C_order), COLOR(RGB,0.,0.,0.)), ORIENTATION(30.,30.),
TITLE("graph of z=xy^2"), STYLE(LINE), AXESSTYLE(BOX),
AXESLABELS(x,y,""))

```

Maple associates a graphics object with a function in two variables on a rectangular grid. This object, which is a regular Maple object, will be used for straight line interpolation between adjacent plot points.

```
> P; # display the surface
```

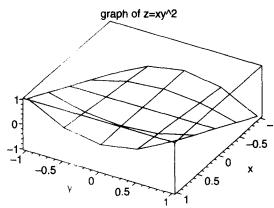


Figure 15.83. Surface  $z = x y^2$  with small grid.

First, the argument in the `plot3d` call that describes the function is evaluated. Hereafter, the function values at the grid points are computed numerically, row by row. To increase the speed of plotting, this computation of function values is usually done in hardware floating-point arithmetic. Unlike two-dimensional plotting, no automatic refinement of the grid takes place when kink angles become large.

We say that inside plotting routines Maple *usually* carries out evaluation of function values through hardware floating-point arithmetic. Of course, when you set **Digits** to a higher value than is supported by hardware floating-points, Maple has no other choice than omitting the hardware floating-point arithmetic. But you must also watch out for parts in a function that do not allow hardware floating-point arithmetic. For example, use of complex numbers in plotting disables the ability of hardware arithmetic.

```
> evalhf(2+sqrt(3)*I);
```

Error, unable to evaluate expression to hardware floats

This has a tremendous effect on the speed of plotting. Check: plotting the surface  $z = \sqrt{x^2 + y^2}$  via the function

```
> f := proc(x,y) sqrt(x^2+y^2) end proc;
```

is about 25 times faster than via the function

```
> g := proc(x,y) z := x+I*y; evalc(abs(z)) end proc;
```

If you find this example too artificial, let us consider the plotting of the filled-in Julia set of the complex function  $f : z \mapsto z^2 - 1$ , which is the set of those points in the complex plane whose orbits under the function  $f$  do not approach infinity. In Figure 15.84, this set is approximated by points that escape to infinity according to the following criteria: a point  $z_0$  escapes to infinity if the orbit of  $z_0$  contains a point with absolute value greater than 3 within the first 25 iterates. In that case we associate a function value  $F(z_0) = 0.5$  with it; otherwise  $F(z_0) = 0$ . Figure 15.84 is actually the surface plot of this new function  $F$  with gray shading based on the function value and viewed from the top. The Maple code

```
> F := proc(x,y)
>   local z;
>   z := evalf(x+y*I);
>   to 25 while abs(z)<=3 do z := z^2 - 1 end do;
>   if abs(z)>3 then 0.5 else 0 end if
> end proc;
> plot3d(F, -2..2, -1..1, grid=[200,200],
> orientation=[-90,0], style=patchnogrid, axes=box,
> labels=[x,y,''], shading=zgrayscale);
```

does the job, but is much slower than the ‘real’ version below that benefits from hardware arithmetic.

```
> F := proc(x, y)
>   local X, Y, XCOPY;
>   X := evalf(x); Y := evalf(y);
>   to 25 while X^2 + Y^2 <= 9 do
>     XCOPY := X: X := X^2-Y^2-1: Y := 2*XCOPY*Y
>   end do;
>   if X^2 + Y^2 > 9 then 0.5 else 0 end if
> end proc;
> plot3d(F, -2..2, -1..1, grid=[200,200],
> orientation=[-90,0], style=patchnogrid, axes=box,
> labels=[x,y," "], shading=zgrayscale);
```

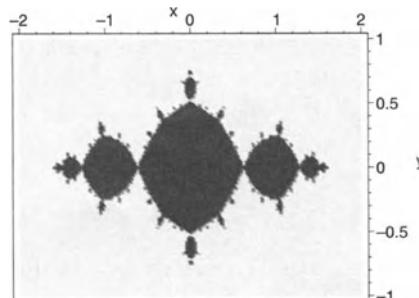


Figure 15.84. Filled-in Julia set of complex map  $z \mapsto z^2 - 1$ .

The same graph in a different coloring can be obtained by drawing the plane  $z = 0$  with the function  $F$  as the color function in the HUE color model. We only give the command and omit the resulting graph.

```
> plot3d(0, -2..2, -1..1, grid=[200,200],
> orientation=[-90,0], style=patchnogrid,
> axes=box, labels=[x,y,""], color=F);
```

The graph of the boundary of the filled-in Julia set, which actually forms the Julia set, can be created by drawing a contour plot of the surface  $z = F(x, y)$  with only one contour line at a level between the two extreme values (Figure 15.85).

```
> plot3d(F, -2..2, -1..1, grid=[200,200],
> orientation=[-90,0], style=contour, axes=box,
> labels=[x,y,""], contours=[0.25], shading=zgrayscale);
```

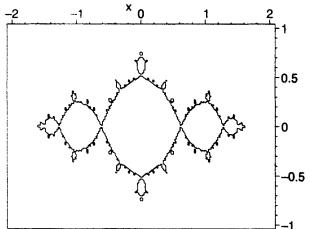


Figure 15.85. Julia set of complex map  $z \mapsto z^2 - 1$ .

Like in two dimensions, the general format of a three-dimensional graphics primitive is a function call

*ObjectName( ObjectInformation, LocalInformation ).*

In the first example of this section, you encountered the object name **POLYGONS** with the same meaning as in the two-dimensional case. In general, two-dimensional object names like **POINT** and **TEXT** can be used in three dimensions as well. The only new object names are **GRID** and **MESH**. **GRID** is the object name that describes a functional grid (in two dimensions) and **MESH** is a structure that describes a list of lists of three-dimensional points describing a surface. For example, the next function call describes a surface consisting of two triangles (Figure 15.86).

```
> PLOT3D(MESH([[1,1,1],[0,1,0],[0,0,1]],
> [[1,1,1],[1,0,0],[0,0,1]]), STYLE(PATCH), AXES(BOX),
> SHADING(NONE), LIGHT(70,0,.8,.8,0.2));
```

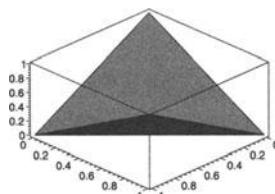


Figure 15.86. A three-dimensional mesh of two triangles.

Many of the names for local information in 2D-graphics can also be used in 3D-graphics: **AXES**, **FONT**, and **SCALING** are three names among many. Of course, there are names that only have a meaning in 3D-graphics; **AMBIENTLIGHT**, **GRIDSTYLE**, **ORIENTATION**, and **SHADING** are some of them. Most names are similar to names of options that are allowed in plotting commands. You can actually see how an option is converted into a graphics primitive.

```
> convert([color=red, orientation=[20,60]], PLOT3DOptions);
[ORIENTATION(20., 60.), COLOUR(RGB, 1.00000000, 0., 0.)]
```

This conversion of plot options into graphics primitives comes in handy sometimes when you want to change a low-level graphics primitive of an existing graphics object.

## 15.12 Special Three-Dimensional Plots

The Maple packages **plots** and **plottools**, which provide utility routines and building blocks for two- and three-dimensional plotting, respectively, have been discussed earlier in this chapter for mostly two-dimensional graphics. Special three-dimensional plots can be generated with procedures from these packages as well. Therefore, let us load the packages and do some examples.

```
> with(plots): with(plottools):
```

We start with examples of procedures from the **plots** package.

### Parametric Plots

A three-dimensional space curve can be drawn with **spacecurve** (Figure 15.87).

```
> spacecurve({[t*cos(2*Pi*t), t*sin(2*Pi*t), 2+t],
> [2+t, t*cos(2*Pi*t), t*sin(2*Pi*t)],
> [t*cos(2*Pi*t), 2+t, t*sin(2*Pi*t)]}, t=0..10,
> numpoints=400, orientation=[40,70],
> style=line, thickness=3, axes=box);
```

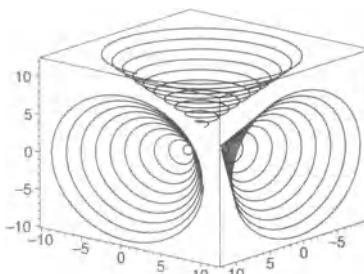


Figure 15.87. Three-dimensional space curve.

The general format for drawing a three-dimensional surface defined in the Cartesian coordinate system by parametric *formulas*

$$x = f(s, t), \quad y = g(s, t), \quad z = h(s, t)$$

is

```
plot3d([f(s, t), g(s, t), h(s, t)], s = a .. b, t = c .. d, options);
```

where  $a .. b$  and  $c .. d$  are the ranges of the independent variables  $s$  and  $t$ , and where *options* describes zero or more options.

As an example, we draw the helicoid (Figure 15.88) defined by the parameterization

$$(\phi, z) \mapsto (r \cos \phi, r \sin \phi, \phi),$$

where  $0 \leq r \leq 1$  and  $0 \leq \phi \leq 6\pi$ .

```
> plot3d([r*cos(phi), r*sin(phi), phi],
>        r=0..1, phi=0..6*Pi, grid=[15,45], style=patch,
>        orientation=[55,70], shading=zhue, axes=box);
```

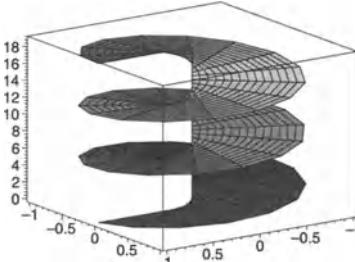


Figure 15.88. helicoid.

## Coordinate Systems

Spherical (Figure 15.89), cylindrical (Figure 15.90), and many more coordinate systems are also supported by Maple. A few examples, in which we first give the graphics object a name and then render it:

```
> S := plot3d(1, theta=0..2*Pi, phi=0..Pi, style=patch,
>             coords=spherical, scaling=constrained): S;
```

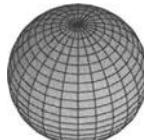


Figure 15.89. Sphere.

```
> C := plot3d(1/2, theta=0..2*Pi, z=-2..2, style=patch,
>             coords=cylindrical, scaling=constrained): C;
```

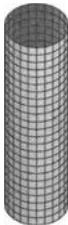


Figure 15.90. Cylinder.

```
> subs(STYLE(PATCH)=STYLE(PATCHCONTOUR), display({S,C},
> axes=box, orientation=[20,70], scaling=constrained));
```

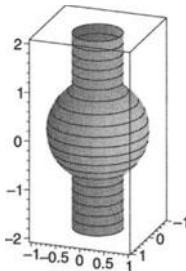


Figure 15.91. Combination of a sphere and a cylinder.

You must change the style via the above substitution instead of via the option `style=patchcontour` to obtain Figure 15.91. The reason is that in the procedure `display`, global options like color and style are moved to local ones attached to the corresponding graphical objects.

### Tube Plots

The `tubeplot` procedure defines a tube about one or more three-dimensional space curves. Let us draw in this way the torus of type (4,7) (Figure 15.92).

```
> r := a + b*cos(n*t): z := c*sin(n*t):
> curve:=[r*cos(m*t), r*sin(m*t), z]:
> a:=2: b:=4/5: c:=1: m:=4: n:=7:
> tubeplot(curve, t=0..2*Pi, radius=1/4, numpoints=200,
> tubepoints=20, orientation=[45,10], style=patch,
> shading=xyz);
```

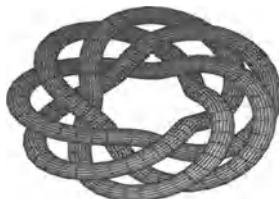


Figure 15.92. Torusknot of type (4,7).

### Implicit Plot

Nonsingular surfaces defined by an equation in any coordinate system whatsoever can be plotted with **implicitplot3d**. The catenoid is defined in the Cartesian coordinate system by  $\cosh z = \sqrt{x^2 + y^2}$ , and can be drawn as follows (Figure 15.93).

```
> implicitplot3d(cosh(z)=sqrt(x^2+y^2),
>                  x=-3..3, y=-3..3, z=-2.5..2.5, grid=[15,15,20],
>                  style=patchcontour, contours=15, axes=box,
>                  orientation=[30,70]);
```

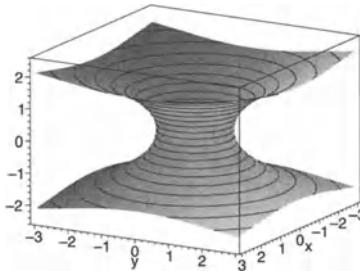


Figure 15.93. Implicit plot of the catenoid.

The option **style=patchcontour** ensures that the surface not only is shaded but also contains contour lines.

A more spectacular implicit plot with contours is the following example in spherical coordinates (Figure 15.94).

```
> r * (4 + 2*cos(3*theta) - cos(3*theta+2*phi)
>      - cos(-3*theta+2*phi) + 2*cos(2*phi)) = 2;
r(4 + 2 cos(3 theta) - cos(3 theta + 2 phi) - cos(3 theta - 2 phi) + 2 cos(2 phi)) = 2
> implicitplot3d(% , r=0..2, theta=0..2*Pi, phi=0..Pi,
>                  coords=spherical, grid=[10,30,30],
>                  style=patchcontour, thickness=2, orientation=[35,30]);
```

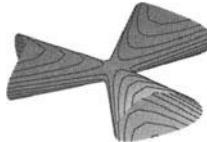


Figure 15.94. Implicit plot in spherical coordinates.

### Complex Plotting

The procedure **complexplot3d** can be used to plot complex functions from  $\mathbb{C}$  to  $\mathbb{C}$ : the height of the surface at position  $(x, y)$  is the magnitude of a function value at  $x + yi$ , and the color at this point is determined by the argument of the function value. An example: the gamma function (Figure 15.95).

```
> complexplot3d(GAMMA(z), z= -Pi-Pi*I .. Pi+Pi*I,
> view=0..5, grid=[30,30], orientation=[-120,45],
> axes=frame, style=patchcontour, thickness=2);
```

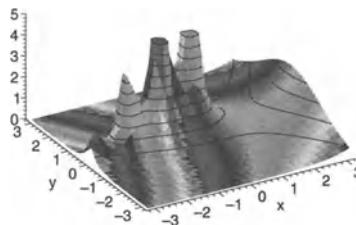


Figure 15.95. Plot of complex gamma function.

Compare this picture with Figure 2.8 of §2.6.

### Polyhedra

Some popular polyhedra such as the tetrahedron, octahedron, and dodecahedron (Figure 15.96) can easily be drawn with **polyhedraplot**. From the help file:

```
> polyhedraplot([0,0,0], polytype=dodecahedron,
> style=patch, scaling=constrained, orientation=[71,66]);
```



Figure 15.96. Dodecahedron.

### Sierpinsky's Tetrahedron

We shall start the examples that illustrate the features of the **plottools** package with the creation of Sierpinsky's tetrahedron. The example illustrates the commands to scale and to translate an existing 3D-graphics object. It is a small example of graphics programming in Maple (Figure 15.97).

We first create the tetrahedron that will form the starting object in an iteration. We can easily do this with the **tetrahedron** routine:

```
> base := tetrahedron([0,0,0], 1):
> display(base, style=patch, scaling=constrained,
> orientation=[60,50], shading=zgrayscale);
```



Figure 15.97. Standard tetrahedron as basic polytope in construction of Sierpinsky's tetrahedron.

Next we scale the tetrahedron by a factor  $\frac{1}{2}$  and make four copies of the scaled tetrahedron by translating it in the directions of the four vertices (Figure 15.98).

```
> vertices := {[0, 0, sqrt(3)],
> [0, 2*sqrt(2)/sqrt(3), -1/sqrt(3)],
> [-sqrt(2), -sqrt(2)/sqrt(3), -1/sqrt(3)],
> [sqrt(2), -sqrt(2)/sqrt(3), -1/sqrt(3)}]:
> translations := map(x->1/2*x, vertices):
> tetrahedra := {seq(translate(scale(base,
> 1/2, 1/2, 1/2), op(t)), t=translations)}:
> display(tetrahedra, style=patch, scaling=constrained,
> orientation=[60,50], shading=zgrayscale, thickness=2);
```



Figure 15.98. First step in construction of Sierpinsky's tetrahedron.

This process can be repeated over and over again, thus creating Sierpinsky's tetrahedron. Let us look at the picture after four iterations consisting of 1024 tetrahedra (Figure 15.99).

```
> translations := map(x->1/2*x, vertices):
> N := 4: # number of iterations
> for k from 2 to N do
>   newtranslations := map(x->x/2^k, vertices):
>   translations := {seq(seq(t+tnew, t=translations),
>                         tnew=newtranslations)}:
> end do:
> tetrahedra := {seq(translate(scale(base,
> 1/2^N, 1/2^N, 1/2^N), op(t)), t=translations)}:
> display(tetrahedra, style=patchnogrid, scaling=
> constrained, orientation=[55,50], shading=zgrayscale);
```



Figure 15.99. Sierpinsky's tetrahedron after four iterations.

### Canadian Flag

The following fun example of a look-alike of the Canadian flag illustrates transformations from two-dimensional to three-dimensional graphics and other mappings on graphics objects. We start with the Maple leaf that we created in §15.1.

```

> S := t -> 100/(100+(t-Pi/2)^8): # for scaling
> R := t -> S(t)*(2-sin(7*t)-cos(30*t)/2):
> mapleleaf := plot([R, t->t, -Pi/2..3/2*Pi],
> coords=polar, axes=none, color=red, numpoints=1000):

```

These commands create a closed curve resembling the Maple leaf. In the Canadian flag, the inner space should be colored red. The easiest way to do this is to replace the CURVES object name by POLYGONS. Moreover, we add two rectangles to get a plane Canadian flag and a border to indicate the white background (we assume that the default background color is white) (Figure 15.100).

```

> mapleleaf := subs(CURVES=POLYGONS, mapleleaf):
> rectangles := rectangle([-5,-1],[-3,4], color=red),
>                  rectangle([3,-1],[5,4],color=red):
> border := plot({-1,4},-3..3, color=black):
> flag2d := display([mapleleaf, rectangles, border],
>                   view=[-5..5,-1..4], scaling=constrained):
> flag2d;

```



Figure 15.100. Look-alike of Canadian flag.

Finally, we use the **transform** procedure from the **plottools** package with the mapping  $(x, y) \mapsto (x, y, 0)$  to convert the two-dimensional graphics object into a three-dimensional one. Finally, we apply the mapping  $(x, y, z) \mapsto (x, y, 1 + \frac{1}{15} \sin(x))$  to project the plane flag onto the sine surface (Figure 15.101).

```

> flag3d := transform((x,y)->[x,y,0])(flag2d):
> wavingflag3d :=
>      transform((x,y,z)->[x,y,1+1/15*sin(x)])(flag3d):
> display({flag3d, wavingflag3d},
>           scaling unconstrained, orientation=[-110,60], axes=none,
>           style=patchnogrid, shading=none);

```



Figure 15.101. Flat and waving Canadian flags.

## 15.13 Data Plotting

When a plot is to be made of some data or numbers instead of functions or expressions then specialized graphics commands must be used. We distinguish *list plotting commands*, whose command names start with **list** or **point**, and statistical plotting commands, which reside in the **stats** package.

### List Plotting Commands

Some function/expression plotting commands have equivalent list plotting commands in the **plots** package (Table 15.2).

List Plotting	Similar To	Graphics Object
<b>listplot</b>	<b>plot</b>	2D-plot of a list of data
<b>listplot3d</b>	<b>plot3d</b>	3D-plot of a list of data
<b>listcontplot</b>	<b>contourplot</b>	2D contour plot of a data array
<b>listcontplot3d</b>	<b>contourplot3d</b>	3D contour plot of a data array
<b>listdensityplot</b>	<b>densityplot</b>	a density plot of a data array
<b>pointplot</b>	<b>plot</b>	a plot of a list of 2D points
<b>pointplot3d</b>	<b>plot3d</b>	a plot of a list of 3D points

Table 15.2. List plotting.

For some of the following examples, we take a small data set about Dutch consumption of beverages in the last two decades from the electronic database *StatLine* of Statistics Netherlands (URL: <http://www.cbs.nl>).

Beverage	Unit	1980	1985	1990	1995	2000
beer	liter	86	85	91	86	83
soft drink	liter	64	66	86	97	106
wine	liter	12.9	15	14.5	16.6	18.8

Table 15.3. Dutch consumption of beverages per inhabitant.

The following table **beverage** contains lists of lists representing the annual consumption of beverages per inhabitant in the last two decades, collected every five years.

```
> beverage[beer] :=  
> [[1980,86], [1985,85], [1990,91], [1995,86], [2000,83]]:  
> beverage[soft_drink] :=  
> [[1980,64], [1985,66], [1990,86], [1995,97], [2000,106]]:  
> beverage[wine] :=  
> [[1980,12.9], [1985,15], [1990,14.5], [1995,16.6],  
> [2000,18.8]]:
```

All data are put in a Maple table called **beverage**. Let us load the **plots** package and plot the beer data as points joined with line segments (Figure 15.102).

```
> with(plots):
> listplot(beverage[beer]);
```

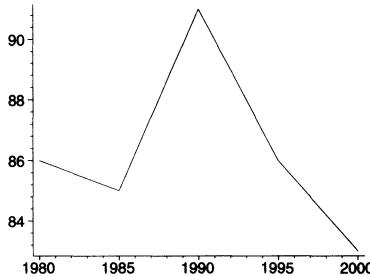


Figure 15.102. List plot of annual beer consumption per Dutch inhabitant.

By changing the drawing style to `point` you can graph the data points only. Instead you can also use the procedure `pointplot` (Figure 15.99).

```
> pointplot(beverage[beer], symbol=circle,
>           symbolsize=14, color=blue);
```

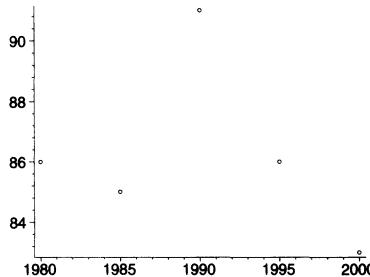


Figure 15.103. Point plot of annual beer consumption per Dutch inhabitant.

Both plots can be combined and displayed with a user-defined view frame (Figure 15.104).

```
> display({%,%}, view=[1979..2001,82..92],
>          labels=["year", "consumption"],
>          labeldirections=[horizontal,vertical]);
```

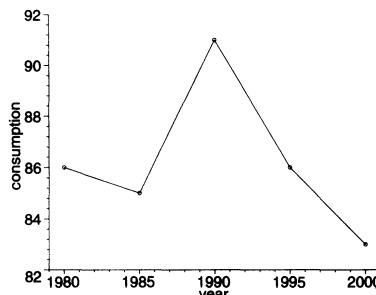


Figure 15.104. Annual beer consumption per Dutch inhabitant.

Two or more beverages can be plotted in one picture with different line and point styles, and a plot legend that explains the meaning of various elements (Figure 15.105).

```
> bp := listplot(beverage[beer],
>                 color=blue, legend="beer"):
> sp := listplot(beverage[soft_drink],
>                 color=black, legend="soft drink"):
> display({bp,sp}, labels=["year", "consumption"],
>          labeldirections=[horizontal,vertical]);
```

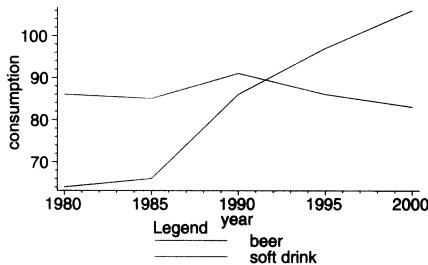


Figure 15.105. Annual beer and soft drink consumption per Dutch inhabitant.

If you want more control about style of the plot (e.g., if you want to have data points connected with segments or if you want to specify the style and location of the legend), then you must use the graphics package to get the work done yourself in the hard way. An example is shown in Figure 15.106.

```
> # data plot
> bp := pointplot(beverage[beer], symbol=circle,
>                  symbolsize=14, color=blue):
> bl := listplot(beverage[beer]):
> sp := pointplot(beverage[soft_drink], symbol=diamond,
>                  symbolsize=14, color=magenta):
> sl := listplot(beverage[soft_drink], linestyle=3):
> # plot legend
> with(plottools, [line,point]):
> bL := line([1990,72], [1992,72]):
> bP := point([1991,72], symbol=circle, symbolsize=14,
>             color=blue):
> bT := textplot([1992.4, 72, "beer"],
>                align=RIGHT, color=blue):
> sL := line([1990,70], [1992,70], linestyle=3):
> sP := point([1991,70], symbol=diamond, symbolsize=14,
>             color=magenta):
> sT := textplot([1992.4, 70, "soft drink"],
>                align=RIGHT, color=magenta):
> # Complete picture:
> display({bp,bl,sp,sl,bL,bP,bT,sL,sP,sT},
>          labels=["year", "consumption"],
>          labeldirections=[horizontal,vertical],
>          title="annual consumption of beverages\
per inhabitant (in liter)", view=[1979..2001,63..107]);
```

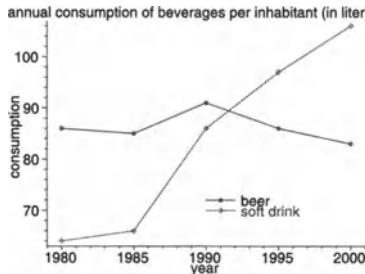


Figure 15.106. Annual beer and soft drink consumption per Dutch inhabitant.

Bar charts and pie charts can easily be built from graphics primitives or be generated as special cases of three-dimensional plotting routines. In the next subsection we shall see how the statistical plotting routine **histogram** can be used to generate a bar chart. Here, we show some other representations. We first convert our table of data into a matrix.

```
> M := map(d->map2(op,2,op(d)), [entries(beverage)]):
> setattr(M, matrix);
```

$$\begin{bmatrix} 86 & 85 & 91 & 86 & 83 \\ 12.9 & 15 & 14.5 & 16.6 & 18.8 \\ 64 & 66 & 86 & 97 & 106 \end{bmatrix}$$

Note that *M* is internally a list of lists object, regardless of its matrix-like display. The rows in *M* describe the following drinks:

```
> indices(beverage);
[beer], [wine], [soft_drink]
```

The procedure **matrixplot** from the **plots** package can be used to show these data as a histogram (Figure 15.107).

```
> matrixplot(M, heights=histogram, style=patch,
> shading=zgrayscale, orientation=[-75,75], axes=box,
> tickmarks=[[1.5="beer",2.5="wine",3.5="soft drink"],
> [1.5="1980",3.5="1990",5.5="2000"],5], labels=
> ["","","quantity"], font=[HELVETICA,DEFAULT,8]);
```

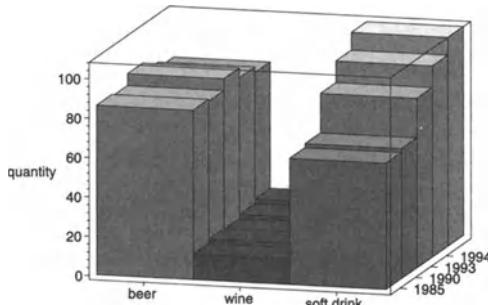


Figure 15.107. Histogram of annual consumption of beverages per inhabitant.

The bar chart of beverages consumed in 2000 (Figure 15.108) can be created as follows:

```
> data2000 := <<op(map(z->op(5,z), M))>>;
> matrixplot(data2000, heights=histogram, style=patch,
> shading=zgrayscale, orientation=[-90,90], axes=box,
> tickmarks=[[1.5="beer",2.5="wine",3.5="soft drink"],
> 0,5], labels=["","","quantity"], font=[HELVETICA,
> DEFAULT,8], title="consumption of beverages in 2000");
```

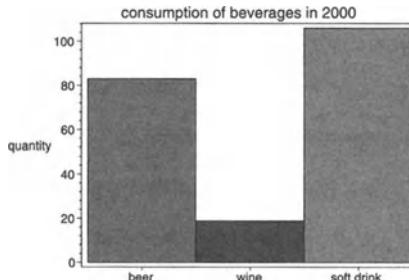


Figure 15.108. Bar chart of consumption of beverages in 2000 per Dutch inhabitant.

A circular bar chart (Figure 15.109) can easily be built with the help of the procedure **pieslice** from the **plottools** package. Below we implement a procedure that does the work for lists consisting of pairs [ *name of drink*, *quantity* ];

```
> circular_bar_chart := proc(d::listlist)
>   local n,i, pies, text, grayshades:
>   n := nops(d):
>   grayshades := map(x->COLOR(RGB,x,x,x),
>   [0.5,0.7,0.9]):
>   for i to n do
>     pies[i] := plottools['pieslice'](
>       [0,0],d[i][2],2*Pi*(i-1)/n..2*Pi*i/n,
>       color=grayshades[eval(1+(i mod 3))]):
>     text[i] := plots['textplot']([
>       1/2*d[i][2]*cos(Pi*(2*i-1)/n),
>       1/2*d[i][2]*sin(Pi*(2*i-1)/n),
>       convert(d[i][1], name)],
>       font=[HELVETICA,BOLD,12])
>   end do:
>   plots[display]([seq(pies[i], i=1..n),
>   seq(text[i], i=1..n)], axes=none,
>   scaling=constrained)
> end proc:
```

Let us apply it to the following data.

```
> data2000 := [seq([op(indices(beverage)[i]),
> data2000[i,1]], i=1..3)];
> data2000 := [[beer, 83], [wine, 18.8], [soft_drink, 106]];
> circular_bar_chart(data2000);
```



Figure 15.109. Circular bar chart of consumption of beverages in 1994.

The above procedure can easily be adapted to produce a pie chart or even an “exploded pie chart,” in which pies are separated a bit. This is left as an exercise.

A so-called *Pareto plot* of the 2000 data (Figure 15.110) can be generated by the **pareto** command. A Pareto diagram is a chart with the following elements: a tagged histogram of decreasing frequencies and a curve indicating the cumulative frequencies. Instead of frequencies we shall use percentages by scaling the data.

```
> Values := map(d->op(2,d), data2000);
      Values := [83, 18.8, 106]
> Percentages := map((x,s)->100*x/s,  Values,
>   +'(op(Values)));
      Percentages := [39.94225217, 9.047160731, 51.01058710]
> Labels := map(d->op(1,d), data2000);
      Labels := [beer, wine, soft_drink]
> pareto(Percentages, tags=Labels);
```

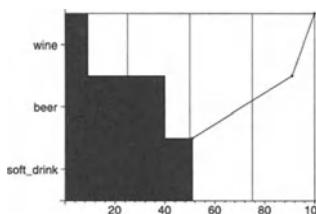


Figure 15.110. Pareto plot of consumption of beverages in 2000.

We end this subsection on list plotting commands with an example of **listdensityplot**, viz., a map of the set of Gaussian integers in the complex plane. A square  $(x, y)$  is drawn in black if the Gaussian integer  $x + yi$  is prime, and in white otherwise. For this, we first define a function that takes the value 0 if the corresponding Gaussian integer is prime. Maple has a package called **GaussInt** for arithmetic of Gaussian integers and it is this package that provides the primality test.

```

> isgaussianprime :=
>   (x,y) -> if GaussInt[GIpriime](x+I*y) then
>     0
>   else
>     1
>   end if:

```

Next, we generate a  $50 \times 50$  grid and draw the list density plot on this grid (Figure 15.111).

```

> data := array([seq([seq( isgaussianprime(i,j),
>   i=-25..25)], j=-25..25)]):
> ticks := [1=-25, 26=0, 51=25]:
> display(listdensityplot(data, style=patchnogrid,
>   axes=box, scaling=constrained),
>   xtickmarks=ticks, ytickmarks=ticks);

```

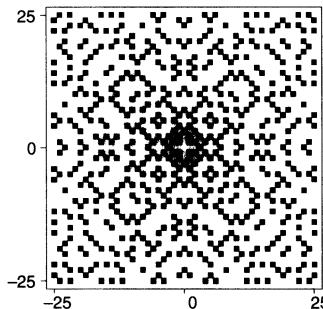


Figure 15.111. List density plot of Gaussian primes.

### Statistical Plotting

In the statistics package **stats** and more specifically in the subpackage **statplots** reside classical statistical plotting commands as listed in Table 15.4.

Plotting Command	Graphics Object
<b>boxplot</b>	box plot summarizing data
<b>histogram</b>	histogram of data
<b>scatterplot</b>	scatter plot of data

Table 15.4. Statistical plotting commands in **statplots** subpackage.

With the option **format=notched** of the **boxplot** command you can ask for a notched-box plot. The **scatterplot** procedure with the option **format=quantile** generates either a quantile plot, or a quantile-quantile plot, depending on the number of statistical lists passed. The option **format=symmetry** generates a symmetry plot. The utility functions **changecolor**, **xshift**, and **xchange** can be used to adapt the plots. Furthermore, the **fit** subpackage provides a tool for fitting curves to statistical data. Only linear fits are available in Maple.

We shall only show two examples of statistical plots. As promised before, we shall create a bar chart of the 2000 consumption of beverages (Figure 15.112) via the **histogram** procedure. Recall the data.

```
> data2000;
[[beer, 83], [wine, 18.8], [soft_drink, 106]]
```

Supply these data to the **histogram** procedure as weights.

```
> values2000 := [seq(Weight(i..i+1, data2000[i,2]),
> i=1..3)]:
```

Load the necessary packages and plot the bar chart.

```
> with(stats): with(statplots):
> histogram(values2000, color=yellow, tickmarks=
> [[1.5="beer", 2.5="wine", 3.5="soft drink"], default],
> font=[HELVETICA,DEFAULT,14]);
```

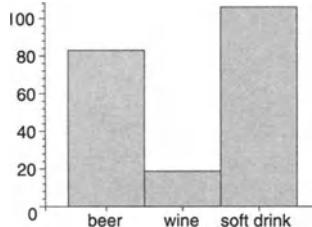


Figure 15.112. Bar chart of consumption of beverages in 2000.

The second example is the application of the linear regression method to numerical data and a plot of the least-squares fit. We assume that the numerical data are stored in a file, say **datafile**. You can display the file as follows:

```
> ssystem("more datafile")[2]; # display data
"0      1
 0.1   1.1
 0.2   1.2
 0.3   1.4
 0.4   1.6
 0.5   1.9
 0.6   2.3
 0.7   2.8
 0.8   3.5
 0.9   4.6
 1     6.3
  "
```

We decrease numerical precision and import these data. There are several ways of reading in the data; here, we shall use the procedure **ImportMatrix** and the **importdata** command from the **stats** package.

```
> with(stats): with(statplots): Digits := 5:
```

The **ImportMatrix** command reads tabular data. Just mention the name of the file from which you want to read columns of numbers.

```
> M := ImportMatrix("datafile");
```

$M := [11 \times 2 \text{ Matrix} \text{ Data Type: anything}$   
 $\text{Storage: rectangular Order: Fortran\_order}]$

You get an **rtable** object that is not shown because of its size. After conversion into a **matrix** object, you will see the data.

```
> convert(M, matrix);
```

$$\begin{bmatrix} 0 & 1 \\ 0.1 & 1.1 \\ 0.2 & 1.2 \\ 0.3 & 1.4 \\ 0.4 & 1.6 \\ 0.5 & 1.9 \\ 0.6 & 2.3 \\ 0.7 & 2.8 \\ 0.8 & 3.5 \\ 0.9 & 4.6 \\ 1 & 6.3 \end{bmatrix}$$

However, some routines in Maple expect a list of lists. So we carry out the following conversion.

```
> xydata := convert(M, listlist);
```

$xydata := [[0, 1], [0.1, 1.1], [0.2, 1.2], [0.3, 1.4], [0.4, 1.6], [0.5, 1.9],$   
 $[0.6, 2.3], [0.7, 2.8], [0.8, 3.5], [0.9, 4.6], [1, 6.3]]$

Some routines in the **stats** package do not accept data points, but expect that the columns in the data table are collected as lists. This can be done by transforming the data after reading. However, the **ImportMatrix** command has several options to deal with such things and with special formats. In the following command, the last two options are not necessary because they are the default options. We included them to show you how a different delimiter in the data file can be specified.

```
> Mt := ImportMatrix("datafile", transpose=true,
> source=delimited, delimiter="\t");
```

$Mt := [2 \times 11 \text{ Matrix} \text{ Data Type: anything}$   
 $\text{Storage: rectangular Order: Fortran\_order}]$

```
> datamatrix := convert(Mt, matrix);
```

$datamatrix := \begin{bmatrix} 0 & 0.1 & 0.2 & 0.3 & 0.4 & 0.5 & 0.6 & 0.7 & 0.8 & 0.9 & 1 \\ 1 & 1.1 & 1.2 & 1.4 & 1.6 & 1.9 & 2.3 & 2.8 & 3.5 & 4.6 & 6.3 \end{bmatrix}$

```
> datalist := convert(datamatrix, listlist);
```

```
datalist := [[0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1],
            [1, 1.1, 1.2, 1.4, 1.6, 1.9, 2.3, 2.8, 3.5, 4.6, 6.3]]
```

The **importdata** procedure of the **stats** package can be used as well to read the columns of numbers.

```
> data := importdata("datafile", 2);

data := [0., 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.],
         [1., 1.1, 1.2, 1.4, 1.6, 1.9, 2.3, 2.8, 3.5, 4.6, 6.3]
```

The data points can be shown in a scatter plot (Figure 15.113):

```
> dataplot := scatterplot(data, symbol=diamond,
>                         symbolsize=14, color=black):
> dataplot; # display the scatter plot
```

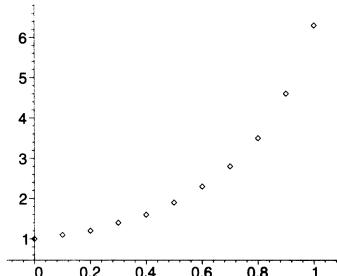


Figure 15.113. Scatter plot of data.

Next, we fit these data to the function  $x \mapsto a + b \sin(x) + c \sin(2x)$  via the least squares method, and plot the original data together with the fitted curve (Figure 15.114).

```
> fit[leastsquare[[x,y], y = a + b*sin(x) + c*sin(2*x),
> {a,b,c}]]([data]);

y = 1.1563 + 12.770 \sin(x) - 6.5290 \sin(2\,x)

> lsqcurve := plot(rhs(%), x=0..1, color=blue):
> display({dataplot, lsqcurve});
```

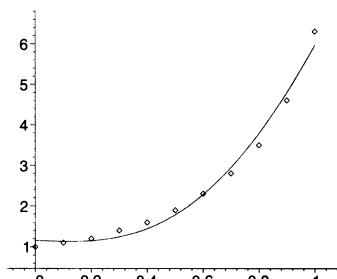


Figure 15.114. Least squares fit of data.

The **CurveFitting** package contains commands for various data fitting methods such as polynomial interpolation and spline approximation. We give two approximations of our data, viz., the above least squares approximation and the B-spline approximation of order 4 (Figure 15.115).

```
> with(CurveFitting):
> LeastSquares(M, x, 'curve'=a+b*sin(x)+c*sin(2*x));

$$1.1554 + 12.758 \sin(x) - 6.5202 \sin(2x)$$

> bspline := BSplineCurve(xydata, x):
> bsplinecurve := plot(bspline, color=blue):
> display({dataplot, bsplinecurve});
```

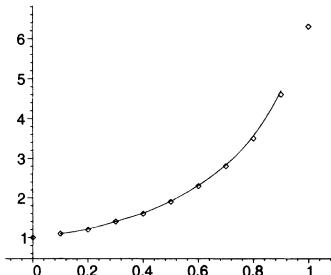


Figure 15.115. Spline approximation of data.

The **Interactive** procedure of the **CurveFitting** package provides a special GUI for spline approximation.

## 15.14 Animation

We end this chapter with some animations. Maple animations are a sequence of pictures (“frames”) that are displayed in rapid succession in an animation window or that are embedded in a worksheet like an active plot (the default number of frames is 16). In Figure 15.116, you see the first picture of a moving sine wave that has been generated with the procedure **animate** from the **plots** package. When you select this picture in the worksheet, control buttons appear for playing the movie clip. The playback mode can be chosen from **forward**, **backward**, and **circular**. The animation can be run in faster or slower mode, or step by step. These options are controlled in the Worksheet interface. But all other plot options can be placed inside the **animate** command and its three-dimensional analogue **animate3d**.

```
> animate(sin(2*Pi*(x+t)), x=0..4, t=0..1,
> numpoints=500, frames=8, color=black);
```

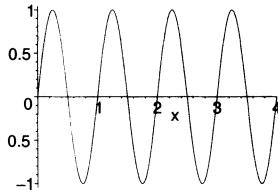


Figure 15.116. First picture in the animation of a sine wave.

The eight frames of the animation are actually displayed in a graphics array by application of the **display** command on the animation data structure. (Figure 15.117).

```
> display(% , tickmarks=[2,3]);
```

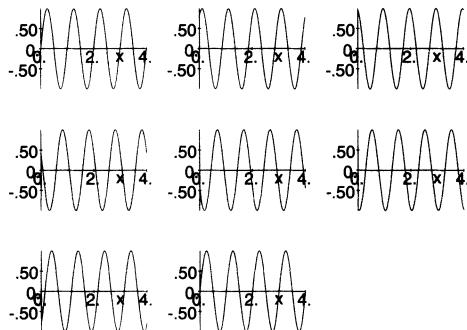


Figure 15.117. Eight frames in the animation of a sine wave.

Another two-dimensional animation can be generated by the procedure **animatecurve**. It visualizes the drawing of a real function. The following command visualizes the drawing of a sine graph. We only display the 8 frames of the movie clip in Figure 15.118.

```
> animatecurve(sin(2*Pi*x), x=0..1, color=black):
> display(% , tickmarks=[2,2]);
```

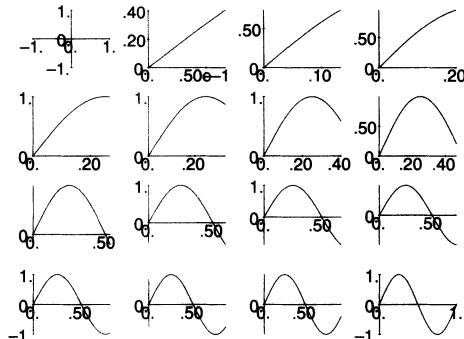


Figure 15.118. Eight frames in the animation of plotting a sine function.

Animations of three-dimensional surfaces that are generated by a formula or function and surfaces that are represented in parametric form can be made with the **animate3d** command or by combining several plots via **display** with the option **insequence = true**. Experiment with deformations of your favorite surface.

As an illustration of the option **insequence**, we produce below an animation of implicitly defined curves, by building up the necessary animation structure. First, we generate a table of implicit plots.

```
> for i from 0 to 8 do
>   P[i] := implicitplot(x^3 + y^3 - 5*x*y = 1 - i/4,
>   x=-3..3, y=-3..3, color=black)
> end do:
```

We use the **display** routine from the **plots** package to display the implicit plots in sequence, i.e., to produce the animation.

```
> display([seq(P[i], i=0..8)], insequence=true);
```

The animation is omitted, but the subsequent frames are shown in Figure 15.119.

```
> display(%, tickmarks=[2,2]);
```

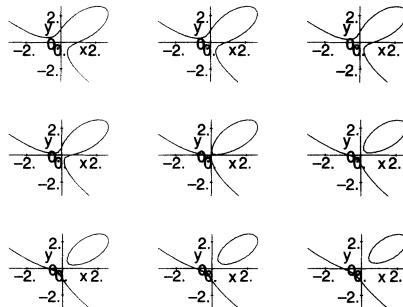


Figure 15.119. Frames of the animation of algebraic curves.

We end with the example of a “spin show” around the  $z$ -axis of an octahedron, i.e., rotations of the octahedron such that it looks, in an animation, like a spinning object.

```
> with(plottools): with(plots):
> base := display(octahedron([0,0,0],1),
>   color=blue, scaling=constrained, axes=none,
>   style=hidden, thickness=3, orientation=[30,60]):
> N := 16:
> angles := [seq(Pi*k/(2*N), k=0..(N-1))]:
> for phi in angles do
>   P[phi] := rotate(base,0,0,phi)
> end do:
> display([seq(P[phi], phi=angles)], insequence=true);
```

In circular playback mode the animation looks like a smoothly rotating octahedron.

## 15.15 List of Plot Options

In this section we list all options of plot routines. They are divided into options that can be used in all plot routines and options specific to the graphics mode. Each table entry consists of the name of the option, its possible values, and its purpose. Note that values of options can be specified in lower-case as well as in upper-case characters (unless we explicitly use upper-case characters in the lists below); both British and American spellings are allowed. Usual defaults of options have been underlined in the tables below. The special names **DEFAULT** and **default** are for built-in and platform dependent choices.

### Common plot options

**axes = style**

The following axes styles are available.

Axes Style	Meaning
<b>box</b>	plot in a bounding box
<b>DEFAULT</b>	default axes
<b>frame</b>	axes along the edges of the plot
<b>none</b>	no axes
<b>normal</b>	Cartesian axes

**axesfont = [family, style, size]**

Use the font for labels of tick marks on the axes. See the option **font** for more details.

**color = colorspec**

Colors can be specified in the following way.

Color Specification	Meaning
<b>COLOR(RGB, r, g, b)</b>	use the RGB color
<b>COLOR(HSV, h, s, v)</b>	use the HSV color
<b>HUE( h )</b>	use this shortcut for <b>COLOR(HSV, h, 0.9, 1)</b>
<i>color name</i>	use the predefined color name (aquamarine, black, blue, etc.) stored in <b>plot/colortable</b> (enter <b>print(`plot/colortable`)</b> to print.)

**coords = name**

Use the specified coordinate system. Names to think of for two-dimensional graphics are **bipolar**, **cardiod**, **cartesian**, and so on (enter **?plot,coords** to see more allowed names). In three-dimensional graphics, popular coordinate systems are **cartesian**, **cylindrical**, **spherical**, and so on (enter **?plot3d,coords** to see more allowed names).

**font** = [*family*, *style*, *size*]

Specify the font used for text objects. *size* specifies the font size in points. The following combinations of *family* and *style* are available.

<i>Font Family</i>	<i>Font Style</i>
COURIER	BOLD, BOLD OBLIQUE, DEFAULT, OBLIQUE
HELVETICA	BOLD, BOLD OBLIQUE, DEFAULT, OBLIQUE
TIMES	ROMAN, BOLD, ITALIC, BOLD ITALIC
SYMBOL	—

**labeldirections** = [symbol *d<sub>x</sub>*, symbol *d<sub>y</sub>* (, string *d<sub>z</sub>*)]

Specify the direction in which labels are printed along the axes. The direction must be a symbol **horizontal** or **vertical**. The default direction of any labels is horizontal, in case none is specified.

**labelfont** = [*family*, *style*, *size*]

Use the font for labels on the axes. See the option **font** for more details.

**labels** = [string *s<sub>x</sub>*, string *s<sub>y</sub>* (, string *s<sub>z</sub>*)]

Add labels to the main axes.

**linestyle** = nonnegative integer *n* or a string

Draw the curves in the graph using the dash pattern, which is specified by a nonnegative integer between 0 (default) and 7. Instead of an integer from {1, 2, ..., 6} you may use the corresponding name **SOLID**, **DOT**, **DASH**, **DASHDOT**, **LONGDASH**, **SPACEDASH**, or **SPACEDOT**.

**numpoints** = natural number *n*

Specify the minimum number of sample points used (default: *n* = 49 for 2D plots, *n* = 625 for 3D plots).

**scaling** = *mode*

Three scaling modes are available:

<i>Scaling</i>	<i>Meaning</i>
<b>constrained</b>	use the same scaling of the graph in all directions
<b>DEFAULT</b>	default scaling of the graph
<b>unconstrained</b>	use independent scaling of graph

**symbol** = *name*

When the option **style** = **point** is used, you can specify the following symbols for drawing at each sample point or vertex of a polygon.

<i>Specification</i>	box	circle	cross	<b>DEFAULT</b>	diamond	point
<i>Symbol</i>	□	○	+	usually +	◇	·

**symbolsize** = natural number *n*

Specify the size (in points) of a symbol used in plotting. The default symbol size is 10 points.

**thickness** = nonnegative integer  $n$ 

Specify the thickness of lines on the plot by a nonnegative integer and by modular arithmetic bring it to the range [0, 15]. Default value is 0 and a higher value means a thicker line.

**tickmarks** = [*xspec*, *yspec* (, *zspec*)]

Specify the number, location, and labeling of the tick marks on the main axes. Each specification may be an integer (i.e., the minimum number of tick marks), a list of numbers (i.e., a list of locations of tick marks), a list of equations (i.e., a list of locations and labels of tick marks), or the special value **default**.

**title** = string *s*

Give the graph a title (default: *s*=NULL, i.e., no title).

**titlefont** = [*family*, *style*, *size* ]

Use the font for a title of the graph. See the option **font** for more details.

**view** = *v*

Specify the display region. Each specification *v* may be a range of function values, the special value **default**, or a list of such specifications for the main directions. **default** means a range that allows display of entire curve or surface in that direction. Default value of **view** is **default** along all main axes.

### Options specific to plot

**adaptive** = false/**true**

Disable/Enable adaptive plotting.

**discont** = **true**

(*Only for expressions*) By default, Maple does not care about discontinuities. But with the option set the value **true**, Maple tries to resolve them.

**filled** = **true**

Fill the space between the curve and the horizontal axis.

**legend** = *specs*

Specify a legend for a plot by either a string or a list of strings. When more than one curve is being plotted then they must be specified as a list whose length is equal to the number of curves.

**resolution** = natural number  $n$ 

Specify the number of pixels across the display device (default: *n* = 200).

**sample** = list of parameter values

Specify what parameter values are used for the initial sampling.

**style = name**

The following drawing styles are available.

<i>Style</i>	<i>Meaning</i>
<u>line</u>	connect sample points by straight line segments and omit the filled interior of polygons
<u>patch</u>	render points as symbols, curves as line segments, and polygons as filled regions with a border
<u>patchnogrid</u>	As <u>patch</u> , but omit the border on polygons
<u>point</u>	display sample points and vertices of polygons only

**xtickmarks = [xspec]**

Specify the tick marks on the horizontal axis. See the option **tickmarks** for more details.

**y tickmarks = [yspec]**

Specify the tick marks on the vertical axis. See the option **tickmarks** for more details.

### Options specific to **plot3d**

**ambientlight = [r, g, b]**

Use a nondirectional shading with red, green, and blue intensities in the RGB color model for user defined lighting.

**color = colorfunc**

Beside all specifications of colors that were listed in the table of common plot option, you can color a space curve or surface. *colorfunc* can be a *function* or *expression* mapping the resulting values over the plotting domain to the HUE coloring scheme. *colorfunc* can also be a list [*r*, *g*, *b*], where *r*, *g*, and *b* are either functions or expressions that depend only on the plotting domain variables. The resulting values are mapped to the RGB coloring scheme.

**contours = specification**

You can specify the number *n* of contours when the drawing style equals **contour** or **patchcontour**. Default: *n* = 10. Moreover, the specification may be a *list of function values* to be used for contours when the drawing style equals **contour** or **patchcontour**.

**filled = false/true**

If the **filled** option in **plot3d**, is set to **true**, the region between the surface and the *xy*-plane is displayed as solid. The default value is **false**.

**grid = [natural number *n<sub>x</sub>*, natural number *n<sub>y</sub>*]**

Specify the dimensions of the grid on which the points will be generated (equally spaced). Default dimensions are *n<sub>x = *n<sub>y</sub>* = 25.</sub>*

**gridstyle = rectangular/triangular**

Use a rectangular/triangular grid.

**light =  $[\phi, \theta, r, g, b]$**

Simulate a light source with given *red*, *green*, and *blue* intensities in the RGB color model shining on the graphics object from the direction specified in the spherical coordinates  $[\phi, \theta]$  (in degrees).

**lightmodel = name**

In the following table, the predefined lighting models are specified in terms of the option **ambientlight** and **light**.

<i>Model</i>	<i>Ambient Light</i>	<i>Lights</i>
<code>light1</code>	[ 0.5, 0.5, 0.5 ]	[ 90, -45, 0, 0.9, 0 ], [ 45, -45, 0, 0, 0.9 ], [ 90, 45, 0.9, 0, 0 ]
<code>light2</code>	[ 0.5, 0.5, 0.5 ]	[ 45, 90, 0.9, 0, 0 ], [ 45, 45, 0, 0.9, 0 ], [ 90, -45, 0, 0, 0.9 ]
<code>light3</code>	[ 0.5, 0.5, 0.5 ]	[ 45, 45, 0, 0., 0.9 ], [ 45, 45, 0, 0.9, 0 ], [ 0, 135, 0.9, 0, 0 ]
<code>light4</code>	[ 0.6, 0.6, 0.6 ]	[ 85, 60, 0.8, 0.8, 0.8 ]

**orientation =  $[\theta, \phi]$**

Specify the direction from which the plot is viewed; the direction of the eyepoint is in spherical coordinates:  $\phi$  is the angle with respect to the *z*-axis,  $\theta$  is the angle of the projection of the view point to the *xy*-plane (in degrees). See Figure 15.75; default values are  $\theta = 45$  and  $\phi = 45$ .

**projection = *p***

The following perspectives are available.

<i>p</i>	<i>Projection Value</i>	<i>Meaning</i>
<code>fisheye</code>	0	wide angle perspective
<code>normal</code>	1/2	intermediate perspective
<code>orthogonal</code>	1	no perspective
—	real number $p \in [0, 1]$	a user defined perspective

**shading = name**

Several shadings are available.

<i>Shading</i>	<i>Meaning</i>
<code>none</code>	no shading scheme used
<code>xyz</code>	shade in color ranges along all axes
<code>xy</code>	shade in color ranges along <i>x</i> and <i>y</i> axes only
<code>z</code>	shade in one color range along the <i>z</i> axis
<code>zgrayscale</code>	shade according to the <i>z</i> -value using only gray tints
<code>zhue</code>	shade according to the <i>z</i> -value in the HUE color model

**style = name**

The following drawing styles are available.

<i>Style</i>	<i>Meaning</i>
<code>contour</code>	draw only the contour lines of the surface
<code>hidden</code>	as <code>wireframe</code> , but with hidden-line removal, i.e., all line segments that are hidden behind other parts of the surface are removed
<code>line</code>	synonym for <code>wireframe</code>
<code>patch</code>	draw the surface with shaded patches on a wireframe grid with hidden-surface removal, i.e., all patches that are hidden behind other parts of the surface are removed
<code>patchcontour</code>	as <code>patch</code> but with contour lines
<code>patchnogrid</code>	as <code>patch</code> but omit the grid
<code>point</code>	display sample points and vertices of polygons only
<code>wireframe</code>	draw a wireframe surface, connecting adjacent sampled points with line segments

### Options related to animations

`frames` = natural number  $n$

$n$  frames are displayed in an animation (default:  $n=16$ ).

`framescaling` = *name*

The scaling of subsequent frames in an animation can be as follows.

<i>Frame Scaling</i>	<i>Meaning</i>
<code>default</code>	default relationship
<code>nonuniform</code>	scale each frame in an animation individually
<code>uniform</code>	scale all frames in an animation by an equal amount

`insequence` = `true`

By default, `display` combines several plots in one picture. With the option set the value `true`, plots are displayed as separate frames in an animation.

## 15.16 Exercises

- Plot the function  $x \mapsto \frac{\sin 2x}{\sin x}$  on the interval  $(0, 4\pi)$ .
- The sine integral  $Si$  is defined as  $Si(x) = \int_0^x \frac{\sin t}{t} dt$ . Plot the sine integral over the interval  $(0, 100)$ , compute  $\lim_{x \rightarrow \infty} Si(x)$ , and compare this with the result that you read off the picture.
- Plot the factorial function  $x \mapsto x!$  and its derivative over the interval  $(-4, 4)$ .

4. Enter the command

```
plot( cosh(x)^2 - sinh(x)^2 - 1, x=0..10 );
```

and explain what you see. Can you improve the result?

5. Plot the function  $x \mapsto x \sin(1/x)$  over an “interesting” domain around zero.

6. Plot the function  $t \mapsto \int_{-\infty}^t e^{-\theta^4} d\theta$  on the interval  $(-2, 2)$ .

7. Plot the function  $x \mapsto \int_0^x \sin(\sin \xi) d\xi$  on the interval  $(0, 2\pi)$ .

8. Plot some Lissajous figures, which are defined in parametric form by

$$t \mapsto (\sin(m t), \sin(n t)),$$

for integers  $m$  and  $n$ . Pay special attention to the case where  $m$  and  $n$  are consecutive Fibonacci numbers.

9. Draw the Lemniscate of Bernoulli defined by

(a) the equation  $(x^2 + y^2)^2 = (x^2 - y^2)$ ,

(b) the polar equation  $r^2 = \cos 2\phi$ , and

(c) the parameterization  $x = \frac{\cos t}{1 + \sin^2 t}$ ,  $y = \frac{\cos t \sin t}{1 + \sin^2 t}$ , for  $-\pi \leq t \leq \pi$ .

10. Draw the following parametric plots.

(a)  $t \mapsto \left( \frac{t^2 - 1}{t^2 + 1}, \frac{2t}{t^2 + 1} \right)$ , for  $t \in (-\infty, \infty)$ .

(b)  $t \mapsto \left( \frac{4}{9}t^3 - \frac{14}{9}t^2 + \frac{1}{9}t + 1, -\frac{4}{9}t^3 - \frac{1}{9}t^2 + \frac{14}{9}t \right)$ , for  $t \in (0, 1)$ .

(c) Draw the previous parametric plot together with the plot of  $t \mapsto (\cos(\frac{\pi}{2}t), \sin(\frac{\pi}{2}t))$  for  $t \in (0, 1)$ .

11. Plot the Heaviside step function  $H$  and next draw the graphs of the functions below.

(a)  $H(x - 1) - H(x - 2)$

(b)  $\sum_{i=0}^{10} (-1/2)^i H(x - i/2)$

(c)  $(1 - |x|)(H(x + 1) - H(x - 1))$

12. Draw the parametric plot  $t \mapsto (FresnelC(t), FresnelS(t))$  for  $t$  in  $(0, 6)$  and describe the asymptotic behavior of this parametric curve.

13. Plot the Folium, which is defined as a polar plot with radial function  $\cos \theta(4 \sin^2 \theta - 1)$ .

14. Plot the function  $x \mapsto x + \cos(\pi x)$  on the interval  $(-49, 49)$

(a) without changing the defaults of **plot**.

(b) with different options.

15. Plot the function  $x \mapsto e^x + \ln|4 - x|$  on the interval  $(0, 5)$ . What do you think of it?

16. Plot the graph of the function  $(x, y) \mapsto \frac{x}{x^2 + y^2}$ , for  $x$  and  $y$  ranging from  $-1$  to  $1$ .
17. Plot the “monkey saddle” defined by the function  $(x, y) \mapsto x(x^2 - 3y^2)$  under various options.
18. Plot the function  $(x, y) \mapsto \sin(2\pi x) \sin(2\pi y)$ , for  $x$  and  $y$  ranging from  $0$  to  $25$ , without changing the default options. What do you think of it?
19. Plot the surface parameterized by

$$(\phi, \theta) \mapsto (\cos \phi \sin(2\theta), \sin \phi \sin(2\theta), \sin \theta),$$

where  $\phi$  and  $\theta$  range from  $0$  to  $2\pi$ .

20. Draw a contour plot and a density plot of the strength of the potential field of a configuration of unit charges at positions  $(-1, 0)$  and  $(1, 0)$ .
21. Draw Klein’s bottle in Dieudonné’s parameterization [72]

$$\begin{aligned} x &= (2 + \cos(u/2) \sin t - \sin(u/2) \sin(2t)) \cos u, \\ y &= (2 + \cos(u/2) \sin t - \sin(u/2) \sin(2t)) \sin u, \\ z &= \sin(u/2) \sin t + \sin(u/2) \sin(2t), \end{aligned}$$

where  $0 \leq u \leq 2\pi$ ,  $0 \leq t \leq 2\pi$ . Compare the result with the first example in the “graphics gallery” in [163].

22. Draw Klein’s bottle in Banchoff’s parameterization [10]
- $$\begin{aligned} x &= \left( \cos(\theta/2) (\sqrt{2} + \cos \phi) + \sin(\theta/2) (\cos \phi \sin \phi) \right) \cos \theta, \\ y &= \left( \cos(\theta/2) (\sqrt{2} + \cos \phi) + \sin(\theta/2) (\cos \phi \sin \phi) \right) \sin \theta, \\ z &= -\sin(\theta/2) (\sqrt{2} + \cos \phi) + \cos(\theta/2) (\cos \phi \sin \phi), \end{aligned}$$

where  $0 \leq \theta \leq 4\pi$ ,  $0 \leq \phi \leq 2\pi$ . Compare the result with the color graph in [140].

23. Draw the Möbius strip. (Hint: make a parameterization by rotating a line segment along a circle with varying elevation angle.)
24. Draw the catenoid defined by the parameterization

$$(\theta, z) \mapsto (\cos \theta \cosh z, \sin \theta \cosh z, z),$$

where  $0 \leq \phi \leq 2\pi$  and  $-1 \leq z \leq 1$ , directly with this parameterization, and as a cylindrical plot.

25. Draw Enneper’s minimal surface defined by the parameterization
- $$(u, v) \mapsto \left( \frac{u}{2} - \frac{u^3}{6} + \frac{uv^2}{2}, -\frac{v}{2} + \frac{v^3}{6} - \frac{vu^2}{2}, \frac{u^2}{2} - \frac{v^2}{2} \right).$$
26. Draw Scherk’s minimal surface defined by the equation  $\exp(z) \cos x = \cos y$ .

27. Generate a contour plot and a density plot of the function  $(x, y) \mapsto \sin(xy)$ .
28. Plot the two-dimensional gradient vector field of the function  $(x, y) \mapsto \sin x \cos y$ .
29. Make an animation of a rotating spiral.
30. Build a picture of a house using graphics primitives only.
31. The RGB specification of a gray shading is given by three equal numbers between 0 and 1. Build up a color chart for gray shadings.
32. Prove and visualize Pascal's theorem with the `geometry` package.

**Pascal's Theorem.**

Let  $A, B, C, D, E$ , and  $F$  be six points on a circle. Let  $P$  be the intersection of the lines  $AB$  and  $DF$ ,  $Q$  be the intersection of the lines  $BC$  and  $FE$ , and let  $R$  be the intersection of the lines  $CD$  and  $EA$ . Then  $P, Q$ , and  $R$  are collinear.

33. Prove and visualize the following theorem of Euler.

**Euler's Theorem.**

In a triangle, the orthocenter (intersection of altitudes), the centroid (intersection of medians), the circumcenter, and the incenter (center of the circle circumscribing, resp. inscribed in, the triangle) lie on a line.

34. Verify and visualize Feuerbach's nine-point circle. Verify with Maple that Feuerbach's nine-point circle is the same as the Euler circle.

**Feuerbach's nine-point circle.**

Consider a triangle  $T$  with vertices  $A, B$ , and  $C$ . Let  $A', B', C'$  be the midpoints of the edges, let  $D, E, F$  be the feet of the altitudes, and let  $X, Y, Z$  be the midpoints of segments connecting the orthocenter  $H$  (intersection of altitudes) to the vertices of  $T$ . Then, the points  $A', B', C', D, E, F, X, Y$ , and  $Z$  lie on a circle whose center  $N$  is the midpoint of the segment joining  $H$  to the circumcenter  $O$  of  $T$ , and whose radius is half the circumradius of  $T$ .

35. Use the `plottools` package to make an “exploded” version of the pie chart of Figure 15.109 in which pie segments are separated a bit.

# 16

## Solving Equations

In this chapter we shall discuss several methods implemented in Maple to solve (systems of) equations and inequalities of various types. Special attention is paid to systems of polynomial equations; the use of the Gröbner basis package is discussed briefly. A recurrence relation is another type of equation that will be discussed in detail. We shall consider exact methods (over various domains) as well as approximate numerical methods. Examples in this chapter come from application areas such as electronic circuit design, chemical kinetics, neural networks, geodesy, and dimensional analysis.

### 16.1 Equations in One Unknown

In its simplest form, the Maple procedure **solve** takes an equation in one unknown and tries to solve it analytically.

```
> eqn := (x-1)*(x^2+x+1);  
eqn := (x - 1) (x2 + x + 1)  
> sol := solve(eqn, x);  
sol := 1, - $\frac{1}{2}$  +  $\frac{1}{2}I\sqrt{3}$ , - $\frac{1}{2}$  -  $\frac{1}{2}I\sqrt{3}$ 
```

To check the solutions, substitute them back into the original equation.

```
> subs(x=sol[2], eqn);  
(- $\frac{3}{2}$  +  $\frac{1}{2}I\sqrt{3}$ ) ((- $\frac{1}{2}$  +  $\frac{1}{2}I\sqrt{3}$ )2 +  $\frac{1}{2}$  +  $\frac{1}{2}I\sqrt{3}$ )  
> expand(%);  
0
```

Although this example does not show it, it is in general more convenient to verify solutions using the procedure **eval**.

```
> eval(eqn, x=sol[3]);

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

> expand(%);
0
```

The equations may contain several unknowns and still you can ask Maple to solve it for one of the unknowns, in terms of the others.

```
> eqn := x^3 + 2*a*x^2 + a*x = 1;
eqn :=  $x^3 + 2ax^2 + ax = 1$ 
> solve(eqn, x);

$$\begin{aligned} &\frac{\%1^{(1/3)}}{6} - \frac{6\%2}{\%1^{(1/3)}} - \frac{2a}{3}, \\ &-\frac{\%1^{(1/3)}}{12} + \frac{3\%2}{\%1^{(1/3)}} - \frac{2a}{3} + \frac{1}{2}I\sqrt{3}\left(\frac{\%1^{(1/3)}}{6} + \frac{6\%2}{\%1^{(1/3)}}\right), \\ &-\frac{\%1^{(1/3)}}{12} + \frac{3\%2}{\%1^{(1/3)}} - \frac{2a}{3} - \frac{1}{2}I\sqrt{3}\left(\frac{\%1^{(1/3)}}{6} + \frac{6\%2}{\%1^{(1/3)}}\right) \end{aligned}$$


$$\begin{aligned} \%1 &:= 72a^2 + 108 - 64a^3 + 12\sqrt{-84a^3 - 12a^4 + 108a^2 + 81} \\ \%2 &:= \frac{1}{3}a - \frac{4}{9}a^2 \end{aligned}$$

```

Maple finds the three (complex) solutions, which are represented as a sequence of formulae. From the example, it is clear how Maple represents complicated expressions; the system looks for common subexpressions and gives them names like  $\%1$ ,  $\%2$ , and so on. You can refer to these labels as long as they are not replaced by other values.

```
> expr := %2 / %1^(1/3);
expr :=  $\frac{\frac{1}{3}a - \frac{4}{9}a^2}{(72a^2 + 108 - 64a^3 + 12\sqrt{-84a^3 - 12a^4 + 108a^2 + 81})^{(1/3)}}$ 
> rationalize(expr^3);

$$-\frac{a^2}{648} - \frac{1}{432} + \frac{a^3}{729} + \frac{\sqrt{-84a^3 - 12a^4 + 108a^2 + 81}}{3888}$$

```

In practice, only immediate reference is safe; the system itself may reuse the labels and destroy former information. In the worksheet interface, there is an alternative way to select the right-hand side of the label: copy the formula into an input line and give it a name, if you wish.

## 16.2 Abbreviations in **solve**

Maple expects an equation or a set of equations as the first argument in a call to **solve**, but the system kindly supplements expressions with " $= 0$ ".

```
> solve(a+ln(x-3)-ln(x), x);

$$\frac{3e^a}{-1 + e^a}$$

```

As the second argument, Maple expects a variable or a set of variables. When this argument is absent, Maple finds all indeterminates in the first argument with a command similar to

```
indets(eqns, name) minus {constants}
```

and uses the result as the second argument of **solve**. This is convenient, but sometimes it has a strange effect.

```
> solve(a+ln(x-3)-ln(x));

$$\{a = -\ln(x - 3) + \ln(x), x = x\}$$

```

Maple solved the equation for  $x$  and  $a$ . The solution  $x = x$  means that  $x$  can have any value.

You have seen that Maple uses labels to abbreviate large common sub-expressions. The system sometimes uses another kind of abbreviation, viz., the **RootOf** description of algebraic numbers.

```
> poly := x^7 - 2*x^6 - 4*x^5 - x^3 + x^2 + 6*x + 4;

$$poly := x^7 - 2x^6 - 4x^5 - x^3 + x^2 + 6x + 4$$

> sol := solve(poly);

$$sol := 1 + \sqrt{5}, 1 - \sqrt{5}, \text{RootOf}(-Z^5 - Z - 1, index = 1),$$


$$\text{RootOf}(-Z^5 - Z - 1, index = 2), \text{RootOf}(-Z^5 - Z - 1, index = 3),$$


$$\text{RootOf}(-Z^5 - Z - 1, index = 4), \text{RootOf}(-Z^5 - Z - 1, index = 5)$$

```

In this way, Maple informs you that it has found two real solutions, viz.,  $1 + \sqrt{5}$  and  $1 - \sqrt{5}$ , in analytical form, and that the other solutions are roots of the polynomial equation  $Z^5 - Z - 1 = 0$  in  $Z$ . The roots are indexed and Maple uses this information in calculations. We give a few examples.

```
> evalf([sol]); # numerical approximations
[3.236067978, -1.236067978, 1.167303978, 0.1812324445 + 1.083954101 I,
-0.7648844336 + 0.3524715460 I, -0.7648844336 - 0.3524715460 I,
0.1812324445 - 1.083954101 I]
> sol[5]+sol[6]; # sum of conjugated roots
RootOf(-Z^5 - Z - 1, index = 3) + RootOf(-Z^5 - Z - 1, index = 4)
```

```

> is(%, real); # the sum is a real number
                                         true
> prod := product(sol[i], i=3..7):
> simplify(prod);

RootOf( $-Z^5 - Z - 1$ , index = 1) RootOf( $-Z^5 - Z - 1$ , index = 2)
RootOf( $-Z^5 - Z - 1$ , index = 3) RootOf( $-Z^5 - Z - 1$ , index = 4)
RootOf( $-Z^5 - Z - 1$ , index = 5)

```

No simple result; you should have used the simplification in the context of algebraic numbers.

```
> evala(Simplify(prod));
```

```
1
```

You can ask directly for real solutions by using the **RealDomain** package or you can select real solutions afterwards.

```

> select(x->Im(x)=0, [sol]); # select real solutions
[ $1 + \sqrt{5}$ ,  $1 - \sqrt{5}$ , RootOf( $-Z^5 - Z - 1$ , index = 1)]
> use RealDomain in solve(poly) end use;
> # calculate only real solutions
[ $1 + \sqrt{5}$ ,  $1 - \sqrt{5}$ , RootOf( $-Z^5 - Z - 1$ , index = 1)]

```

By setting the environment variable **\_EnvExplicit** to **false**, all solutions that are not rational will be reported as **RootOfs**.

```

> _EnvExplicit := false:
> solve(poly);

```

$2\text{RootOf}(-Z^2 - Z - 1, \text{label} = \text{_L9}), \text{RootOf}(-Z^5 - Z - 1, \text{label} = \text{_L8})$

The labels allow us to distinguish between several roots of the same (polynomial) equation. If you apply the procedure **evalf**, Maple will compute the value of the principal branch. With **allvalues** you can get all solutions.

```

> map(evalf, [%]);
[3.236067978, 1.167303978]
> map(allvalues, [%]);
[ $1 + \sqrt{5}$ ,  $1 - \sqrt{5}$ , RootOf( $-Z^5 - Z - 1$ , index = 1),
RootOf( $-Z^5 - Z - 1$ , index = 2), RootOf( $-Z^5 - Z - 1$ , index = 3),
RootOf( $-Z^5 - Z - 1$ , index = 4), RootOf( $-Z^5 - Z - 1$ , index = 5)]

```

## 16.3 Some Difficulties

The main difficulties with **solve** are that no solutions are found in cases where it is known that solutions exist, that not all solutions are found, and that superfluous “solutions” are found. We shall give examples of all cases mentioned and make suggestions to assist Maple.

### No solution found, but there exists at least one.

Often it is known that there exist solutions for an equation or system of equations, but no general method for finding the solutions is available. There are cases where, from a mathematical point of view, it is hopeless to find a general solution in analytical form. It is well known from Galois theory that a general formula using radicals only for roots of polynomials of degree five or higher does not exist. From Maple you cannot expect more; it only provides you with a set of reasonably good algorithms for solving equations. Nevertheless, in quite a few mathematical problems general methods fail whereas the user recognizes a way to find a solution. Often this is based on recognizing some pattern within a formula, which the computer algebra system does not see.

When Maple does not find any solution, three things can happen:

1. Maple returns the equation back in the form of a **RootOf** so that you can still process it further.

```
> solve(cos(x)=x, x);
RootOf(-_Z + cos(_Z))
> evalf(%);
0.7390851332
```

2. The system keeps silent because there does not exist a solution.

```
> solve(x=x+1, x);
```

3. The system simply returns a **RootOf** expression or keeps silent because it did not find a solution but there might be one, which Maple affirms by setting the variable **\_SolutionsMayBeLost** to **true**.

```
> solve(sin(x)=exp(x), x);
RootOf(-_Z + ln(sin(_Z)))
> _SolutionsMayBeLost;
true
```

Numerical evaluation leads to a complex solution, where we know that there exists real solutions.

```
> evalf(%);
```

$$0.3627020561 - 1.133745919 I$$

Maple is more communicative when you increase the value of the variable `infolevel[solve]`. Since `solve` uses option `remember` to keep track of answers of previous problems, we first **forget** the previous results of `solve`.

```
> forget(solve); # forget previous results of solve
> infolevel[solve] := 2; # make Maple more communicative
> solve(sin(x)=exp(x), x);

solve: Warning: solutions may have been lost
```

$$\text{RootOf}(-Z + \ln(\sin(Z)))$$

It is known that no closed-form solution of this equation exists. So, in this case, you cannot blame Maple for not having found a solution in another format than the **RootOf** expression. However, sometimes results of `solve` are a bit disappointing. For example, computations that require branch selection or that involve radicals, logarithms, or trigonometric functions usually do not come easily to computer algebra systems, and Maple is no exception in this respect.

```
> infolevel[solve] := 1; # little userinfo
> eqn := x+x^(1/3)=-2;

eqn := x + x^(1/3) = -2
> solve(eqn, x);

solve: Warning: no solutions found
```

The real solution  $x = -1$  has not been found. In this case, you can help Maple by first transforming the radicals to algebraic numbers in the **RootOf** representation, solve the equation, convert back to radicals, and check for superfluous ‘solutions’.

```
> eqn := convert(eqn, RootOf);
eqn := x + RootOf(_Z^3 - x, index = 1) = -2
> solve(eqn, x);

solve: Warning: no solutions found
solve: Warning: solutions may have been lost
```

Still no success. The reason is that  $x^{(1/3)}$  is converted to the first complex root, where we want a general cubic root. We define a conversion procedure that transforms indexed **RootOfs** into non-indexed ones and apply it to the equation.

```
> 'convert/noIndexedRootOF' := proc(expr)
>   local rootofs, e;
>   e := convert(expr, RootOf);
>   rootofs := map(r->r=subsop(2=_Z,r), indets(e, RootOf));
>   eval(e, rootofs)
> end proc:
```

```
> eqn := convert(eqn, noIndexedRootOf);
      eqn := x + RootOf(_Z^3 - x, _Z) = -2
```

Now we get results.

```
> solve(eqn, x);
      -1, - $\frac{5}{2}$  +  $\frac{1}{2}I\sqrt{7}$ , - $\frac{5}{2}$  -  $\frac{1}{2}I\sqrt{7}$ 
```

In fact, all complex roots of the polynomial  $(x+2)^3 + x$  are found.

Another example:

```
> restart;
> solve(sin(x)=3*x/Pi, x);
      RootOf(-3_Z + sin(_Z) π)
> evalf(%);
      0.
```

From a plot of the sine function and the function  $x \mapsto 3x/\pi$  it is clear that there exist three solutions, and you know them:  $\pi/6$ ,  $-\pi/6$ , and 0.

But, do not get from the above examples a bad impression of Maple's **solve** capacities. A few gems to repair this view:

```
> (x^6-1)^x=0;  solve(% , x);
      (x^6 - 1)^x = 0
      1,  $\frac{\sqrt{-2 + 2I\sqrt{3}}}{2}, \frac{\sqrt{-2 - 2I\sqrt{3}}}{2}$ 
> x+x^(1/2)+x^(1/3)+x^(1/4)=4;  solve(% , x);
      x +  $\sqrt{x} + x^{(1/3)} + x^{(1/4)} = 4$ 
      1
> x^3*(ln(5)-ln(x))=3;  solve(% , x);
      x^3 (ln(5) - ln(x)) = 3
      5 e^(1/3 LambertW( $\frac{-9}{125}$ )), 5 e^(1/3 LambertW(-1,  $\frac{-9}{125}$ ))
> arccos(3*x)=2*arcsin(x);  solve(% , x);
      arccos(3 x) = 2 arcsin(x)
      - $\frac{3}{4}$  +  $\frac{\sqrt{17}}{4}$ 
> arccos(2*x)=arctan(3*x);  solve(% , x);
      arccos(2 x) = arctan(3 x)
```

```


$$\frac{\sqrt{-2 + 2\sqrt{10}}}{6}$$

> max(x, 2*x-2)=min(x^2-1, 5-x); solve(%, x);
max(x, 2x - 2) = min(x^2 - 1, 5 - x)

$$\frac{1}{2} - \frac{1}{2}\sqrt{5}, \frac{1}{2}\sqrt{5} + \frac{1}{2}, \frac{7}{3}$$

> evalf(%);
[-0.6180339880, 1.618033988, 2.333333333]

```

Figure 16.1 provides a visual check of this result.

```
> plot({max(x, 2*x-2), min(x^2-1, 5-x)}, x=-4..4);
```

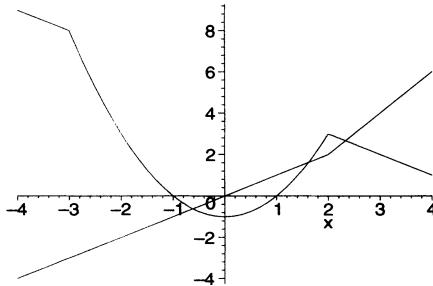


Figure 16.1. Graph of  $\max(x, 2x - 2)$  and  $\min(x^2 - 1, 5 - x)$  on interval  $(-4, 4)$ .

```

> abs(x+abs(x+1))=1; solve(%, x);
|x + |x + 1|| = 1
0, RealRange(-∞, -1)

```

A range of solutions! When using sets, the result will be expressed in set notation as well.

```
> solve({%%}, {x});
{x = 0}, {x ≤ -1}
```

### Too few solutions.

A classical example of getting too few solutions is the following:

```
> solve(sin(x)=1/2, x);
```

$$\frac{1}{6}\pi$$

Maple is satisfied with one solution instead of the many solutions  $\frac{1}{6}\pi + 2k\pi$  and  $\frac{5}{6}\pi + 2k\pi$  for  $k \in \mathbb{Z}$ . These solutions are only obtained when the environment variable `_EnvAllSolutions` has been set `true`.

```
> forget(solve); # forget previous answer
> _EnvAllSolutions := true:
> solve(sin(x)=1/2, x);
```

$$\frac{1}{6}\pi + \frac{2}{3}\pi\_B1 + 2\pi\_Z1$$

with assumptions on  $_B1$  and  $_Z1$

The variable  $_B1$  denotes a binary number (0 or 1) and  $_Z1$  denotes an arbitrary integer. The sentence about assumptions reminds you that these variables are special

```
> map(about, indets(%,name)):
```

**Pi:**  
 is assumed to be: Pi  
 Originally  $_B1$ , renamed  $_B1^~$ :  
 is assumed to be: OrProp(0,1)  
 Originally  $_Z1$ , renamed  $_Z1^~$ :  
 is assumed to be: integer

Two other examples of this kind:

```
> solve(exp(x)=2, x);
```

$$\ln(2) + 2I\pi\_Z2$$

with assumptions on  $_Z2$

```
> solve(exp(-x)=x, x);
```

$$\text{LambertW}(_NN1, 1)$$

with assumptions on  $_NN1$

Here, you get solutions in different branches of the logarithm and Lambert's W function.

Occasionally, you might want to limit the number of solutions sought for. This can be controlled by the variable **\_MaxSols**, which is by default  $\infty$ .

```
> eqn := product(x-k, k=1..101)=0:
> nops({solve(eqn, x)});
```

$$101$$

```
> _MaxSols := 10:
> solve(eqn, x); # only ten solutions at most
```

$$1, 2, 3, 4, 5, 6, 7, 8, 9, 10$$

### Too many solutions.

A trigonometric equation for which you better assist Maple:

```
> _EnvAllSolutions := true:
> eqn := sin(x)^3 - 13/2*sin(x)^2 + 11*sin(x) = 4;
```

$$eqn := \sin(x)^3 - \frac{13}{2}\sin(x)^2 + 11\sin(x) = 4$$

```
> solve(eqn, x);

$$\frac{1}{6}\pi + \frac{2}{3}\pi \cdot B1 + 2\pi \cdot Z1, \arcsin(2) - 2\arcsin(2) \cdot B2 + 2\pi \cdot Z2 + \pi \cdot B2,$$


$$\arcsin(4) - 2\arcsin(4) \cdot B3 + 2\pi \cdot Z3 + \pi \cdot B3$$

with assumptions on _B1, _Z1, _B2, _Z2, _B3 and _Z3
```

All complex solutions are found, but most probably you were only searching for the real solutions. Here, more insight is obtained by considering **eqn** as an equation in **sin(x)**.

```
> solve(eqn, sin(x));

$$\frac{1}{2}, 2, 4$$

```

The three equations  $\sin x = 2$ ,  $\sin x = \frac{1}{2}$ , and  $\sin x = 4$  can now be handled separately.

The next example illustrates that you must be careful with using function calls as names to solve for; Maple does not apply much mathematical knowledge while considering the equation in these names.

```
> solve(cos(2*x)=cos(x), x);

$$2\pi \cdot Z4, \frac{2}{3}\pi - \frac{4}{3}\pi \cdot B4 + 2\pi \cdot Z4$$

with assumptions on _Z4 and _B4
> solve(cos(2*x)=cos(x), cos(x));

$$\cos(2x)$$

```

Be careful,  $\cos 2x$  is not considered in the above equation as a polynomial in  $\cos x$ .

```
> solve(2*cos(x)^2-1=cos(x), cos(x));

$$1, \frac{-1}{2}$$

> seq(solve(cos(x)=s, x), s=%);

$$2\pi \cdot Z8, \frac{2}{3}\pi - \frac{4}{3}\pi \cdot B7 + 2\pi \cdot Z9$$

with assumptions on _Z8, _B7 and _Z9
```

We end this section with two more practical issues. The first one is in the preprocessing of an equation. Before trying to find solutions, Maple first tries to simplify the equation without any scruples.

```
> (x-1)^2/(x^2-1)=0;

$$\frac{(x-1)^2}{x^2-1} = 0$$

```

```
> solve(%);
```

1

Maple can check itself that its solution is not valid when  $\frac{(x-1)^2}{x^2-1}$  is considered as a real function.

```
> subs(x=1, %%);
```

```
Error, numeric exception: division by zero
```

But of course, considered as elements from the quotient field  $\mathbb{R}(x)$ , the expressions  $\frac{(x-1)^2}{x^2-1}$  and  $\frac{x-1}{x+1}$  are equivalent. For the analytical continuation it is true that  $x = 1$  is a solution.

```
> limit(lhs(%%), x=1);
```

0

Finally, an example of the problem of not being able to specify criteria for parameterization of solutions of systems of equations.

```
> eqns := {w+x+y+z=1, w+y=0, 2*w+z=2, v+z=0}:
> vars := indets(eqns);
```

$vars := \{y, v, z, x, w\}$

```
> solve(eqns, vars);
```

$\{y = -w, x = 2w - 1, z = -2w + 2, v = 2w - 2, w = w\}$

When you solve a system of equations with respect to all unknowns, Maple chooses the parameterization of the solutions on the basis of criteria like “select the equation with the least number of terms” or “select the equation with the simplest coefficients,” and so on. But what is there to do if you have different criteria? You can use simplification with respect to side relations to express a variable in terms of the others.

```
> simplify({v,w}, eqns, [v,w,x,y,z]);
```

$$\left\{-\frac{z}{2} + 1, -z\right\}$$

```
> simplify({v,w}, eqns, [v,w,z,y,x]);
```

$$\left\{x - 1, \frac{x}{2} + \frac{1}{2}\right\}$$

But this does not mean that you have solved the original system of equations. This method only works when you know beforehand which variables can be used as parameters. Then you can also leave the parameters out of the set of unknowns.

```
> solve(eqns, {v,w,y,z});
```

$$\left\{y = -\frac{x}{2} - \frac{1}{2}, v = x - 1, w = \frac{x}{2} + \frac{1}{2}, z = 1 - x\right\}$$

Alternatively you can use the procedure **solvefor**.

```
> solvefor[v,w,y,z](eqns);
```

$$\{y = -\frac{x}{2} - \frac{1}{2}, v = x - 1, w = \frac{x}{2} + \frac{1}{2}, z = 1 - x\}$$

## 16.4 Systems of Equations

The last example of the previous section shows that **solve** can be used for solving systems of equations. For **solve**, the equations and unknowns should be presented as sets. Even with a system of equations in one unknown this variable must be presented as a set. In this section, we shall look at some practical examples of how to solve systems of equations. But all examples involve only few equations; in case you want to solve large systems of linear equations linear algebra methods are a far better approach.

### Linear equations

First we shall look at a system of equations that describes the relations between voltages, currents, and resistors in the electronic circuit of resistors shown in Figure 16.2 (taken from [81]).

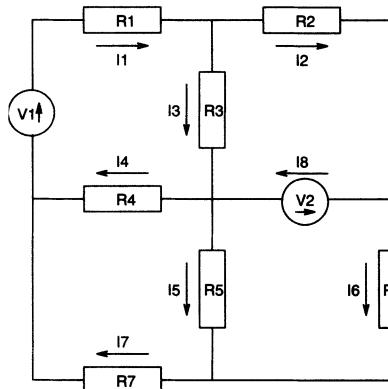


Figure 16.2. Electronic circuit of resistors.

Since our example is from electrical engineering, it is convenient to follow the practice of  $j$  as notation for  $\sqrt{-1}$  instead of  $i$ .

```
> interface(imaginaryunit=j):
```

Applying Kirchhoff's laws you get the following system of equations.

```

> eqns := {R[1]*I[1]+R[3]*I[3]+R[4]*I[4]-V[1]=0,
> R[2]*I[2]-V[2]-R[3]*I[3]=0, R[5]*I[5]-R[6]*I[6]-V[2]=0,
> R[5]*I[5]+R[7]*I[7]-R[4]*I[4]=0, I[1]-I[2]-I[3]=0,
> I[2]-I[6]-I[8]=0, I[5]+I[6]-I[7]=0, I[4]+I[7]-I[1]=0,
> I[3]+I[8]-I[4]-I[5]=0};

```

$eqns := \{R_1 I_1 + R_3 I_3 + R_4 I_4 - V_1 = 0, R_2 I_2 - V_2 - R_3 I_3 = 0,$   
 $R_5 I_5 - R_6 I_6 - V_2 = 0, R_5 I_5 + R_7 I_7 - R_4 I_4 = 0, I_1 - I_2 - I_3 = 0,$   
 $I_2 - I_6 - I_8 = 0, I_5 + I_6 - I_7 = 0, I_4 + I_7 - I_1 = 0, I_3 + I_8 - I_4 - I_5 = 0\}$

Consider it as a linear system of equations in which the resistors and voltages are parameters and the currents actually are the unknowns.

```

> currents := {seq(I[i], i=1..8)};
> resistors := {seq(R[i], i=1..7)};
> voltages := {V[1], V[2]};
> sol := solve(eqns, currents);

```

The solution is not shown. Instead we shall look for a simple formula of current  $I_5$ . But first we **assign** the expressions of the currents in the solution, because this is not done automatically by **solve**, and print the formula for  $I_5$  obtained with **solve**.

```
> assign(sol); I[5];
```

$$(R_3 R_4 R_1 V_2 + R_3 R_1 R_7 V_2 + R_3 R_4 R_7 V_2 + R_2 R_4 R_1 V_2 + R_2 R_4 R_3 V_2 \\ + R_2 R_1 R_7 V_2 + R_2 R_3 R_7 V_2 + R_2 R_4 R_7 V_2 + R_4 V_2 R_3 R_6 + R_3 R_4 R_6 V_1 \\ + R_4 R_6 R_2 V_1) / (R_3 R_4 R_1 R_5 + R_3 R_4 R_1 R_6 + R_3 R_1 R_7 R_6 \\ + R_3 R_1 R_5 R_6 + R_3 R_1 R_7 R_5 + R_3 R_4 R_7 R_6 + R_3 R_4 R_5 R_6 + R_3 R_4 R_7 R_5 \\ + R_2 R_4 R_1 R_5 + R_2 R_4 R_1 R_6 + R_2 R_4 R_3 R_6 + R_2 R_1 R_7 R_6 + R_2 R_1 R_5 R_6 \\ + R_2 R_1 R_7 R_5 + R_2 R_3 R_7 R_6 + R_2 R_3 R_5 R_6 + R_2 R_3 R_7 R_5 + R_2 R_4 R_7 R_6 \\ + R_2 R_4 R_5 R_6 + R_2 R_4 R_7 R_5 + R_3 R_4 R_5 R_2)$$

To simplify this formula we introduce short names for subexpressions and simplify  $I_5$  in terms of the new variables.

```

> relations := {
>   A = R[1]*R[2]*R[4] + R[1]*R[3]*R[4] + R[2]*R[3]*R[4],
>   B = R[5]*R[6] + R[5]*R[7] + R[6]*R[7], C = R[1]*R[2]
>     + R[1]*R[3] + R[2]*R[3] + R[2]*R[4] + R[3]*R[4],
>   D = R[4]*R[6]};
> I[5] := map(simplify, I[5], relations,
> [op(resistors),A,B,C,D]);

```

$$I_5 := \frac{V_2 A + R_7 V_2 C + V_1 R_2 D + (V_2 + V_1) R_3 D}{R_5 A + B C + R_6 A}$$

```
> I[5] := map(collect, I[5], [V1,V2,D]);
```

$$I_5 := \frac{(V_1 R_2 + (V_2 + V_1) R_3) D + R_7 V_2 C + V_2 A}{R_5 A + B C + R_6 A}$$

As a second example we consider a system of equations describing the *pseudo steady state* of an enzyme-catalyzed reaction. We shall apply the Michaelis-Menten theory to the dimeric enzymatic reaction shown in Figure 16.3.

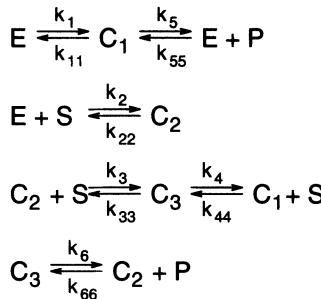


Figure 16.3. Dimeric enzymatic reaction.

Here,  $S, P, E, C_1, C_2$ , and  $C_3$  are the concentrations of the substrate, product, free enzyme, and the three enzyme substrate complexes, respectively. The deterministic mathematical model that describes the kinetics of this reaction is the following system of differential equations.

$$\begin{aligned}
 S' &= k_{22}C_2 + (k_{33} + k_4)C_3 - (k_2E + k_{44}C_1 + k_3C_2)S \\
 P' &= k_5C_1 + k_6C_3 - (k_{55}E + k_{66}C_2)P \\
 E' &= (k_{11} + k_5)C_1 + k_{22}C_2 - (k_1 + k_2S + k_{55}P)E \\
 C'_1 &= (k_1 + k_{55}P)E + k_4C_3 - (k_{11} + k_5 + k_{44}S)C_1 \\
 C'_2 &= k_2ES + (k_6 + k_{33})C_3 - (k_{22} + k_3S + k_{66}P)C_2 \\
 C'_3 &= (k_{66}P + k_3S)C_2 + k_{44}C_1S - (k_{33} + k_4 + k_6)C_3
 \end{aligned}$$

Let us concentrate on the first of these equations. For convenience, we introduce with the **alias** and **macro** facility some shortcuts for various names.

```

> alias(S=S(t), P=P(t)): # s, p as functions of time t
> sequence := seq( k||i = k[i], i=1..6 ),
>      seq( k||i||i = k[i,i], i=1..6 ),
>      seq( C||i = C[i], i=1..6 ):
> eval('macro(s)', s=sequence):
> RS := k22*C2 + (k33+k4)*C3 - (k2*E+k3*C2+k44*C1)*S;

```

$$RS := k_{2,2}C_2 + (k_{3,3} + k_4)C_3 - (k_2E + k_3C_2 + k_{4,4}C_1)S$$

In the pseudo steady state, it is assumed that the concentrations of the free enzyme and the complexes change very slowly, so that we may even assume them to be constant in time. Thus we get the following system of equations (linear in  $C_1, C_2, C_3$ , and  $E$ ).

```
> sys := {0 = (k11+k5)*C1 + k22*C2 - (k1+k2*S+k55*P)*E,
>         0 = (k1+k55*P)*E + k4*C3 - (k11+k5+k44*S)*C1,
>         0 = k2*E*S + (k6+k33)*C3 - (k22+k3*S+k66*P)*C2,
>         0 = (k66*P+k3*S)*C2 + k44*C1*S - (k33+k4+k6)*C3,
>         E0 = C1+C2+C3+E};
```

$$\begin{aligned} sys := \{ & 0 = (k_{1,1} + k_5) C_1 + k_{2,2} C_2 - (k_1 + k_2 S + k_{5,5} P) E, \\ & 0 = (k_1 + k_{5,5} P) E + k_4 C_3 - (k_{1,1} + k_5 + k_{4,4} S) C_1, \\ & 0 = k_2 E S + (k_6 + k_{3,3}) C_3 - (k_{2,2} + k_3 S + k_{6,6} P) C_2, \\ & 0 = (k_{6,6} P + k_3 S) C_2 + k_{4,4} C_1 S - (k_{3,3} + k_4 + k_6) C_3, \\ & E0 = C_1 + C_2 + C_3 + E \} \end{aligned}$$

Maple can solve this system of equations so that the reaction rate  $S'$  can be expressed in terms of  $S$ ,  $P$ ,  $E_0$ , and the rate constants. We do not show the large solution. Instead we shall show the expression when  $k_{55} = 0$ ,  $k_{66} = 0$ .

```
> solve(sys, {C1,C2,C3,E}):  
> assign(%):  
> k55 := 0: k66 := 0:
```

The differential equation for substrate concentration becomes

```
> diff(S,t) = collect(normal(RS), S, factor);
```

$$\begin{aligned} \frac{\partial}{\partial t} S = & -S E0 (k_3 S^2 k_6 k_2 k_{4,4} \\ & + k_3 (k_2 k_4 k_{1,1} + k_1 k_6 k_{4,4} + k_2 k_6 k_{1,1} + k_2 k_4 k_5 + k_2 k_6 k_5) S \\ & - k_{4,4} k_{3,3} k_1 k_{2,2}) / (k_2 k_{4,4} S^3 k_3 + \\ & (k_2 k_{3,3} k_{4,4} + k_{4,4} k_1 k_3 + k_2 k_5 k_3 + k_2 k_6 k_{4,4} + k_2 k_4 k_3 + k_2 k_{1,1} k_3) S^2 \\ & + (k_4 k_{1,1} k_3 + k_1 k_{3,3} k_{4,4} + k_1 k_6 k_{4,4} + k_4 k_1 k_3 + k_2 k_4 k_{1,1} + k_2 k_6 k_{1,1} \\ & + k_{4,4} k_6 k_{2,2} + k_{4,4} k_{3,3} k_{2,2} + k_4 k_5 k_3 + k_{4,4} k_1 k_{2,2} + k_2 k_{3,3} k_5 \\ & + k_2 k_6 k_5 + k_2 k_4 k_5 + k_2 k_{3,3} k_{1,1}) S \\ & + k_{2,2} (k_{3,3} + k_4 + k_6) (k_1 + k_{1,1} + k_5)) \end{aligned}$$

Maple offers various methods for solving this differential equation with **dsolve**. But more on differential equations in the next chapter.

## Nonlinear equations

Enough about systems of linear equations; time and memory are the only computational limitations to solving them. There are two algorithms implemented, viz., a normal Gaussian elimination with a sparse representation and a “primitive” fraction-free algorithm with a sparse representation in case coefficients are integers, radicals, or polynomials, among others. For large linear systems of equations, the **LinearAlgebra** provides better methods for solving these systems numerically. Let us now concentrate on nonlinear systems of equations. Consider the following equations of a circle

and a parabola

$$x^2 + y^2 = 25, \quad y = x^2 - 5,$$

and solve this system with Maple.

```
> eqns := {x^2+y^2=25, y=x^2-5}:
> vars := {x,y}:
> solve(eqns, vars);
```

$$\{x = 0, y = -5\}, \{x = 0, y = 5\}, \{x = 3, y = 4\}, \{x = -3, y = 4\}$$

These sets of solutions can be checked by back substitution. We show how to do this with the procedures **subs** and **eval**.

```
> map(subs, {}, eqns);
{{25 = 25, -5 = -5}, {25 = 25, 4 = 4}}
> map2(''eval'', eqns, {%%});
{ eval(%1, {x = 0, y = -5}), eval(%1, {x = 3, y = 4}),
  eval(%1, {x = -3, y = 4}) }
%1 := {x^2 + y^2 = 25, y = x^2 - 5}
> %; # extra evaluation to get the results
{{25 = 25, -5 = -5}, {25 = 25, 4 = 4}}
```

Solving the above system of polynomial equations was simple, but very soon solving nonlinear equations gets more complicated. Consider

$$x^2 + y^2 = 1, \quad \sqrt{x+y} = x^2 - y^2,$$

and solve this system of equations with Maple.

```
> eqns := x^2+y^2=1, sqrt(x+y)=x^2-y^2:
> vars := x,y:
> sols := solve({eqns}, {vars});
```

```
sols := {y = RootOf(2_Z^2 - 1), x = -RootOf(2_Z^2 - 1)},
{y = 0, x = 1}, {x = 2 - 4 RootOf(%1, -0.8090169944 + 0.3930756889 I)^3
+ 3 RootOf(%1, -0.8090169944 + 0.3930756889 I)
- 2 RootOf(%1, -0.8090169944 + 0.3930756889 I)^2,
y = RootOf(%1, -0.8090169944 + 0.3930756889 I)},
{x = 2 - 4%2^3 + 3%2 - 2%2^2, y = %2}
%1 := 4_Z^4 + 4_Z^3 - 2_Z^2 - 4_Z - 1
%2 := RootOf(%1, -0.3269928304)
```

Apparently Maple finds four solutions of the given system of equations and represents them as a sequence of sets. Within each set the values of the variables in the particular solution are denoted as equations. This makes substitution of the solution in the original system of equations easy.

```
> simplify(eval({eqns}, sols[2]));
{1 = 1}
> simplify(eval({eqns}, sols[1]));
{1 = 1, 0 = 0}
```

To check a solution, you can also use **testeq**.

```
> map(testeq, eval({eqns}, sols[1]));
{true}
> convert(%, 'and');
true
```

The first solution

```
> sols[1];
{y = RootOf(2_Z^2 - 1), x = -RootOf(2_Z^2 - 1)}
```

is actually a set of two or four solutions. When both **RootOf** expressions correspond with the same complex number, then there are two solutions, which can be obtained by the procedure **allvalues**.

```
> s1 := allvalues(sols[1], 'dependent');
s1 := {y =  $\frac{\sqrt{2}}{2}$ , x =  $-\frac{\sqrt{2}}{2}$ }, {y =  $-\frac{\sqrt{2}}{2}$ , x =  $\frac{\sqrt{2}}{2}$ }
```

The keyword **dependent** in **allvalues** is used to specify that the **RootOfs** in the first solution represent the same value and should not be evaluated independently of one another. This is Maple's default behavior when applying **allvalues**; so it could have been left out of the command.

You can also consider the first solution as a set of four candidate solutions and check them separately.

```
> candidates := allvalues(sols[1], 'independent');
candidates := {y =  $\frac{\sqrt{2}}{2}$ , x =  $-\frac{\sqrt{2}}{2}$ }, {x =  $-\frac{\sqrt{2}}{2}$ , y =  $-\frac{\sqrt{2}}{2}$ },
{y =  $\frac{\sqrt{2}}{2}$ , x =  $\frac{\sqrt{2}}{2}$ }, {y =  $-\frac{\sqrt{2}}{2}$ , x =  $\frac{\sqrt{2}}{2}$ }
> for c in candidates do simplify(subs(c, {eqns})) end do;
{1 = 1, 0 = 0}
{I 2^(1/4) = 0, 1 = 1}
{2^(1/4) = 0, 1 = 1}
{1 = 1, 0 = 0}
```

You see that only two out of four candidate solutions are valid. You can select them as follows.

```
> issol := proc(sol, eqns)
>   convert(map(testeq, subs(sol, eqns)), 'and')
> end proc: # the selection routine
> select(issol, {candidates}, {eqns});
```

$$\{\{y = \frac{\sqrt{2}}{2}, x = -\frac{\sqrt{2}}{2}\}, \{y = -\frac{\sqrt{2}}{2}, x = \frac{\sqrt{2}}{2}\}\}$$

The third and fourth solutions show another interesting phenomenon, viz., the use of **RootOfs** with an extra argument to select a specific root. You can convert them into radical notation.

```
> sols[3];
```

$$\begin{aligned} & \{x = 2 - 4 \%1^3 + 3 \%1 - 2 \%1^2, y = \%1\} \\ & \%1 := \text{RootOf}(4 \cdot Z^4 + 4 \cdot Z^3 - 2 \cdot Z^2 - 4 \cdot Z - 1, \\ & \quad -0.8090169944 + 0.3930756889 I) \end{aligned}$$

```
> convert(%, radical);
```

$$\begin{aligned} & \{y = -\frac{1}{4} - \frac{\sqrt{5}}{4} + \frac{\sqrt{2 - 2\sqrt{5}}}{4}, x = \frac{5}{4} - 4 \left(-\frac{1}{4} - \frac{\sqrt{5}}{4} + \frac{\sqrt{2 - 2\sqrt{5}}}{4}\right)^3 \\\ & \quad - \frac{3\sqrt{5}}{4} + \frac{3\sqrt{2 - 2\sqrt{5}}}{4} - 2 \left(-\frac{1}{4} - \frac{\sqrt{5}}{4} + \frac{\sqrt{2 - 2\sqrt{5}}}{4}\right)^2\} \end{aligned}$$

```
> s3 := simplify(%);
```

$$s3 := \{y = -\frac{1}{4} - \frac{\sqrt{5}}{4} + \frac{1}{4} I \sqrt{-2 + 2\sqrt{5}}, x = -\frac{1}{4} - \frac{\sqrt{5}}{4} - \frac{1}{4} I \sqrt{-2 + 2\sqrt{5}}\}$$

```
> convert(sols[4], radical);
```

$$\begin{aligned} & \{y = -\frac{1}{4} + \frac{\sqrt{5}}{4} - \frac{\sqrt{2 + 2\sqrt{5}}}{4}, x = \frac{5}{4} - 4 \left(-\frac{1}{4} + \frac{\sqrt{5}}{4} - \frac{\sqrt{2 + 2\sqrt{5}}}{4}\right)^3 \\\ & \quad + \frac{3\sqrt{5}}{4} - \frac{3\sqrt{2 + 2\sqrt{5}}}{4} - 2 \left(-\frac{1}{4} + \frac{\sqrt{5}}{4} - \frac{\sqrt{2 + 2\sqrt{5}}}{4}\right)^2\} \end{aligned}$$

```
> s4 := simplify(%);
```

$$s4 := \{y = -\frac{1}{4} + \frac{\sqrt{5}}{4} - \frac{\sqrt{2 + 2\sqrt{5}}}{4}, x = -\frac{1}{4} + \frac{\sqrt{5}}{4} + \frac{\sqrt{2 + 2\sqrt{5}}}{4}\}$$

So, Maple found in fact five solutions

```
> s1, sols[2], s3, s4;
```

$$\begin{aligned} & \{y = \frac{\sqrt{2}}{2}, x = -\frac{\sqrt{2}}{2}\}, \{y = -\frac{\sqrt{2}}{2}, x = \frac{\sqrt{2}}{2}\}, \{y = 0, x = 1\}, \\ & \{y = -\frac{1}{4} - \frac{\sqrt{5}}{4} + \frac{1}{4} I \sqrt{-2 + 2\sqrt{5}}, x = -\frac{1}{4} - \frac{\sqrt{5}}{4} - \frac{1}{4} I \sqrt{-2 + 2\sqrt{5}}\}, \\ & \{y = -\frac{1}{4} + \frac{\sqrt{5}}{4} - \frac{\sqrt{2 + 2\sqrt{5}}}{4}, x = -\frac{1}{4} + \frac{\sqrt{5}}{4} + \frac{\sqrt{2 + 2\sqrt{5}}}{4}\} \end{aligned}$$

But it used by itself **RootOfs** to specify two or more solutions in one expression. You may be somewhat disappointed that Maple in this example did not come up right away with the solutions in radical notation. After all, the polynomials inside the **RootOfs** are of degree less than five and therefore can be expressed in radical notation. Your wish is Maple's command if you set the environment variable **\_EnvExplicit** to **true**.

```
> forget(solve); # forget previous results
> _EnvExplicit := true:
> {solve({eqns})};
```

$$\begin{aligned} & \{x = 1, y = 0\}, \{y = \frac{1}{2} \sqrt{2}, x = -\frac{1}{2} \sqrt{2}\}, \{y = -\frac{1}{2} \sqrt{2}, x = \frac{1}{2} \sqrt{2}\}, \{ \\ & x = \frac{5}{4} - 4(-\frac{1}{4} - \frac{1}{4} \sqrt{5} + \frac{1}{4} \%2)^3 - \frac{3}{4} \sqrt{5} + \frac{3}{4} \%2 - 2(-\frac{1}{4} - \frac{1}{4} \sqrt{5} + \frac{1}{4} \%2)^2, \\ & y = -\frac{1}{4} - \frac{1}{4} \sqrt{5} + \frac{1}{4} \%2\}, \{y = -\frac{1}{4} + \frac{1}{4} \sqrt{5} - \frac{1}{4} \%1, \\ & x = \frac{5}{4} - 4(-\frac{1}{4} + \frac{1}{4} \sqrt{5} - \frac{1}{4} \%1)^3 + \frac{3}{4} \sqrt{5} - \frac{3}{4} \%1 - 2(-\frac{1}{4} + \frac{1}{4} \sqrt{5} - \frac{1}{4} \%1)^2 \\ & \} \\ \%1 := \sqrt{2 + 2\sqrt{5}} \\ \%2 := \sqrt{2 - 2\sqrt{5}} \\ > nops(%); \end{aligned}$$

5

By taking squares, the original system of equations can be changed into a system of polynomial equations. The newly obtained system can be solved with traditional methods for polynomial equations such as the elimination method. First, you take squares and write the system of equations as polynomials.

```
> restart: _EnvExplicit:= true:
> eqns := x^2+y^2=1, sqrt(x+y)=x^2-y^2;
eqns := x^2 + y^2 = 1,  $\sqrt{x + y} = x^2 - y^2$ 
> P[1] := (lhs-rhs)(eqns[1]);
P1 := x^2 + y^2 - 1
> P[2] := (lhs^2-rhs^2)(eqns[2]);
```

```


$$P_2 := x + y - (x^2 - y^2)^2$$

> Peqns := {P[1], P[2]};


$$Peqns := \{x^2 + y^2 - 1, x + y - (x^2 - y^2)^2\}$$


```

Next, you eliminate  $y$  by computing the **resultant** with respect to this variable. For the definition of resultant, check your favorite algebra textbook or [69, 98, 165, 168, 176].

```

> eqn_x := resultant(P[1], P[2], y) = 0;
eqn_x := -6x^2 - 2x - 8x^5 + 8x^3 + 24x^4 + 16x^8 - 32x^6 = 0
> eqn_x := factor(eqn_x);
eqn_x := 2x(x - 1)(2x^2 - 1)(4x^4 + 4x^3 - 2x^2 - 4x - 1) = 0

```

This polynomial equation in  $x$  can easily be solved with Maple.

```

> x_roots := solve(eqn_x, x);


$$x\_roots := 0, 1, \frac{\sqrt{2}}{2}, -\frac{\sqrt{2}}{2}, -\frac{1}{4} + \frac{\sqrt{5}}{4} + \frac{\sqrt{2+2\sqrt{5}}}{4},$$


$$-\frac{1}{4} + \frac{\sqrt{5}}{4} - \frac{\sqrt{2+2\sqrt{5}}}{4}, -\frac{1}{4} - \frac{\sqrt{5}}{4} + \frac{\sqrt{2-2\sqrt{5}}}{4}$$


$$-\frac{1}{4} - \frac{\sqrt{5}}{4} - \frac{\sqrt{2-2\sqrt{5}}}{4}$$


```

Simplification with respect to side relations can express  $y$  in terms of  $x$ .

```

> simplify(P[2], {P[1]}, [y, x]);

$$x + y - 4x^4 + 4x^2 - 1$$

> eqn_y := y = solve(% ,y);
eqn_y := y = -x + 1 + 4x^4 - 4x^2

```

The new system of equations  $\{eqn_x, eqn_y\}$  has more solutions than the original system. ‘Ghost solutions’ have been introduced by taking squares.

The elimination process could more easily have been done with the library function **eliminate**.

```

> eliminate({P[1], P[2]}, y);


$$[\{y = -x + 1 + 4x^4 - 4x^2\},$$


$$\{x(x - 1)(2x^2 - 1)(4x^4 + 4x^3 - 2x^2 - 4x - 1)\}]$$


```

This routine makes use of pseudo resultants and pseudo subresultants to do the work (q.v., [6, 69, 98, 177]). From the result of **eliminate** you can easily find the solutions.

Let us redo our original example with three variables. First the system of equations is transformed into a system of polynomial equations for which

the solution set contains the solutions of the original system of equations. To this end an extra variable, say  $z$ , is introduced; it describes the square root.

```
> eqns := {x^2+y^2=1, z=x^2-y^2, z^2=x+y};  
eqns := { $x^2 + y^2 = 1$ ,  $z = x^2 - y^2$ ,  $z^2 = x + y$ }
```

Next consider the set of polynomials that defines the system of equations.

```
> polys := map(lhs-rhs, eqns);  
polys := { $x^2 + y^2 - 1$ ,  $z^2 - x - y$ ,  $z - x^2 + y^2$ }
```

Now eliminate  $y$  and  $z$ :

```
> eliminate(polys, {y,z});  
[{ $y = -x + 1 + 4x^4 - 4x^2$ ,  
 $z = 8x^2 + 2x + 8x^5 - 8x^3 - 1 - 24x^4 - 16x^8 + 32x^6$ ,  
 $x(x-1)(2x^2-1)(4x^4+4x^3-2x^2-4x-1)$ }]
```

Basically, we have the same result as was found previously.

In general, analytical solutions of systems of polynomial equations are not always so easily found by the elimination method. A more sophisticated method for solving polynomial equations will be described in the next section.

## 16.5 The Gröbner Basis Method

The Gröbner basis is an important mathematical notion in the computational theory of polynomials. In §14.7 the Gröbner basis method was applied to simplification with respect to side relations. In this section, its role in the process of finding solutions of systems of polynomial equations will be discussed briefly. More details, e.g., how to test whether a finite set of solutions exists, how to work over rings instead of fields, can be found in [3, 16, 37, 66, 85, 91, 98, 156] and in overview articles [5, 35, 36, 61, 105, 127, 175].

Let us see how the last example of the previous section is treated in the Gröbner basis method.

```
> polys := {x^2+y^2-1, x^2-y^2-z, x+y-z^2};  
polys := { $x^2 - y^2 - z$ ,  $x + y - z^2$ ,  $x^2 + y^2 - 1$ }
```

The minimal Gröbner basis with respect to the pure lexicographical ordering of the unknowns  $x$ ,  $y$ , and  $z$ , induced by  $z \succ y \succ x$  is

```
> with(Groebner): # load the Groebner basis package  
> G := gbasis(polys, plex(z,y,x));
```

```

G := [-3 x2 - 4 x5 + 8 x8 - 16 x6 + 12 x4 + 4 x3 - x,
      x + y - 4 x4 + 4 x2 - 1, z - 2 x2 + 1]
> G := factor(G);

G := [x (x - 1) (2 x2 - 1) (4 x4 + 4 x3 - 2 x2 - 4 x - 1),
      x + y - 4 x4 + 4 x2 - 1, z - 2 x2 + 1]

```

The set of common zeros of the Gröbner basis is equivalent to the set of common zeros of the original set of polynomials. However, the Gröbner basis has in this case achieved a complete separation of the variables  $z$  and  $y$  as polynomials in  $x$  and a remaining univariate polynomial in  $x$ . The first two polynomials can be trivially solved for  $y$  and  $z$ .

$$z = 2x^2 - 1, \quad y = 4x^4 - 4x^2 - x + 1$$

The roots of the third polynomial

$$x(x - 1)(2x^2 - 1)(4x^4 + 4x^3 - 2x^2 - 4x - 1)$$

can be found in analytical form. Each root can be plugged into the equations for  $y$  and  $z$ .

```

> _EnvExplicit := true:
> assign({solve(G[3], {z}), solve(G[2], {y}))}):
> rootlist := [solve(G[1])];

```

$$\begin{aligned} \text{rootlist} := & [0, 1, \frac{\sqrt{2}}{2}, -\frac{\sqrt{2}}{2}, -\frac{1}{4} + \frac{\sqrt{5}}{4} + \frac{\sqrt{2+2\sqrt{5}}}{4}, \\ & -\frac{1}{4} + \frac{\sqrt{5}}{4} - \frac{\sqrt{2+2\sqrt{5}}}{4}, -\frac{1}{4} - \frac{\sqrt{5}}{4} + \frac{\sqrt{2-2\sqrt{5}}}{4}, \\ & -\frac{1}{4} - \frac{\sqrt{5}}{4} - \frac{\sqrt{2-2\sqrt{5}}}{4}] \end{aligned}$$

```

> for x in rootlist do
>   simplify( ['x'=x, 'y'=y, 'z'=z] );
>   if simplify(subs(% , polys)) = {0}
>     then print("valid solution")
>     else print("INVALID solution")
>   end if
> end do;

```

$$[x = 0, y = 1, z = -1]$$

“valid solution”

$$[x = 1, y = 0, z = 1]$$

“valid solution”

$$[x = \frac{\sqrt{2}}{2}, y = -\frac{\sqrt{2}}{2}, z = 0]$$

“valid solution”

$$[x = -\frac{\sqrt{2}}{2}, y = \frac{\sqrt{2}}{2}, z = 0]$$

“valid solution”

$$\begin{aligned} [x &= -\frac{1}{4} + \frac{\sqrt{5}}{4} + \frac{\sqrt{2+2\sqrt{5}}}{4}, y = -\frac{1}{4} + \frac{\sqrt{5}}{4} - \frac{\sqrt{2+2\sqrt{5}}}{4}, \\ z &= -\frac{\sqrt{2+2\sqrt{5}}}{4} + \frac{\sqrt{5}\sqrt{2+2\sqrt{5}}}{4}] \end{aligned}$$

“valid solution”

$$\begin{aligned} [x &= -\frac{1}{4} + \frac{\sqrt{5}}{4} - \frac{\sqrt{2+2\sqrt{5}}}{4}, y = -\frac{1}{4} + \frac{\sqrt{5}}{4} + \frac{\sqrt{2+2\sqrt{5}}}{4}, \\ z &= \frac{\sqrt{2+2\sqrt{5}}}{4} - \frac{\sqrt{5}\sqrt{2+2\sqrt{5}}}{4}] \end{aligned}$$

“valid solution”

$$\begin{aligned} [x &= -\frac{1}{4} - \frac{\sqrt{5}}{4} + \frac{1}{4}I\sqrt{-2+2\sqrt{5}}, y = -\frac{1}{4} - \frac{\sqrt{5}}{4} - \frac{1}{4}I\sqrt{-2+2\sqrt{5}}, \\ z &= \frac{-1}{4}I\sqrt{-2+2\sqrt{5}}(1+\sqrt{5})] \end{aligned}$$

“valid solution”

$$\begin{aligned} [x &= -\frac{1}{4} - \frac{\sqrt{5}}{4} - \frac{1}{4}I\sqrt{-2+2\sqrt{5}}, y = -\frac{1}{4} - \frac{\sqrt{5}}{4} + \frac{1}{4}I\sqrt{-2+2\sqrt{5}}, \\ z &= \frac{1}{4}I\sqrt{-2+2\sqrt{5}}(1+\sqrt{5})] \end{aligned}$$

“valid solution”

The advantage of the Gröbner basis with respect to the elimination method of the previous section is that more insight into the structure of the problem can be gained by experimenting with the term orderings.

The second example is a system of equations that describes the steady state of an ordinary differential equation from a neural network [186]:

$$\{cx + xy^2 + xz^2 = 1, \quad cy + yx^2 + yz^2 = 1, \quad cz + zx^2 + zy^2 = 1\}.$$

We consider it as a system of equations in the unknowns  $x, y$ , and  $z$ , and with parameter  $c$ . The following solution obtained with **solve** contains algebraic functions.

```
> eqns := [c*x+x*y^2+x*z^2=1,
>          c*y+y*x^2+y*z^2=1, c*z+z*x^2+z*y^2=1];
```

$$\text{eqns} := [cx + xy^2 + xz^2 = 1, cy + yx^2 + yz^2 = 1, cz + zx^2 + zy^2 = 1]$$

When asked about explicit solutions, Maple computes 21 solutions.

```
> _EnvExplicit := true: # explicit solutions
> nops([solve({op(eqns)}, {x,y,z})]);
```

21

Let us see what these solutions look like. We return to the default choice of **RootOf** expressions.

```
> _EnvExplicit := false: # stick to RootOfs
> solve({op(eqns)}, {x,y,z});
```

$$\begin{aligned} & \{z = \text{RootOf}(c_Z - 1 + 2_Z^3), y = \text{RootOf}(c_Z - 1 + 2_Z^3), \\ & x = \frac{1}{c + 2 \text{RootOf}(c_Z - 1 + 2_Z^3)^2}\}, \\ & \{z = \%2, y = -\frac{-1 + 2c\%2 + 2\%2^3}{c}, x = -\frac{c^2}{-1 + 3c\%2 + 2\%2^3}\}, \\ & \{z = -\frac{-1 + 2c\%2 + 2\%2^3}{c}, y = \%2, x = -\frac{c^2}{-1 + 3c\%2 + 2\%2^3}\}, \\ & \{x = \frac{1}{2} \frac{\%2^2 c^2 - 4 + 7c\%2 + 6\%2^3 + c^3}{c^2 \%2 + c\%2^3 - 2c - \%2^2}, z = \%2, \\ & y = -\frac{1}{2} \frac{c^2 + c\%2^2 + \%2}{c\%2 + \%2^3 - 1}\}, \{y = \text{RootOf}(c + _Z^2 + \%1_Z + \%1^2), \\ & x = \frac{c + \%1^2}{\text{RootOf}(c + _Z^2 + \%1_Z + \%1^2)}, z = \%1\} \\ & \%1 := \text{RootOf}(1 + c_Z + Z^3) \\ & \%2 := \text{RootOf}(c^2 + 3c_Z^2 + 2_Z^4 - Z) \end{aligned}$$

This answer does not give you much structural insight. For systems of polynomial equations the procedure **gsolve** in the **Groebner** package is quite useful. **gsolve** requires the equations to be expressed as pure polynomials, which are understood to be equal to 0.

```
> polys := map(lhs-rhs, eqns);
polys := [cx + xy^2 + xz^2 - 1, cy + yx^2 + yz^2 - 1, cz + zx^2 + zy^2 - 1]
> with(Groebner): # load Groebner basis package
> sys := gsolve(polys, [c,x,y,z]);
sys := {[xy^2 + yz^2 - 1, -xy^2 - xz^2 + 1 + x^3 - xy z, c + x^2 - y z],
plex(c, x, y, z), {y - z}},
[[y - z, x - z, cz + 2z^3 - 1], plex(c, x, y, z), {}],
[[y - z, zx^2 + xz^2 - 1, c - xz + z^2], plex(c, x, y, z), {x - z}]}
```

You get a list of new systems of polynomials whose common zeros are the solutions of the original system. The third argument in the new systems of equations mentions the polynomials for which the solutions of the equations should not vanish. For example,  $[[zy^2 + yz^2 - 1, -xy^2 - xz^2 + 1 + x^3 - xyz, c + x^2 - yzx + y + z], \text{plex}(c, x, y, z), \{y - z\}]$  means that for the solutions of this polynomial system the expression  $y - z$  should not vanish, i.e., that  $y$  and  $z$  differ from each other. An empty set means that there are no conditions. Often these new systems of equations obtained with **gsolve** are easier to solve. At least, they give you more insight into how the problem can be split into smaller subproblems.

```
> for s in sys do solve({op(s[1])}, {x,y,z}) end do;


$$\{y = \frac{c + \%2^2}{\text{RootOf}(c + \%2 \cdot Z + \_Z^2 + \%2^2)},$$


$$z = \text{RootOf}(c + \%2 \cdot Z + \_Z^2 + \%2^2), x = \%2\}, \{x = \%1,$$


$$z = \text{RootOf}(c \cdot Z^2 + (2 \%1^3 + c \%1 - 1) \cdot Z + c \%1^2 + c^2),$$


$$y = \frac{c + \%1^2}{\text{RootOf}(c \cdot Z^2 + (2 \%1^3 + c \%1 - 1) \cdot Z + c \%1^2 + c^2)}\}$$


$$\%1 := \text{RootOf}(2 \cdot Z^4 + 3 c \cdot Z^2 - \_Z + c^2)$$


$$\%2 := \text{RootOf}(1 + \_Z^3 + c \cdot Z)$$



$$\{x = \text{RootOf}(2 \cdot Z^3 + c \cdot Z - 1), z = \text{RootOf}(2 \cdot Z^3 + c \cdot Z - 1),$$


$$y = \text{RootOf}(2 \cdot Z^3 + c \cdot Z - 1)\}$$



$$\{z = \%1, x = -\frac{-1 + 2 c \%1 + 2 \%1^3}{c}, y = \%1\}$$


$$\%1 := \text{RootOf}(2 \cdot Z^4 + 3 c \cdot Z^2 - \_Z + c^2)$$

```

Let us compare the **gsolve** approach to computing the Gröbner basis of the original system of polynomials.

```
> G := gbasis(polys, plex(c,x,y,z));
> G := factor(G);
```

$$G := [(y - z)(z y^2 + y z^2 - 1), (x - z)(z x^2 + x z^2 - 1),$$

$$(x - y)(y x^2 + x y^2 - 1), c z + z x^2 + z y^2 - 1, c y + y x^2 + y z^2 - 1,$$

$$c x + x y^2 + x z^2 - 1]$$

In the **gsolve** approach, whenever a polynomial occurs that can be factorized, Maple splits the original problem into subproblems corresponding to the factors found. This leads to a better efficiency, which is one of the key issues in applying the Gröbner basis method.

With the above Gröbner basis, Maple can quickly answer the question whether the system of polynomial equations has a finite set of solutions.

```
> is_finite(G);
false
```

The answer is false when the system is considered as polynomials in the unknowns  $c$ ,  $x$ ,  $y$ , and  $z$ . The answer would be true if we consider the problem in three unknowns  $x$ ,  $y$ ,  $z$ , i.e., if we consider  $c$  as a parameter. Under this assumption we can split the original problem into simpler subproblems via the procedure **gsolve**.

```
> for s in sys do gsolve({op(s[1])}, {x,y,z}) end do;
{[[1 + z^3 + c z, y^2 + y z + z^2 + c, x + y + z], plex(x, y, z), {}], [
[c^2 - z + 3 c z^2 + 2 z^4, c y + 2 c z + 2 z^3 - 1, x - z], plex(x, y, z),
{1 + z^3 + c z}], [[c^2 z^2 + 1 + 2 c z - 2 z^3 + 2 c z^4,
-z c^3 - 2 c^2 z^3 - c^2 + 4 c z^2 - 2 z + 2 y,
-z c^3 - 2 c^2 z^3 - c^2 + 4 c z^2 - 2 z + 2 x], plex(x, y, z),
{1 + z^3 + c z, c^3 - 3 c z + 2 c^2 z^2 - 1 + 2 z^3}]}
{[[c x + 2 x^3 - 1, -x + z, y - x], plex(y, z, x), {}]}
{[[2 y^4 + 3 c y^2 - y + c^2, -1 + c x + 2 c y + 2 y^3, -y + z], plex(z, x, y), {}]}
```

Our third example of a system of polynomial equations attacked by Gröbner basis methods comes from computational geodesy. Here we shall only sketch the problem and its solution by Maple; for a detailed account we refer to [125].

The relationships between the geocentric Cartesian coordinates  $x$ ,  $y$ , and  $z$  of a point P on or near the surface of the earth and the geodetic coordinates  $h$  (height),  $\lambda$  (longitude), and  $\phi$  (latitude) of its Helmert's projection on the geocentric reference ellipsoid are

$$\begin{aligned} x &= (N + h) \cos \phi \cos \lambda, \\ y &= (N + h) \cos \phi \sin \lambda, \\ z &= (N(1 - e^2) + h) \sin \phi, \end{aligned}$$

where the prime vertical radius of curvature  $N$  and the eccentricity  $e$  of the reference ellipsoid are defined by

$$N = \frac{a}{\sqrt{1 - e^2 \sin^2 \phi}}$$

and

$$e = \sqrt{\frac{a^2 - b^2}{a^2}},$$

$a$  and  $b$  being semi-major and semi-minor axes of the reference ellipsoid. With the above equations, the Cartesian coordinates  $x$ ,  $y$ , and  $z$  can be computed directly from the geodetic coordinates  $h$ ,  $\lambda$ , and  $\phi$ . The inverse problem is more difficult: you are asked to solve the above nonlinear system of equations in the unknowns  $h$ ,  $\lambda$ , and  $\phi$  for given  $x$ ,  $y$ , and  $z$ .

When you ask Maple to solve this system of trigonometric equations, it returns an answer for the latitude in which roots of an 8th degree polynomials are involved. But you can do better! First, associate with the system of trigonometric equations a system of polynomial equations by introducing variables for the trigonometric entities and by adding polynomial equations that originate from well-known trigonometric identities. Henceforth, we shall use the following variables:  $cf = \cos \phi$ ,  $sf = \sin \phi$ ,  $tf = \tan \phi$ ,  $cl = \cos \lambda$ , and  $sl = \sin \lambda$ . We shall also use the variable  $S$  to deal with the square root  $S = \sqrt{1 - e^2 sf^2}$  and define  $d = (N + h) cf$ . In this way, we come up with the following system of ten equations:

```
> sys := [ x - (N+h)*cf*cl, y - (N+h)*cf*sl,
>          z - (N*(1-e^2)+h)*sf, cf^2 + sf^2 - 1, cl^2 + sl^2 - 1,
>          tf*cf - sf, N*S - a, S^2 + e^2*sf^2 - 1, (N+h)*cf - d,
>          d^2 - x^2 - y^2 ];
```

$$\begin{aligned} sys := & [x - (N + h) cf cl, y - (N + h) cf sl, z - (N(1 - e^2) + h) sf, \\ & cf^2 + sf^2 - 1, cl^2 + sl^2 - 1, tf cf - sf, NS - a, S^2 + e^2 sf^2 - 1, \\ & (N + h) cf - d, d^2 - x^2 - y^2] \end{aligned}$$

We compute the Gröbner basis with respect to the following pure lexicographical ordering of the variables.

```
N > S > x > y > h > cl > sl > cf > sf > tf
> with(Groebner): # load Groebner basis package
> vars := N,S,x,y,h,cl,sl,cf,sf,tf:
> gsys := gbasis(sys, plex(vars)):
```

The complete Gröbner basis is too big to be presented here. Besides, we are only interested in the univariate polynomial in  $tf$ , which is expected to be the last polynomial in the Gröbner basis.

```
> collect(gsys[1], tf); # get the polynomial in tf
(-d^2 + d^2 e^2) tf^4 + (-2 e^2 z d + 2 d z) tf^3
+ (-d^2 + z^2 e^2 + e^4 a^2 - z^2) tf^2 + 2 z d t f - z^2
> map(convert, %, sqrfree); # rewrite the polynomial
d^2 (-1 + e^2) tf^4 - 2 d z (-1 + e^2) tf^3
+ (a^2 e^4 - d^2 + z^2 e^2 - z^2) tf^2 + 2 d z t f - z^2
> sort(subs(e^2*z^2 = (e^2-1)*z^2 + z^2, %), tf);
```

$$\begin{aligned} & d^2 (-1 + e^2) tf^4 - 2 dz (-1 + e^2) tf^3 \\ & + (a^2 e^4 - d^2 + (-1 + e^2) z^2) tf^2 + 2 dz tf - z^2 \end{aligned}$$

So, we end up with a 4th degree polynomial in  $\tan \phi$ , which can be solved analytically (q.v., [191]).

The same answer can be found by the procedure **univpoly**.

```
> univpoly(tf, sys, {vars});
```

$$\begin{aligned} & tf^2 a^2 e^4 + e^2 tf^4 d^2 - 2 e^2 tf^3 z d + e^2 tf^2 z^2 - tf^4 d^2 \\ & - d^2 tf^2 + 2 z d tf + 2 tf^3 z d - z^2 - tf^2 z^2 \\ > & \text{collect}(\%, [tf, d, z]); \\ & (-1 + e^2) d^2 tf^4 + (-2 e^2 + 2) z d tf^3 \\ & + (-d^2 + (-1 + e^2) z^2 + a^2 e^4) tf^2 + 2 d z tf - z^2 \end{aligned}$$

This procedure uses the total degree ordering of variables to compute a Gröbner basis and uses this to construct the univariate polynomial (in  $tf$ ) of least degree in the ideal generated by the polynomials. In this case, you have limited control over the ordering of the variables used by Maple. As a matter of fact, you can be very unlucky and need much the computing time and computer resources.

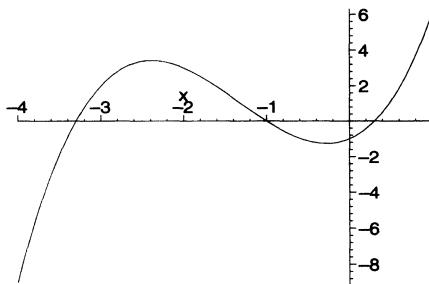
The examples in this section gave you an idea of the strength of the Gröbner basis method as an alternative to the elimination method or Ritt-Wu's characteristic sets method, which is implemented and contributed by Dongming Wang [228] to the Maple share library. One thing to remember: time and memory complexity of the Gröbner basis algorithm can be a serious drawback.

## 16.6 Inequalities

The examples below will give you an impression of Maple's capabilities in solving (systems of) inequalities in one unknown. A necessary condition for success is that the equality points of the inequalities can be properly ordered and that only one variable is involved.

Consider the 3rd degree polynomial function  $x \mapsto x^3 + 4x^2 + 2x - 1$ , the graph of which is shown in Figure 16.4.

```
> plot(x^3+4*x^2+2*x-1, x=-4..1);
```

Figure 16.4. Graph of  $x^3 + 4x^2 + 2x - 1$  on interval  $(-4, 1)$ .

Determine where the function is positive.

```
> solve(x^3+4*x^2+2*x-1>0, x);
```

$$\begin{aligned} &\text{RealRange}\left(\text{Open}\left(-\frac{3}{2} - \frac{\sqrt{13}}{2}, \text{Open}(-1)\right), \right. \\ &\quad \left. \text{RealRange}\left(\text{Open}\left(-\frac{3}{2} + \frac{\sqrt{13}}{2}, \infty\right)\right)\right) \end{aligned}$$

As with one equation in one unknown, the solution contains no variable but is expressed in terms of **RealRange**'s. When you specify the problem as a set of inequalities, you will obtain (as in the case of an equation) solutions containing the unknown.

```
> solve({x^3+4*x^2+2*x-1>0}, {x});
```

$$\left\{-\frac{3}{2} - \frac{\sqrt{13}}{2} < x, x < -1\right\}, \left\{-\frac{3}{2} + \frac{\sqrt{13}}{2} < x\right\}$$

You may omit the braces around the inequality and still get solution sets.

```
> solve(x+ln(x)<0, {x});
```

$$\{0 < x, x < \text{LambertW}(1)\}$$

```
> solve(a*x+b>=c, {x});
```

$$\left\{-\text{signum}(a)x \leq -\frac{\text{signum}(a)(c-b)}{a}\right\}$$

Inequalities and equalites can be used together in **solve**.

```
> solve({x^2-x=0, x<>0}, x); # nonzero solution
```

$$\{x = 1\}$$

```
> _EnvExplicit := true:
```

```
> solve({x^4=9, x^2-3<>0});
```

$$\{x = \sqrt{3}I\}, \{x = -I\sqrt{3}\}$$

## 16.7 Numerical Solvers

For numerical approximations of solutions of equations or systems of equations Maple offers you the procedure **fsolve**. The use of this procedure is similar to the use of **solve**.

```
> x^7 - 2*x^6 - 4*x^5 - x^3 + x^2 + 6*x + 4;
x7 - 2 x6 - 4 x5 - x3 + x2 + 6 x + 4
> fsolve(%);
-1.236067977, 1.167303978, 3.236067977
```

But there are additional options as listed in Table 16.1.

Option	Description
<b>complex</b>	complex-valued root(s)
<i>a..b</i>	search range over the real numbers, from <i>a</i> to <i>b</i>
<i>x = v</i>	set for variable <i>x</i> the initial value <i>v</i>
<b>maxsols=n</b>	maximum number of solutions set to <i>n</i>
<b>avoid</b>	avoid certain values when searching for roots
<b>fulldigits</b>	use floating-point number with <b>Digits</b> precision

Table 16.1. Options of **fsolve**.

Some examples:

```
> fsolve(%%, x, complex);
-1.236067977, -0.7648844336 - 0.3524715460 I,
-0.7648844336 + 0.3524715460 I, 0.1812324445 - 1.083954101 I,
0.1812324445 + 1.083954101 I, 1.167303978, 3.236067977
> fsolve(%%, x, 0..2);
1.167303978
```

In the latter case, Maple was told to try and find only real solutions between 0 and 2.

For polynomial equations, the procedure **fsolve** returns in general (but not always) all real solutions, and with the option **complex** all complex solutions. For equations of other types, **fsolve** is usually satisfied when one solution has been found.

```
> eqn := sin(x)=x/2;
eqn := sin(x) =  $\frac{1}{2} x$ 
> fsolve(eqn, x);
0.
```

A valid solution, but there exist more solutions. We try to find another by avoiding the previous solution  $x = 0$ .

```
> fsolve(eqn, x, avoid={x=0});
                                         -1.895494267
```

Another attempt:

```
> fsolve(eqn, x, avoid={x=0, x=%});
                                         1.895494267
> fsolve(eqn, x, avoid={x=0, x=%%, x=%});
                                         
```

`fsolve(sin(x) =  $\frac{x}{2}$ , x, avoid = {x = 0, x = -1.895494267, x = 1.895494267})`

The signal that Maple has not find a solution anymore. Thus, a general method for finding *all* numerical solutions could go as follows.

```
> sols := {}:
> do
>   newsol := fsolve(eqn, x, avoid=sols):
>   if type(newsol, 'function') then
>     break
>   else
>     sols := {op(sols), x=newsol}
>   end if
> end do:
> sols;
```

$\{x = 0., x = -1.895494267, x = 1.895494267\}$

Alternatively you can specify in the procedure `fsolve` the interval in which you want to search for.

```
> fsolve(eqn, x, 0.1..infinity);
                                         1.895494267
> fsolve(eqn, x, -0.1..0.1);
                                         0.
> fsolve(eqn, x, -infinity..-0.1);
                                         -1.895494267
```

`fsolve` is based on two methods, the (multidimensional) Newton method and, when this fails, the (multidimensional) secant method [214]. You can specify the starting point in these numerical methods.

```
> fsolve(sin(x), x);
                                         0.
> fsolve(sin(x), x=3);
                                         3.141592654
```

The procedure **realroot** uses Descartes' rule of signs (q.v., [176]) to find a list of isolating intervals for all real roots of a univariate polynomial. The width of the interval is optional.

```
> x^7 - 2*x^6 - 4*x^5 - x^3 + x^2 + 6*x + 4;
x7 - 2 x6 - 4 x5 - x3 + x2 + 6 x + 4
> realroot(%);
[[0, 2], [2, 4], [-2, -1]]
> realroot(%%, 1/100);
[[149/128, 75/64], [207/64, 415/128], [-159/128, -79/64]]
```

The procedure **sturm** allows you to compute the number of real roots of a polynomial in an interval. It is based on Sturm's theorem (q.v., [176]).

```
> x^7 - 2*x^6 - 4*x^5 - x^3 + x^2 + 6*x + 4;
x7 - 2 x6 - 4 x5 - x3 + x2 + 6 x + 4
> sturm(% , x, -infinity, infinity); # three real roots
3
> sturm(% , x, 0, infinity); # two positive roots
2
> sturm(%%, x, 2, 4); # one root between 2 and 4
1
```

## 16.8 Other Solvers in Maple

### **isolve**

With **isolve** you look for integer solutions of (systems of) equations. The next example is an application in dimensional analysis.

The drag  $F$  of a fast-moving object in the air is supposed to depend upon its speed  $v$ , diameter  $d$ , the air density  $\rho$ , the velocity of sound  $c$ , and the kinematic viscosity  $\nu$ . All these quantities have dimensions related to the dimensions mass  $m$ , length  $l$ , and time  $t$ . You can determine dimensional groupings of the quantities  $F, v, d, \rho, c$ , and  $\nu$  in the following way.

```
> nondimensional := force^f * speed^v * diameter^d
>      * density^r * acoustic_velocity^c
>      * kinematic_viscosity^n;
```

$$\text{nondimensional} := \text{force}^f \text{ speed}^v \text{ diameter}^d \text{ density}^r \text{ acoustic\_velocity}^c \\ \text{kinematic\_viscosity}^n$$

```

> subs({force = M*L/T^2, speed = L/T, diameter = L,
> density = M/L^3, acoustic_velocity = L/T,
> kinematic_viscosity = L^2/T}, nondimensional);

$$\left(\frac{M L}{T^2}\right)^f \left(\frac{L}{T}\right)^v L^d \left(\frac{M}{L^3}\right)^r \left(\frac{L}{T}\right)^c \left(\frac{L^2}{T}\right)^n$$

> simplify(% , 'symbolic');


$$M^{(f+r)} L^{(f+v+d-3r+c+2n)} T^{(-2f-v-c-n)}$$

> eqns := {seq(op(2,i)=0, i=%)};
```

*eqns* := { $f + r = 0, f + v + d - 3r + c + 2n = 0, -2f - v - c - n = 0$ }

```

> isolve(%); # find all integral solutions

{r = -_Z2, v = -_Z1 + _Z3, c = _Z1,
n = -2_Z2 - _Z3, f = _Z2, d = _Z3}
```

```

> subs(% , nondimensional);

force-Z2 speed(-Z1+Z3) diameter-Z3 density(-Z2) acoustic_velocity-Z1
kinematic_viscosity(-2_Z2-Z3)
```

Some tricks enable you to group powers with the same exponent.

```

> simplify(ln(%), 'symbolic');


$$-_Z2 \ln(\text{force}) - _Z1 \ln(\text{speed}) + _Z3 \ln(\text{speed}) + _Z3 \ln(\text{diameter})$$


$$- _Z2 \ln(\text{density}) + _Z1 \ln(\text{acoustic\_velocity})$$


$$- 2_Z2 \ln(\text{kinematic\_viscosity}) - _Z3 \ln(\text{kinematic\_viscosity})$$

> [coeffs(% , {_Z1,_Z2,_Z3} , 'vars')];

[-ln(speed) + ln(acoustic_velocity),
ln(force) - 2 ln(kinematic_viscosity) - ln(density),
ln(diameter) + ln(speed) - ln(kinematic_viscosity)]
```

```

> combine(% , ln, anything , 'symbolic');


$$[\ln\left(\frac{\text{acoustic\_velocity}}{\text{speed}}\right), \ln\left(\frac{\text{force}}{\text{kinematic\_viscosity}^2 \text{density}}\right),$$


$$\ln\left(\frac{\text{diameter} \text{speed}}{\text{kinematic\_viscosity}}\right)]$$

> zip((a,b)->exp(a)^b, %, [vars]);
```

```

[( $\frac{\text{acoustic\_velocity}}{\text{speed}}$ )-Z1, ( $\frac{\text{force}}{\text{kinematic\_viscosity}^2 \text{density}}$ )-Z2,
 $(\frac{\text{diameter} \text{speed}}{\text{kinematic\_viscosity}})^{-Z3}]$ 
```

```
> convert(%, `*`);
```

$$\left(\frac{\text{acoustic\_velocity}}{\text{speed}}\right)^{-Z1} \left(\frac{\text{force}}{\text{kinematic\_viscosity}^2 \text{density}}\right)^{-Z2}$$

$$\left(\frac{\text{diameter speed}}{\text{kinematic\_viscosity}}\right)^{-Z3}$$

Here, you have obtained a family of dimensionless variables with three integer parameters. Two combinations are well known in aerodynamics and fluid dynamics.

```
> subs(_Z1=-1, _Z2=0, _Z3=0, %); # the Mach number
```

$$\frac{\text{speed}}{\text{acoustic\_velocity}}$$

```
> subs(_Z1=1, _Z2=0, _Z3=1, %% ); # the Reynold's number
```

$$\frac{\text{acoustic\_velocity diameter}}{\text{kinematic\_viscosity}}$$

### **msolve**

Modular arithmetic is also provided for. One example is a cubic analogue of Pell's equation over  $\mathbb{Z}_7$ .

```
> msolve(y^2=x^3-28, 7);
```

$$\{x = 0, y = 0\}, \{y = 6, x = 2\}, \{y = 1, x = 2\}, \{x = 1, y = 1\},$$

$$\{x = 1, y = 6\}, \{x = 4, y = 1\}, \{x = 4, y = 6\}$$

### **rsolve**

Maple can also solve recurrence equations. It uses standard techniques like generating functions and z-transforms, and methods based on substitutions and characteristic equations. The procedure **rsolve** can currently solve linear recurrences with constant coefficients, systems of linear recurrences with constant coefficients, divide and conquer recurrences with constant coefficients, many first order linear recurrences, and some nonlinear first order recurrences. A few examples give you an idea of its power.

- Generalized Fibonacci polynomials

```
> rsolve({f(n+2)=f(n+1)+f(n), f(0)=0, f(1)=1}, f(n)); # Fibonacci numbers
```

$$\frac{\sqrt{5}\left(\frac{1}{2} + \frac{\sqrt{5}}{2}\right)^n}{5} - \frac{\sqrt{5}\left(\frac{1}{2} - \frac{\sqrt{5}}{2}\right)^n}{5}$$

```
> rsolve({f(n+2)=x*f(n+1)+y*f(n), f(0)=0, f(1)=1}, f(n));
```

```


$$\frac{(-x^2 + x \sqrt{x^2 + 4y} - 4y) (-\frac{2y}{x - \sqrt{x^2 + 4y}})^n}{(x^2 + 4y)(x - \sqrt{x^2 + 4y})}$$


$$+ \frac{(-x \sqrt{x^2 + 4y} - x^2 - 4y) (-\frac{2y}{x + \sqrt{x^2 + 4y}})^n}{(x^2 + 4y)(x + \sqrt{x^2 + 4y})}$$

> normal(% , 'expanded');


$$(\frac{2y}{x - \sqrt{x^2 + 4y}})^n - (\frac{2y}{x + \sqrt{x^2 + 4y}})^n$$


$$\sqrt{x^2 + 4y}$$

> # 5th generalized Fibonacci polynomial
> normal(subs(n=5, %), 'expanded');


$$x^4 + 3yx^2 + y^2$$


```

By the way, the conjecture that  $f(n)$  is irreducible if and only if  $n$  is a prime number holds for  $n < 100$ .

- Complexity of Gauss elimination

```

> rsolve({T(n)=T(n-1)+n^2, T(1)=0}, T(n));

$$n - 3(n + 1)(\frac{n}{2} + 1) + 2(n + 1)(\frac{n}{2} + 1)(\frac{n}{3} + 1)$$

> factor(%);


$$\frac{(n - 1)(2n^2 + 5n + 6)}{6}$$


```

- Complexity of merge sort

```

> rsolve({T(n)=2*T(n/2)+n-1, T(1)=0}, T(n));

$$\frac{\ln(2) + \ln(n)n - n\ln(2)}{\ln(2)}$$

> combine(%);


$$\frac{n\ln(\frac{n}{2}) + \ln(2)}{\ln(2)}$$


```

- Complexity of Karatsuba multiplication

```

> rsolve({T(n)=3*T(n/2)+n, T(1)=1}, T(n));

$$3n^{(\frac{\ln(3)}{\ln(2)})} - 2n$$


```

Sometimes, only partial solutions of a summation problem are found.

```
> rsolve(a(n+1)=ln(n+2)*a(n)+1, a(n));
```

$$\left( \prod_{n=0}^{n-1} \ln(n\theta + 2) \right) \left( \left( \sum_{n=0}^{n-1} \frac{1}{\prod_{n=0}^{n-1} \ln(n\theta + 2)} \right) + a(0) \right)$$

Even when **rsolve** cannot find a closed formula, it may still give you information on the asymptotic behavior. An example:

```
> rsolve(u(n+1)=ln(u(n)+1), u(n));
rsolve(u(n + 1) = ln(u(n) + 1), u(n))
> asympt(% , n, 4);

```

$$\begin{aligned} & \frac{2}{n} + \frac{-C + \frac{2}{3} \ln(n)}{n^2} \\ & + \frac{-16 O(1) - \frac{35}{9} - \frac{C}{3} + \frac{C^2}{2} - \left(-\frac{2-C}{3} + \frac{2}{9}\right) \ln(n) + \frac{2}{9} \ln(n)^2}{n^3} \\ & + O\left(\frac{1}{n^4}\right) \end{aligned}$$

Our final example will be the solution of a problem posed by Knuth [149]. Solve the recurrence

$$x_0 = a, \quad x_1 = b, \quad x_{n+2} = x_{n+1} + x_n/(n+1), \quad \text{for } n = 0, 1, 2, \dots$$

both analytically (in terms of familiar functions of  $n$ ) and asymptotically.

```
> infolevel[rsolve]:=1:
> rsolve({x(n+2)=x(n+1)+x(n)/(n+1), x(0)=a, x(1)=b}, x(n));
```

**LREtools/resolveinit:** Warning: no solution found  
with the given initial conditions

$$(n+1)\left(a - \frac{(-b+2a)\Gamma(n-n\theta)}{\Gamma(n+1)} + \frac{(-b+2a)\Gamma(n-n\theta)}{\Gamma(n+2)}\right)$$

We were too optimistic. New tools in the **LREtools** package, which manipulate and solve linear recurrence equations, fail in this example; a strange answer with some new variable  $n\theta$  is returned. So let us try to assist Maple and first compute the  $z$ -transform of the recurrence equation (written in a different but equivalent way).

```
> x(0) := a: x(1) := b:
> expand( subs(ztrans(x(n),n,z)=F(z),
> ztrans((n+1)*x(n+2)=(n+1)*x(n+1)+x(n), n, z)) );

```

$$-z^2 F(z) - z^3 \left(\frac{d}{dz} F(z)\right) + a z^2 = -z^2 \left(\frac{d}{dz} F(z)\right) + F(z)$$

Next we have to solve the differential equation.

```
> dsolve(%, F(z));
```

$$F(z) = \frac{z^2 a}{(z-1)^2} + \frac{z \cdot C1}{(z-1)^2 e^{(\frac{1}{z})}}$$

The last step of computing the inverse  $z$ -transform is the most difficult one.

```
> subs(invztrans(F(z), z, n)=x(n), invztrans(%, z, n));
```

$$x(n) = a n + a + \text{C1} \operatorname{invztrans}\left(\frac{z}{e^{(\frac{1}{z})}(z^2 - 2z + 1)}, z, n\right)$$

```
> simplify(factor(%));
```

$$x(n) = a n + a + \text{C1} \operatorname{invztrans}\left(\frac{z e^{(-\frac{1}{z})}}{(z-1)^2}, z, n\right)$$

```
> f := subs(body=invztrans(exp(-1/z), z, k), k->body);
```

$$f := k \rightarrow \frac{(-1)^k}{k!}$$

```
> g := subs(body=invztrans(z/(z-1)^2, z, k), k->body);
```

$$g := k \rightarrow k$$

So, the inverse  $z$ -transform that remains to be computed can be found as a convolution.

```
> lhs(%%%)=subsop(3=_C1*Sum(f(n-k)*g(k),k=0..n), rhs(%%%));
```

$$x(n) = a n + a + \text{C1} \left( \sum_{k=0}^n \frac{(-1)^{(n-k)} k}{(n-k)!} \right)$$

```
> value(subs(n=1, %));
```

$$b = 2a + \text{C1}$$

```
> solve(%, _C1);
```

$$b - 2a$$

```
> subs(_C1=%, %%);
```

$$x(n) = a n + a + (b - 2a) \left( \sum_{k=0}^n \frac{(-1)^{(n-k)} k}{(n-k)!} \right)$$

Maple can even find a closed formula for the sum.

```
> value(%);
```

$$x(n) = a n + a + \frac{(b - 2a) (-1)^{(n-1)} \operatorname{hypergeom}([2, -n+1], [], 1)}{(n-1)!}$$

We end with the remark that the `genfunc` package contains functions for manipulating rational generating functions. It is a convenient tool when you

want to apply the method of generating functions manually to recurrence relations.

## Pattern Matching

Currently, Maple has limited facilities for pattern matching. The **match** function puts some computational effort into matching an expression with a pattern in one main variable. A simple example: do there exist  $a$  and  $b$  such that  $x^2 + 2x + 3 = (x + a)^2 + b$  for all  $x$ ? If so, what are these variables equal to?

```
> match(x^2+2*x+3=(x+a)^2+b, x, 'sol');
true
```

Maple affirms a true matching. The suitable values for  $a$  and  $b$  are stored in the third variable **sol**.

```
> sol;
{a = 1, b = 2}
```

Please understand the difference between **match**, which does algebraic pattern matching, and **typematch**, which just matches the form of objects.

```
> typematch((x+1)^2+2, (a::anything)&+(b::anything));
true
> a,b;
(x + 1)^2, 2
> typematch((x+1)^2+2,
> (((u::anything)&+(v::anything))^2)&+(w::anything));
true
> typematch(x^2+2*x+3,
> (((p::anything)&+(q::anything))^2)&+(r::anything));
false
```

The procedure **match** resembles the procedure **solve/identity**. This function can be used to find a solution of an equation that is considered as an identity in terms of one unknown. It is automatically called when the procedure **solve** is applied to an identity.

```
> identity(x^2+alpha*x+3 = (x+a)^2+b, x);
identity(x^2 +  $\alpha$  x + 3 = (x + a)^2 + b, x)
> solve(%, {a,b});
{b = 3 -  $\frac{\alpha^2}{4}$ , a =  $\frac{\alpha}{2}$ }
```

The following example shows that one should not have too high expectations.

```
> infolevel := 1;
> solve(identity(cos(t)=sin(omega*t+phi), t), {omega,phi});
{ω = 1, φ = π/2}, {ω = -1, φ = -π/2}
> solve(identity(cos(t)+sin(t)=A*sin(omega*t+phi), t),
> {A,omega,phi});

solve: Warning: no solutions found
solve: Warning: solutions may have been lost
```

## 16.9 Exercises

1. Compute the 6th degree polynomial mapping of which the graph goes through the points  $(-5, -120)$ ,  $(-3, 48)$ ,  $(-2, 36)$ ,  $(1, 120)$ ,  $(4, 2400)$ ,  $(10, 220380)$ , and  $(12, 57408)$ .
2. Check whether

$$3x^2 + 3y^2 + 6xy + 6x + 6y + 2$$

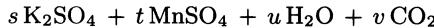
can be written in the form

$$a(x + by + c)^n + d$$

for suitable values of  $a$ ,  $b$ ,  $c$ ,  $d$ , and  $n$ .

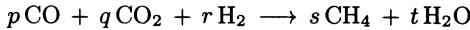
3. Consider the stoichiometry of the following two chemical reactions.

- (a) For what values of  $p, q, r, s, t, u$ , and  $v$  is the reaction equation



balanced?

- (b) For what values of  $p, q, r, s$ , and  $t$  is the reaction equation



balanced?

4. Solve the following equation in  $x$  by **solve** and **fsolve**.

$$48x^5 + 8x^4 - 6x^3 + 114x^2 - 37x + 18 = 0$$

5. Solve the following equations in  $x$ .

- (a)  $(x + 1)^{(x+a)} = (x + 1)^2$
- (b)  $x + (1 + x)^{(1/2)} + (2 + x)^{(1/3)} + (3 + x)^{(1/4)} = 5$
- (c)  $2 \arctan x = \arctan(\frac{2x}{1 - x^2})$
- (d)  $2 - \sin(1 - x) = 2x$

6. The Cartesian coordinates  $(x, y, z)$  can easily be expressed in spherical coordinates  $(r, \theta, \phi)$ :

$$\begin{aligned} x &= r \cos \theta \sin \phi \\ y &= r \sin \theta \sin \phi \\ z &= r \cos \phi \end{aligned}$$

Express the spherical coordinates in the Cartesian ones.

7. Solve the system  $\{x^2 + y^2 = 5, xy = y^2 - 2\}$  with **solve** and **fsolve**.  
 8. Solve the following system of polynomial equations in the unknowns  $x, y$ , and  $z$  over  $\mathbb{R}$  (here,  $a$  is a real constant).

$$\begin{cases} z^2 - x^2 - y^2 + 2ax + 2az - a^2 = 0, \\ yz - ay - ax + a^2 = 0, \\ -2a + x + y = 0 \end{cases}$$

9. Let  $f$  be the homogeneous polynomial  $x_0^3 + x_1^3 + x_2^3 + x_3^3$ . It defines the Fermat surface in the projective space  $\mathbb{P}^3$  as

$$\{x_0:x_1:x_2:x_3 \in \mathbb{P}^3 \mid f(x_0, x_1, x_2, x_3) = 0\}.$$

- (a) There are 27 lines on the surface; use Maple to determine these lines.  
 Hint: the line through the two points  $x_0:x_1:x_2:x_3$  and  $y_0:y_1:y_2:y_3$  is described in so-called Plücker coordinates by

$$p^{ij} := x_i y_j - x_j y_i, \quad \text{with } i, j = 0, 1, 2, 3 \text{ and } i \neq j.$$

In this coordinate system, a line is on the surface iff

$$f(p^{01}u_1 + p^{02}u_2 + p^{03}u_3, p^{10}u_0 + p^{12}u_2 + p^{13}u_3,$$

$$p^{20}u_0 + p^{21}u_1 + p^{23}u_3, p^{30}u_0 + p^{31}u_1 + p^{32}u_2) = 0$$

for all  $u_0, u_1, u_2, u_3$  in  $\mathbb{R}$ , and  $p^{01}p^{23} + p^{02}p^{31} + p^{03}p^{12} = 0$ .

- (b) Determine the singular points on the Fermat surface. Recall that a point is singular when the function values and all partial derivatives in this point are equal to zero.

10. Solve the following inequalities in  $x$ .
- $|x - 3| \cdot |3 - x| > |x|$
  - $|x^3 - x^2 - x - 1| > \frac{1}{|x^2 - 1|}$
11. Solve the recurrence equation  $a_{n+1} = (8/5)a_n - a_{n-1}$ ,  $a_0 = 0$ ,  $a_1 = 1$ .
12. Solve the recurrence equation  $a_{n+1} = 3n a_n - 2n(n-1)a_{n-1}$ ,  $a_1 = 5$ ,  $a_2 = 54$ .

# 17

## Differential Equations

Maple can solve many ordinary differential equations analytically in explicit and implicit form. Traditional techniques such as the method of Laplace transformations, integrating factors, etc., as well as more advanced techniques like the method of Lie symmetries are available through the ordinary differential equation solver **dsolve**. Lie symmetries enable to generate new solutions from a particular solution of a differential equation; they form a systematic way of solving differential equations, in particular nonlinear differential equations. The procedure **pdsolve** provides classical methods to solve partial differential equations such as the method of characteristics and techniques for uncoupling systems of equations. Lie symmetry methods are also available for partial differential equations, viz., in the **liesymm** package. Approximate methods such as Taylor series and power series methods have been implemented for ordinary differential equations. And if all fails, one can still use the numerical solver based on the Runge-Kutta method or other numerical methods. The **DEtools** package contains procedures for graphical presentation of solutions of differential equations, and utilities such as change of variables (dependent as well as independent variables) or computation of Lie symmetries of ordinary differential equations. The **PDEtools** package play a similar role for partial differential equations. Moreover, Maple provides all the tools to apply perturbation methods, like the Poincaré-Lindstedt method and the method of multiple scales up to high order. In this chapter, we shall discuss the tools available in Maple for studying differential equations. Many examples come from applied mathematics.

## 17.1 First Glance at ODEs

Recall that an *ordinary differential equation* (abbreviated ODE) is an equation of the form

$$\omega(x, y, y', y'', \dots, y^{(n)}) = 0,$$

which holds on a particular interval, where  $y', y'', \dots, y^{(n)}$  are short notations for the derivatives of  $y(x)$ , and where  $\omega$  is a real function defined over (a subdomain of)  $\mathbb{R}^{n+2}$ . The ODE is of *order n*, when the function  $\omega$  does depend on the  $(n+2)^{\text{th}}$  argument. When the function  $\omega$  is linear in its last  $n+1$  arguments, the ODE is called *linear*; it is of the form

$$a_n(x)y^{(n)} + a_{n-1}(x)y^{(n-1)} + \dots + a_1(x)y' + a_0(x)y + a(x) = 0.$$

When  $\omega$  is a polynomial mapping, then the *degree* of the ODE of order  $n$  is defined as the exponent of the  $(n+2)^{\text{th}}$  argument in  $\omega$ . In other words, the order of an ODE is equal to  $n$  if the  $n^{\text{th}}$  derivative is the highest derivative that occurs in the ODE, and the degree is the exponent of this highest derivative. Some examples:

$$\begin{aligned} y'' - x^2y - x^3 &= 0 && \text{a linear ODE of order 2 and degree 1,} \\ y'' - y^3 &= 0 && \text{a nonlinear ODE of order 2 and degree 1,} \\ (y'')^3 - y &= 0 && \text{a nonlinear ODE of order 2 and degree 3.} \end{aligned}$$

Mathematicians have developed a whole range of methods for solving differential equations. The interested reader is referred to [171] for a survey of the use of computer algebra in solving ordinary differential equations. Some of these methods have been implemented in Maple. Once again all methods are provided by one procedure, viz., **dsolve** (differential equation solver). You can leave it up to the Maple system to chose a method for solving a differential equation or you can select one yourself. For example, when you want to apply the method of Laplace transforms or find a Taylor series solution, then you only have to say so.

The **dsolve** procedure uses the following approach

- It first checks if a numeric solution is requested via the option `type=numeric`. If so, Maple will apply such a method.
- If the option `type=series` is encountered, then a Taylor series method is used.
- When the input ODE is a homogeneous linear ODE with polynomial coefficients, and the optional arguments `type=formal_series` and `coeffs=coeff_type` are given, **dsolve** will return a set of formal power series solutions with the specified coefficients at all candidate points of expansion.

- Hereafter, it is checked in the **dsolve** procedure whether a method of integral transforms is asked for via the option **method=integral transform**. Recognized integral transforms are Laplace, Fourier, Fourier-Sine, and Fourier-Cosine.
- **dsolve** also allows a solution set of linear ODEs in the form of a list of functions that span the solution space. If the differential equation is nonhomogeneous, **dsolve** returns a list containing the list of basis functions followed by a single particular solution.
- After the previous tests, Maple knows that an analytic solution is requested. It follows the following strategy of solving for a single ODE:
  1. Try to solve the ODE using classification methods, i.e., see if the ODE matches a recognizable pattern, for which a solving method is implemented. For example, check if the ODE is 1st order linear, separable, of Bernoulli type, or Riccati type, and so on. If a pattern matches, use the method that is implemented for the recognized case.
  2. If classification methods fail, then apply Lie symmetry methods. **dsolve** first looks for the generators of symmetry groups of the given ODE, and then uses this information to integrate it, or at least reduce its order.

Of course you can change the above strategy: by the keywords **class** and **Lie**, you request that only classification methods or only Lie symmetry methods are applied, respectively.

- Systems of ODEs and initial or boundary value problems are treated separately.

Maple's capabilities in solving ODEs are impressive: about 97% of all examples of [147] are solved by simply calling the procedure **dsolve** without any extra arguments. In addition, Maple provides valuable facilities for solving ordinary differential equations in such a way that the user on the one hand does not need to know all details of the mathematical methods used and on the other hand still has much control on the choice of the solution strategy. In the next sections we shall have a look at the facilities through many examples from applied mathematics. For the interested reader, Maple provides much information about its methods and gives references to papers about implemented algorithms in its help system: enter **?dsolve,algorithms** or **?dsolve,references**. The most recent version of the Maple tools for solving differential equations is maintained by E.S. Cheb-Terrab at the website <http://lie.uwaterloo.ca/odetools.htm>.

## 17.2 Analytic Solutions

Consider the differential equation

$$xy' = y \ln(xy) - y,$$

of order 1 and degree 1. If you have no strong preference for a particular method of solving the ODE, then you call **dsolve** without all the trimmings.

```
> ODE := x*diff(y(x),x) = y(x)*ln(x*y(x)) - y(x);
ODE := x ( $\frac{d}{dx}$  y(x)) = y(x) ln(x y(x)) - y(x)
> sol := dsolve(ODE, y(x));
sol := y(x) =  $\frac{e^{(-\frac{x}{C1})}}{x}$ 
```

This solution of the ODE can easily be checked (and it is good advice to do this as much as you can): substitute the solution into the ODE and simplify it to a trivial equation

```
> simplify(eval(ODE, sol));

$$\frac{e^{(-\frac{x}{C1})} (-x + _C1)}{x\_C1} = \frac{e^{(-\frac{x}{C1})} (\ln(e^{(-\frac{x}{C1})}) - 1)}{x}$$

```

Not enough simplification: you have to assume that variables are positive.

```
> simplify(% , assume=positive);

$$\frac{e^{(-\frac{x}{C1})} (x - _C1)}{x\_C1} = \frac{e^{(-\frac{x}{C1})} (x - _C1)}{x\_C1}$$

```

In case you do not believe your eyes, let **testeq** do the work.

```
> testeq(%);
true
```

Maple provides a utility procedure called **odetest** to do the work of checking a (possible) solution of an ODE. If the solution is valid, the returned result will be 0; otherwise, the algebraic remaining expression will be returned.

```
> odetest(sol, ODE);
0
```

A natural and convenient notation is provided for by the **PDEtools** procedure **declare** in combination with the **alias** construct in Maple. Let us apply it to the same example.

```
> with(PDEtools, declare):
> declare(y(x), prime=x, quiet);
```

The quiet argument in the previous command suppresses the following remark.

```
> declare();
```

*Declared :  $y(x)$  to be displayed as  $y$   
 derivatives with respect to :  $x$  of functions of one variable are being  
 displayed with '*

Although the output is now displayed in a shorter way, the input isn't. For this purpose we can use the procedure **alias**.

```
> alias(y=y(x));
> ODE := x*diff(y,x) = y*ln(x*y) - y;
```

$$ODE := x y' = y \ln(xy) - y$$

```
> dsolve(ODE, y);
```

$$y = \frac{e^{\left(\frac{x}{-C_1}\right)}}{x}$$

You see once more that Maple chose to find an explicit solution of the differential equation. If you prefer an implicit expression for the function  $y$ , you can use the option **implicit** in the **dsolve** command. Later on, you can convert the implicit solution into an explicit solution via the procedure **solve** or **isolate**.

```
> dsolve(ODE, y, 'implicit');
ln(x) - _C1 - ln(ln(xy)) = 0
```

```
> isolate(% , y);
```

$$y = \frac{e^{\left(\frac{x}{e^{-C_1}}\right)}}{x}$$

Note that this is not the most readable expression for the general solution.

```
> subs(exp(_C1)=1/c, %);
```

$$y = \frac{e^{(x c)}}{x}$$

Let us also check this solution of the ODE.

```
> odetest(%, ODE);
```

$$c e^{(x c)} - \frac{e^{(x c)} \ln(e^{(x c)})}{x}$$

```
> simplify(%, assume=real);
```

$$0$$

As a second example, we consider the differential equation

$$y y' = \sqrt{y^2 + 2y + 1}.$$

In order to get more information about what is going on during the computation, we set a higher value to the variable **infolevel[dsolve]**.

```
> infolevel[dsolve] := 3:
> alias(y=y): # unalias y, but rest of notation
```

For a change, we shall introduce the differential equation with the **D** operator instead of with **diff**.

```
> ODE := (y*D(y)=sqrt(y^2-2*y+1))(x);
```

$$ODE := y D(y)(x) = \sqrt{(-1 + y)^2}$$

**dsolve** understands this notation.

```
> dsolve(ODE, y(x));
```

```
Methods for first order ODEs:
--- Trying classification methods ---
trying a quadrature
trying 1st order linear
trying Bernoulli
trying separable
<- separable successful
```

$$x + \frac{(1 - y)(y + \ln(-1 + y))}{\sqrt{(-1 + y)^2}} + _C1 = 0$$

You can try to further simplify this result

```
> simplify(%);
```

$$x - \text{csgn}(-1 + y)y - \text{csgn}(-1 + y)\ln(-1 + y) + _C1 = 0$$

But as you see, not much happens. Maple needs more information about the complex sign of  $y(x) - 1$ . Let us assume that  $y(x) > 1$  so that the complex sign equals 1.

```
> eval(% , csgn(y(x)-1)=1);
```

$$x - y - \ln(-1 + y) + _C1 = 0$$

```
> solve(% , y(x));
```

$$\text{LambertW}(e^{(-1+x+_C1)}) + 1$$

It is easy to verify that the assumption  $y(x) < 1$  leads to the solution

$$y(x) = \text{LambertW}(e^{-(x-_C1)}) + 1.$$

So, with some assistance we find two explicit solutions.

A differential equation may contain several variables. For example, look at the differential equation which describes the trajectory of an object that is pulled with a rope of length  $a$  by someone who walks along the  $x$ -axis to the right.

$$y' = -\frac{y}{\sqrt{a^2 - y^2}}$$

```
> restart: PDEtools[declare](y(x), prime=x, quiet):
> alias(y=y(x)): ODE := diff(y,x) = -y/sqrt(a^2-y^2);
```

$$ODE := y' = -\frac{y}{\sqrt{a^2 - y^2}}$$

```
> solution := dsolve(ODE, y);
solution := x + \sqrt{a^2 - y^2} - \frac{a^2 \ln(\frac{2a^2 + 2\sqrt{a^2} \sqrt{a^2 - y^2}}{y})}{\sqrt{a^2}} + _C1 = 0
```

The solution is simpler when we assume that unknowns, and in particular  $a$ , are positive.

```
> solution := simplify(% , assume=positive);
solution := x + \sqrt{a^2 - y^2} - a \ln(2) - a \ln(a) - a \ln(\frac{a + \sqrt{a^2 - y^2}}{y}) + _C1 = 0
```

Note that the constant function 0 is not recognized by Maple as a solution of the differential equation.

When the person starts at the origin and the object is at that time at position  $(0, a)$ , i.e., when the initial condition is  $y(0) = a$ , then you can attempt to determine the constant  $_C1$ .

```
> eval(solution, {x=0,y=a});
-a \ln(2) - a \ln(a) + _C1 = 0
```

Simplification of the solution with respect to this side relation gives the following result.

```
> solution := simplify(solution, %);
solution := x + \sqrt{a^2 - y^2} - a \ln(\frac{a + \sqrt{a^2 - y^2}}{y}) = 0
```

The solution of the differential equation could also have been expressed in terms of the inverse hyperbolic tangent function.

```
> solution := sqrt(a^2-y^2) - a*arctanh(sqrt(a^2-y^2)/a)
> + x = C;
solution := \sqrt{a^2 - y^2} - a \operatorname{arctanh}(\frac{\sqrt{a^2 - y^2}}{a}) + x = C
```

Let us verify this implicit solution.

```
> diff(solution, x);
- \frac{y y'}{\sqrt{a^2 - y^2}} + \frac{y y'}{\sqrt{a^2 - y^2} (1 - \frac{a^2 - y^2}{a^2})} + 1 = 0
> diff(y,x) = solve(%, diff(y,x));
y' = -\frac{y}{\sqrt{a^2 - y^2}}
```

Let us look again at the initial value problem  $y(0) = a$ .

```
> eval(solution, {x=0,y=a});  
0 = C
```

You might have been tempted to evaluate as follows.

```
> eval(solution, {x=0, y(0)=a});  

$$\sqrt{a^2 - y(0)^2} - a \operatorname{arctanh}\left(\frac{\sqrt{a^2 - y(0)^2}}{a}\right) = C$$
  
> subs(y(0)=0, %);  

$$\sqrt{a^2 - y(0)^2} - a \operatorname{arctanh}\left(\frac{\sqrt{a^2 - y(0)^2}}{a}\right) = C$$

```

The reason why this does not work as expected is that  $y$  has first been aliased to  $y(x)$ . After the first evaluation with  $x=0$ , an object  $y(0)$  is returned that differs from the one obtained when you enter  $y(0)$  without any alias. This is why the substitution does not work. You have to use one of the following tricks.

```
> eval(% , eval(y,x=0) = a);  
0 = C  
> eval(%%, op(0,y)(0) = a);  
0 = C
```

However, these are tricky solutions compared to the way we have chosen above. We end this example with a description of the curve, which is known as the Tractrix (q.v., [89]).

```
> Tractrix := eval(solution, C=0);  
Tractrix :=  $\sqrt{a^2 - y^2} - a \operatorname{arctanh}\left(\frac{\sqrt{a^2 - y^2}}{a}\right) + x = 0$ 
```

By the way, in Maple you can immediately specify an initial value or boundary value problem. We illustrate this by studying the differential equation

$$u'' + \omega^2 u = 0$$

of the mathematical pendulum (see Figure 17.3,  $u$  is the horizontal displacement of the end point). The same problems with aliases as described in the previous example would occur; hence we avoid them.

```
> ODE := diff(u(t),[t$2]) + omega^2*u(t) = 0;  
ODE :=  $(\frac{d^2}{dt^2} u(t)) + \omega^2 u(t) = 0$   
> # initial value problem  
> dsolve({ODE, u(0)=2, D(u)(0)=3}, u(t));
```

$$u(t) = \frac{3 \sin(\omega t)}{\omega} + 2 \cos(\omega t)$$

You specify the first derivative at zero as `D(u)(0)`. Higher derivatives  $u''(0)$ ,  $u'''(0)$ , ... are given as `(D@@2)(u)(0)`, `(D@@3)(u)(0)`, ...

```
> # boundary value problem
> dsolve({ODE, u(0)=1, u(2)=3}, u(t));
```

$$u(t) = -\frac{(\cos(2\omega) - 3)\sin(\omega t)}{\sin(2\omega)} + \cos(\omega t)$$

```
> combine(%);
```

$$u(t) = \frac{-\sin(\omega t - 2\omega) + 3 \sin(\omega t)}{\sin(2\omega)}$$

In Maple you can easily use the *method of integral transforms* for solving differential equations. As an example, we shall look at the step response of a linear damped oscillator via the method of Laplace transforms. The initial value problem is the following:

```
> ODE := diff(u(t), [t$2]) + 2*d*omega*diff(u(t), t)
>      + omega^2*u(t) = Heaviside(t);
```

$$ODE := (\frac{d^2}{dt^2} u(t)) + 2d\omega(\frac{d}{dt} u(t)) + \omega^2 u(t) = \text{Heaviside}(t)$$

```
> initvals := u(0)=0, D(u)(0)=0:
> solution := dsolve({ODE, initvals}, u(t),
> method=laplace);
```

*solution* :=

$$u(t) = \frac{\frac{1}{\omega} + e^{(-t d \omega)} \left( -\frac{\cosh(t \sqrt{d^2 \omega^2 - \omega^2})}{\omega} - \frac{d \sinh(t \sqrt{d^2 \omega^2 - \omega^2})}{\sqrt{d^2 \omega^2 - \omega^2}} \right)}{\omega}$$

Maple does not distinguish by itself the cases of no damping ( $d = 0$ ), underdamping ( $0 < d < 1$ ), critical damping ( $d = 1$ ), and overdamping ( $d > 1$ ). In most cases however, the above formula for the solution is simplified to its most convenient form. This may happen because you make explicit assumptions or because you provide explicit values for the unknowns.

```
> d := 0: # no damping
> simplify(solution, assume=positive);
```

$$u(t) = -\frac{-1 + \cos(\omega t)}{\omega^2}$$

```
> d := 1/6: # underdamping
> simplify(solution, assume=positive);
```

$u(t) =$

$$-\frac{1}{35} \frac{-35 + 35 e^{(-1/6 \omega t)} \cos(\frac{1}{6} t \omega \sqrt{35}) + e^{(-1/6 \omega t)} \sqrt{35} \sin(\frac{1}{6} t \omega \sqrt{35})}{\omega^2}$$

You can plot this solution for a range of frequency values.

```
> plot3d(rhs(%), omega=2/3..4/3, t=0..20,
> style=hidden, orientation=[-30,45], axes=box);
```

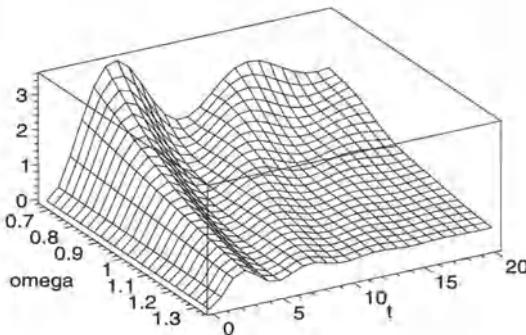


Figure 17.1. Step response of an underdamped oscillator for a range of frequencies.

Similarly, you can plot the solutions for a fixed frequency and a range of damping values.

```
> d := 'd': omega := 1:
> plot3d(rhs(solution), d=1/5..2, t=0..20,
> style=hidden, orientation=[-10,45], axes=BOXED);
```

From picture 17.2 the trade-off between system damping and system response is clear; low damping values result in overshooting, while high damping values are slow to respond to input signals.

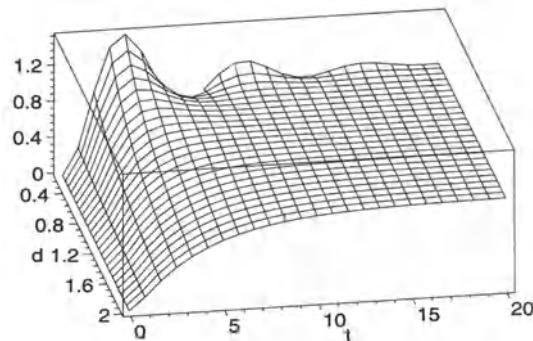


Figure 17.2. Step response of an oscillator for a range of damping factors.

Other integral transforms that can be applied as well for solving differential equations in Maple are **fourier**, **fouriersin**, and **fouriercos**. For Hilbert transforms or Hankel transforms you have to do the transformations and inverse transformations yourself. An example of the use of the Fourier transform:

```
> diff(u(t),[t$2]) + 3*diff(u(t),t)+ 2*u(t) = exp(-abs(t));
```

```


$$\left( \frac{d^2}{dt^2} u(t) \right) + 3 \left( \frac{d}{dt} u(t) \right) + 2 u(t) = e^{-|t|}$$

> dsolve(%, u(t), method=fourier);

$$u(t) = \frac{1}{6} (4 e^{-2t} - 3 e^{-t} + 6 t e^{-t}) \text{Heaviside}(t) + \frac{1}{6} e^t \text{Heaviside}(-t)$$


```

Systems of linear differential equations can also be solved with **dsolve**.

```

> ODE := diff(f(t),[t$2]) - 6*diff(g(t),t) = 6*sin(t),
>       6*diff(g(t),[t$2]) + c^2*diff(f(t),t) = 6*cos(t);

```

*ODE* :=

```


$$\left( \frac{d^2}{dt^2} f(t) \right) - 6 \left( \frac{d}{dt} g(t) \right) = 6 \sin(t), 6 \left( \frac{d^2}{dt^2} g(t) \right) + c^2 \left( \frac{d}{dt} f(t) \right) = 6 \cos(t)$$

> initvals := f(0)=0, g(0)=1, D(f)(0)=0, D(g)(0)=1:
> funcs := f(t),g(t):
> dsolve({ODE, initvals}, {funcs}, method=laplace );

$$\begin{aligned} f(t) &= -\frac{12 \sin(ct)}{(c-1)(c+1)c} + \frac{6(-\cos(ct)+1)}{c^2} + \frac{12 \sin(t)}{(c-1)(c+1)}, \\ g(t) &= \frac{\sin(ct)}{c} + \frac{-2 \cos(ct) + (c^2+1) \cos(t)}{(c-1)(c+1)} \} \\ > collect(%, \{\cos(c*t), \sin(c*t)\}, simplify); \end{aligned}$$


$$\begin{aligned} f(t) &= -\frac{6 \cos(ct)}{c^2} - \frac{12 \sin(ct)}{(c^2-1)c} + \frac{6(c^2-1+2 \sin(t)c^2)}{c^2(c^2-1)}, \\ g(t) &= -\frac{2 \cos(ct)}{c^2-1} + \frac{\sin(ct)}{c} + \frac{(c^2+1) \cos(t)}{c^2-1} \} \end{aligned}$$


```

Table 17.1 and Table 17.2 list types of ordinary differential equations that can presently be solved with **dsolve**. How long these lists already are, they are still not exhaustive: all classification patterns can be found in the on-line help system by entering **?odeadvisor,types**.

Moreover, Kovacic's algorithm [153] has been implemented in Maple; it tries to solve linear homogeneous second order ODEs of the form

$$P(x)y'' + Q(x)y' + R(x)y = 0,$$

where  $P(x)$ ,  $Q(x)$ , and  $R(x)$  are rational functions. It is a decision procedure like the Risch algorithm: given the linear second order ODE, Kovacic's algorithm decides whether or not there exists a closed form Liouvillian solution, and if the answer is yes, provides the solution. For higher order linear differential equations, the following two important decision procedures exist: Abramov's algorithm RATLODE (RATional Linear ODE, [1]), which determines whether or not a closed form solution exists in the domain of coefficients of the ODE; and Bronstein's algorithm EXPLODE (EXPonential Linear ODE, [29]), which determines solutions of

Type of ODE	Shape of ODE
quadrature	$y' = F(x)$ or $y' = F(y(x))$
linear	$y' + P(x)y = Q(x)$
exact	$y' = -\frac{P(x,y)}{Q(x,y)}$ , with $\frac{\partial Q}{\partial x} = \frac{\partial P}{\partial y}$
inexact	$F(x,y)y' + G(x,y) = 0$ , when an integrating factor can be found
separable	$y' = f(x)g(y)$
homogeneous	$y' = F(\frac{y}{x})$ , $y' = F(\frac{y}{x^\alpha})\frac{y}{x}$ , and others
high degree	$x = F(y, y')$
Abel	$y' = P(x)y^3 + Q(x)y^2 + R(x)y + S(x)$
Bernoulli	$y' + P(x)y = Q(x)y^n$ , where $n \in \mathbb{Z}$
Chini	$y' = P(x)y^n - Q(x)y + R(x)$ , where $n \in \mathbb{Z}$
Clairaut	$y = xy' + F(y')$
d'Alembert	$y = xF(y') + G(y')$
Riccati	$y' = P(x)y^2 + Q(x)y' + R(x)$

Table 17.1. Some solvable first order ODEs.

the form  $\exp(\int f(x) dx)$ . For a leisurely account on some methods implemented in Maple for linear differential equations we refer to [158]. In Maple version 8, a new systematic approach has lead to radical improvements in the determination of solutions, for second and higher-order linear ODEs, in term of special functions. Enter `?odeadvisor,linear_ODEs` for details about the methods that Maple can apply for linear ODEs. A few examples will give you an impression of how things work.

```
> alias(y=y(x)): infolevel[dsolve] := 2:
> x^2*diff(y, [x$3]) + (x-3*x^2)*diff(y, [x$2])
>      + (4*x^2-2*x)*diff(y,x) + (x-2*x^2)*y = 0;

$$x^2 \left( \frac{\partial^3 y}{\partial x^3} \right) + (x - 3x^2) \left( \frac{\partial^2 y}{\partial x^2} \right) + (4x^2 - 2x) \left( \frac{\partial y}{\partial x} \right) + (x - 2x^2)y = 0$$

> dsolve(% , y);
```

```
Methods for third order ODEs:
--- Trying classification methods ---
trying a quadrature
checking if the LODE has constant coefficients
checking if the LODE is of Euler type
trying high order exact linear fully integrable
trying to convert to a linear ODE with constant coefficients
trying a solution in terms of MeijerG functions
    trying a solution in terms of MeijerG functions
    checking if the LODE is of Euler type
    expon. solutions partially successful. Result(s) = [exp(x)]
        trying a quadrature
        checking if the LODE has constant coefficients
```

Type of ODE	Shape of ODE
quadrature	$y'' = F(x)$
linear, constant coeffs.	$ay'' + by' + cy = d(x)$ , where $a, b, c \in \mathbb{C}$
reducible	$y'' = F(x, y)$ , for which there exist an integrating factor that only depends on $y'$
Euler	$x^2y'' + axy' + by = c(x)$
Bessel	$x^2y'' + (2k+1)xy' + (\alpha^2x^{2r} + \beta^2)y = 0$ , where $k, \alpha, r, \beta \in \mathbb{C}$ and $\alpha r \neq 0$
Bernoulli	$y' + P(x)y = Q(x)y^n$
Gegenbauer	$(x^2 - 1)y'' - (2m+3)xy' + \lambda y = 0$ , where $m \in \mathbb{Z}$
Hermite	$y'' = xy' - ny$ , where $n \in \mathbb{Z}$
Hypergeometric	$x(1-x)y'' + ((c - (a+b+1)x)y' - aby = 0$ , where $a, b, c \in \mathbb{C}$
Liouville	$y'' + P(y)(y')^2 + Q(x)y' = 0$
Riemann-Papperitz	$y'' + P y' + Q y = 0$ , where $P = \frac{1-\alpha_1-\beta_1}{x-a} + \frac{1-\alpha_2-\beta_2}{x-b} + \frac{1-\alpha_3-\beta_3}{x-c}$ , $Q = \left( \frac{\alpha_1\beta_1(a-b)(a-c)}{x-a} + \frac{\alpha_2\beta_2(b-a)(b-c)}{x-b} + \frac{\alpha_3\beta_3(c-a)(c-b)}{x-c} \right) \times \frac{1}{(x-a)(x-b)(x-c)}$
Van der Pol	$y'' - \mu(1 - y^2)y' + y = 0$

Table 17.2. Some solvable second order ODEs.

```

checking if the LODE is of Euler type
trying a symmetry of the form [xi=0, eta=F(x)]
checking if the LODE is missing 'y'
trying a Liouvillian solution using Kovacic's algorithm
  No Liouvillian solutions exists
trying a solution in terms of special functions:
  -> Bessel
  <- Bessel successful
  <- solution in terms of special functions successful
<- differential factorization successful

```

$$\begin{aligned}
y &= -C1 e^x + -C2 x(-2 \text{BesselJ}(0, x) - \pi \text{StruveH}(0, x) \text{BesselJ}(1, x) \\
&\quad + \pi \text{StruveH}(1, x) \text{BesselJ}(0, x))e^x + -C3 x(-2 \text{BesselY}(0, x) \\
&\quad - \pi \text{StruveH}(0, x) \text{BesselY}(1, x) + \pi \text{StruveH}(1, x) \text{BesselY}(0, x))e^x
\end{aligned}$$

An example of a Bessel equation.

```

> ODE := x*diff(y,[x$2]) + diff(y,x) - x*y = 0;
ODE := ( $\frac{\partial^2}{\partial x^2} y$ )x + ( $\frac{\partial}{\partial x} y$ ) - y x = 0
> dsolve(ODE, y);

```

Methods for second order ODEs:

```

--- Trying classification methods ---
trying a quadrature
checking if the LODE has constant coefficients
checking if the LODE is of Euler type
trying a symmetry of the form [xi=0, eta=F(x)]
checking if the LODE is missing 'y'
trying a Liouvillian solution using Kovacic's algorithm
No Liouvillian solutions exists
trying a solution in terms of special functions:
-> Bessel
-< Bessel successful
-< solution in terms of special functions successful

```

$$y = -C1 \text{BesselI}(0, x) + -C2 \text{BesselK}(0, x)$$

You could have asked Maple to apply immediately a method involving Bessel functions.

```

> sol_b := dsolve(ODE, y, [Bessel]);
Classification methods on request
Successful isolation of d^2y/dx^2
Methods to be used are: [Bessel]
-----
* Tackling ODE using method: Bessel
--- Trying classification methods ---
-> searching for a solution in terms of Bessel functions
-< solution in terms of Bessel functions successful
sol_b := y = -C1 BesselI(0, x) + -C2 BesselK(0, x)

```

All methods used for tackling ODEs are listed in the table **dsolve/methods**. The indices of this table are the following.

```

> indices('dsolve/methods');
[1], [high, linear_nonhomogeneous], [2, "hypergeometric"],
[1, development], [2, linear_homogeneous], [2, development],
[3, nonlinear], [high, nonlinear], [3, development],
[high, linear_homogeneous], [high, development],
[2, "linear_homogeneous all methods"], [2, nonlinear],
[2, "linear_homogeneous as given"],
[2, "linear_homogeneous other"], [2, linear_nonhomogeneous],
[1, high_degree], [2, "linear_homogeneous in Normal Form"],
[3, linear_homogeneous], [2, "special_functions"], [1, semiclass],
[3, linear_nonhomogeneous]

```

The natural numbers in the indices refer to the order of the ODE that can be tackled by the methods under consideration. Let us have a look at the methods that Maple has for computing solutions that involve special functions for second order linear homogeneous ODEs.

```
> 'dsolve/methods' [2,"special_functions"];
```

[Bessel, elliptic, Legendre, Whittaker, hypergeometric, cylindrical]

All these methods are available directly at user level by giving optional extra arguments to the procedure **dsolve**. For example, we could have asked Maple to solve the Bessel equation in terms of hypergeometric functions.

```
> sol_h := dsolve(ODE, y, [hypergeometric]);
Classification methods on request
Methods to be used are: [hypergeometric]
-----
* Tackling ODE using method: hypergeometric
--- Trying classification methods ---
trying a solution in terms of hypergeometric functions
-> heuristic approach
-< heuristic approach successful
-< hypergeometric solution successful
```

$$sol\_h := y = _C1 e^x \text{KummerM}\left(\frac{1}{2}, 1, -2x\right) + _C2 e^x \text{KummerU}\left(\frac{1}{2}, 1, -2x\right)$$

Maybe you are surprised that the answer is not in terms of hypergeometric functions, but in terms of another special function. This is because Maple has many conversion possibilities between special functions. For example, the above solution can be rewritten in terms of Whittaker functions.

```
> convert(%, Whittaker);
y = 
$$\frac{-C1 \text{WhittakerM}(0, 0, -2x)}{\sqrt{-2x}} + \frac{-C2 \text{WhittakerW}(0, 0, -2x)}{\sqrt{-2x}}$$

```

From the penultimate output, any of the two special functions entering **sol\_h** is equal to the linear combination of the Bessel functions entering **sol\_b**, for some particular values of  $_C1$  and  $_C2$ . We express

```
> h := eval(rhs(sol_h), {_C1=1, _C2=0});
h := e^x \text{KummerM}\left(\frac{1}{2}, 1, -2x\right)
> h := convert(h, hypergeometric, only=KummerM);
h := e^x \text{hypergeom}\left([\frac{1}{2}], [1], -2x\right)
```

in terms of Bessel functions (finally). The following expression must be equal to 0.

```
> h - rhs(sol_b);
e^x \text{hypergeom}\left([\frac{1}{2}], [1], -2x\right) - _C1 \text{BesselI}(0, x) - _C2 \text{BesselK}(0, x)
```

So, the following series should be 0 for any  $x$ .

```
> series(% , x, 2);
(1 - _C1 - _C2 (\ln(2) - \ln(x) - \gamma)) + O(x^2)
```

In this case, we do not have to use **solve** for finding the solution: obviously,  $\_C1=1$  and  $\_C2=0$ .

```
> b := eval(rhs(sol_b), {_C1=1, _C2=0});  
b := BesselI(0, x)
```

Thus,

$$e^x \text{hypergeom}\left(\left[\frac{1}{2}\right], [1], -2x\right) = \text{BesselI}(0, x).$$

We could also have found this identity in the following way.

```
> h = convert(h, StandardFunctions);  
  
e^x \text{hypergeom}\left(\left[\frac{1}{2}\right], [1], -2x\right) = \text{BesselI}(0, x)
```

Let us have a look at another Bessel equation.

```
> infolevel[dsolve] := 0: # no further information  
> ODE := diff(y, [x$2]) + 3*diff(y, x)/x  
> + (x^2-15)/x^2*y - 1/x;  
  
ODE := (\frac{\partial^2}{\partial x^2} y) + \frac{3(\frac{\partial}{\partial x} y)}{x} + \frac{(x^2 - 15)y}{x^2} - \frac{1}{x}  
  
> dsolve(ODE, y);  
  
y = \frac{\text{BesselJ}(4, x)\_C2}{x} + \frac{\text{BesselY}(4, x)\_C1}{x} + \frac{192 + 16x^2 + x^4}{x^5}
```

An alternative output form is available in terms of a *solution basis*.

```
> dsolve(ODE, y, output=basis);  
  
[[\frac{\text{BesselY}(4, x)}{x}, \frac{\text{BesselJ}(4, x)}{x}], \frac{192 + 16x^2 + x^4}{x^5}]
```

Even if Maple recognizes the type of an ODE it may still be useful to assist Maple. An example.

```
> restart; alias(y=y(x));  
> ODE := diff(y, x) + y^2 - y*sinh(2*x) + cosh(2*x) = 2;  
  
ODE := (\frac{\partial}{\partial x} y) + y^2 - y \sinh(2x) + \cosh(2x) = 2
```

This is a Riccati equation, as Maple confirms, when you ask for the type of equation with the procedure **odeadvisor** from the **DEtools** package.

```
> DEtools[odeadvisor](ODE);  
[_Riccati]
```

Maple does its best to solve the ODE and it can do it all by itself.

```
> dsolve(ODE, y);
```

$$y = \frac{-C1 e^{((\cosh(x)-1)(\cosh(x)+1))}}{\cosh(x)^2 \left( -C1 \int \frac{e^{((\cosh(x)-1)(\cosh(x)+1))}}{\cosh(x)^2} dx + 1 \right)} + \frac{\sinh(x)}{\cosh(x)}$$

Let us now help Maple a bit. It is easily checked that  $\tanh x$  is indeed a solution.

```
> odetest(y=tanh(x), ODE);
```

0

Write the required solution  $y(x)$  as  $\tanh x + z(x)$ , derive the ODE for the function  $z(x)$ , and try to solve this one.

```
> odetest(y=tanh(x)+z(x), ODE);
```

$$\frac{1}{2} \frac{2(\frac{d}{dx} z(x)) \cosh(x) + 3 \sinh(x) z(x) + 2 z(x)^2 \cosh(x) - z(x) \sinh(3x)}{\cosh(x)}$$

```
> expand(%);
```

$$(\frac{d}{dx} z(x)) + \frac{2 \sinh(x) z(x)}{\cosh(x)} + z(x)^2 - 2 z(x) \sinh(x) \cosh(x)$$

```
> dsolve(%, z(x));
```

$$z(x) = \frac{e^{(\sinh(x)^2)}}{\left( \int \frac{e^{(\sinh(x)^2)}}{\cosh(x)^2} dx + _C1 \right) \cosh(x)^2}$$

Check the general solution, which looks a bit simpler than the one found by Maple itself.

```
> odetest(y=tanh(x)+rhs(%), ODE);
```

0

Sometimes Maple comes back with a partial solution.

```
> restart: alias(y=y(x));
> ODE := diff(y,x$3)+4*diff(y,x$2)+4*diff(y,x)-12*y/x = 0;
```

$$ODE := (\frac{\partial^3}{\partial x^3} y) + 4 (\frac{\partial^2}{\partial x^2} y) + 4 (\frac{\partial}{\partial x} y) - \frac{12 y}{x} = 0$$

```
> dsolve(%, y);
```

$$y = _C1 (27 x + 24 x^2 + 4 x^3) + \int \text{DESol} \left( \left\{ \frac{(2916 x + 11340 x^2 + 11904 x^3 + 5136 x^4 + 960 x^5 - 729 + 64 x^6) \text{Y}(x)}{x (27 + 24 x + 4 x^2)^2} \right\} \right)$$

$$\left. + \frac{(156x + 108x^2 + 16x^3 + 27)(\frac{\partial}{\partial x} - Y(x))}{27 + 24x + 4x^2} + x(\frac{\partial^2}{\partial x^2} - Y(x)) \right\}, \{-Y(x)\} \right) \\ / (27x + 24x^2 + 4x^3) dx(27x + 24x^2 + 4x^3)$$

This can be considered as the analog of **RootOf**: **DESol** represents a solution of a differential equation and can be manipulated with **diff**, **series**, **evalf**, and so on. Enter **?DESol** to see examples of this feature. So, we can check the above solution by

```
> odetest(% , ODE);
0
```

and compute a series solution:

```
> series(rhs(%), x, 4);
```

$$2\_C2 + (\frac{16\_C2}{9} + 27\_C1)x + (\frac{1676\_C2}{81} + 12\_C2 \ln(x) + 25\_C1)x^2 + \\ (-\frac{4832\_C2}{243} - 16\_C2 \ln(x) + \frac{8\_C1}{3})x^3 + O(x^4)$$

Other inert structures for solutions of ODEs such as **ODESolStruc** and **FPSstruct** exist. We refer to the on-line help system for further information

Table 17.3 summarizes the options of the ordinary differential equation solver **dsolve** when attempting to find an analytical solution (optionally specified as **type=exact**).

Option	Meaning
method = fourier	Fourier method
method = fouriercos	method of Fourier cosine transform
method = fouriersin	method of Fourier sine transform
method = laplace	Laplace method
output = basis	generate solution basis and particular solution
class	use only classification methods
Lie	apply only Lie symmetry methods
explicit	attempt to find explicit solution
implicit	implicit solution

Table 17.3. Options of **dsolve(ODE, type=exact)**.

## 17.3 Lie Point Symmetries for ODEs

Lie symmetry methods systematically deal with problems like transforming differential equations to standard ones that we know solutions of. They form

the main tools in Maple for solving ODEs. In this section we shall only sketch the main methods and illustrate them through the built-in utilities of the `DEtools` package. The interested reader is referred to [136, 189, 206, 213] for a thorough theoretical discussion of the topic and more examples. The Maple implementation of these methods is described in detail in [50, 51, 52, 53, 54].

A *Lie point symmetry* for the ODE

$$\omega(x, y, y', y'', \dots) = 0$$

is a mapping

$$x \rightarrow \bar{x}(x, y), \quad y \rightarrow \bar{y}(x, y)$$

such that the new variables obey the original equation. Usually, one considers only one-parameter groups of transformations

$$\bar{x} = \bar{x}(x, y; \varepsilon), \quad \bar{y} = \bar{y}(x, y; \varepsilon)$$

with  $x = \bar{x}(x, y; 0)$  and  $y = \bar{y}(x, y; 0)$  defined in the  $x$ - $y$ -plane. So, the identity transformation corresponds to the parameter value  $\varepsilon = 0$ . The corresponding infinitesimal mappings are

$$x \rightarrow x + \varepsilon \xi, \quad y \rightarrow y + \varepsilon \eta,$$

for some parameter  $\varepsilon$ , and functions  $\xi, \eta$  defined by

$$\xi(x, y) = \frac{\partial \bar{x}}{\partial \varepsilon}|_{\varepsilon=0}, \quad \eta(x, y) = \frac{\partial \bar{y}}{\partial \varepsilon}|_{\varepsilon=0}.$$

The derivatives  $y', y'', \dots$ , then transform by “prolongation”:

$$y' \rightarrow y' + \varepsilon \eta^{(1)}, \quad y'' \rightarrow y'' + \varepsilon \eta^{(2)}, \quad \dots$$

where the  $\eta^{(k)}$  are recursively defined by

$$\begin{aligned} \eta^{(1)} &= D(\eta) - y'D(\xi), & \eta^{(2)} &= D(\eta^{(1)}) - y''D(\xi), \\ \eta^{(k)} &= D(\eta^{(k-1)}) - y^{(k)}D(\xi) \end{aligned}$$

for the operator  $D$  of total differentiation with respect to  $x$ , which is

$$D = \frac{\partial}{\partial x} + y' \frac{\partial}{\partial y} + y'' \frac{\partial}{\partial y'} + \dots + y^{(n)} \frac{\partial}{\partial y^{(n-1)}}.$$

Be careful: we use here for the derivatives of the function  $y$  both the notation  $y', y'', y''', \dots$ , and  $y^{(1)}, y^{(2)}, y^{(3)}, \dots$ . However,  $\eta^{(1)}, \eta^{(2)}, \dots$ , are not the derivatives of  $\eta$ . The  $\eta^{(k)}$  soon become complicated expressions: below we compute the prolongations up to order 3 from first principles.

```

> alias( y=y(x), 'y'='diff(y(x),x),
> 'y''='diff(y(x),x$2)', 'y'''='diff(y(x),x$3)',
> xi=xi(x,y(x)), eta=eta(x,y(x)),
> xi[x]=D[1](xi)(x,y(x)), xi[y]=D[2](xi)(x,y(x)),
> eta[x]=D[1](eta)(x,y(x)), eta[y]=D[2](eta)(x,y(x)),
> xi[xx]=D[1,1](xi)(x,y(x)), xi[xy]=D[1,2](xi)(x,y(x)),
> xi[yy]=D[2,2](xi)(x,y(x)), eta[xx]=D[1,1](eta)(x,y(x)),
> eta[xy] = D[1,2](eta)(x,y(x)),
> eta[yy] = D[2,2](eta)(x,y(x)),
> xi[xxx] = D[1,1,1](xi)(x,y(x)),
> xi[xxy] = D[1,1,2](xi)(x,y(x)),
> xi[xyy] = D[1,2,2](xi)(x,y(x)),
> xi[yyy] = D[2,2,2](xi)(x,y(x)),
> eta[xxx] = D[1,1,1](eta)(x,y(x)),
> eta[xxy] = D[1,1,2](eta)(x,y(x)),
> eta[xyy] = D[1,2,2](eta)(x,y(x)),
> eta[yyy] = D[2,2,2](eta)(x,y(x)) );

```

The purpose of the **alias** statement is just to introduce a short readable notation for partial derivatives of the functions  $\xi(x, y)$  and  $\eta(x, y)$ . For example,  $\xi_{xx}$  denotes  $\frac{\partial^2 \xi}{\partial x^2}$ , and so on. Below, we shall use X and Y instead of  $\bar{x}$  and  $\bar{y}$ ,  $dYdX$  instead of  $\frac{dY}{dX}$ ,  $dY2dX2$  instead of  $\frac{d^2Y}{dX^2}$ , and  $dY3dX3$  instead of  $\frac{d^3Y}{dX^3}$ .

```
> X := x + epsilon*xi;
```

$$X := x + \varepsilon \xi$$

```
> Y := y + epsilon*eta;
```

$$Y := y + \varepsilon \eta$$

According to the chain rule

$$\frac{d\bar{y}}{dx} = \frac{d\bar{y}}{d\bar{x}} \frac{d\bar{x}}{dx}.$$

So

$$\frac{d\bar{y}}{d\bar{x}} = \frac{d\bar{y}}{dx} \Big/ \frac{d\bar{x}}{dx}.$$

Thus

```
> dYdX := diff(Y,x) / diff(X,x);
```

$$dYdX := \frac{y' + \varepsilon (\eta_x + \eta_y y')}{1 + \varepsilon (\xi_x + \xi_y y')}$$

We compute the first order term of the Taylor series approximation about  $\varepsilon = 0$ .

```
> coeftayl(% , epsilon=0, 1);
```

$$\eta_x + \eta_y y' - y' (\xi_x + \xi_y y')$$

Finally we sort with respect to the derivative of  $y$ .

```
> collect(%,'y'');

$$-\xi_y y'^2 + (\eta_y - \xi_x) y' + \eta_x$$

```

Thus we have proved

$$\eta^{(1)} = \eta_x + (\eta_y - \xi_x) y' - \xi_y y'^2$$

We continue with the computation of the second order prolongation  $\eta^{(2)}$ .

```
> dY2dX2 := diff(dYdX,x) / diff(X,x):
> coeftayl(% , epsilon=0, 1):
> collect(% , ['y','y'''], distributed);
```

$$\begin{aligned} & -\xi_{yy} y'^3 + (\eta_{yy} - 2\xi_{xy}) y'^2 + (2\eta_{xy} - \xi_{xx}) y' - 3\xi_y y'' y' + \eta_{xx} \\ & + (-2\xi_x + \eta_y) y'' \end{aligned}$$

Expressions become more and more complicated: the third order prolongation  $\eta^{(3)}$  is the following:

```
> dY3dX3 := diff(dY2dX2,x) / diff(X,x):
> coeftayl(% , epsilon=0, 1):
> collect(% , ['y','y''','y'''], distributed);


$$\begin{aligned} & -\xi_{yyy} y'^4 + (\eta_{yyy} - 3\xi_{xyy}) y'^3 + (3\eta_{xyy} - 3\xi_{xxy}) y'^2 - 6\xi_{yy} y'' y'^2 \\ & + (3\eta_{xxy} - \xi_{xxx}) y' + (-9\xi_{xy} + 3\eta_{yy}) y'' y' - 4\xi_y y''' y' - 3\xi_y y''^2 \\ & + (-3\xi_{xx} + 3\eta_{xy}) y'' + \eta_{xxx} + (\eta_y - 3\xi_x) y''' \end{aligned}$$

```

Of course Maple contains a procedure to compute these prolongations, viz., the procedure **eta\_k** in the **DEtools** package. Let us verify the first two prolongations found above.

```
> X := [xi,eta];

$$X := [\xi, \eta]$$

```

We use the procedure **PDEtools[declare]** to install short notation of derivatives and we introduce aliases to display the variables  $_y1$  and  $_y2$ , which are generated by Maple, as  $y'$  and  $y''$ , respectively.

```
> PDEtools[declare](xi, eta, quiet):
> alias('y'='y1, 'y''='y2'):
> with(DEtools, eta_k):
> eta1 = eta_k(X, 1, y);

$$\eta1 = -y'^2 \xi_y + (\eta_y - \xi_x) y' + \eta_x$$

```

```
> eta2 = eta_k(X, 2, y);
```

$$\begin{aligned} \eta2 = & -y'^3 \xi_{yy} + (-2\xi_{xy} + \eta_{yy}) y'^2 + (2\eta_{xy} - \xi_{xx}) y' - 3y'' \xi_y \\ & + (\eta_y - 2\xi_x) y'' + \eta_{xx} \end{aligned}$$

The procedure `eta.k` also computes prolongations of so-called *contact transformations*, in which the functions  $\xi$  and  $\eta$  not only depend on  $x$  and  $y$ , but also on the derivative  $y'$ , and more generally, it also computes so-called *dynamical symmetries*, which involve even higher order derivatives.

The differential equation

$$\omega(x, y, y', y'', \dots, y^{(n)}) = 0$$

is left invariant by the transformation with the infinitesimal generator

$$U = \xi(x, y) \frac{\partial}{\partial x} + \eta(x, y) \frac{\partial}{\partial y}$$

if the  $n$ -th order prolongation  $U^{(n)}$  defined by

$$\begin{aligned} U^{(n)} = & \xi(x, y) \frac{\partial}{\partial x} + \eta(x, y) \frac{\partial}{\partial y} + \eta^{(1)}(x, y, y') \frac{\partial}{\partial y'} + \eta^{(2)}(x, y, y', y'') \frac{\partial}{\partial y''} \\ & + \dots + \eta^{(n)}(x, y, y', y'', \dots, y^{(n)}) \frac{\partial}{\partial y^{(n)}} \end{aligned}$$

satisfies the relation

$$U^{(n)} \omega(x, y, y', y'', \dots, y^{(n)}) = 0$$

on the manifold  $\omega = 0$  in the space of variables  $x, y, y', y'', \dots, y^{(n)}$ .

**Example.** Take  $\omega(x, y, y', y'') = y'' + y$  and  $\bar{x} = x$ ,  $\bar{y} = (1 + \varepsilon)y$ , i.e.,  $\xi(x, y) = 0$ ,  $\eta(x, y) = y$ . Then the prolongation  $U^{(2)}$  of the infinitesimal operator  $U = y \frac{\partial}{\partial y}$  can easily be computed and is given by

$$U^{(2)} = y \frac{\partial}{\partial y} + y' \frac{\partial}{\partial y'} + y'' \frac{\partial}{\partial y''}.$$

Hence,

$$U^{(2)}(\omega(x, y, y', y'')) = y + y'' = 0 \pmod{\omega = 0}.$$

The property  $U^{(n)} \omega(x, y, y', y'', \dots, y^{(n)}) = 0 \pmod{\omega = 0}$  leads to a system of partial differential equations for the functions  $\xi$  and  $\eta$ . This is called the *determining system*.

**Example.** A first order ODE of the form

$$y' = \Phi(x, y)$$

is invariant under the infinitesimal operator

$$U = \xi(x, y) \frac{\partial}{\partial x} + \eta(x, y) \frac{\partial}{\partial y}$$

if the prolongation

$$U^{(1)} = \xi(x, y) \frac{\partial}{\partial x} + \eta(x, y) \frac{\partial}{\partial y} + \eta^{(1)}(x, y, y') \frac{\partial}{\partial y'}$$

with

$$\eta^{(1)} = \eta_x + (\eta_y - \xi_x) y' - \xi_y y'^2$$

satisfies the relation

$$U^{(1)}(y' - \Phi(x, y)) = 0 \quad (\text{mod } y' - \Phi(x, y) = 0).$$

Let us use Maple to work this out. In the session below, `y1` aliases  $y'$  and `eta1` denotes  $\eta^{(1)}$ .

```
> restart: alias('y'='y1, Phi=Phi(x,y)):
> ODE := y1 = Phi;
```

$$ODE := y' = \Phi$$

```
> omega := (lhs-rhs)(%);
```

$$\omega := y' - \Phi$$

```
> eta1 := eta[x] + (eta[y]-xi[x])*y1 - xi[y]*y1^2;
```

$$\eta1 := \eta_x + (\eta_y - \xi_x) y' - \xi_y y'^2$$

Invariance means:

```
> xi*diff(omega,x) + eta*diff(omega,y)
> + eta1*diff(omega,'y') = 0;
-<\left(\frac{\partial}{\partial x}\Phi\right) - \eta\left(\frac{\partial}{\partial y}\Phi\right) + \eta_x + (\eta_y - \xi_x) y' - \xi_y y'^2 = 0
> eval(% , ODE);
-\left(\frac{\partial}{\partial x}\Phi\right) - \eta\left(\frac{\partial}{\partial y}\Phi\right) + \eta_x + (\eta_y - \xi_x)\Phi - \xi_y\Phi^2 = 0
```

This equation must be an identity in  $x$  and  $y$ . By the way, the `DEtools` package provides the procedure `odepde` to compute the determining system right from the differential equation.

```
> ODE := subs([y=y(x), 'y'='diff(y(x),x)], ODE);
ODE :=  $\frac{d}{dx}y(x) = \Phi(x, y(x))$ 
> PDEtools[declare](_xi(x,y), _eta(x,y), quiet):
> alias(xi=_xi, eta=_eta):
> DEtools[odepde](ODE);
 $\eta_x + (\eta_y - \xi_x)\Phi - \xi_y\Phi^2 - \xi\Phi_x - \eta\Phi_y$ 
```

Supposing that we can solve the determining system of an ODE, how would that help? To answer this question we look at first and second order ordinary differential equations. This will give you the general idea of the usefulness of Lie point symmetry methods.

## First-Order ODEs

Let us first look at our example of a first-order ODE of the form

$$y' = \Phi(x, y).$$

Once a solution to the determining system

$$-\xi \left( \frac{\partial}{\partial x} \Phi \right) - \eta \left( \frac{\partial}{\partial y} \Phi \right) + \eta_x + (\eta_y - \xi_x) \Phi - \xi_y \Phi^2 = 0$$

is found, there are two ways to proceed:

- Using the expressions for  $\xi(x, y)$  and  $\eta(x, y)$ , we can build a solution in implicit form by the line integral

$$\int \frac{dy - \Phi dx}{\eta - \xi \Phi} = \text{constant}.$$

Stated differently:  $\frac{1}{\eta - \xi \Phi}$  is an *integrating factor* of the first-order ODE.

- Separation of variables of variables is achieved if  $u(x, y)$  and  $v(x, y)$  are determined as solutions of

$$U u = 0, \quad U v = 1.$$

This reduces the ODE to an equation that can be solved by quadrature, i.e., solved with just an integration. The new variables  $u, v$  are called the *canonical variables*.

Finding solutions of the determining system looks as a much more complicated problem than solving the original differential equation. However, by heuristic methods mixed with formal methods coming from differential algebra and differential algebra Gröbner bases, particular solutions are often found. This makes Lie point symmetry methods successful.

The **DEtools** package contains utility functions to carry out individual steps in the Lie point symmetry approach. To make things more concrete we consider the following example of a first-order ordinary differential equation of Riccati type (taken from [50]).

$$y' = \frac{(y - x \ln x)^2}{x^2} + y$$

As we have already seen, the procedure **odepde** can be used to compute the determining system.

```
> restart: with(DEtools):
> PDEtools[declare](y(x), prime=x, (_xi,_eta)(x,y), quiet):
> alias(xi=_xi, eta=_eta):
> ODE := diff(y(x), x) = (y(x)-x*ln(x))^2/x^2 + ln(x);
```

```

ODE :=  $y' = \frac{(y - x \ln(x))^2}{x^2} + \ln(x)$ 
> odeadvisor(ODE);
[[[-1st_order, _with_linear_symmetries], _Riccati]]
> determining_system := odepde(ODE);

determining_system :=  $\eta_x + (\eta_y - \xi_x) \left( \frac{(y - x \ln(x))^2}{x^2} + \ln(x) \right)$ 
 $- \xi_y \left( \frac{(y - x \ln(x))^2}{x^2} + \ln(x) \right)^2$ 
 $- \xi \left( \frac{2(y - x \ln(x))(-\ln(x) - 1)}{x^2} - \frac{2(y - x \ln(x))^2}{x^3} + \frac{1}{x} \right)$ 
 $- \frac{2\eta(y - x \ln(x))}{x^2}$ 

```

In general, the Maple procedure **pdsolve** for solving partial differential equation will not be able to solve the determining system. But what we are really looking for is a simple solution of the determining system. For this purpose, we collect the logarithmic terms.

```

> collect(% , ln);

 $-\xi_y \ln(x)^4 - \xi_y \left( -\frac{4y}{x} + 2 \right) \ln(x)^3 + (\eta_y - \xi_x - \xi_y \left( \frac{2y^2}{x^2} + (-\frac{2y}{x} + 1)^2 \right)) \ln(x)^2$ 
 $+ \left( (\eta_y - \xi_x) \left( -\frac{2y}{x} + 1 \right) - \xi \left( \frac{2y}{x^2} + \frac{2}{x} \right) + \frac{2\eta}{x} - \frac{2\xi_y y^2 \left( -\frac{2y}{x} + 1 \right)}{x^2} \right) \ln(x)$ 
 $+ \eta_x + \frac{(\eta_y - \xi_x) y^2}{x^2} - \frac{2\eta y}{x^2} - \frac{\xi_y y^4}{x^4} - \xi \left( -\frac{2y}{x^2} + \frac{1}{x} - \frac{2y^2}{x^3} \right)$ 

```

From the first terms we see that  $\xi_y = 0$ ,  $\xi_x = \eta_y$ . Let us use this information and simplify the determining system.

```

> eval(% , {diff(xi(x,y), y) = 0,
>           diff(eta(x,y), y) = diff(xi(x,y), x)});
 $(-\xi \left( \frac{2y}{x^2} + \frac{2}{x} \right) + \frac{2\eta}{x}) \ln(x) + \eta_x - \frac{2\eta y}{x^2} - \xi \left( -\frac{2y}{x^2} + \frac{1}{x} - \frac{2y^2}{x^3} \right)$ 
> collect(% , ln);

 $(-\xi \left( \frac{2y}{x^2} + \frac{2}{x} \right) + \frac{2\eta}{x}) \ln(x) + \eta_x - \frac{2\eta y}{x^2} - \xi \left( -\frac{2y}{x^2} + \frac{1}{x} - \frac{2y^2}{x^3} \right)$ 

```

Look at the logarithmic term.

```

> normal(coeff(% , ln(x)));
 $-\frac{2(\xi x + \xi y - \eta x)}{x^2}$ 

```

So we also have

$$(x + y) \xi = \eta x.$$

Because  $\xi$  does not depend on  $y$ , we have  $\eta_y = \xi/x$ . Since  $\eta_y = \xi_x$ , we get the differential equation  $\xi_x = \xi/x$ , which is easily solved as  $\xi = c x$  (for constant  $c$ ). Therefore, a possible solution of the determining system is.

$$\xi = x, \quad \eta = x + y.$$

Let us verify that it is indeed a solution.

```
> eval(determining_system, {xi(x,y)=x, eta(x,y)=x+y});
```

$$1 - x \left( \frac{2(y - x \ln(x))(-\ln(x) - 1)}{x^2} - \frac{2(y - x \ln(x))^2}{x^3} + \frac{1}{x} \right) \\ - \frac{2(x + y)(y - x \ln(x))}{x^2}$$

```
> normal(%);
```

$$0$$

Above, we solved the determining system ourselves in step. Maple provides the procedure **symgen** in the **DEtools** package to solve the determining system by heuristic methods. In our example, it will find the above infinitesimals.

```
> infinitesimals := symgen(ODE);
```

$$\text{infinitesimals} := [\xi = x, \eta = x + y]$$

The procedure **symtest** is for testing a guess or verifying an answer: if the result equals 0, the test is successful.

```
> symtest(infinitesimals, ODE);
```

$$0$$

There can be more than one kind of solution of the determining system. In the following command we explicitly require Maple to use the procedure **pdsolve** of the **PDEtools** package. This leads here to different sets of symmetry generators. We added an extra evaluation to get the final answer.

```
> symgen(ODE, way=pdsolve): %;
```

$$[\xi = 0, \eta = \frac{1}{5} \frac{(-2y + 2x \ln(x) + x)^2}{x} - x], \\ [\xi = 0, \eta = -\frac{1}{5} (\sqrt{5} \ln(x) - 2 \operatorname{arctanh}(\frac{1}{5} \frac{(-2y + 2x \ln(x) + x) \sqrt{5}}{x})) \sqrt{5} \\ (\frac{1}{5} \frac{(-2y + 2x \ln(x) + x)^2}{x^2} - 1)x] \\ > symtest(%[1], ODE);$$

$$0$$

Knowing the infinitesimals  $\xi$  and  $\eta$ , there are, as we have mentioned before, two ways to proceed: using an integrating factor or using canonical variables. Let us first look at the method of an integrating factor. The ordinary differential equation has an integrating factor

$$\mu = \frac{1}{\eta - \xi \Phi},$$

where

$$\Phi = \frac{(y - x \ln x)^2}{x^2} + \ln x.$$

Let us verify with Maple that the differential equation obtained from the original one by multiplying with the integrating factor is indeed of exact type.

```
> mu := eval(1/(eta-xi*Phi),
> {xi=x, eta=x+y(x), Phi=rhs(ODE)}) ;

$$\mu := \frac{1}{x + y - x \left( \frac{(y - x \ln(x))^2}{x^2} + \ln(x) \right)}$$

> normal(mu) ;

$$-\frac{x}{y^2 - 2xy \ln(x) + x^2 \ln(x)^2 + \ln(x)x^2 - x^2 - yx}$$

```

The procedure **odeadvisor** confirms that the differential equation obtained from the original one with the integrating factor is indeed of exact type.

```
> odeadvisor(mu*ODE) ;
[[_1st_order, _with_linear_symmetries], _exact, _Riccati]
```

The procedure that computes the integrating factor immediately from the ODE is called **intfactor**.

```
> intfactor(ODE) ;

$$\frac{x}{y^2 - 2xy \ln(x) + x^2 \ln(x)^2 + \ln(x)x^2 - x^2 - yx}$$

```

The method of canonical variables works for our example as follows. For the infinitesimal generator

$$U = \xi \frac{\partial}{\partial x} + \eta \frac{\partial}{\partial y}$$

we search solutions  $u(x, y)$ ,  $v(x, y)$  of the equations

$$Uu = 0, \quad Uv = 1.$$

```
> Uu := xi*diff(u(x,y),x) + eta*diff(u(x,y),y);
Uu := \xi u_x + \eta u_y
```

```
> PDE_u := eval(Uu=0, {xi=x, eta=x+y});  
PDE_u := x u_x + (x + y) u_y = 0
```

We use the procedure Maple procedure **pdsolve** to solve this partial differential equation.

```
> pdsolve(PDE_u);  
u(x, y) = _F1(-y + x ln(x))  
x
```

Here,  $\_F1$  is an arbitrary function. However, we are looking for a simple solution and therefore we choose the identity mapping for  $\_F1$ .

```
> eval(%, _F1=(z->z));  
u(x, y) = -y + x ln(x)  
x  
> expand(%);  
u(x, y) = y/x - ln(x)  
> Uv := xi*diff(v(x,y),x) + eta*diff(v(x,y),y);  
Uv :=  $\xi v_x + \eta v_y$   
> PDE_v := eval(Uv=1, {xi=x, eta=x+y});  
PDE_v := x v_x + (x + y) v_y = 1  
> pdsolve(PDE_v);  
v(x, y) = ln(x) + _F1(-y + x ln(x))  
x
```

Again we choose a solution as simple as possible.

```
> eval(%, _F1=0);  
v(x, y) = ln(x)
```

So, we have obtained the canonical variables

$$u(x, y) = \frac{y - x \ln(x)}{x}, \quad v(x, y) = \ln(x).$$

Maple provides the procedure **canoni** in the **DEtools** package to compute the canonical variables  $u$  and  $v$  directly from the infinitesimals  $\xi$  and  $\eta$ .

```
> u_v_vars := canoni(infinitesimals, y(x), v(u));  
u_v_vars := \{u = -\frac{y + x \ln(x)}{x}, v(u) = \ln(x)\}
```

In order to reduce the ODE to a quadrature in the canonical variables, we first express the old variables in terms of the new ones.

```
> x_y_vars := solve(%, {x,y(x)});
```

$$x\_y\_vars := \{x = e^{v(u)}, y = u e^{v(u)} + e^{v(u)} v(u)\}$$

Next, we use the procedure **dchange** in the **PDEtools** package to carry out the coordinate transformation.

```
> with(PDEtools, dchange):
> dchange(x_y_vars, ODE, [u,v(u)]);


$$\frac{e^{v(u)} + u v_u e^{v(u)} + v_u e^{v(u)} v(u) + v_u e^{v(u)}}{v_u e^{v(u)}} =$$


$$\frac{(u e^{v(u)} + e^{v(u)} v(u) - e^{v(u)} \ln(e^{v(u)}))^2}{(e^{v(u)})^2} + \ln(e^{v(u)})$$


> simplify(% , symbolic);


$$\frac{1 + u v_u + v_u v(u) + v_u}{v_u} = u^2 + v(u)$$


> isolate(% , diff(v(u),u));


$$v_u = \frac{1}{-u - 1 + u^2}$$

```

So, our original problem is reduced to solving the above quadrature; a much simpler task to do.

```
> sol_v := map(integrate, %, u);

sol_v := v(u) = -\frac{2}{5} \sqrt{5} \operatorname{arctanh}\left(\frac{(2u-1)\sqrt{5}}{5}\right)
```

We write the answer down in the original coordinates and solve for  $y(x)$ .

```
> sol_y := isolate( dchange(u_v_vars, sol_v), y(x) );

sol_y := y = \frac{1}{2} \left(-\tanh\left(\frac{1}{2} \sqrt{5} \ln(x)\right) \sqrt{5} + 1\right) x + x \ln(x)
```

As we have seen in the above example, the most important step in solving the ODE by Lie point symmetries is the calculation of the infinitesimals  $\xi(x, y)$  and  $\eta(x, y)$ . For first-order ODEs of the form  $y' = \Phi(x, y(x))$ , the procedure **symgen** tries to find the infinitesimals via heuristic methods. The following algorithms, which involve certain functional forms for  $[\xi, \eta]$ , have been implemented in Maple.

$[\xi = f(x), \eta = 0],$	$[\xi = g(y), \eta = 0]$
$[\xi = 0, \eta = f(x)],$	$[\xi = 0, \eta = g(y)]$
$[\xi = f(x)g(y), \eta = 0],$	$[\xi = f(x) + g(y), \eta = 0]$
$[\xi = 0, \eta = f(x)g(y)],$	$[\xi = 0, \eta = f(x) + g(y)]$
$[\xi = ax + by + c, \eta = dx + ey + f],$	

for arbitrary functions  $f$  and  $g$ , and arbitrary constants  $a, b, \dots, f$ . Note that more than one algorithm may lead to a symmetry generator: if you insist on application of a particular listed scheme, you can specify this in the **symgen** procedure as an extra argument of the form

$$\text{way} = \text{algorithm}$$

where *algorithm* = **abaco1**, 2, 3, 4, 5, 6, 7, **abaco2**, **exp\_sym**, **formal**, **mu**, **patterns**, and **pdsolve**. The first algorithm, **abaco1**, tries the eight schemes listed above in which one of the infinitesimals equals 0. The **patterns** algorithm determines both the existence and the symmetries themselves when they are of the form  $\xi = f(x), \eta = p(x)y + q(x)$ , for some functions  $f, p, q$ . See the on-line help system for information about the other possible choices. A complete list of available methods of **symgen** can be obtained by entering the command `print('symgen/methods');`

If Maple fails to solve an ODE by Lie point symmetries or produces too complicated formulas, then it does so in most cases because it can only compute the infinitesimals on the basis of the implemented schemes. By the optional argument

$$\text{HINT} = \text{ansatz}$$

you can make a suggestion to Maple about the possible functional form of the infinitesimals. For example, let us consider the first-order differential equation of Riccati type

$$y' + y^2 - y \sinh(2x) + \cos(2x) = 2$$

that we solved in the previous section. The reason that Maple could solve it all by itself is that it was able to find a Lie point symmetry. But if for some reason we have the idea that the infinitesimals are of the form

$$\xi = 0, \quad \eta = f(x)(y \cosh x - \sinh x)^2$$

then Maple can use this hint and determine the exact form of the infinitesimals. This leads to a simple formula.

```
> restart: with(DEtools): alias(xi=_xi, eta=_eta):
> diff(y(x),x) + y(x)^2 - y(x)*sinh(2*x) + cosh(2*x) = 2;
          ( $\frac{d}{dx} y(x)) + y(x)^2 - y(x) \sinh(2x) + \cosh(2x) = 2$ 
> symgen(%); # no hint

$$\left[ \xi = 0, \eta = \frac{(-y \cosh(x) - \sinh(x))^2}{e^{(\cosh(x)^2)}}, \dots \right]$$

```

We have omitted the second complicated result.

```
> symgen(%%, HINT=[0,f(x)*(cosh(x)*y-sinh(x))^2]);

$$[\xi = 0, \eta = e^{(-1/2 \cosh(2x))} (y \cosh(x) - \sinh(x))^2]$$

```

You can also add such hints in the **dsolve** command and then Maple will use them when it tries Lie symmetry methods to solve an ordinary differential equation.

## Second-Order and Higher Order ODEs

Henceforth, we shall look at how Lie point symmetries help for second-order and higher-order equations. As we shall see the trick of the trade is to use symmetries to introduce new variables that reduce the order of the problem. If we have a differential equation of order  $n$  and  $n$  independent Lie point symmetries that generate a solvable Lie algebra, then we can reduce the problem to order 0, i.e., basically solve the original differential equation.

The following concrete example, taken from [137], will illustrate some main methods. The ordinary differential equation to start with is

$$y'' = \frac{y'}{y^2} - \frac{1}{xy}.$$

```
> restart: with(DEtools):
> PDEtools[declare](y(x), prime=x, (_xi, _eta)(x,y), quiet)
> alias('y'='y1, xi=_xi, eta=_eta):
> ODE := diff(y(x),x$2) = diff(y(x),x)/y(x)^2-1/(x*y(x));
```

$$ODE := y'' = \frac{y'}{y^2} - \frac{1}{xy}$$

The procedure **odepde** computes the determining system.

```
> odepde(ODE, [_xi=xi(x,y), _eta=eta(x,y)]);

$$\begin{aligned} & -\eta \left( -\frac{2y'}{y^3} + \frac{1}{xy^2} \right) + (-3y'\xi_y - 2\xi_x + \eta_y) \left( \frac{y'}{y^2} - \frac{1}{xy} \right) - \frac{\xi}{x^2y} \\ & + \frac{-y'\eta_y + y'^2\xi_y + y'\xi_x - \eta_x}{y^2} + \eta_{x,x} + \eta_{y,y}y'^2 - \xi_{y,y}y'^3 - 2\xi_{x,y}y'^2 \\ & + 2\eta_{x,y}y' - y'\xi_{x,x} \end{aligned}$$

```

Here, we have added the requested form of the infinitesimals because otherwise **odepde** will compute the determining system of contact transformations. The variable `_y1` in the result of Maple has been aliased to the first derivative of  $y$ . Let us collect terms with respect to this variable.

```
> collect(%,'y');

$$\begin{aligned} & -\xi_{y,y}y'^3 + \left( -\frac{2\xi_y}{y^2} - 2\xi_{x,y} + \eta_{y,y} \right) y'^2 \\ & + \left( \frac{-2\xi_x + \eta_y}{y^2} + \frac{3\xi_y}{xy} + \frac{-\eta_y + \xi_x}{y^2} + \frac{2\eta}{y^3} - \xi_{x,x} + 2\eta_{x,y} \right) y' - \frac{\eta}{xy^2} \end{aligned}$$

```

$$-\frac{\xi}{x^2 y} - \frac{-2\xi_x + \eta_y}{xy} + \eta_{x,x} - \frac{\eta_x}{y^2}$$

It follows among other things that  $\xi_{yy} = 0$  and  $\eta_{xx} = 2\xi_{xy} + 2\xi_y/y^2$ . With the procedure **symgen** Maple finds solutions of the determining system.

```
> infinitesimals := symgen(ODE);
```

$$\text{infinitesimals} := [\xi = 2x, \eta = y], [\xi = x^2, \eta = xy]$$

Let us verify the two solutions.

```
> map(symtest,{%}, ODE);
```

$$\{0\}$$

Knowing these infinitesimals, we try to integrate the ODE by a group-theoretical quadrature method. This can be done in various ways; the following five methods have been implemented in Maple and can be requested by the optional argument that specifies the integration scheme. In case of no specification, Maple tries the schemes in the order as they are written down.

- gon** Using one pair of infinitesimals to build a first integral, then introduce it as ‘new’ variable and look for a first-order ODE with known symmetry and formally equivalent to the solution.
- can2** Using two pairs of infinitesimals at once to try to integrate the second-order ODE in one step.
- gon2** Using two pairs of infinitesimals at once to build two first integrals, then using these first integrals to obtain an explicit/implicit solution to the ODE.
- can** Determining canonical coordinates, say  $u, v(u)$ , of the associated Lie group on the basis of one pair of infinitesimals.
- dif** Determining differential invariants of order 0 and 1 of the symmetry group and using them to reduce the ODE to a quadrature.

We shall only illustrate some integration methods for the given example. First we proceed in the same way as for first-order ODEs: use of a symmetry to introduce canonical variables.

```
> Uu := xi*diff(u(x,y),x) + eta*diff(u(x,y),y);
```

$$Uu := \xi u_x + \eta u_y$$

```
> PDE_u := eval(Uu=0, {xi=x^2, eta=x*y});
```

$$PDE\_u := x^2 u_x + xy u_y = 0$$

```
> pdsolve(PDE_u);
```

$$u(x, y) = \text{_F1}\left(\frac{y}{x}\right)$$

```

> eval(%, _F1=(z->z));

$$u(x, y) = \frac{y}{x}$$

> Uv := xi*diff(v(x,y),x) + eta*diff(v(x,y),y);

$$Uv := \xi v_x + \eta v_y$$

> PDE_v := eval(Uv=1, {xi=x^2, eta=x*y});

$$PDE_v := x^2 v_x + x y v_y = 1$$

> pdsolve(PDE_v);

$$v(x, y) = -\frac{1}{x} + _F1\left(\frac{y}{x}\right)$$

> eval(%, _F1=0);

$$v(x, y) = -\frac{1}{x}$$


```

We could have found this immediately in the following way.

```

> u_v_vars := canoni(infinitesimals[2], y(x), v(u));

$$u\_v\_vars := \{u = \frac{y}{x}, v(u) = -\frac{1}{x}\}$$


```

We solve for  $x, y$  to find the inverse transformation

```

> x_y_vars := solve(%, {x,y(x)});

$$x\_y\_vars := \{x = -\frac{1}{v(u)}, y = -\frac{u}{v(u)}\}$$


```

In the new coordinates, the ODE is as follows:

```

> PDEtools[dchange](x_y_vars, ODE, [u,v(u)]):
> normal((lhs-rhs)(%)) = 0;

$$\frac{v(u)^3 (v_{u,u} u^2 + v_u^2)}{v_u^3 u^2} = 0$$


```

Thus, we get the following differential equation, which is of order 1 for  $\frac{dv}{du}$ .

```
> numer(lhs(%))/v(u)^3 = 0;
```

$$v_{u,u} u^2 + v_u^2 = 0$$

Let us write  $w(u)$  instead of  $\frac{dv}{du}$ .

```
> eval(%, diff(v(u),u)=w(u));
```

$$w_u u^2 + w(u)^2 = 0$$

It is a separable ODE that can easily be solved.

```
> odeadvisor(%);
```

[*\_separable*]

```
> dsolve(%, w(u));
```

$$w(u) = \frac{u}{-1 + _C1 u}$$

This leads to the following first-order ODE for  $v(u)$

```
> eval(% , w(u)=diff(v(u),u));
```

$$v_u = \frac{u}{-1 + _C1 u}$$

Its solution is as follows:

```
> dsolve(%, v(u));
```

$$v(u) = \frac{u}{_C1} + \frac{\ln(-1 + _C1 u)}{-_C1^2} + _C2$$

In terms of the original variables:

```
> eval(%, u_v_vars);
```

$$-\frac{1}{x} = \frac{y}{_C1 x} + \frac{\ln(-1 + \frac{-C1 y}{x})}{-_C1^2} + _C2$$

In explicit form:

```
> solve(%, y(x));
```

$$\frac{x (\text{LambertW}(e^{(-\frac{x + -C1^2 + -C2 x - C1^2}{x})) + 1))}{-_C1}$$

This method is actually applied in the **dsolve** procedure when the argument **can** (of canonical variables, using one pair of infinitesimals) is added in the **dsolve** command.

```
> ODE;
```

$$y'' = \frac{y'}{y^2} - \frac{1}{x y}$$

```
> dsolve(ODE, y(x), can);
```

$$y = \frac{x \left( \text{LambertW} \left( \frac{e^{(-1)}}{(e^{(\frac{-C1^2}{x})} e^{(-C2 - C1^2)})} \right) + 1 \right)}{-_C1}$$

```
> simplify(%);
```

$$y = \frac{x (\text{LambertW}(e^{(-\frac{x + -C1^2 + -C2 x - C1^2}{x})) + 1))}{-_C1}$$

The method of differential invariants works essentially as follows. For a second-order ODE that is invariant under the Lie group with infinitesimal

generator

$$U = \xi(x, y) \frac{\partial}{\partial x} + \eta(x, y) \frac{\partial}{\partial y}$$

determine functions  $u(x, y)$  and  $v(x, y, y')$  such that

$$Uu = 0, \quad U^{(1)}v = 0,$$

and solve for  $y$  and  $y'$ . Substitution in the original ODE leads to an ODE in  $u$  and  $v(u)$  that is of order one less than the original equation. Let us see how it works for our example.

```
> ODE;
```

$$y'' = \frac{y'}{y^2} - \frac{1}{xy}$$

Again, we first determine infinitesimal generators

```
> infinitesimals := symgen(ODE);
```

$$\text{infinitesimals} := [\xi = 2x, \eta = y], [\xi = x^2, \eta = xy]$$

and use the second generator to reduce the order of the problem. The computation of a suitable  $u$  is as before.

```
> Uu := xi*diff(u(x,y),x) + eta*diff(u(x,y),y);
```

$$Uu := \xi u_x + \eta u_y$$

```
> PDE_u := eval(Uu=0, infinitesimals[2]);
```

$$PDE\_u := x^2 u_x + xy u_y = 0$$

```
> pdsolve(PDE_u);
```

$$u(x, y) = \text{_F1}\left(\frac{y}{x}\right)$$

```
> eval(% , _F1=(z->z));
```

$$u(x, y) = \frac{y}{x}$$

Next, we compute the first prolongation  $U^{(1)}$  by the **eta\_k** procedure.

```
> eta1 := eta_k(infinitesimals[2], 1, y(x));
```

$$\eta1 := y - y' x$$

The partial differential equation  $U^{(1)}v = 0$  is as follows

```
> U1v := xi*diff(v(x,y,'y''), x)
>      + eta*diff(v(x,y,'y''), y)
>      + eta1*diff(v(x,y,'y''), 'y'');
> PDE_v := eval(U1v=0, infinitesimals[2]);
```

$$PDE\_v := x^2 v_x + xy v_y + (y - y' x) v_{y'} = 0$$

and its solutions are

```
> pdsolve(PDE_v);
```

$$v(x, y, y') = -F1\left(\frac{y}{x}, y' x - y\right)$$

for arbitrary function  $_F1$ . We choose a simple solution.

```
> eval(% , _F1=((a,b)->b));
```

$$v(x, y, y') = y' x - y$$

So, the new variables can be as follows:

```
> u_v_vars := {u = y(x)/x,
   > v(u) = diff(y(x),x)*x - y(x)};
```

$$u\_v\_vars := \{u = \frac{y}{x}, v(u) = y' x - y\}$$

Express  $y$  and  $y'$  in these variables.

```
> eval(% , {diff(y(x),x)='y' , y(x)=y});
```

$$\{u = \frac{y}{x}, v(u) = y' x - y\}$$

```
> y_y1_vars := eval(solve(% , {y,'y'}),
   > {y=y(x) , 'y'='diff(y(x),x)} );
```

$$y\_y1\_vars := \{y' = \frac{v(u) + ux}{x}, y = ux\}$$

Next, we express  $y''$  in terms of the new variables and  $v'$ . This is a bit cumbersome because of substitutions and back-substitutions.

```
> diff(y(x),x) =
> eval(eval(diff(y(x),x) , y_y1_vars) , u=y(x)/x);
```

$$y' = \frac{v\left(\frac{y}{x}\right) + y}{x}$$

```
> eval(diff(y(x),x$2) , %);
```

$$\frac{D(v)\left(\frac{y}{x}\right)\left(-\frac{y}{x^2} + \frac{y'}{x}\right) + y'}{x} - \frac{v\left(\frac{y}{x}\right) + y}{x^2}$$

```
> eval(%, y(x)/x=u);
```

$$\frac{D(v)(u)\left(-\frac{y}{x^2} + \frac{y'}{x}\right) + y'}{x} - \frac{v(u) + y}{x^2}$$

```
> eval(%, y_y1_vars );
```

$$\frac{D(v)(u)\left(-\frac{u}{x} + \frac{v(u) + ux}{x^2}\right) + \frac{v(u) + ux}{x}}{x} - \frac{v(u) + ux}{x^2}$$

```
> normal(%);
```

$$\frac{D(v)(u)v(u)}{x^3}$$

Finally we get the expression for  $y''$ .

```
> diff(y(x), x$2) = convert(%, diff);
```

$$y'' = \frac{v_u v(u)}{x^3}$$

Recall the original ODE.

```
> ODE;
```

$$y'' = \frac{y'}{y^2} - \frac{1}{x y}$$

In the new variables we get:

```
> eval(%, {%%} union y_y1_vars );
```

$$\frac{v_u v(u)}{x^3} = \frac{v(u) + u x}{x^3 u^2} - \frac{1}{x^2 u}$$

```
> normal((lhs-rhs)(%));
```

$$\frac{v(u) (v_u u^2 - 1)}{x^3 u^2}$$

```
> ode := numer(%) / v(u) = 0;
```

$$ode := v_u u^2 - 1 = 0$$

Indeed a first-order ODE in  $v(u)$  that is easily solved.

```
> dsolve(ode, v(u));
```

$$v(u) = -\frac{1}{u} + _C1$$

Going back to the original coordinates, we get another first-order ODE to solve.

```
> eval(%, u_v_vars);
```

$$y' x - y = -\frac{x}{y} + _C1$$

```
> simplify(dsolve(%, y(x)));
```

$$y = \frac{x (\text{LambertW}(e^{(\frac{-x - _C1^2 + _C2 x - _C1^2}{x})}) + 1)}{-_C1}$$

A lot of work to find the solution, but luckily this is completely automated in the **dsolve** command when you add the option **dif** (of differential invariants).

```
> dsolve(ODE, y(x), dif);
```

$$\begin{aligned} y = & \left( -(\text{LambertW}\left(e^{(-\frac{-_C1^2 + _C2 x - _C1^2}{x})}\right)) - \frac{-_C1^2 + \ln(x)x + _C2 x - x}{x} \right) x \\ & + _C1^2 - \ln(x)x - _C2 x / _C1 \\ > & \text{simplify}(%); \end{aligned}$$

$$y = -\frac{x (\text{LambertW}(-e^{(\frac{-C1^2 + C2 x - x}{x})}) + 1)}{-C1}$$

You may have wondered why we only used one pair of infinitesimal generators, whereas the ODE has at least two pairs. The only reason is that for most ODEs finding one pair of symmetries is already difficult enough and the only possible achievement. However, once we have found two pairs generating a two-dimensional Lie algebra, say  $X_1 = [\xi_1, \eta_1]$  and  $X_2 = [\xi_2, \eta_2]$ , we can apply Lie's classification of all possibilities. The fact is that there are only four possibilities; they are distinguished by the fact

- whether the operators commute or not (in which case we may assume  $[X_1, X_2] = 0$  or  $[X_1, X_2] = X_1$ ).
- whether the expression  $X_1 \vee X_2 = \xi_1 \eta_2 - \xi_2 \eta_1$  is equal to zero or not.

Table 17.4 classifies the second-order ODEs that admit a two-dimensional Lie algebra generated by infinitesimal generators  $X_1, X_2$ . The table also lists the generators in canonical variables and the type of differential equation in these canonical coordinates.

Lie algebra structure	Canonical basis	Equation
$[X_1, X_2] = 0, X_1 \vee X_2 \neq 0$	$X_1 = \frac{\partial}{\partial x}, X_2 = \frac{\partial}{\partial y}$	$y'' = f(y')$
$[X_1, X_2] = 0, X_1 \vee X_2 = 0$	$X_1 = \frac{\partial}{\partial y}, X_2 = x \frac{\partial}{\partial y}$	$y'' = f(x)$
$[X_1, X_2] = X_1, X_1 \vee X_2 \neq 0$	$X_1 = \frac{\partial}{\partial y}, X_2 = x \frac{\partial}{\partial x} + y \frac{\partial}{\partial y}$	$y'' = f(y')$
$[X_1, X_2] = X_1, X_1 \vee X_2 = 0$	$X_1 = \frac{1}{2} \frac{\partial}{\partial y}, X_2 = y \frac{\partial}{\partial y}$	$y'' = f(x)y'$

Table 17.4. Lie's classification of second-order ODEs that admit a 2-dimensional symmetry algebra.

There exist algorithms to compute the canonical variables once the classification type of an ODE is known. They have been implemented in Maple and are used in the **dsolve** command when the option **can2** is added. Let us apply this method to our example. We start by computing  $[X_1, X_2]$  and  $X_1 \vee X_2$ .

```
> ODE;

$$y'' = \frac{y'}{y^2} - \frac{1}{x y}$$

> infinitesimals := symgen(ODE);
```

*infinitesimals* := [ $\xi = 2x, \eta = y$ ], [ $\xi = x^2, \eta = xy$ ]

Define the differential algebra  $\mathbb{Q}[x, y, \frac{\partial}{\partial x}, \frac{\partial}{\partial y}]$ .

```
> _Envdiffopdomain := [Dx, Dy, x, y]:
```

We show that  $[X_1, X_2] \neq 0$  and  $X_1 \vee X_2 \neq 0$ .

```

> X1 := eval(xi*Dx+eta*Dy, infinitesimals[1]);

$$X1 := 2x \, Dx + y \, Dy$$

> X2 := eval(xi*Dx+eta*Dy, infinitesimals[2]);

$$X2 := x^2 \, Dx + xy \, Dy$$


```

The commutator  $[X_1, X_2]$  is

```

> mult(X1,X2) - mult(X2,X1);

$$x^2 y$$


```

$X_1 \vee X_2$  equals

```

> LinearAlgebra[Determinant](<<2*x|y>, <x^2|x*y>>);

$$x^2 y$$


```

So, we are in case 3 of the above classification table. Using canonical variables, Maple can solve the ODE.

```

> dsolve(ODE, y(x), can2);

$$\begin{aligned} & -\frac{y}{x^2 \sqrt{\frac{y^2}{x^2}}} + 2 \sqrt{\frac{-C1 y^2}{x^2}} + 2 \cdot C1 \ln\left(\sqrt{\frac{-C1 y^2}{x^2}} - 2 \cdot C1\right) \\ & - 2 \cdot C1 \ln\left(2 \cdot C1 + \sqrt{\frac{-C1 y^2}{x^2}}\right) + 2 \cdot C1 \ln\left(4 \cdot C1 - \frac{y^2}{x^2}\right) - \cdot C2 = 0 \\ > \text{simplify}(\%, \text{symbolic}); \\ & (-1 + 2y\sqrt{-C1} + 2 \cdot C1 x \ln(y\sqrt{-C1} - 2 \cdot C1 x) - 4 \cdot C1 x \ln(x) \\ & - 2 \cdot C1 x \ln(2 \cdot C1 x + y\sqrt{-C1}) + 2 \cdot C1 x \ln(4 \cdot C1 x^2 - y^2) \\ & - \cdot C2 x)/x = 0 \end{aligned}$$


```

As you see an implicit solution that looks different from the solution that we found before.

## Summary of DEtools Commands

A compact summary of the commands of the **DEtools** package that are related to Lie point symmetry methods and were discussed in the above examples is listed in Table 17.5. Basic knowledge about these commands and how they work will enable you to solve more ordinary differential equations than by just applying the **dsolve** command and hope for the best. For example, you can give hints and/or select a particular algorithm that is suitable for your problem.

Command	Purpose
<b>canoni</b>	look for a pair of canonical coordinates
<b>eta_k</b>	returns the k-extended infinitesimal
<b>intfactor</b>	look for an integrating factor of 1st-order ODE
<b>odeadvisor</b>	classify ODEs and pop up help-pages
<b>odepde</b>	return the PDE for the infinitesimals
<b>symgen</b>	look for a pair of infinitesimals
<b>symtest</b>	test a given symmetry

Table 17.5. Main commands in **DETools** package for Lie symmetry approach.

## 17.4 Taylor Series Method

When an analytical solution for an ordinary differential equation cannot be found, there is still the possibility of using Maple to find a Taylor series approximation. We shall apply this method to the (large oscillation) pendulum, shown below and described by the differential equation

$$l \theta'' = -g \sin \theta,$$

where  $l$  is the pendulum length,  $g$  is the gravitational acceleration, and  $\theta$  is the angle between the rod and the vertical.

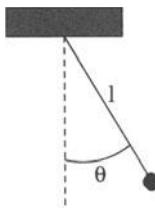


Figure 17.3. Pendulum.

```
> ODE := l*diff(theta(t), [t$2]) = -g*sin(theta(t));
```

$$ODE := l \left( \frac{d^2}{dt^2} \theta(t) \right) = -g \sin(\theta(t))$$

```
> initvals := theta(0)=0, D(theta)(0)=v[0]/l:
```

```
> IVP := ODE, initvals: Order := 9:
```

```
> solution := dsolve(IPV, theta(t), type=series);
```

$$\begin{aligned} solution := \theta(t) = & \frac{v_0}{l} t - \frac{1}{6} \frac{g v_0}{l^2} t^3 + \frac{1}{120} \frac{g v_0 (g l + v_0^2)}{l^4} t^5 - \\ & \frac{1}{5040} \frac{g v_0 (g^2 l^2 + 11 g l v_0^2 + v_0^4)}{l^6} t^7 + O(t^9) \end{aligned}$$

Below, we shall rewrite this solution in dimensionless variables. First, we check dimensions.

```
> eval(% , {v[0]=L/T, g=L/T^2, l=L});
```

$$\theta(t) = \frac{1}{T} t - \frac{1}{6T^3} t^3 + \frac{1}{60T^5} t^5 - \frac{13}{5040T^7} t^7 + O(t^9)$$

Next, we determine dimensionless parameters  $a$  and  $b$ .

```
> nondimensional := v[0]^ev[0] * g^eg * l^el * t^et;
      nondimensional := v0^ev0 g^eg l^el t^et
> simplify(eval(nondimensional,
> {v[0]=L/T, g=L/T^2, l=L, t=T}), assume=positive);
      L^(ev0+eg+el) T^{(-ev0-2 eg+et)}
> isolve({seq(op(2,i)=0, i=%)} );
      {ev0 = -2 Z1 - Z2, eg = -Z1 + Z2, el = -Z1, et = -Z2}
> assign(%):
> {a=subs(_Z1=-1, _Z2=1, nondimensional),
> b=subs(_Z1=-1, _Z2=0, nondimensional)};
      {b =  $\frac{v_0^2}{gl}$ , a =  $\frac{v_0 t}{l}$ }
```

We express the gravitation  $g$  and length  $l$  of the rod in terms of the dimensionless variables, and we substitute these expressions into the solution of the ODE.

```
> solve(% , {g,l});
      {g =  $\frac{v_0 a}{b t}$ , l =  $\frac{v_0 t}{a}$ }
> subs(% , convert(rhs(solution), 'polynom')):
> map(collect, %);
      a - 1/6  $\frac{a^3}{b}$  +  $\frac{1}{120} \frac{a^5 (1+b)}{b^2}$  -  $\frac{1}{5040} \frac{a^7 (1+11b+b^2)}{b^3}$ 
```

Compare this with the series expansion of  $\sin t$ .

```
> series(sin(t), t, 9);
      t -  $\frac{1}{6} t^3 + \frac{1}{120} t^5 - \frac{1}{5040} t^7 + O(t^9)$ 
```

## 17.5 Power Series Method

Naturally, in a modern computer algebra system like Maple, the method of power series to solve differential equation should not be absent. This facility resides in the **powseries** package and in the **LSODE** package. However, the first package only works for linear differential equations with polynomial coefficients. The **LSODE** also works for linear ODEs with rational, hypergeometric or  $m$ -sparse,  $m$ -hypergeometric coefficients over the rational field.

Our first example will be a differential equation of Bessel type.

$$xy'' + y' + 4x^2y = 0$$

Maple can find the exact solution.

```
> ODE := x*diff(y(x), [x$2]) + diff(y(x), x) +
>        4*x^2*y(x) = 0;
ODE := x ( $\frac{d^2}{dx^2} y(x)$ ) + ( $\frac{d}{dx} y(x)$ ) + 4  $x^2 y(x) = 0$ 
> dsolve(ODE, y(x));
y(x) =  $-C1 \text{BesselJ}(0, \frac{4x^{(3/2)}}{3}) + -C2 \text{BesselY}(0, \frac{4x^{(3/2)}}{3})$ 
```

Below, we shall compute the power series solution for the initial values  $y(0) = 1$ ,  $y'(0) = 0$ .

```
> initvals := y(0)=1, D(y)(0)=0:
> with(powseries): # load the power series package
> solution := powsolve({ODE, initvals});
solution := proc(powparm) ... end proc
> tpsform(solution, x, 15); # truncated powseries
1 -  $\frac{4}{9}x^3 + \frac{4}{81}x^6 - \frac{16}{6561}x^9 + \frac{4}{59049}x^{12} + O(x^{15})$ 
```

Maple can give you the recurrence relation of the coefficients.

```
> solution(_k);
```

$$-\frac{4a_{-k-3}}{k^2}$$

Recall that names starting with an underscore are used by Maple itself and interpret the last result as the relation

$$a_k = -4 \frac{a_{k-3}}{k^2}.$$

The above formula is more easily found via the **Slode** package for computing formal power series solutions of linear ordinary differential equations.

```
> with(Slode): # load library package
```

You can create a formal power series via the **FPseries** command:

```
> FPseries(ODE, y(x), a(n));
```

$$\text{FPSstruct}(-C_0 + (\sum_{n=3}^{\infty} a(n)x^n), 4a(n-3) + n^2a(n))$$

It has created a special Maple object that describes the formal power series by giving the general format and the recurrence relation of the coefficients. You can specify what name(s) the initial coefficient(s) will have and how

many terms are explicitly worked out. Below, we use  $c_0$  instead of the default  $_C_0$  and we explicitly show the terms up to and including order 12.

```
> FPseries(ODE, y(x), a(n), 0, c, 12);
```

$$\begin{aligned} \text{FPSstruct}\left(c_0 - \frac{4}{9}c_0x^3 + \frac{4}{81}c_0x^6 - \frac{16}{6561}c_0x^9 + \frac{4}{59049}c_0x^{12}\right. \\ \left.+ \left(\sum_{n=13}^{\infty} a(n)x^n\right), 4a(n-3) + n^2a(n)\right) \end{aligned}$$

Let us select the recurrence relation of the  $a_n$ 's.

```
> op(2,%)=0;
```

$$4a(n-3) + n^2a(n) = 0$$

```
> isolate(% , a(n));
```

$$a(n) = -\frac{4a(n-3)}{n^2}$$

A formal series is restricted to a formal Taylor series when you use the procedure **FTseries** (note the format of the solution below).

```
> FTseries(ODE, y(x), a(n));
```

$$\text{FPSstruct}\left(_C_0 + \left(\sum_{n=3}^{\infty} \frac{a(n)x^n}{n!}\right), (4n^2 - 12n + 8)a(n-3) + a(n)n\right)$$

With some terms worked out:

```
> FTseries(ODE, y(x), a(n), 0, c, 12);
```

$$\begin{aligned} \text{FPSstruct}\left(c_0 - \frac{4}{9}c_0x^3 + \frac{4}{81}c_0x^6 - \frac{16}{6561}c_0x^9 + \frac{4}{59049}c_0x^{12}\right. \\ \left.+ \left(\sum_{n=13}^{\infty} \frac{a(n)x^n}{n!}\right), (4n^2 - 12n + 8)a(n-3) + a(n)n\right) \end{aligned}$$

In this case, the recurrence relation of the  $a_n$ 's is as follows:

```
> op(2,%)=0;
```

$$(4n^2 - 12n + 8)a(n-3) + a(n)n = 0$$

```
> isolate(% , a(n));
```

$$a(n) = -\frac{(4n^2 - 12n + 8)a(n-3)}{n}$$

```
> factor(%);
```

$$a(n) = -4 \frac{(n-1)(n-2)a(n-3)}{n}$$

In case of homogenous linear ordinary differential equations over the rational field, you can use the **dsolve** procedure as a user frontend for some functions in the **Slode** package. For example, instead of using the procedure **polynomial\_series** you may specify the options **type=formal\_series** and **coeffs=polynomial** to compute a formal Taylor series. An example:

```
> ODE := (x-1)^2*diff(y(x),[x$2]) +
>        4*(x-1)*diff(y(x),x)+2*y(x) = 0;
ODE := (x - 1)^2 ( $\frac{d^2}{dx^2}$  y(x)) + 4 (x - 1) ( $\frac{d}{dx}$  y(x)) + 2 y(x) = 0
> dsolve(ODE, y(x), type=formal_series, coeffs=polynomial);
```

$$y(x) = \sum_{n=0}^{\infty} (-C1 + n \cdot C2) x^n$$

Maple can actually determine a closed formula for this formal series.

```
> value(%);
y(x) = - $\frac{-C1 x - -C1 - -C2 x}{(x - 1)^2}$ 
> normal(subs(_C2=_C2+_C1, %));
y(x) =  $\frac{-C1 + -C2 x}{(x - 1)^2}$ 
```

This is in agreement with the exact solution of the ODE.

```
> dsolve(ODE, y(x));
y(x) =  $\frac{-C1}{(x - 1)^2} + \frac{-C2}{x - 1}$ 
> normal(subs(_C1=_C1+_C2, %));
y(x) =  $\frac{-C1 + -C2 x}{(x - 1)^2}$ 
```

Our next example is a classical one in quantum mechanics — the solutions of the one-dimensional harmonic oscillator. The Schrödinger equation can be given in dimensionless units as

$$\frac{d^2y(x)}{dx^2} + (\lambda - x^2) y(x) = 0.$$

See §17.8 for a derivation of this equation.

```
> alias(y=y(x), h=h(x)):
> ODE := diff(y,[x$2]) + (lambda-x^2)*y = 0;
```

$$ODE := (\frac{\partial^2}{\partial x^2} y) + (\lambda - x^2) y = 0$$

Asymptotic analysis suggests the substitution  $y(x) = h(x)e^{-x^2/2}$ . The differential equation for  $h$  is as follows:

```

> subs(y=exp(-x^2/2)*h, %):
> collect(%, exp(-x^2/2)) / exp(-x^2/2);

$$-h + x^2 h - 2x \left(\frac{\partial}{\partial x} h\right) + \left(\frac{\partial^2}{\partial x^2} h\right) + (\lambda - x^2) h = 0$$

> ODE := collect(%, [diff(h,[x$2]), diff(h,x), h]);

$$ODE := \left(\frac{\partial^2}{\partial x^2} h\right) - 2x \left(\frac{\partial}{\partial x} h\right) + (-1 + \lambda) h = 0$$


```

We solve this differential equation via the power series method.

```

> with(powseries): H := powsolve(ODE):
> tpsform(H,x,8); # a few terms

```

$$\begin{aligned}
& C_0 + C_1 x - \frac{(-1 + \lambda) C_0}{2} x^2 - \frac{(-3 + \lambda) C_1}{6} x^3 \\
& + \frac{(-5 + \lambda)(-1 + \lambda) C_0}{24} x^4 + \frac{(-7 + \lambda)(-3 + \lambda) C_1}{120} x^5 \\
& - \frac{(-9 + \lambda)(-5 + \lambda)(-1 + \lambda) C_0}{720} x^6 \\
& - \frac{(-11 + \lambda)(-7 + \lambda)(-3 + \lambda) C_1}{5040} x^7 + O(x^8) \\
> & \text{collect(convert(%, 'polynom'), [C0, C1])}; \\
& (1 - \frac{(-1 + \lambda) x^2}{2} + \frac{(-5 + \lambda)(-1 + \lambda) x^4}{24} \\
& - \frac{(-9 + \lambda)(-5 + \lambda)(-1 + \lambda) x^6}{720}) C_0 \\
& + (x - \frac{(-3 + \lambda) x^3}{6} + \frac{(-7 + \lambda)(-3 + \lambda) x^5}{120} \\
& - \frac{(-11 + \lambda)(-7 + \lambda)(-3 + \lambda) x^7}{5040}) C_1
\end{aligned}$$

Let us look at the recurrence relation for the coefficients of the series  $h$ .

```

> H(_k);

$$-\frac{(3 + \lambda - 2k) a_{-k-2}}{_k (-k - 1)}$$


```

This must be interpreted as

$$a_k = -\frac{(3 + \lambda - 2k) a_{k-2}}{k(k-1)},$$

or equivalently as

$$(k+1)(k+2)a_{k+2} = (2k+1-\lambda)a_k,$$

for all  $k$ . A finite series  $h$  is obtained if and only if  $\lambda = 2k+1$  for some integer  $k$ . This is the famous quantization of the energy levels of the harmonic oscillator. One example of a wave function:

```
> C0 := 1: C1 := 0: lambda := 9:
> tpsform(H,x,9): convert(%,'polynom');
```

$$1 - 4x^2 + \frac{4}{3}x^4$$

It is a multiple of the fourth Hermite polynomial.

```
> simplify(1/12*HermiteH(4,x));
```

$$1 - 4x^2 + \frac{4}{3}x^4$$

## 17.6 Numerical Solutions

You can solve initial value problems (IVP) and boundary value problems (BVP) numerically in Maple by adding the option `type=numeric`. The type of problem (BVP or IVP) is automatically detected by `dsolve`, and an appropriate method is used. By default, the system uses a Fehlberg fourth-fifth order Runge-Kutta method also known as algorithm RKF45 [79] for initial value problems. For boundary value problems, a finite difference technique (trapezoid) with Richardson extrapolation is used. But other numerical methods are available as well. First, we shall apply the Runge-Kutta method to the van der Pol equation

$$y'' - (1 - y^2)y' + y = 0,$$

with initial values  $y(0) = 0, y'(0) = -0.1$ .

To find a numerical solution of the system of differential equations, call the procedure `dsolve` with the option `type=numeric`.

```
> interface(displayprecision=5):
> PDEtools[declare](y(t), prime=t, quiet):
> alias(y=y(t), y0=y(0), yp0=D(y)(0)):
> ODE := diff(y,[t$2]) - (1-y^2)*diff(y,t) + y = 0:
> initvals := y0=0, yp0=-0.1:
> IVP := :={ODE, initvals};

IVP := {y'' - (1 - y^2) y' + y = 0, y0 = 0, yp0 = -0.1}
> F := dsolve(IVP, y, type=numeric);

F := proc(rkf45_x) ... end proc
```

You get a numeric solver  $F$  that computes the function value and the derivative at a requested point, i.e., that returns a list consisting of an equation describing the value of the independent variable  $t$ , an equation describing the value of the dependent variable  $y$  at that point, and an equation describing the value of the derivative  $y'$  at that point. Two examples:

```
> F(0);
```

$$[t = 0.00000, y = 0.00000, y' = -0.10000]$$

```
> F(1);
```

$$[t = 1.00000, y = -0.14477, y' = -0.17810]$$

You get a better understanding of the solution by plotting it over a domain of interest. Let us first convert the numeric solution into a function. Evaluation of the dependent variable  $y$  with respect to the side relations given by  $F$  does this job.

```
> # The actual numerical solution
> Y := theta -> eval(y, F(theta));
```

Now you can use the **plot** command to draw the graph of the solution.

```
> plot(Y, 0..14, title=
> "solution of van der Pol's Equation");
```

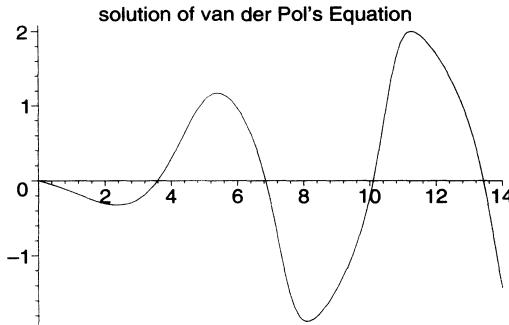


Figure 17.4. Graph of solution of the van der Pol equation.

If non-adaptive plotting of the solution is good enough, you could have used the procedure **odeplot** from the **plots** package.

```
> with(plots, odeplot):
> odeplot(F, [t,y], 0..32, title=
> "odeplot of the solution of van der Pol's Equation");
```

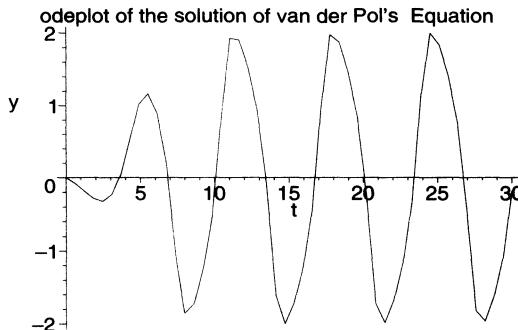


Figure 17.5. **odeplot** of solution of the van der Pol equation.

You see that the quality of the plot has indeed decreased. To avoid this, you can do the following: Firstly, add in the call of **dsolve** the requested

range for the solution. In this case, **dsolve** will compute the solution over the desired range before returning, stores that solution for later calls to the returned procedure, which will then compute the return values through interpolation. Secondly, use the **refine** option in **odeplot** to allow adaptive plotting. Below we choose **refine=2**, which means that twice the number of points in the stored solution are used.

```
> F := dsolve(IVP, y, type=numeric, range=0..32):
> odeplot(F, [t,y], 0..30, refine=2, title=
> "odeplot of the solution of van der Pol's Equation");
```

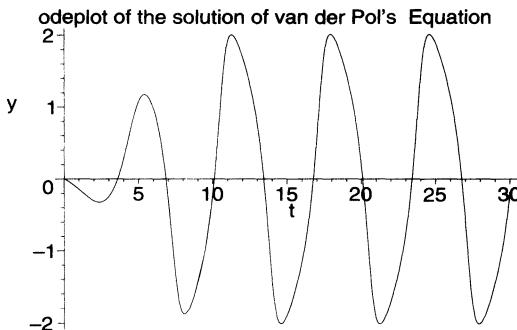


Figure 17.6. Nice **odeplot** of solution of the van der Pol equation.

In addition to the two methods of storing and not storing the numeric solution, you can also compute solution values at a predetermined set of points. The procedure **odeplot** also understands the object of type *Matrix* that is returned by **dsolve**, but naturally cannot do any refinement of the plot.

```
> pts := Array(1..10, i->(i-1)*0.5):
> numsol := dsolve(IVP, type=numeric, output=pts);
```

$$numsol := \left[ \begin{array}{ccc} & [t, y, y'] \\ \left[ \begin{array}{ccc} 0.00000 & 0.00000 & -0.10000 \\ 0.50000 & -0.06220 & -0.14757 \\ 1.00000 & -0.14477 & -0.17810 \\ 1.50000 & -0.23370 & -0.16900 \\ 2.00000 & -0.30336 & -0.09786 \\ 2.50000 & -0.31927 & 0.04644 \\ 3.00000 & -0.24533 & 0.26006 \\ 3.50000 & -0.04966 & 0.53024 \\ 4.00000 & 0.28664 & 0.80520 \\ 4.50000 & 0.72193 & 0.87231 \end{array} \right] \right]$$

```
> odeplot(numsol, style=point, symbol=circle,
> symbolsize=20);
```

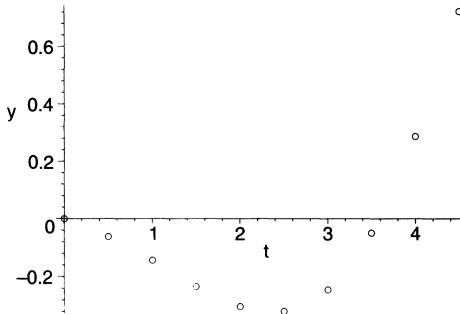


Figure 17.7. `odeplot` at predetermined set of points.

In the second, more lengthy, example we apply the numerical ODE-solver to the system of differential equations that describes the dynamics of a frictionless, rigid, two-link robot manipulator without torque (double pendulum, cf. 17.8) and we shall use the 7th-8th order continuous Runge-Kutta method `dverk78` [75].

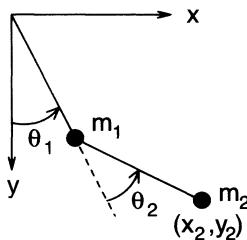


Figure 17.8. Double pendulum.

First, we derive the equations of motion via the *Euler-Lagrange* formalism. The interested reader is referred to [155] for the derivation of the equations of motion via the Newton-Euler formalism. Let  $\theta = (\theta_1, \theta_2)$  and  $\dot{\theta} = (\dot{\theta}_1, \dot{\theta}_2)$ , and define the *Lagrangian function* as

$$L(\theta, \dot{\theta}) = T(\theta, \dot{\theta}) - V(\theta),$$

where  $T(\theta, \dot{\theta})$  is the kinetic energy and  $V(\theta)$  is the potential energy. For the above configuration the kinetic energy is computed as the sum of the kinetic energies  $T_1$  and  $T_2$  of the masses  $m_1$  and  $m_2$ , respectively. The kinetic energy of mass  $m_1$  can be written directly as

$$T_1(\dot{\theta}) = \frac{1}{2}m_1l_1^2\dot{\theta}_1^2.$$

The Cartesian coordinates  $(x_2, y_2)$  of the endpoint are given by

$$\begin{aligned} x_2 &= l_1 \sin \theta_1 + l_2 \sin(\theta_1 + \theta_2), \\ y_2 &= l_1 \cos \theta_1 + l_2 \cos(\theta_1 + \theta_2). \end{aligned}$$

Hence, the Cartesian components of the velocity are

$$\begin{aligned}\dot{x}_2 &= l_1(\cos \theta_1)\dot{\theta}_1 + l_2 \cos(\theta_1 + \theta_2)(\dot{\theta}_1 + \dot{\theta}_2), \\ \dot{y}_2 &= -l_1(\sin \theta_1)\dot{\theta}_1 - l_2 \sin(\theta_1 + \theta_2)(\dot{\theta}_1 + \dot{\theta}_2).\end{aligned}$$

Squaring the magnitude of the velocity yields

$$T_2(\theta, \dot{\theta}) = \frac{1}{2}m_2 \left( (l_1^2 \dot{\theta}_1^2 + (l_2^2 (\dot{\theta}_1 + \dot{\theta}_2)^2) + 2l_1 l_2 (\cos \theta_2) \dot{\theta}_1 (\dot{\theta}_1 + \dot{\theta}_2)) \right).$$

The potential energy is determined by the height of the mass.

$$\begin{aligned}V_1(\theta) &= -m_1 g l_1 \cos \theta_1, \\ V_2(\theta) &= -m_2 g l_1 \cos \theta_1 - m_2 g l_2 \cos(\theta_1 + \theta_2).\end{aligned}$$

All above formulae could have been found with Maple.

```
> restart: interface(displayprecision=5):
> PDEtools[declare]((theta[1],theta[2])(t), prime=t, quiet):
> alias(theta[1]=theta[1](t), theta[2]=theta[2](t)):
> macro(m1=m[1], m2=m[2], l1=l[1], l2=l[2],
>       x1=x[1], x2=x[2], y1=y[1], y2 = y[2],
>       t1=theta[1](t), t2=theta[2](t)):
> x1 := l1*sin(t1): y1 := l1*cos(t1):
> x2 := l1*sin(t1) + l2*sin(t1+t2):
> y2 := l1*cos(t1) + l2*cos(t1+t2):
```

The kinetic energy of mass  $m_1$  is computed by

```
> T[1] := simplify(1/2*m1*(diff(x1,t)^2 + diff(y1,t)^2));
```

$$T_1 := \frac{1}{2} m_1 l_1^2 (\theta_1')^2$$

The kinetic energy of mass  $m_2$  is obtained in a similar way.

```
> T[2] := expand(1/2*m2*(diff(x2,t)^2 + diff(y2,t)^2),
>                 cos(t1+t2), sin(t1+t2)):
> simplify(%);
```

$$\begin{aligned}&\frac{1}{2}m_2(2l_1 \cos(\theta_1)(\theta_1')^2 l_2 \cos(\%1) + 2l_1 \cos(\theta_1)\theta_1' l_2 \cos(\%1)\theta_2' + l_1^2(\theta_1')^2 \\ &+ 2l_1 \sin(\theta_1)(\theta_1')^2 l_2 \sin(\%1) + 2l_1 \sin(\theta_1)\theta_1' l_2 \sin(\%1)\theta_2' \\ &+ l_2^2(\theta_1')^2 + 2l_2^2\theta_1'\theta_2' + l_2^2(\theta_2')^2) \\ \%1 &:= \theta_1 + \theta_2\end{aligned}$$

More simplification is needed, such as computing the finite Fourier series.

```
> map(combine, collect(% , diff(t1,t)), 'trig'):
> T[2] := collect(% , [l1,l2,m2,cos(t2)], factor);
```

$$T_2 := \frac{1}{2}m_2(\theta_1')^2 l_1^2 + \theta_1'(\theta_1' + \theta_2') \cos(\theta_2) m_2 l_2 l_1 + \frac{1}{2}(\theta_1' + \theta_2')^2 m_2 l_2^2$$

The Euler-Lagrange equations are

$$\frac{d}{dt} \left( \frac{\partial L}{\partial \dot{\theta}_i} \right) - \frac{\partial L}{\partial \theta_i} = 0,$$

for  $i = 1, 2$ , which yield in this case the vector equation

$$M(\theta)\ddot{\theta} + C(\theta, \dot{\theta}) + G(\theta) = 0,$$

where  $\ddot{\theta} = (\ddot{\theta}_1, \ddot{\theta}_2)$ ,

$$M(\theta) = \begin{pmatrix} m_1 l_1^2 + m_2 l_1^2 + m_2 l_2^2 + 2m_2 l_1 l_2 \cos \theta_2 & m_2 l_2^2 + m_2 l_1 l_2 \cos \theta_2 \\ m_2 l_2^2 + m_2 l_1 l_2 \cos \theta_2 & m_2 l_2^2 \end{pmatrix},$$

the centripetal or Coriolis term  $C(\theta, \dot{\theta})$  is defined as

$$\begin{pmatrix} -m_2 l_1 l_2 \sin(\theta_2) \dot{\theta}_2 (2\dot{\theta}_1 + \dot{\theta}_2) \\ m_2 l_1 l_2 \sin(\theta_2) \dot{\theta}_1^2 \end{pmatrix},$$

and the gravitational term  $G(\theta)$  is defined as

$$\begin{pmatrix} m_1 g l_1 \sin \theta_1 + m_2 g l_1 \sin \theta_1 + m_2 g l_2 \sin(\theta_1 + \theta_2) \\ m_2 g l_2 \sin(\theta_1 + \theta_2) \end{pmatrix}.$$

Let us see how this can be computed by Maple. First, we compute the potential energy of masses  $m_1$  and  $m_2$ , respectively.

```
> V[1] := - m1*g*l1*cos(t1):
> V[2] := - m2*g*l1*cos(t1) - m2*g*l2*cos(t1+t2):
> L := T[1] + T[2] - V[1] - V[2]: # Lagrangian
```

We cannot easily compute the derivatives  $\frac{\partial L}{\partial \dot{\theta}_i}$  and  $\frac{\partial L}{\partial \theta_i}$ . We solve this problem by introducing some auxiliary variables.

```
> L := subs({t1=t_1, t2=t_2, diff(t1,t)=t1p,
> diff(t2,t)=t2p}, L):
> dL_dt1p := diff(L, t1p): dL_dt2p := diff(L, t2p):
> dL_dt1 := diff(L, t_1): dL_dt2 := diff(L, t_2):
> dL_dt1p := subs({t_1=t1, t_2=t2, t1p=diff(t1,t),
> t2p=diff(t2,t)}, dL_dt1p):
> dL_dt2p := subs({t_1=t1, t_2=t2, t1p=diff(t1,t),
> t2p=diff(t2,t)}, dL_dt2p):
> dL_dt1 := subs({t_1=t1, t_2=t2, t1p=diff(t1,t),
> t2p=diff(t2,t)}, dL_dt1):
> dL_dt2 := subs({t_1=t1, t_2=t2, t1p=diff(t1,t),
> t2p=diff(t2,t)}, dL_dt2):
```

Now, we are ready to compute the differential equations via the Euler-Lagrange equations.

```
> ode[1] := collect(diff(dL_dt1p, t) - dL_dt1 = 0,
> [diff(t1,[t$2]),diff(t2,[t$2]),diff(t1,t),diff(t2,t)]);
```

$$\begin{aligned}
ode_1 := & (m_1 l_1^2 + m_2 l_1^2 + m_2 l_2^2 + 2 \cos(\theta_2) m_2 l_2 l_1) \theta_1'' \\
& + (m_2 l_2^2 + \cos(\theta_2) m_2 l_2 l_1) \theta_2'' - 2 \theta_1' \sin(\theta_2) \theta_2' m_2 l_2 l_1 \\
& - (\theta_2')^2 \sin(\theta_2) m_2 l_2 l_1 + m_2 g l_2 \sin(\theta_1 + \theta_2) + m_1 g l_1 \sin(\theta_1) \\
& + m_2 g l_1 \sin(\theta_1) = 0
\end{aligned}$$

> `ode[2] := collect(diff(dL_dt2p, t) - dL_dt2 = 0,`  
> `[diff(t1,[t$2]),diff(t2,[t$2]),diff(t1,t),diff(t2,t)]);`

$$\begin{aligned}
ode_2 := & (m_2 l_2^2 + \cos(\theta_2) m_2 l_2 l_1) \theta_1'' + \theta_2'' m_2 l_2^2 + \sin(\theta_2) m_2 l_2 l_1 (\theta_1')^2 \\
& + m_2 g l_2 \sin(\theta_1 + \theta_2) = 0
\end{aligned}$$

For simplicity, we shall consider the case that both masses and lengths are equal, say  $m_1 = m_2 = m$  and  $l_1 = l_2 = l$ . Then the equations of motion are as follows.

> `m1 := m2: m2 := m: l1 := l2: l2 := 1:`  
> `map(x->x/(m*l^2), lhs(ode[1])):`  
> `ode[1] := map(normal, %) = 0;`

$$\begin{aligned}
ode_1 := & (3 + 2 \cos(\theta_2)) \theta_1'' + (1 + \cos(\theta_2)) \theta_2'' - 2 \theta_1' \sin(\theta_2) \theta_2' \\
& - (\theta_2')^2 \sin(\theta_2) + \frac{g \sin(\theta_1 + \theta_2)}{l} + \frac{2 g \sin(\theta_1)}{l} = 0
\end{aligned}$$

> `map(x->x/(m*l^2), lhs(ode[2])):`  
> `ode[2] := map(normal, %) = 0;`

$$ode_2 := (1 + \cos(\theta_2)) \theta_1'' + \theta_2'' + \sin(\theta_2) (\theta_1')^2 + \frac{g \sin(\theta_1 + \theta_2)}{l} = 0$$

Let us take  $l = 1$  and  $g = 9.8$ . As initial values we choose

$$\theta_1(0) = 1.1, \theta_2(0) = 0, \dot{\theta}_1(0) = 0, \dot{\theta}_2(0) = 0.$$

Now we are ready to solve the initial value problem numerically.

> `l := 1: g := 9.8:`  
> `alias(t1_0=theta[1](0), t2_0=theta[2](0),`  
> `t1p_0=D(theta[1])(0), t2p_0=D(theta[2])(0) ):`

Once again, all we have to do is to call the procedure `dsolve` with the option `type=numeric`. Here, we choose the method `dverk78`.

> `F := dsolve({ode[1], ode[2], t1_0=1.1, t2_0=0, t1p_0=0,`  
> `t2p_0=0}, {t1,t2}, type=numeric, method=dverk78):`  
> `F(0.5); # an example`

$$[t = 0.50000, \theta_1 = 0.30064, \theta_1' = -2.04315, \theta_2 = 0.45013, \theta_2' = -0.60460]$$

The numeric solver returns a list consisting of an equation describing the value of the independent variable  $t$ , two equations describing the value of the dependent variables  $\theta_1$  and  $\theta_2$  at that point, and two equations describing the value of the derivatives of the dependent variables at that point.

You can add the option `output=listprocedure` for numerical solving of ODEs so that the result is a list of equations where the left-hand sides are the names of the independent variable and the dependent functions, and the right-hand sides are procedures. This makes it easy to obtain numerical functions for angles ( $\theta_1, \theta_2$ ) and derivatives ( $\dot{\theta}_1, \dot{\theta}_2$ ):

```
> F := dsolve({ode[1], ode[2], t1_0=1.1, t2_0=0, t1p_0=0,
> t2p_0=0}, {t1,t2}, type = numeric, method=dverk78,
> output=listprocedure);
```

```
F := [t = (proc(t) ... end proc), theta1 = (proc(t) ... end proc),
theta1' = (proc(t) ... end proc), theta2 = (proc(t) ... end proc),
theta2' = (proc(t) ... end proc)]
> Theta[1] := eval(t1, F): Theta[1](0.5);
0.30064
> Theta[2] := eval(t2, F):
```

Let  $\Theta_{1,1}$  denote the first derivative of the function  $\Theta_1$ :

```
> Theta[1,1] := eval(diff(t1,t), F):
```

We plot the graphs of  $\theta_1$  and  $\theta_2$  to get an impression of the solution found.

```
> plot(Theta[1], 0..10, title="shoulder angle");
```

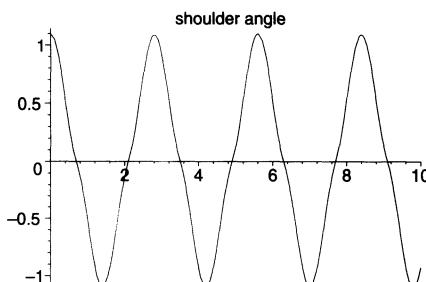


Figure 17.9. Graph of shoulder angle of double pendulum.

```
> plot(Theta[2], 0..10, title="elbow angle");
```

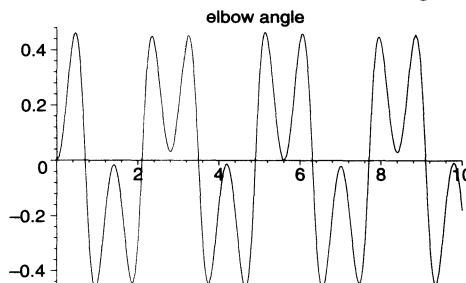


Figure 17.10. Graph of elbow angle of double pendulum.

Is this what you expected for the angles: a nice periodic motion? A three-dimensional solution curve for the angles  $(\theta_1, \theta_2)$  can be plotted with the procedure **odeplot** from the **plots** package. By modifying the orientation of the three-dimensional plot, it is possible instead to obtain the above two-dimensional projections of the solution curve. Periodicity is better illustrated by drawing the curve  $t \rightarrow [\theta_1, \theta_2, \dot{\theta}_1]$  in 3-space.

```
> plots[spacecurve]([Theta[1], Theta[2], Theta[1,1]], 0..10,
> view = [-1.25..1.25, -0.5..0.5, -3..3], numpoints=200,
> orientation=[-75,35], color=blue, axes=box, title=
> "periodic motion at starting angle of 63 degrees");
```

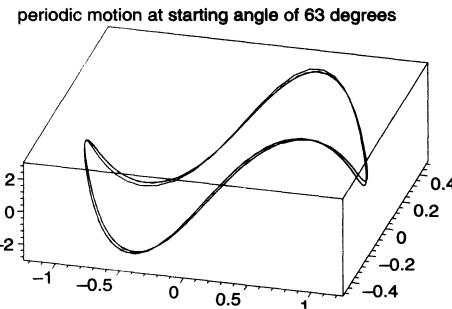


Figure 17.11. periodic solution of double pendulum.

But for other initial conditions, the nonlinear character of the differential equations may lead to chaotic motion. To see this effect we choose  $\theta_1(0) = 1.5$  and  $\theta_2(0) = 0$ .

```
> F := dsolve( {ode[1], ode[2], t1_0=1.5, t2_0=0, t1p_0=0,
> t2p_0=0}, {t1,t2}, type=numeric, method=dverk78,
> output=listprocedure):
> Theta[1] := eval(t1, F): Theta[2] := eval(t2, F):
> Theta[1,1] := eval(diff(t1,t), F):
```

We plot the graphs of  $\theta_1$  and  $\theta_2$  to get an impression of the solution found.

```
> plot(Theta[1], 0..15, title="shoulder angle");
shoulder angle
```

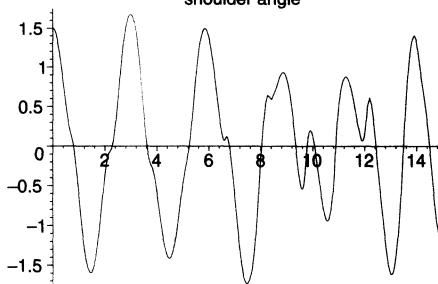


Figure 17.12. Graph of shoulder angle of double pendulum.

```
> plot(Theta[2], 0..15, title="elbow angle");
```

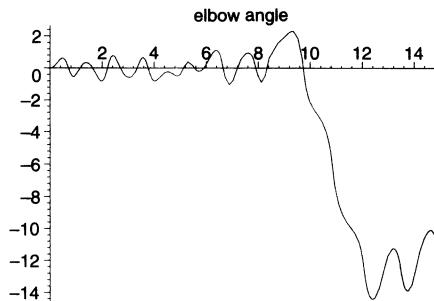


Figure 17.13. Graph of elbow angle of double pendulum.

This chaotic motion is even better illustrated by the following space curve.

```
> plots[spacecurve]([Theta[1], Theta[2], Theta[1,1]], 0..15,
> view=[-1.75..1.75, -15..3, -5..7], numpoints=200,
> orientation=[-75,35], color=black, axes=box, title=
> "chaotic motion at starting angle of 86 degrees");
chaotic motion at starting angle of 86 degrees
```

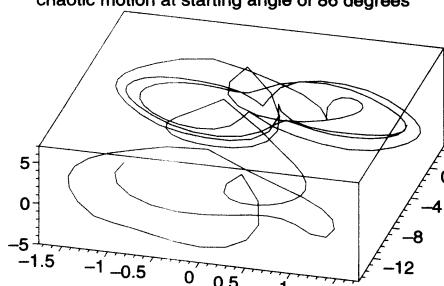


Figure 17.14. chaotic solution of double pendulum

Better insight into the motion of the double pendulum can be obtained from an animation. Below, we give the commands that generate such an animation. Note that this is a continuation of the previous computation. Mimic the session and experience the chaotic motion of the double pendulum.

Firstly, we define functions which describe the position of the kink and the tip of the two-link robot arm.

```
> x1 := t -> sin(Theta[1](t)):
> y1 := t -> -cos(Theta[1](t)):
> x2 := t -> sin(Theta[1](t))
>      + sin(Theta[1](t) + Theta[2](t)):
> y2 := t -> -cos(Theta[1](t))
>      - cos(Theta[1](t) + Theta[2](t)):
```

Secondly, we generate a sequence of plots that describe the robot-arm at different times in the interval  $(0, 25)$ . This is time-consuming step if the number of time steps is large.

```

> for i from 0 to 30 by 1/10 do
>   P[i] := plot([[0,0], [x1(i),y1(i)], [x2(i),y2(i)]],
>   style=line, view=[-2..2,-2..2])
> end do:
> plotsequence := [seq(P[i/10], i=0..300)]:
```

Thirdly, we display them in sequence with the **display** routine from the **plots** package, i.e., we produce the animation.

```

> plots[display](plotsequence, insequence=true,
> view=[-2..2,-2..2]);
```

You have encountered two Runge-Kutta methods for solving ODEs numerically. But Maple contains many more numerical methods. Table 17.6 summarizes the numerical methods provided by **dsolve** for initial value problems via the options **type=numeric**, **method=approach**.

Method	Meaning
classical	forward Euler method (default)
classical[adambash]	Adams-Bashford ('predictor') method
classical[abmoulton]	Adams-Bashford-Moulton ('predictor-corrector') method
classical[foreuler]	forward Euler method
classical[heunform]	Heun's method
classical[impoly]	improved polygon method (also known as modified Euler method)
classical[rk2]	2nd order classical Runge-Kutta method
classical[rk3]	3rd order classical Runge-Kutta method
classical[rk4]	4th order classical Runge-Kutta method
dverk78	7th-8th order continuous Runge-Kutta method
gear	Gear single-step extrapolation method
gear[bstoer]	Bulirsch-Stoer rational extrapolation method
gear[polyextr]	polynomial extrapolation method
lsode	Livermore Stiff ODE solver
lsode[adamsband]	implicit Adams method using chord iteration with a banded Jacobian matrix
lsode[adamsdiag]	implicit Adams method using chord iteration with a diagonal Jacobian matrix
lsode[adamsfunc]	implicit Adams method with a functional iteration (default)
lsode[adamsfull]	implicit Adams method using chord iteration with a full Jacobian matrix
lsode[backband]	method of backward differentiation formulas with a banded Jacobian matrix
lsode[backdiag]	method of backward differentiation formulas with a diagonal Jacobian matrix

*continued on next page*

<i>continued from previous page</i>	
Method	Meaning
lsode[backfunc]	method of backward differentiation formulas with a functional iteration
lsode[backfull]	method of backward differentiation formulas with a full Jacobian matrix
mgear	Gear multi-step extrapolation method
mgear[adamspc]	Adams predictor-corrector method
mgear[msteppart]	multi-step method for stiff systems with evaluation of Jacobian matrix at each step
mgear[mstepnum]	multi-step method for stiff systems with numeric differencing of derivatives for the computation of Jacobian matrix (default)
rkf45	Fehlberg 4th-5th order Runge-Kutta method
rosenbrock	implicit Rosenbrock 3rd-4th order Runge
taylorseries	Kutta method for stiff systems numerical Taylor series method

Table 17.6. Numerical methods provided by **dsolve** for initial value problems.

A description of the classical numerical approaches can be found in almost any text on numerical solving of ODEs. For the Gear single- and multistep methods we refer to [92]. The code GEAR was further improved to LSODE, Livermore Stiff ODE solver, and is part of a larger program package called ODEPACK (see [130]). The numerical solver for systems of first order ODEs using a Taylor series method is described in [14]. A gentle introduction for numerical methods using Maple is the standard text on numerical analysis of Burden and Faires [38].

Table 17.7 summarizes the numerical methods provided by **dsolve** for boundary value problems via the options **type=numeric**, **method=approach**. The **bvp[traprich]** method is the default choice.

Method	Meaning
bvp[middefer]	midpoint based method of deferred corrections
bvp[midrich]	midpoint based Richardson extrapolation
bvp[traprich]	trapezoid based Richardson extrapolation
bvp[trapdefer]	trapezoid based method of deferred corrections

Table 17.7. Numerical methods provided by **dsolve** for boundary value problems.

We refer to the on-line help system for immediate help on how to call the various numerical solvers of ODEs. Especially, enter the command **?dsolve,numeric,IVP** and **?dsolve,numeric,BVP** to get information about various options. Here, we only give two examples. In the first example of an initial value problem, we apply the Taylor series method to a system of ODEs representing an anharmonic oscillator.

```
> ODEs := diff(q(t),t) = p(t),
> diff(p(t),t) = -q(t) - 4*q(t)^2 - 2*q(t)^3;
```

$$ODEs := \frac{d}{dt} q(t) = p(t), \quad \frac{d}{dt} p(t) = -q(t) - 4q(t)^2 - 2q(t)^3$$

with the following initial conditions.

```
> initvals := q(0)=0, p(0)=1:
> dsolve({ODEs, initvals}, {q(t), p(t)}, type=numeric,
> method=taylorseries, output=listprocedure):
```

We have added the option `output = listprocedure` for numerical solving of ODEs so that the result is a list of equations where the left-hand sides are the names of the independent variable and the dependent functions, and the right-hand sides are procedures. This makes it easy to obtain numerical functions for position ( $q$ ) and momentum ( $p$ ):

```
> Q := eval(q(t), %): # position
> P := eval(p(t), %): # momentum
```

The plot of position and momentum with respect to time is shown in Figure 17.15. It is computed as follows:

```
> plot([Q, P], 0..10, linestyle=[1,2], color=[blue,red]);
```

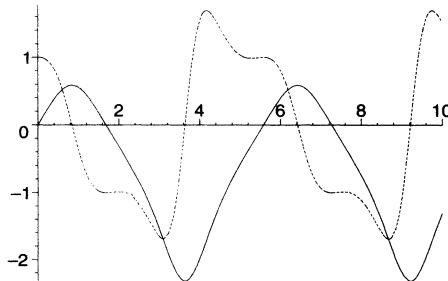


Figure 17.15. Graph of position and momentum of an anharmonic oscillator.

A numerical approximation of the period of the anharmonic oscillator can be obtained easily.

```
> fsolve(Q, 5..6);
5.583431947
```

Let us verify the position after 100 oscillations.

```
> Q(100*%);

-0.38254494773640 10^-6
```

The oscillator trajectory in phase space (momentum versus position) is shown in Figure 17.16.

```
> plot([Q, P, 0..5.6]); # phase plot
```

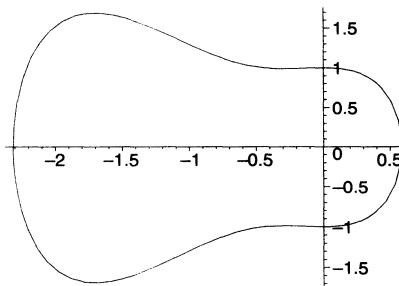


Figure 17.16. Phase plot of an anharmonic oscillator.

The kinetic energy versus position looks as follows.

```
> plot([Q, P^2/2, 0..5.6]);
```

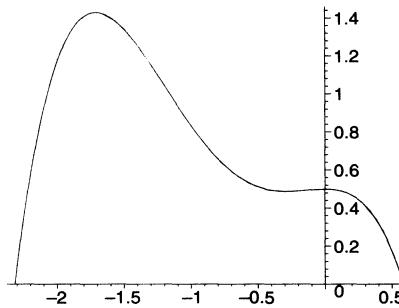


Figure 17.17. Kinetic energy versus position for an anharmonic oscillator.

Figure 17.18 illustrates that the corresponding Hamiltonian

$$H = \frac{1}{2}p^2 + \frac{1}{2}q^2 + \frac{4}{3}q^3 + \frac{1}{2}q^4$$

is a constant of motion.

```
> plot(1/2*p^2 + 1/2*q^2 + 4/3*q^3 + 1/2*q^4,
>      0..5.6, 0.4..0.6, numpoints=25, adaptive=false);
```

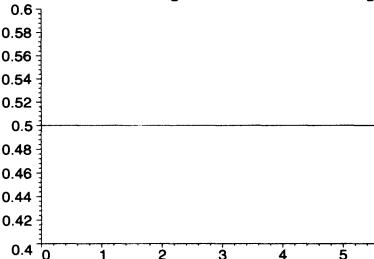


Figure 17.18. Hamiltonian as constant of motion.

We end with the following boundary value problem:

$$y'' + 9y = 0, \quad y\left(-\frac{\pi}{2}\right) = -1, \quad y\left(\frac{\pi}{2}\right) = 1.$$

Indeed, the differential equation of the mathematical pendulum and the solution  $y(x) = \sin(3x)$  matches the boundary condition so that it is easy to verify the numerical solution method. And that is what we shall do below.

```

> BVP := {diff(y(x),[x$2])+9*y(x)=0,
> y(-Pi/2)=-1, y(Pi/2)=1};

BVP := {(\frac{d^2}{dx^2} y(x)) + 9 y(x) = 0, y(-\frac{\pi}{2}) = -1, y(\frac{\pi}{2}) = 1}

> F := dsolve(BVP, y(x), type=numeric, method=bvp[midrich]);

F := proc(x_bvp) ... end proc

> plots[odeplot](F, [x,y(x)], -Pi/2..Pi/2, color=black,
> style=point, symbol=circle, symbolsize=12);

```

Figure 17.19. Numerical solution of a boundary value problem.

```

> plot(-sin(3*x), x=-Pi/2..Pi/2, color=blue):
> plots[display](%, %); # graphical comparison


```

Figure 17.20. Comparison of numerical and exact solution.

## 17.7 Graphical Methods

In all examples in this section we shall assume that the **DEtools** package has been loaded and that the procedure **declare** of the **PDEtools** package has been used to declare dependent functions for the purpose of short notation.

```

> with(DEtools):
> PDEtools[declare]((x,y,z,p,q)(t), prime=t, quiet):

```

You can use the procedure **DEplot** of the **DEtools** package to solve an ordinary differential equation graphically. An example from the previous section: the van der Pol equation

$$y'' - (1 - y^2)y' + y = 0,$$

with initial values  $y(0) = 0, y'(0) = -0.1$  (Figure 17.21).

```
> ODE := diff(y(t),[t$2]) - (1-y(t)^2)*diff(y(t),t)
>      + y(t) = 0;
ODE :=  $y'' - (1 - y^2)y' + y = 0$ 
> initvals := [[y(0) = 0, D(y)(0) = -0.1]];
initvals := [[y(0) = 0, D(y)(0) = -0.1]]
> DEplot(ODE, {y(t)}, t=0..30, initvals,
>      linecolor=red, stepsize=0.1);
```

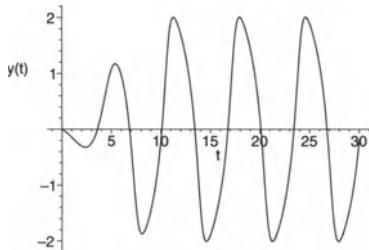


Figure 17.21. Solution curve of the van der Pol equation.

You see that after an initially driven motion a periodic motion occurs. This can also be seen in the phase portrait; the ideally periodic curve is called the limit cycle. You can also use the procedure **DEplot** for this purpose (Figure 17.22).

```
> eqns := diff(y(t), t) = x(t),
>      diff(x(t), t) - (1-y(t)^2)*x(t) + y(t) = 0;
eqns :=  $y' = x, x' - (1 - y^2)x + y = 0$ 
> initvals := [[x(0)=-0.1, y(0)=0], [x(0)=1, y(0)=1]]:
> DEplot({eqns}, [x(t), y(t)], 0..10, initvals,
>      linecolor=[black, blue], stepsize=0.1);
```

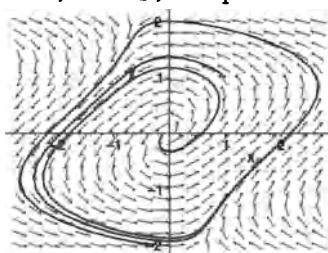


Figure 17.22. Phase portrait with two solution curves of the van der Pol equation.

As a second example, we shall use the **DEplot** procedure to draw the phase portrait with two integral curves of the anharmonic oscillator of the previous section.

```
> ODEs := diff(q(t),t) = p(t),
> diff(p(t),t) = -q(t) - 4*q(t)^2 - 2*q(t)^3;
ODEs := q' = p, p' = -q - 4 q^2 - 2 q^3
> DEplot({ODEs}, [q(t),p(t)], t=0..8,
> [[q(0)=0,p(0)=1], [q(0)=1,p(0)=0]],
> method=dverk78, linecolor=black );
```

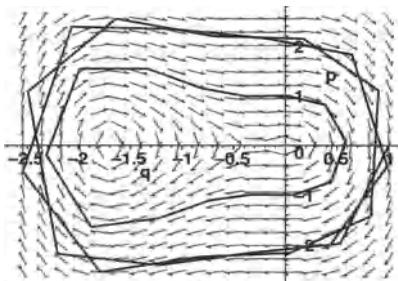


Figure 17.23. Phase plot of an anharmonic oscillator

As a third example, we compute field lines of an electric field due to two parallel uniform line charges of one unit charge per unit length, of opposite polarities, and at a distance of two unit length of each other. Actually, we consider the same example as in §15.5. The electric field of the charge distribution in the  $xy$ -plane is given by  $\vec{E} = (E_x, E_y) = -\nabla\phi$ , where

$$\phi = \ln(\sqrt{(x+1)^2 + y^2}) - \ln(\sqrt{(x-1)^2 + y^2}).$$

A field line can now be thought of as the trajectory of a unit charge under the influence of the electric field. It can be calculated by solving the differential equation  $\frac{dx}{dt} = E_x, \frac{dy}{dt} = E_y$ , where  $t$  represents progress down the trajectory.

```
> phi := ln(sqrt((x+1)^2+y^2)) - ln(sqrt((x-1)^2+y^2));
phi := ln(sqrt(x^2 + 2 x + 1 + y^2)) - ln(sqrt(x^2 - 2 x + 1 + y^2))
> E := map(normal, VectorCalculus[Gradient](-phi, [x,y]));
E := 
$$\frac{2 (x^2 - 1 - y^2)}{(x^2 + 2 x + 1 + y^2) (x^2 - 2 x + 1 + y^2)} \bar{e}_x +$$


$$\frac{4 y x}{(x^2 + 2 x + 1 + y^2) (x^2 - 2 x + 1 + y^2)} \bar{e}_y$$

> ODEs := diff(x(t),t)=E[1], diff(y(t),t)=E[2];
```

$$ODEs := x' = \frac{2(x^2 - 1 - y^2)}{(x^2 + 2x + 1 + y^2)(x^2 - 2x + 1 + y^2)},$$

$$y' = \frac{4yx}{(x^2 + 2x + 1 + y^2)(x^2 - 2x + 1 + y^2)}$$

```

> initvals := [seq([x(0)=1+0.05*cos(Pi/20*s),
> y(0)=0.05*sin(Pi/20*s)], s=5..19)]:
> initvals := evalf(initvals):
> interface(warnlevel=0): # suppress warning messages
> DEplot({ODEs}, [x,y], t=0..20, initvals, x=-1.5..1.5,
> y=0..2.5, linecolor=black, stepsize=0.001 );

```

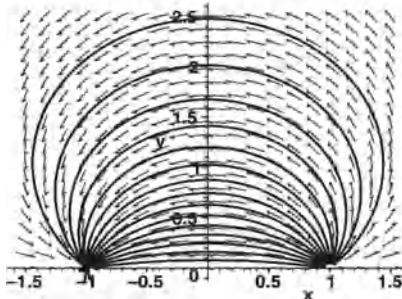


Figure 17.24. Electric field plot.

The ranges and step size are set to make the field lines look reasonable near the singular point  $(-1,0)$  of the electrostatic potential. Drawbacks of these choices are that the computation of the field lines takes quite a long time.

With the **DEplot3d** procedure of the **DEtools** package you can easily produce a three dimensional representation of the solution curves of a system of ordinary differential equations. Below we show two famous examples of nonlinear systems with periodic and chaotic solutions: the Rössler attractor and the Lorenz attractor. The Rössler attractor is described by the equations

$$x' = -(y - z), \quad y' = z + ay, \quad z' = b + z(x - c),$$

with  $x, y, z$  real functions of time  $t$ , and  $a, b$ , and  $c$  positive constants.

```

> vars := [x(t),y(t),z(t)]:
> RoesslerAttractor := {diff(x(t),t) = - (y(t)+z(t)),
> diff(y(t),t) = x(t) + a*y(t),
> diff(z(t),t) = b + z(t)*(x(t)-c)};

```

$$RoesslerAttractor := \{z' = b + z(x - c), x' = -y - z, y' = x + ay\}$$

We choose the ‘standard’ values  $a = b = 0.2$ . The qualitative nature of a trajectory depends very much on the value of  $c$ : you can have periodic solutions as well as chaotic behavior, period doubling takes place, etc. Like in section 7.7.1 of [74], we choose  $c = 5$  and take as initial values  $x(0) = 4$ ,  $y(0) = z(0) = 0$ .

```

> a, b, c := 0.2, 0.2, 5;
      a, b, c := 0.2, 0.2, 5
> initvals := [[x(0)=4, y(0)=0, z(0)=0]];
      initvals := [[x(0) = 4, y(0) = 0, z(0) = 0]]

```

We plot the solution curve, coloring it on the basis of value of time  $t$  and taking a step size in the numerical integration small enough to get a smooth graph (Figure 17.25).

```

> DEplot3d(RoesslerAttractor, vars, t=0..100, initvals,
>   stepsize=0.05, linecolor=t, orientation=[-170,60],
>   axes=frame, title="chaotic solution of Roessler\
>   attractor");

```

chaotic solution of Roessler attractor

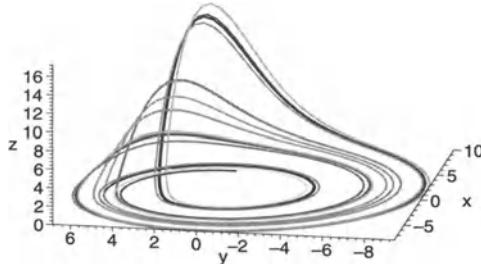


Figure 17.25. Chaotic solution of Rössler attractor.

So, the trajectory is spiraling out from the origin until at some critical radius it jumps away from the  $x$ - $y$  plane, moving in the  $z$  direction until some maximum  $z$  value is attained, and then dropping back into the  $x$ - $y$  plane to undertake a new spiraling out motion, and so on.

The Lorenz attractor is described by the equations

$$x' = \sigma(y - x), \quad y' = \rho x - y - xz, \quad z' = xy - \beta z,$$

with  $x, y, z$  real functions of time  $t$ , and  $\beta, \rho$ , and  $\sigma$  positive constants.

```

> LorenzAttractor :=
>{ diff(x(t),t) = sigma*(y(t) - x(t)),
>  diff(y(t),t) = rho*x(t) - y(t) - x(t)*z(t),
>  diff(z(t),t) = x(t)*y(t) - beta*z(t) };

```

$$\text{LorenzAttractor} := \{y' = \rho x - y - xz, x' = \sigma(y - x), z' = xy - \beta z\}$$

We choose the ‘standard’ values  $\sigma = 10$  and  $\beta = 8/3$ . Like in the Rössler attractor, the qualitative nature of a trajectory depends very much on the value of the so-called bifurcation parameter  $\rho$ : there is a bifurcation point at  $\rho = 24.737$ . We choose  $\rho = 28$  and take as initial values  $x(0) = 2$ ,  $y(0) = 3$ ,  $z(0) = 4$ .

```
> sigma, beta, rho := 10, 8/3, 28;
```

```

 $\sigma, \beta, \rho := 10, \frac{8}{3}, 28$ 
> initvals := [[x(0)=2, y(0)=3, z(0)=4]];
initvals := [[x(0) = 2, y(0) = 3, z(0) = 4]]

```

We plot the solution curve, coloring it on the basis of value of time  $t$ . We choose a gear method for solving the equation numerically and we take a step size small enough to get a smooth graph (Figure 17.26).

```

> DEplot3d(LorenzAttractor, vars, t=0..20,
> initvals, method=gear, stepsize=0.01,
> linecolor=t, orientation=[120,60], title=
> "chaotic solution of Lorenz attractor");
chaotic solution of Lorenz attractor

```

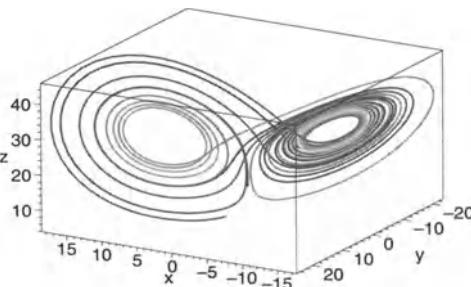


Figure 17.26. Chaotic solution of Lorenz attractor.

With **PDEplot** procedure of the **PDEtools** package you can plot numerical solutions of first order, quasi-linear partial differential equations of the form

$$P(x, t, u) \frac{\partial u}{\partial x} + Q(x, t, u) \frac{\partial u}{\partial t} = R(x, t, u).$$

As an example, we compute how shock waves develop in fluid systems according to the PDE

$$\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} = 0,$$

with initial condition

$$u(x, 0) = \begin{cases} 1 - x^2, & \text{if } |x| \leq 1; \\ 0, & \text{otherwise.} \end{cases}$$

The propagation of the shock wave is shown in Figures 17.27.

```

> with(PDEtools):
> alias(u=u(x,t)):
> PDE := diff(u,t) + u*diff(u,x) = 0:
> initdata := [s,0,1-s^2]:
> PDEplot(PDE, u, initdata, s=-1..1, x=-1..3, t=0..2.5,
> orientation=[-110,60], axes=box, style=hidden,
> color=black);

```

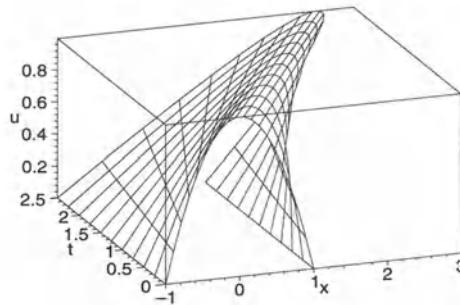


Figure 17.27. Propagation of a shock wave.

## 17.8 Change of Coordinates

Another important application of the **DEtools** and **PDEtools** packages is that they provides tools for performing a change of variables in differential equations. Both dependent and independent variables can be transformed. The procedures in the **DEtools** package are **Dchangevar** and **PDEchangecoords**. In fact, these procedure are superseded by the new procedure **dchange** in the **PDEtools** package, but often they are still convenient.

As an example of a change of variable in a ODE, we consider the one-dimensional Schrödinger equation for a particle in a quadratic potential field:

$$-\frac{\hbar^2}{2m} \frac{d^2\psi}{dx^2} + \frac{1}{2} kx^2\psi = E\psi.$$

```
> restart: with(DEtools): with(PDEtools):
> ODE := -h^2/(2*m)*diff(psi(x),[x$2]) +
> 1/2*k*x^2*psi(x) = E*psi(x);
ODE := -  $\frac{1}{2} \frac{h^2 (\frac{d^2 \psi(x)}{dx^2})}{m} + \frac{1}{2} k x^2 \psi(x) = E \psi(x)$ 
> a := sqrt(h/(m*omega));
a :=  $\sqrt{\frac{h}{m \omega}}$ 
> E := h/2*omega*lambda;
E :=  $\frac{h \omega \lambda}{2}$ 
> omega := sqrt(k/m); # oscillator frequency
omega :=  $\sqrt{\frac{k}{m}}$ 
```

```
> Dchangevar({x=a*xi, psi(x)=phi(xi)}, ODE, x, xi);
```

$$-\frac{1}{2} h \sqrt{\frac{k}{m}} \left( \frac{d^2}{d\xi^2} \phi(\xi) \right) + \frac{1}{2} \frac{k h \xi^2 \phi(\xi)}{m \sqrt{\frac{k}{m}}} = \frac{1}{2} h \sqrt{\frac{k}{m}} \lambda \phi(\xi)$$

With the **dchange** procedure you do it in the following way.

```
> dchange({x=a*xi, psi(x)=phi(xi)}, ODE, [xi,phi]);
```

$$-\frac{1}{2} h \sqrt{\frac{k}{m}} \left( \frac{d^2}{d\xi^2} \phi(\xi) \right) + \frac{1}{2} \frac{k h \xi^2 \phi(\xi)}{m \sqrt{\frac{k}{m}}} = \frac{1}{2} h \sqrt{\frac{k}{m}} \lambda \phi(\xi)$$

```
> simplify((rhs-lhs)(%), assume=positive);
```

$$\frac{1}{2} \frac{h \sqrt{k} (\lambda \phi(\xi) + \left( \frac{d^2}{d\xi^2} \phi(\xi) \right) - \xi^2 \phi(\xi))}{\sqrt{m}}$$

```
> select(has, %, phi);
```

$$\lambda \phi(\xi) + \left( \frac{d^2}{d\xi^2} \phi(\xi) \right) - \xi^2 \phi(\xi)$$

```
> newODE := collect(%, phi);
```

$$newODE := (\lambda - \xi^2) \phi(\xi) + \left( \frac{d^2}{d\xi^2} \phi(\xi) \right)$$

```
> Dchangevar(phi(xi)=exp(-1/2*xi^2)*y(xi), newODE, xi);
```

$$\begin{aligned} & (\lambda - \xi^2) e^{-\frac{\xi^2}{2}} y(\xi) - e^{-\frac{\xi^2}{2}} y(\xi) + \xi^2 e^{-\frac{\xi^2}{2}} y(\xi) \\ & - 2 \xi e^{-\frac{\xi^2}{2}} \left( \frac{d}{d\xi} y(\xi) \right) + e^{-\frac{\xi^2}{2}} \left( \frac{d^2}{d\xi^2} y(\xi) \right) \end{aligned}$$

```
> factor(%);
```

$$e^{-\frac{\xi^2}{2}} (y(\xi) \lambda - y(\xi) - 2 \left( \frac{d}{d\xi} y(\xi) \right) \xi + \left( \frac{d^2}{d\xi^2} y(\xi) \right))$$

```
> select(has, %, y);
```

$$y(\xi) \lambda - y(\xi) - 2 \left( \frac{d}{d\xi} y(\xi) \right) \xi + \left( \frac{d^2}{d\xi^2} y(\xi) \right)$$

```
> HermiteODE := collect(%, y)=0;
```

$$HermiteODE := (\lambda - 1) y(\xi) - 2 \left( \frac{d}{d\xi} y(\xi) \right) \xi + \left( \frac{d^2}{d\xi^2} y(\xi) \right) = 0$$

The power series solution of this differential equation has already been computed in §17.5.

A change of coordinates for PDEs can be done with the procedure **PDEchangecoords** in a similar way as you can change coordinate systems for plotting. For example, the two-dimensional Laplace equation can be rewritten in polar coordinates in the following way.

```
> LaplacePDE := diff(F(x,y),[x$2])  
> + diff(F(x,y),[y$2]) = 0;
```

```

LaplacePDE := ( $\frac{\partial^2}{\partial x^2} F(x, y)$ ) + ( $\frac{\partial^2}{\partial y^2} F(x, y)$ ) = 0
> PDEchangecoords(LaplacePDE, [x,y], polar, [r,phi]);

$$\frac{(\frac{\partial}{\partial r} F(r, \phi)) r + (\frac{\partial^2}{\partial \phi^2} F(r, \phi)) + (\frac{\partial^2}{\partial r^2} F(r, \phi)) r^2}{r^2} = 0$$


```

With the **dchange** procedure you do it in the following way.

```

> dchange({x=r*cos(phi), y=r*sin(phi)}, LaplacePDE, simplify);

$$\frac{(\frac{\partial^2}{\partial \phi^2} F(\phi, r)) + (\frac{\partial}{\partial r} F(\phi, r)) r + (\frac{\partial^2}{\partial r^2} F(\phi, r)) r^2}{r^2} = 0$$


```

The last argument in this command is the simplification routine to be applied to the result to get a simpler result.

```

> newLaplacePDE := numer(lhs(%)) = 0;
newLaplacePDE := ( $\frac{\partial^2}{\partial \phi^2} F(\phi, r)$ ) + ( $\frac{\partial}{\partial r} F(\phi, r)$ ) r + ( $\frac{\partial^2}{\partial r^2} F(\phi, r)$ ) r^2 = 0

```

A big advantage of the **PDEchangecoords** compared to the **dchange** procedure is that it knows about the most popular coordinate systems: above we could just use the keyword **polar** for polar coordinates in the **PDEchangecoords** command whereas we had to really define the transformation in the **dchange** command. If the built-in coordinate systems do not suffice, then you can define your own coordinate transformation via the procedure **addcoords** and use **PDEchangecoords**. We illustrate this by transforming the Fokker-Planck equation

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + x \frac{\partial u}{\partial x} + u$$

into the diffusion equation

$$\frac{\partial v}{\partial \tau} = \frac{\partial^2 v}{\partial \xi^2}.$$

```

> FockerPlanckPDE := diff(u(x,t),t) =
> diff(u(x,t),[x$2]) + x*diff(u(x,t),x) + u(x,t);

```

$$FockerPlanckPDE := \frac{\partial}{\partial t} u(x, t) = \left( \frac{\partial^2}{\partial x^2} u(x, t) \right) + x \left( \frac{\partial}{\partial x} u(x, t) \right) + u(x, t)$$

The first change of variables is  $\xi = xe^t, v = ue^{-t}$ .

```

> Dchangevar({u(x,t)=v(xi,t)*exp(t)},
> FockerPlanckPDE, x, xi);

$$\left( \frac{\partial}{\partial t} v(\xi, t) \right) e^t + v(\xi, t) e^t = \left( \frac{\partial^2}{\partial \xi^2} v(\xi, t) \right) e^t + \xi \left( \frac{\partial}{\partial \xi} v(\xi, t) \right) e^t + v(\xi, t) e^t$$

> addcoords(T1, [x,t], [x*exp(-t),t]):
> PDEchangecoords(%,[xi,t],T1);

```

$$(\xi e^t - \xi e^{(-t)} (e^t)^2) \left( \frac{\partial}{\partial \xi} v(\xi, t) \right) + \left( \frac{\partial}{\partial t} v(\xi, t) \right) e^t - (e^t)^3 \left( \frac{\partial^2}{\partial \xi^2} v(\xi, t) \right) = 0$$

```

> expand(%/exp(t)^3);

$$\frac{\frac{\partial}{\partial t} v(\xi, t)}{(e^t)^2} - \left( \frac{\partial^2}{\partial \xi^2} v(\xi, t) \right) = 0$$

> pde := combine(% , exp);

$$pde := e^{(-2t)} \left( \frac{\partial}{\partial t} v(\xi, t) \right) - \left( \frac{\partial^2}{\partial \xi^2} v(\xi, t) \right) = 0$$


```

So, we arrive at the PDE

$$e^{-2t} \frac{\partial v}{\partial t} = \frac{\partial^2 v}{\partial \xi^2}.$$

Next, we introduce a new variable  $\tau$  by  $\tau = \frac{1}{2}e^{2t}$ . The PDE is in the new coordinate after some simplification in the format known as the diffusion equation.

```

> addcoords(T2, [x,t], [x,1/2*ln(2*t)]):
> PDEchangecoords(pde, [xi,t], T2, [xi,tau]):
> diffusionPDE := simplify(%);

```

$$diffusionPDE := \left( \frac{\partial}{\partial \tau} v(\xi, \tau) \right) - \left( \frac{\partial^2}{\partial \xi^2} v(\xi, \tau) \right) = 0$$

What we have actually proven is that a solution  $u(x, t)$  of the Fokker-Planck equation can be written as

$$u(x, t) = v(xe^t, \frac{e^{2t}}{2}) e^t,$$

where  $v(\xi, \tau)$  is solution of the diffusion equation.

For the **dchange** fans, we show how the computation can be done in this style:

```

> FockerPlanckPDE;

$$\frac{\partial}{\partial t} u(x, t) = \left( \frac{\partial^2}{\partial x^2} u(x, t) \right) + x \left( \frac{\partial}{\partial x} u(x, t) \right) + u(x, t)$$

> dchange({u(x,t) = V(x*exp(t),t)*exp(t)}, 
> FockerPlanckPDE, [V]);

$$(D_1(V)(x e^t, t) x e^t + D_2(V)(x e^t, t)) e^t + V(x e^t, t) e^t =$$


$$D_{1,1}(V)(x e^t, t) (e^t)^3 + x D_1(V)(x e^t, t) (e^t)^2 + V(x e^t, t) e^t$$

> expand((lhs-rhs)(%) / exp(t));

$$D_2(V)(x e^t, t) - (e^t)^2 D_{1,1}(V)(x e^t, t)$$

> simplify(dchange({x=xi*exp(-t)}, %, [xi]));

$$D_2(V)(\xi, t) - e^{(2t)} D_{1,1}(V)(\xi, t)$$

> dchange({t=ln(2*tau)/2, V(xi,t)=v(xi,tau)}, %, [tau,v]);

$$2 \left( \frac{\partial}{\partial \tau} v(\xi, \tau) \right) \tau - 2 \tau \left( \frac{\partial^2}{\partial \xi^2} v(\xi, \tau) \right)$$


```

```
> normal(%/(2*tau))=0;

$$\left(\frac{\partial}{\partial \tau} v(\xi, \tau)\right) - \left(\frac{\partial^2}{\partial \xi^2} v(\xi, \tau)\right) = 0$$

```

## 17.9 Perturbation Methods

When you want to apply perturbation methods to find approximate solutions of ODEs, computer algebra systems are most valuable computational tools. In this section, we shall describe two classical methods, viz., the Poincaré-Lindstedt method and the method of multiple scales; and we shall apply them to the van der Pol equation. The interested reader is also referred to [199, 200], which contain many examples of the use of the computer algebra system MACSYMA in perturbation and bifurcation theory. Mathematical background can be found in [184].

### Poincaré-Lindstedt Method

The van der Pol equation is

$$y'' - \varepsilon(1 - y^2)y' + y = 0.$$

For  $\varepsilon = 0$  it is the ODE of the mathematical pendulum. We already investigated the case  $\varepsilon = 1$  numerically in §17.6. For any  $\varepsilon$ , this differential equation possesses an asymptotically stable periodic solution called the *limit cycle*.

We want to compute a good approximation of the limit cycle for small  $\varepsilon$ . Because there are no explicit time dependent terms in the van der Pol equation we can choose without loss of generality the point that corresponds with  $t = 0$ ; we shall choose the initial value  $y(0) = 0$ . In the Poincaré-Lindstedt method, time is stretched via the transformation

$$\tau = \omega t,$$

where

$$\omega = 1 + \omega_1\varepsilon + \omega_2\varepsilon^2 + \omega_3\varepsilon^3 + \dots$$

Then the van der Pol equation for  $y(\tau)$  becomes

$$\omega^2 y'' - \omega\varepsilon(1 - y^2)y' + y = 0.$$

Let us verify this with Maple.

```
> PDEtools[declare](y(t), prime=tau);
```

*y(t) will now be displayed as y  
derivatives with respect to : tau of functions of one variable will now \  
be displayed with '*

```

> diff(y(t),t$2) - epsilon*(1-y(t)^2)*diff(y(t),t)
>      + y(t) = 0;

$$y_{tt} - \epsilon(1 - y^2)y_t + y = 0$$

> ODE := DEtools[Dchangevar](  

> {t=tau/omega, y(t)=y(tau)}, %, t, tau);

$$ODE := \omega^2 y'' - \epsilon(1 - y(\tau)^2)\omega y' + y(\tau) = 0$$


```

We assume that the solution  $y(\tau)$  can expanded in a Taylor series in  $\epsilon$ ,

$$y(\tau) = y_0(\tau) + y_1(\tau)\epsilon + y_2(\tau)\epsilon^2 + y_3(\tau)\epsilon^3 + \dots$$

Then we substitute the expansions of  $y(\tau)$  and  $\omega(\epsilon)$  in the van der Pol equation, collect terms in  $\epsilon$ , and equate to zero the coefficient of each power of  $\epsilon$ . The equations for small orders of  $\epsilon$  are

$$\begin{aligned} y_0'' + y_0 &= 0, \\ y_1'' + y_1 &= y_0'(1 - y_0^2) - 2\omega_1 y_0'', \\ y_2'' + y_2 &= (1 - y_0^2)y_1' - 2y_0 y_1 y_0' - 2\omega_1 y_1'' \\ &\quad - (2\omega_2 + \omega_1^2)y_0'' + \omega_1(1 - y_0)x_0'. \end{aligned}$$

The initial value  $y(0) = 0$  translates into

$$y_0(0) = 0, \quad y_1(0) = 0, \quad y_2(0) = 0, \quad y_3(0) = 0, \dots$$

Let us check some of the differential equations with Maple, at the same time setting the notation for the rest of the session.

```

> e_order := 6:
> macro(e=epsilon, t=tau):
> alias(seq(y[i]=eta[i](tau), i=0..e_order)):
> e := () -> e: # introduce e as a constant function
> for i from 0 to e_order do
>   eta[i] := t -> eta[i](t)
> end do:
> omega := 1 + sum('w[i]*e^i', 'i'=1..e_order);


$$\omega := 1 + w_1 \epsilon + w_2 \epsilon^2 + w_3 \epsilon^3 + w_4 \epsilon^4 + w_5 \epsilon^5 + w_6 \epsilon^6$$

> y := sum('eta[i]*e^i', 'i'=0..e_order);


$$y := \eta_0 + \eta_1 \epsilon + \eta_2 \epsilon^2 + \eta_3 \epsilon^3 + \eta_4 \epsilon^4 + \eta_5 \epsilon^5 + \eta_6 \epsilon^6$$

> deqn := simplify(collect(ODE,e), {e^(e_order+1)=0}):
> for i from 0 to e_order do
>   ode[i] := coeff(lhs(deqn), e, i) = 0
> end do:
> ode[0];


$$y_0 + \eta_0'' = 0$$

> ode[1];


$$2 w_1 \eta_0'' + \eta_1'' - \eta_0' + y_1 + \eta_0' y_0^2 = 0$$


```

```
> ode[2];
```

$$\begin{aligned} \eta_0'' w_1^2 + \eta_0' w_1 y_0^2 + 2\eta_0' y_0 y_1 + 2w_1 \eta_1'' - \eta_1' + \eta_2'' + y_2 \\ - \eta_0' w_1 + \eta_1' y_0^2 + 2\eta_0'' w_2 = 0 \end{aligned}$$

The initial value problem for  $\eta_0(\tau)$  can easily be solved.

```
> dsolve({ode[0], eta[0](0)=0, D(eta[0])(0)=C[1]},  
        eta[0](t), method=laplace);
```

$$y_0 = C_1 \sin(\tau)$$

We assign this function and proceed with the differential equation for  $\eta_1(\tau)$ .

```
> eta[0] := unapply(rhs(%), t);
```

$$\eta_0 := \tau \rightarrow C_1 \sin(\tau)$$

```
> ode[1];
```

$$-2w_1 C_1 \sin(\tau) + \eta_1'' - C_1 \cos(\tau) + y_1 + C_1^3 \cos(\tau) \sin(\tau)^2 = 0$$

We compute the finite Fourier series with **combine**(..., 'trig').

```
> map(combine, ode[1], 'trig');
```

*ode*<sub>1</sub> :=

$$-2w_1 C_1 \sin(\tau) + (-C_1 + \frac{1}{4} C_1^3) \cos(\tau) - \frac{1}{4} C_1^3 \cos(3\tau) + \eta_1'' + y_1 = 0$$

The  $\sin \tau$  and  $\cos \tau$  terms are called the *resonant terms* or *secular terms*; they are responsible for nonperiodic behavior of the approximation — as is clear from the general solution below.

```
> dsolve({ode[1], eta[1](0)=0, D(eta[1])(0)=C[2]},  
        eta[1](t), method=laplace);
```

$$\begin{aligned} y_1 &= (\frac{1}{32} C_1^3 - C_1 \tau w_1) \cos(\tau) \\ &+ (-\frac{1}{8} C_1 (C_1 - 2) (C_1 + 2) \tau + w_1 C_1 + C_2) \sin(\tau) - \frac{1}{32} C_1^3 \cos(3\tau) \\ &> \text{map(collect, \% , [sin(t), cos(t), t]);} \end{aligned}$$

$$\begin{aligned} y_1 &= (\frac{1}{32} C_1^3 - C_1 \tau w_1) \cos(\tau) \\ &+ (-\frac{1}{8} C_1 (C_1 - 2) (C_1 + 2) \tau + w_1 C_1 + C_2) \sin(\tau) - \frac{1}{32} C_1^3 \cos(3\tau) \end{aligned}$$

So, we choose  $C_1$  and  $w_1$  such that these terms vanish.

```
> solve({coeff(lhs(ode[1]), sin(t)) = 0,  
        coeff(lhs(ode[1]), cos(t)) = 0});
```

$$\{C_1 = 0, w_1 = w_1\}, \{C_1 = 2, w_1 = 0\}, \{C_1 = -2, w_1 = 0\}$$

Because we want to compare it with the numerical method that was discussed in §17.6, we choose a negative amplitude.

```
> w[1] := 0: C[1] := -2: ode[1];
```

$$2 \cos(3\tau) + \eta_1'' + y_1 = 0$$

We solve the differential equation for  $\eta_1(\tau)$  with initial values  $\eta_1(0) = 0$ ,  $\eta_1'(0) = C_2$ .

```
> dsolve({ode[1], eta[1](0)=0, D(eta[1])(0)=C[2]},  
        eta[1](t), method=laplace);
```

$$y_1 = -\frac{1}{4} \cos(\tau) + C_2 \sin(\tau) + \frac{1}{4} \cos(3\tau)$$

```
> eta[1] := unapply(rhs(%), tau);
```

$$\eta_1 := \tau \rightarrow -\frac{1}{4} \cos(\tau) + C_2 \sin(\tau) + \frac{1}{4} \cos(3\tau)$$

Similarly, we deal with  $C_2$  and  $\eta_2(\tau)$ . We omit comments.

```
> map(combine, ode[2], 'trig'):  
> ode[2] := map(collect, %,  
      [sin(t), sin(3*t), cos(t), cos(3*t)]);
```

$$\begin{aligned}ode_2 := & \left( \frac{1}{4} + 4w_2 \right) \sin(\tau) - \frac{3}{2} \sin(3\tau) + \eta_2'' \\& + 2C_2 \cos(\tau) - 3C_2 \cos(3\tau) + \frac{5}{4} \sin(5\tau) + y_2 = 0\end{aligned}$$

```
> solve({coeff(lhs(ode[2]), sin(t)) = 0,  
        coeff(lhs(ode[2]), cos(t)) = 0});
```

$$\{C_2 = 0, w_2 = \frac{-1}{16}\}$$

```
> assign(%):  
> dsolve({ode[2], eta[2](0)=0, D(eta[2])(0)=C[3]},  
        eta[2](t), method=laplace):  
> collect(%,[sin(t), sin(3*t), sin(5*t),  
      cos(t), cos(3*t), cos(5*t)]):  
> eta[2] := unapply(rhs(%), t);
```

$$\eta_2 := \tau \rightarrow \left( \frac{29}{96} + C_3 \right) \sin(\tau) + \frac{5}{96} \sin(5\tau) - \frac{3}{16} \sin(3\tau)$$

We assume that you understand the pattern for finding the higher order terms and compute them repetitively.

```
> for i from 3 to e_order do  
>   map(combine, ode[i], 'trig'):br/>>   ode[i] := map(collect, %,  
     [seq(sin((2*j+1)*t), j=0..i),  
      seq(cos((2*j+1)*t), j=0..i)]):
```

```

> solve({coeff(lhs(ode[i]), sin(t)) = 0,
> coeff(lhs(ode[i]), cos(t)) = 0}):
> assign(%):
> dsolve({ode[i], eta[i](0)=0, D(eta[i])(0)=C[i+1]},
> eta[i](t), method=laplace):
> collect(%, [seq(sin((2*j+1)*t), j=0..i),
> seq(cos((2*j+1)*t), j=0..i)]):
> eta[i] := unapply(rhs(%), t)
> end do:
```

Let us look at the final results.

```
> omega;
```

$$\begin{aligned}
& 1 - \frac{1}{16} \varepsilon^2 + \frac{17}{3072} \varepsilon^4 + \frac{35}{884736} \varepsilon^6 \\
> y(t); \text{ # some higher order terms in output omitted} \\
& -2 \sin(\tau) + \left(-\frac{1}{4} \cos(\tau) + \frac{1}{4} \cos(3\tau)\right) \varepsilon + \left(\frac{5}{96} \sin(5\tau) - \frac{3}{16} \sin(3\tau)\right) \varepsilon^2 \\
& + \left(-\frac{7}{576} \cos(7\tau) - \frac{21}{256} \cos(3\tau) + \frac{5}{72} \cos(5\tau) + \frac{19}{768} \cos(\tau)\right) \varepsilon^3 + \dots \\
& + \left(-\frac{143191}{35389440} \sin(3\tau) - \frac{6017803}{2654208000} \sin(11\tau) + \frac{469795}{31850496} \sin(5\tau)\right. \\
& \left. + \left(-\frac{148447039}{55738368000} + C_7\right) \sin(\tau) + \frac{715247}{3715891200} \sin(13\tau)\right. \\
& \left. - \frac{6871193}{398131200} \sin(7\tau) + \frac{690583}{73728000} \sin(9\tau)\right) \varepsilon^6
\end{aligned}$$

The constant  $C_7$  would be determined in the next step. We shall only consider  $y(\tau)$  up to order 5; results of computations up to order 164 can be found in [8].

```
> y := unapply(simplify(y(t), {e^e_order=0}), t):
```

Let us plot this function for  $\varepsilon = 1$ .

```
> e := 1: y(t);
```

$$\begin{aligned}
& -\frac{1037927}{552960} \sin(\tau) + \frac{5257957}{3317760} \cos(\tau)^3 - \frac{799991}{1105920} \cos(\tau) - \frac{81}{32} \sin(\tau) \cos(\tau)^4 \\
& + \frac{13}{256} \sin(\tau) \cos(\tau)^2 + \frac{1519}{540} \sin(\tau) \cos(\tau)^6 - \frac{61}{80} \sin(\tau) \cos(\tau)^8 \\
& - \frac{1212373}{259200} \cos(\tau)^5 - \frac{114941}{28800} \cos(\tau)^9 + \frac{912187}{129600} \cos(\tau)^7 + \frac{5533}{7200} \cos(\tau)^{11} \\
> plot(y(t), t=0..14);
\end{aligned}$$

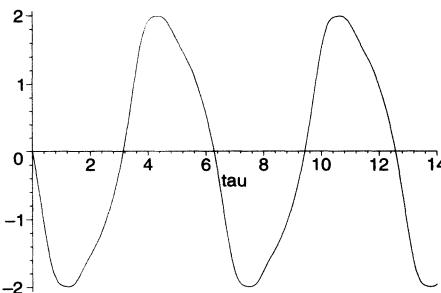


Figure 17.28. Solution of the van der Pol equation by the Poincaré-Lindstedt method.

Compare this with the numerical solution found in §17.6.

### Method of Multiple Scales

The Poincaré-Lindstedt method is useful for finding periodic solutions of ODEs, but it does not help to approximate general solutions of ODEs in the neighborhood of a limit cycle. In the method of multiple scales, the solution of an ODE is not viewed as a function of one variable  $t$  but as a function of two or more independent variables  $t_1, t_2, t_3, \dots$  defined as

$$t_1 = t, \quad t_2 = \varepsilon t, \quad t_3 = \varepsilon^2 t, \quad \dots$$

Then:

$$\begin{aligned} \frac{d}{dt} &= \frac{\partial}{\partial t_1} + \varepsilon \frac{\partial}{\partial t_2} + \varepsilon^2 \frac{\partial}{\partial t_3} + \dots, \\ \frac{d^2}{dt^2} &= \frac{\partial^2}{\partial t_1^2} + 2\varepsilon \frac{\partial^2}{\partial t_1 \partial t_2} + \varepsilon^2 \left( \frac{\partial^2}{\partial t_2^2} + 2 \frac{\partial^2}{\partial t_1 \partial t_3} \right) + \dots \end{aligned}$$

You can verify this with Maple (you can compute higher order terms by increasing the variable `e_order`).

```
> restart: alias(epsilon=e,
>     seq(y[i] = eta[i](seq(t[j], j=1..3)), i=0..2)):
> macro(t1=t[1], t2=t[2], t3=t[3]): e_order := 2:
> e := subs([variables=seq(u||j, j=0..e_order),
>     body=e], (variables->body)):
> subs(D=sum(e^(i-1)*D[i], i=1..e_order+1),
>     (D@@e_order)(y)):
> simplify(collect(%,e), {e^(e_order+1)=0});
```

$$D_{1,1}(y) + 2\varepsilon D_{1,2}(y) + (2D_{1,3}(y) + D_{2,2}(y))\varepsilon^2$$

In classical notation:

```
> convert(%(seq(t[j], j=1..e_order+1)), diff);
```

$$\left( \frac{\partial^2}{\partial t_1^2} \%1 \right) + 2\varepsilon \left( \frac{\partial^2}{\partial t_1 \partial t_2} \%1 \right) + \left( 2 \left( \frac{\partial^2}{\partial t_1 \partial t_3} \%1 \right) + \left( \frac{\partial^2}{\partial t_2^2} \%1 \right) \right) \varepsilon^2$$

$$\%1 := y(t_1, t_2, t_3)$$

In the method of multiple scales, the solution  $y$  of an ODE is expanded in a power series of  $\varepsilon$  as

$$y = y_0(t_1, t_2, t_3, \dots) + \varepsilon y_1(t_1, t_2, t_3, \dots) + \varepsilon^2 y_2(t_1, t_2, t_3, \dots) + \dots$$

This leads to differential equations for  $y_0$ ,  $y_1$ ,  $y_2$ , etc.

We shall apply the three-variables method to the van der Pol equation

$$y'' - \varepsilon(1 - y^2)y' + y = 0,$$

with initial values  $y(0) = 0$ ,  $y'(0) = -1/10$ . We shall actually redo work of Noble and Husain [185].

First, we derive the differential equations for  $y_0(t_0, t_1, t_2)$ ,  $y_1(t_0, t_1, t_2)$ , and  $y_2(t_0, t_1, t_2)$  with Maple.

```
> ODE := (D@@2)(y) - e*(1-y^2)*D(y) + y = 0;
ODE := (D^(2))(y) - ε(1 - y^2) D(y) + y = 0
> subs(D=sum('e^(i-1)*D[i]', 'i'=1..e_order+1), ODE):
> y := sum('eta[i]*e^i', 'i'=0..e_order);
y := η₀ + η₁ ε + η₂ ε²
> diffeqn := simplify(collect(%%, e), {e^(e_order+1)=0}):
> for i from 0 to e_order do
>   ode[i] := coeff(lhs(diffeqn), e, i) = 0
> end do;
ode₀ := D_{1, 1}(η₀) + η₀ = 0
ode₁ := D_{1, 1}(η₁) + 2 D_{1, 2}(η₀) + η₁ - D₁(η₀) + D₁(η₀) η₀² = 0
ode₂ := -D₁(η₁) - D₂(η₀) + η₀² D₁(η₁) + η₀² D₂(η₀) + 2 η₀ η₁ D₁(η₀)
+ 2 D_{1, 2}(η₁) + η₂ + D_{2, 2}(η₀) + D_{1, 1}(η₂) + 2 D_{1, 3}(η₀) = 0
```

So, the required differential equations are:

$$\begin{aligned} \frac{\partial^2 y_0}{\partial t_1^2} + y_0 &= 0, \\ \frac{\partial^2 y_0}{\partial t_1^2} + y_1 &= -2 \frac{\partial^2 y_0}{\partial t_1 \partial t_2} + (1 - y_0^2) \frac{\partial y_0}{\partial t_1}, \\ \frac{\partial^2 y_2}{\partial t_1^2} + y_2 &= -2 \frac{\partial^2 y_0}{\partial t_1 \partial t_2} - \frac{\partial^2 y_0}{\partial t_2^2} - 2 \frac{\partial^2 y_0}{\partial t_1 \partial t_3} + \\ &\quad (1 - y_0^2) \left( \frac{\partial y_1}{\partial t_1} + \frac{\partial y_0}{\partial t_2} \right) - 2 y_0 y_1 \frac{\partial y_0}{\partial t_1}. \end{aligned}$$

The first equation has the general solution

$$y_0(t_1, t_2, t_3) = A(t_2, t_3) \sin(t_1 + B(t_2, t_3)).$$

Substitution in the next equation, give the following differential equations for  $A(t_2, t_3)$  and  $B(t_2, t_3)$ , if we insist on removal of resonant terms.

$$\begin{aligned}\frac{\partial B(t_2, t_3)}{\partial t_2} &= 0, \\ \frac{\partial A(t_2, t_3)}{\partial t_2} &= \frac{1}{2}A(t_2, t_3) - \frac{1}{8}A(t_2, t_3)^3.\end{aligned}$$

Let us verify this with Maple.

```
> ode[1] := convert(ode[1](t1,t2,t3), diff);
ode1 := ( $\frac{\partial^2}{\partial t_1^2} y_1$ ) + 2( $\frac{\partial^2}{\partial t_1 \partial t_2} y_0$ ) +  $y_1$  - ( $\frac{\partial}{\partial t_1} y_0$ ) + ( $\frac{\partial}{\partial t_1} y_0$ ) $y_0^2$  = 0
> eta[0] := (t1,t2,t3) -> A(t2,t3) * sin(t1+B(t2,t3));
eta0 := (t1, t2, t3) → A(t2, t3) sin(t1 + B(t2, t3))
> combine(ode[1], 'trig'): ode[1] :=
> collect(%, [sin(t1+B(t2,t3)),cos(t1+B(t2,t3))]);
ode1 := -2 A(t2, t3) sin(t1 + B(t2, t3)) ( $\frac{\partial}{\partial t_2}$  B(t2, t3))
+ (2 ( $\frac{\partial}{\partial t_2}$  A(t2, t3)) - A(t2, t3) +  $\frac{1}{4}$  A(t2, t3) $^3$ ) cos(t1 + B(t2, t3))
+ ( $\frac{\partial^2}{\partial t_1^2} y_1$ ) -  $\frac{1}{4}$  A(t2, t3) $^3$  cos(3 t1 + 3 B(t2, t3)) +  $y_1$  = 0
```

The resonant terms are

$$\sin(t_1 + B(t_2, t_3)), \cos(t_1 + B(t_2, t_3)).$$

By insisting that coefficients of resonant terms are equal to zero we get the following differential equations.

```
> restrictions := {
>   coeff(lhs(ode[1]), cos(t1+B(t2,t3))) = 0,
>   coeff(lhs(ode[1]), sin(t1+B(t2,t3))) = 0};

restrictions := {2 ( $\frac{\partial}{\partial t_2}$  A(t2, t3)) - A(t2, t3) +  $\frac{1}{4}$  A(t2, t3) $^3$  = 0,
-2 A(t2, t3) ( $\frac{\partial}{\partial t_2}$  B(t2, t3)) = 0}
> 2*diff(F(t),t) - F(t) + 1/4*F(t) $^3$  = 0;
2 ( $\frac{d}{dt}$  F(t)) - F(t) +  $\frac{1}{4}$  F(t) $^3$  = 0
> simplify([dsolve(%, F(t))], symbolic);
[F(t) =  $\frac{2 e^{(1/2)t}}{\sqrt{e^t + 4\_C1}}$ , F(t) = - $\frac{2 e^{(1/2)t}}{\sqrt{e^t + 4\_C1}}]$ 
> simplify(subs(_C1=1/4*exp(t)*C, %), symbolic):
> subs(C=C*exp(-t), %);
```

$$[F(t) = \frac{2}{\sqrt{1 + C e^{(-t)}}}, F(t) = -\frac{2}{\sqrt{1 + C e^{(-t)}}}]$$

It follows that we can take

$$A(t_2, t_3) = \frac{-2}{\sqrt{1 + C(t_3)e^{-t_2}}}.$$

Moreover, we can take  $B(t_2, t_3) = B(t_3)$ . Then, the differential equation satisfied by  $y_1$  becomes

```
> simplify(ode[1], restrictions, convert(
>   [D[1](B)(t2,t3), D[1](A)(t2,t3), A(t2,t3)], diff)):
> combine(%, 'trig');
```

$$y_1 + \left(\frac{\partial^2}{\partial t_1^2} y_1\right) - \frac{1}{4} A(t_2, t_3)^3 \cos(3t_1 + 3B(t_2, t_3)) = 0$$

Now comes a tricky point in the computation: we ignore the solution of the homogeneous differential equation and choose a particular solution  $y_1$  which has a simple form.

```
> eta[1] := (t1, t2, t3) -> 1/32 * A(t2, t3)^3
>   * sin(3*t1+3*B(t2, t3)+3/2*Pi);
```

$$\eta_1 := (t1, t2, t3) \rightarrow \frac{1}{32} A(t2, t3)^3 \sin(3t1 + 3B(t2, t3) + \frac{3}{2}\pi)$$

```
> %%; # verify the solution
```

$$0 = 0$$

We take into account that  $B$  does not depend on  $t_2$  and substitute  $y_1$  into the third differential equation.

```
> eta[1] := 'eta[1]': eta[0] := 'eta[0]':
> ode[2] := convert(ode[2](t1, t2, t3), diff):
> eta[1] := (t1, t2, t3) -> 1/32 * A(t2, t3)^3
>   * sin(3*t1+3*B(t3)+3/2*Pi);
η1 := (t1, t2, t3) → 1/32 A(t2, t3)^3 sin(3t1 + 3B(t3) + 3/2π)
> eta[0] := (t1, t2, t3) -> A(t2, t3) * sin(t1 + B(t3));
η0 := (t1, t2, t3) → A(t2, t3) sin(t1 + B(t3))
> combine(ode[2], 'trig'):
> ode[2] := collect(% , [sin(t1+B(t3)), cos(t1+B(t3))]):
> conditions := {coeff(lhs(ode[2]), cos(t1+B(t3))) = 0,
>   coeff(lhs(ode[2]), sin(t1+B(t3))) = 0};
```

$$conditions := \{2 \left(\frac{\partial}{\partial t_3} A(t_2, t_3)\right) = 0, -\frac{1}{128} A(t_2, t_3)^5 + \left(\frac{\partial^2}{\partial t_2^2} A(t_2, t_3)\right) \\ - \left(\frac{\partial}{\partial t_2} A(t_2, t_3)\right) + \frac{3}{4} A(t_2, t_3)^2 \left(\frac{\partial}{\partial t_2} A(t_2, t_3)\right) - 2 A(t_2, t_3) \left(\frac{d}{dt_3} B(t_3)\right) = 0\}$$

One of the conditions means that  $A(t_2, t_3)$  does not depend on  $t_3$  (so,  $C(t_3)$  is a constant). From the earlier restrictions, we can derive a necessary condition for  $\frac{\partial^2 A(t_2, t_3)}{\partial t_2^2}$ .

```
> op(remove(has, restrictions, B));

$$2 \left( \frac{\partial}{\partial t_2} A(t_2, t_3) \right) - A(t_2, t_3) + \frac{1}{4} A(t_2, t_3)^3 = 0$$

> diff(%, t[2]);

$$2 \left( \frac{\partial^2}{\partial t_2^2} A(t_2, t_3) \right) - \left( \frac{\partial}{\partial t_2} A(t_2, t_3) \right) + \frac{3}{4} A(t_2, t_3)^2 \left( \frac{\partial}{\partial t_2} A(t_2, t_3) \right) = 0$$

```

Together with the second last restriction, we can rewrite the above conditions.

```
> simplify(conditions, {%, %%}, convert(
> [D[1,1](A)(t2,t3), D[1](A)(t2,t3),
> A(t2,t3)], diff));

$$\{2 \left( \frac{\partial}{\partial t_3} A(t_2, t_3) \right) = 0,$$


$$-\frac{7}{128} A(t_2, t_3)^5 + (-2 \left( \frac{d}{dt_3} B(t_3) \right) - \frac{1}{4}) A(t_2, t_3) + \frac{1}{4} A(t_2, t_3)^3 = 0\}$$

> op(select(has, %, B));

$$-\frac{7}{128} A(t_2, t_3)^5 + (-2 \left( \frac{d}{dt_3} B(t_3) \right) - \frac{1}{4}) A(t_2, t_3) + \frac{1}{4} A(t_2, t_3)^3 = 0$$

> 'diff(B(t3), t3)' = solve(% , diff(B(t3), t3));

$$\frac{\partial}{\partial t_3} B(t_3) = -\frac{7}{256} A(t_2, t_3)^4 - \frac{1}{8} + \frac{1}{8} A(t_2, t_3)^2$$

```

Because  $A(t_2, t_3)$  does not really depend on  $t_3$  we get

$$B(t_3) = -\frac{1}{8} \left( 1 - A(t_2, t_3) + \frac{7}{32} A(t_2, t_3)^4 \right) t_3 + B_0,$$

where  $B_0$  is a constant. Strictly speaking, the formula for  $B(t_3)$  contradicts an earlier restriction, viz.,  $B$  does not depend on  $t_2$ . But  $A$  depends on  $t_2$ ! However, the formula for  $A$  shows that it is a slowly varying function of  $t$ , and as long as  $t$  is not too large we may assume that  $A$  is a constant. For a more thorough discussion of how to avoid this contradiction, the interested reader is referred to [185].

Let us see what the approximation is for the initial values  $y(0) = 0$ ,  $y'(0) = -0.1$ , if we only consider the approximation  $y = y_0 + \varepsilon y_1$  and set  $\varepsilon$  equal to one.

```
> restart: alias(e=epsilon): y := eta[0] + e*eta[1];
y :=  $\eta_0 + e \eta_1$ 
> eta[0] := t -> A(t)*sin(t+B(t));
```

```

 $\eta_0 := t \rightarrow A(t) \sin(t + B(t))$ 
> eta[1] := t -> -1/32*A(t)^3*cos(3*t+3*B(t));
 $\eta_1 := t \rightarrow -\frac{1}{32} A(t)^3 \cos(3t + 3B(t))$ 
> B := t-> -1/8*(1-A(t)^2+7/32*A(t)^4)*e^2*t+b;
 $B := t \rightarrow -\frac{1}{8} (1 - A(t)^2 + \frac{7}{32} A(t)^4) e^2 t + b$ 
> A := t-> -2/(1+c*exp(-e*t));
 $A := t \rightarrow -\frac{2}{1 + c e^{-et}}$ 
> e := 1: y := unapply(y(t), t):
> fsolve({y(0)=0, D(y)(0)=-0.1}, {b,c}, {b=-0.1..0.1});
{b = 0.0004073638406, c = 16.51715658}
> assign(%):

```

Let us plot this function.

```
> plot(y(t), t=0..20);
```

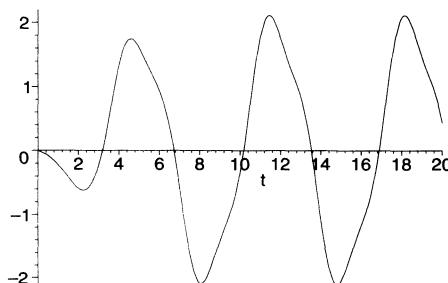


Figure 17.29. Solution of the van der Pol equation by the method of multiple scales.

## 17.10 Partial Differential Equations

For one unknown function  $u$  of two independent variables, say  $x$  and  $t$ , the most general partial differential equation (abbreviated PDE) is an equation of the form

$$F(x, t, u, u_x, u_t, u_{xx}, u_{xt}, u_{tt}, \dots),$$

where the subscripts are a standard notation for partial differentiation with respect to the indicated variables:

$$u_x = \frac{\partial u}{\partial x}, \quad u_{xt} = \frac{\partial^2 u}{\partial x \partial t}, \quad u_{tt} = \frac{\partial^2 u}{\partial t^2}, \dots$$

The *order* of the PDE is defined in analogy with that for an ODE as the highest-order derivative appearing in the PDE. So, the most general first-order PDE may be written as a function of  $x$ ,  $t$ ,  $u$ ,  $u_x$ , and  $u_t$ :

$$F(x, t, u, u_x, u_t) = 0.$$

If  $F$  is a polynomial equation of degree  $r$  in the highest-order partial derivative, then the PDE is said to be of *degree r*. In particular, if  $F$  is linear, then the PDE is also called *linear*. An equation which is of first degree in the highest-order partial derivative but contains other nonlinear terms is called *quasi-linear*. Some famous examples:

$u_t - u_{xx} = 0$	heat equation or diffusion equation: a linear PDE of order 2,
$u_{tt} - u_{xx} = 0$	wave equation: a linear PDE of order 2,
$u_t - u u_x + u_{xxx} = 0$	Korteweg-de Vries equation: a quasi-linear PDE of order 3 and degree 1.

The most general linear partial differential equation of second order in two independent variables, say  $x$  and  $y$ , has the form

$$A \frac{\partial^2 u}{\partial x^2} + 2B \frac{\partial^2 u}{\partial x \partial y} + C \frac{\partial^2 u}{\partial y^2} + D \frac{\partial u}{\partial x} + E \frac{\partial u}{\partial y} + Fu + G = 0,$$

where  $A, \dots, G$  are functions of  $x$  and  $y$ . Equations of this form are classified into three types:

elliptic:  $B^2 - AC < 0$ ,  $A\xi^2 + 2B\xi\eta + C\eta^2 = 1$  is an ellipse;

parabolic:  $B^2 - AC = 0$ ,  $A\xi^2 + 2B\xi\eta + C\eta^2 + D\xi + E\eta = 0$  is a parabola;

hyperbolic:  $B^2 - AC > 0$ ,  $A\xi^2 + 2B\xi\eta + C\eta^2 = 1$  is a hyperbola.

An equation can be of one type in part of the  $x$ - $y$  plane and of another type in a second part. The three types are illustrated respectively by

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0, \quad \frac{\partial u}{\partial x} + \frac{\partial^2 u}{\partial y^2} = 0, \quad \frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = 0,$$

i.e., by the Laplace equation, heat or diffusion equation, and wave equation, respectively.

We have already seen that PDEs can be solved graphically by the procedure **PDEplot** from the **PDEtools** package. But Maple can also solve some PDEs analytically. The procedure **pdsolve** (partial differential equation solver) does the job. Some examples:

```
> restart; with(PDEtools): declare(u(x,t), quiet):
> wavePDE := diff(u(x,t),[x$2]) - 1/v^2*diff(u(x,t),[t$2])
>      = 0;
wavePDE :=  $u_{x,x} - \frac{u_{t,t}}{v^2} = 0$ 
> pdsolve(wavePDE);
```

$$u = \_F1(v t + x) + \_F2(v t - x)$$

Here,  $\_F1$  and  $\_F2$  are arbitrary functions.

The **pdsolve** procedure uses the following strategy of solving PDEs:

1. For first and second order PDEs, if it is perfectly clear that a particular algorithm such as the characteristic strip method, a mapping between PDEs, or pattern matching applies, then such a deterministic method is used.
2. Other PDEs are first tackled by separation of variables. Here, the idea is to build a system of uncoupled ODEs equivalent to the original PDE. A heuristic method tries to find a suitable ansatz; for example, for an indeterminate function  $f(x, y)$ , the general sum of possible products is

$$f_1(x) + f_2(y) + f_1(x)f_2(y).$$

Separation by sum and by product are obtained by selecting only the first two terms or just the last term, respectively.

3. If **pdsolve** is not able to find a suitable ansatz or makes an unwanted choice, then you can make a hint of how to separate variables.

To see how a hint can be made, let us explicitly try separation of variables by product, say  $u(x, t) = f(x)g(t)$ , and in this way find the so-called normal modes of waves.

```
> pdsolve(wavePDE, HINT=f(x)*g(t));
```

$$(u = f(x)g(t)) \& \text{where} [\{g_{t,t} = -c_1 g(t)v^2, f_{x,x} = -c_1 f(x)\}]$$

Separation was successful: Maple returns the two ODEs for  $f$  and  $g$ . By default, the system does not solve the ODEs automatically. Maybe you want to see the ODEs first and then decide if this is a promising way to go or how to adapt it first a bit. For example, you might prefer the expression  $\lambda^2$  instead of  $-c_1$  to get the ODE of a mathematical pendulum for  $f$  and  $g$ , respectively, in ‘standard’ form.

```
> eval(% , _c[1]=-lambda^2);
```

$$(u = f(x)g(t)) \& \text{where} [\{g_{t,t} = -\lambda^2 g(t)v^2, f_{x,x} = -\lambda^2 f(x)\}]$$

If you are happy enough with the separation of variables, solving the ODEs is started off by the procedure **build**.

```
> sol := build(%);
```

$$\begin{aligned} sol := u = & -C3 \sin(v \lambda t) - C1 \sin(\lambda x) + -C3 \sin(v \lambda t) - C2 \cos(\lambda x) \\ & + -C4 \cos(v \lambda t) - C1 \sin(\lambda x) + -C4 \cos(v \lambda t) - C2 \cos(\lambda x) \end{aligned}$$

Let us determine the standing waves between fixed points  $x = 0$  and  $x = \pi$ , i.e., let us determine the solution with initial conditions  $u(0, t) = 0$  and  $u(\pi, t) = 0$ , for all  $t$ .

```

> eval(rhs(sol), x=0) = 0;
      _C3 sin(v λ t) _C2 + _C4 cos(v λ t) _C2 = 0
> eq1 := eval(%, t=0);
      eq1 := _C4 _C2 = 0

```

Thus, either  $_C2$  or  $_C4$  is equal to zero. Let us take  $_C2=0$ .

```

> sol := eval(sol, _C2=0 );
      sol := u = _C3 sin(v λ t) _C1 sin(λ x) + _C4 cos(v λ t) _C1 sin(λ x)
> eval(rhs(sol), x=Pi) = 0;
      _C3 sin(v λ t) _C1 sin(λ π) + _C4 cos(v λ t) _C1 sin(λ π) = 0
> eval(%, t=0);
      _C4 _C1 sin(λ π) = 0

```

Thus, the second condition applied for  $t = 0$  implies that either one of the terms in the above product is equal to zero. Applying the second initial condition to other times  $t$  will convince you that in fact  $\lambda$  should satisfy  $\sin(\lambda\pi) = 0$ .

```

> eq2 := select(has, lhs(%), sin) = 0;
      eq2 := sin(λ π) = 0

```

This means that  $\lambda$  should be an integer, say  $n$ . The solution is given by

```

> factor(sol);
      u = _C1 sin(λ x) (_C3 sin(v λ t) + _C4 cos(v λ t))

```

and can be rewritten as

```

> u(x,t)=A*sin(n*x)*sin(n*v*t+beta);
      u = A sin(n x) sin(n v t + β)

```

With the procedure **pdetest** you can check a possible solution of a PDE in the same way as we used **odetest** for ODEs.

```
> pdetest(%, wavePDE);
```

0

The **PDEtools** package contains the utility procedures **separability** and **casesplit** for analyzing and simplifying PDEs, as well as working out different ways of separating variables. The above example with these PDE-tools goes as follows.

```

> restart; with(PDEtools): declare(u(x,t), quiet):
> wavePDE := diff(u(x,t),[x$2]) - 1/v^2*diff(u(x,t),[t$2])
>      = 0;
      wavePDE := u_{x,x} -  $\frac{u_{t,t}}{v^2} = 0$ 

```

First, we check with the **separability** command whether a solution separable by sum exists.

```
> separability(wavePDE, u(x,t));
0
```

The answer 0 indicates that there are no restrictions on separability. In other word a solution separable by sum or product exists. Therefore, we introduce  $u(x, t) = g(x) + h(t)$ .

```
> eval(wavePDE, u(x,t)=g(x)+h(t));

$$g_{x,x} - \frac{h_{t,t}}{v^2} = 0$$

```

The resulting PDE can be split and triangulized with a call to **casesplit**.

```
> casesplit(%);
```

$$[h_{t,t} = g_{x,x} v^2, g_{x,x,x} = 0] \& \text{where } []$$

You see an uncoupled ODE for  $g(x)$ , which can be solved.

```
> dsolve(op([1,2],%), g(x));

$$g(x) = \frac{-C1 x^2}{2} + _C2 x + _C3$$

```

We substitute this result in the ODE for  $h(t)$  and solve the differential equation.

```
> eval(op([1,1], %%), %);

$$h_{t,t} = -C1 v^2$$

> dsolve(%, h(t));

$$h(t) = \frac{-C1 v^2 t^2}{2} + _C2 t + _C3$$

```

So, we have found the following solution of the PDE:

```
> u(x,t) = rhs(%%) + rhs(%);

$$u = \frac{-C1 x^2}{2} + _C2 x + 2 _C3 + \frac{-C1 v^2 t^2}{2} + _C2 t$$

```

Let us check the answer.

```
> pdetest(%, wavePDE);
0
```

Similarly, a solution separable by product can be obtained. The Maple session below illustrates that simplification of intermediate results allows us to steer to the solution found before.

```
> separability(wavePDE, u(x,t), '*');
0
```

```

> eval(wavePDE, u(x,t)=g(x)*h(t));

$$g_{x,x} h(t) - \frac{g(x) h_{t,t}}{v^2} = 0$$

> casesplit(%);

$$[h(t) = 0] \& \text{where } [], [h_{t,t} = \frac{g_{x,x} h(t) v^2}{g(x)}, g_{x,x,x} = \frac{g_{x,x} g_x}{g(x)}]$$


$$\& \text{where } [g(x) \neq 0], [g(x) = 0] \& \text{where } []$$

> case := %[2];

$$case := [h_{t,t} = \frac{g_{x,x} h(t) v^2}{g(x)}, g_{x,x,x} = \frac{g_{x,x} g_x}{g(x)}] \& \text{where } [g(x) \neq 0]$$

> ode := op([1,2], case);

$$ode := g_{x,x,x} = \frac{g_{x,x} g_x}{g(x)}$$

> dsolve(ode, g(x));

$$g(x) = \frac{1}{2} \sqrt{-\left(e^{\left(\frac{x}{-C1}\right)}\right)^2 \left(e^{\left(\frac{-C2}{-C1}\right)}\right)^2 C3 - 2 C3 - \frac{C3}{\left(e^{\left(\frac{x}{-C1}\right)}\right)^2 \left(e^{\left(\frac{-C2}{-C1}\right)}\right)^2}}$$


$$g(x) = -\frac{1}{2} \sqrt{-\left(e^{\left(\frac{x}{-C1}\right)}\right)^2 \left(e^{\left(\frac{-C2}{-C1}\right)}\right)^2 C3 - 2 C3 - \frac{C3}{\left(e^{\left(\frac{x}{-C1}\right)}\right)^2 \left(e^{\left(\frac{-C2}{-C1}\right)}\right)^2}}$$

> simplify(rhs %[1])^2, symbolic;

$$-\frac{1}{4} C3 \left(e^{\frac{2(x+C2)}{-C1}} + 2 + e^{-\frac{2(x+C2)}{-C1}}\right)$$

> convert(%, 'trig');

$$-\frac{1}{4} C3 \left(2 \cosh\left(\frac{2(x+C2)}{-C1}\right) + 2\right)$$

> expand(subs(x=-C1*xi-C2, %));

$$-C3 \cosh(\xi)^2$$

> simplify(sqrt(%), symbolic);

$$I \cosh(\xi) \sqrt{-C3}$$

> part1 := g(x) = simplify(
>   subs(xi=(x+C2)/_C1, _C3=I^2*alpha^2,
>   _C2=(beta+Pi/2)/lambda, _C1=-I/lambda, %), symbolic);

$$part1 := g(x) = \sin(\lambda x + \beta) \alpha$$

> ode := eval(op([1,1], case), part1);

$$ode := h_{t,t} = -\lambda^2 h(t) v^2$$

> dsolve(ode, h(t));

```

$$h(t) = -C1 \sin(v \lambda t) + -C2 \cos(v \lambda t)$$

We write the solution in a different way.

```
> part2 := h(t) = gamma/alpha*sin(lambda*v*t+delta);
```

$$part2 := h(t) = \frac{\gamma \sin(v \lambda t + \delta)}{\alpha}$$

and get as final answer

```
> u(x,t) = rhs(part1) * rhs(part2);
```

$$u = \sin(\lambda x + \beta) \gamma \sin(v \lambda t + \delta)$$

```
> pdetest(%, wavePDE);
```

$$0$$

A few more examples will give you an idea about Maple's capacities in studying PDEs.

```
> restart; with(PDEtools): declare(u(x,t), quiet):
> PDE := a*x*diff(u(x,t),x) + b*t*diff(u(x,t),t) = 0;
```

$$PDE := a x u_x + b t u_t = 0$$

```
> pdsolve(PDE);
```

$$u = F1\left(\frac{t}{x^{\left(\frac{b}{a}\right)}}\right)$$

```
> PDE := a*x^2*diff(u(x,t),x) + b*t^2*diff(u(x,t),t) = 0;
```

$$PDE := a x^2 u_x + b t^2 u_t = 0$$

```
> pdsolve(PDE);
```

$$u = F1\left(-\frac{b t - a x}{t a x}\right)$$

```
> PDE := diff(u(x,t),x$2) - t^2*diff(u(x,t),t$2)
> - t*diff(u(x,t),t)=0;
```

$$PDE := u_{x,x} - t^2 u_{t,t} - t u_t = 0$$

```
> pdsolve(PDE);
```

$$u = F1(t e^x) + F2(t e^{(-x)})$$

```
> pdetest(%, PDE);
```

$$0$$

Another separation of variables is possible.

```
> pdsolve(PDE, HINT=F(x)+G(t));
```

$$(u = F(x) + G(t)) \& \text{where } [G_{t,t} = \frac{-c_1}{t^2} - \frac{G_t}{t}, F_{x,x} = -c_1]$$

```
> build(%);
```

$$u = \frac{1}{2} -c_1 x^2 + -C1 x + -C2 + \frac{1}{2} -c_1 \ln(t)^2 + -C3 \ln(t) + -C4$$

By adding the optional argument **build** to the **pdsolve** command, you ask Maple to immediately try to solve the uncoupled ODEs that are found when separation of variables was successful.

```
> pdsolve(PDE, HINT=F(x)+G(t), build);
u =  $\frac{1}{2} -c_1 x^2 + -C1 x + -C2 + \frac{1}{2} -c_1 \ln(t)^2 + -C3 \ln(t) + -C4$ 
> pdetest(%, PDE);
0
> pdsolve(PDE, HINT=F(x)*G(t), build);
u =  $-C3 t^{(\sqrt{-c_1})} -C1 \%1 + \frac{-C3 t^{(\sqrt{-c_1})} -C2}{\%1} + \frac{-C4 -C1 \%1}{t^{(\sqrt{-c_1})}} + \frac{-C4 -C2}{t^{(\sqrt{-c_1})} \%1}$ 
\%1 :=  $e^{(\sqrt{-c_1} x)}$ 
> pdetest(%, PDE);
0
```

Of course you can always check with the **pdetest** procedure solutions found in different manners.

```
> u(x,t) = F(t*exp(x)) + G(t*exp(-x));
u = F(t ex) + G(t e(-x))
> pdetest(%, PDE);
0
```

Change of coordinates is useful too. Below, we first solve the Laplace equation in two dimensions in Cartesian coordinates by separation of variables and afterwards we transform the equation in polar coordinates and solve in this coordinate system.

```
> restart; with(PDEtools): declare(u(x,y), quiet):
> LaplacePDE :=
> diff(u(x,y), [x$2]) + diff(u(x,y), [y$2]) = 0;
LaplacePDE :=  $u_{x,x} + u_{y,y} = 0$ 
> pdsolve(LaplacePDE, HINT=F(x)*G(y));
(u = F(x) G(y)) &where [{Gy,y = -c1 G(y), Fx,x = -c1 F(x)}]
> build(subs(_c[1]=omega^2, %));

```

$$u = -C3 \sin(\omega y) -C1 e^{(\omega x)} + \frac{-C3 \sin(\omega y) -C2}{e^{(\omega x)}} \\ + -C4 \cos(\omega y) -C1 e^{(\omega x)} + \frac{-C4 \cos(\omega y) -C2}{e^{(\omega x)}}$$

```

> collect(%, {sin,cos}, simplify);


$$u = -C_4 (-C_1 e^{(\omega x)} + -C_2 e^{(-\omega x)}) \cos(\omega y) \\
+ -C_3 (-C_1 e^{(\omega x)} + -C_2 e^{(-\omega x)}) \sin(\omega y)$$


> map(factor, %);


$$u = (-C_3 \sin(\omega y) + -C_4 \cos(\omega y)) (-C_1 e^{(\omega x)} + -C_2 e^{(-\omega x)})$$


> factor(convert(
>   subs({_C1=(_C1+_C2)/2, _C2=(_C1-_C2)/2}, %), trig));


$$u = (-C_3 \sin(\omega y) + -C_4 \cos(\omega y)) (-C_1 \cosh(\omega x) + -C_2 \sinh(\omega x))$$


> dchange({x=r*cos(phi), y=r*sin(phi)}, LaplacePDE,
> [r,phi], simplify);


$$\frac{u_r r + u_{\phi,\phi} + u_{r,r} r^2}{r^2} = 0$$


> pdsolve(%, HINT=F(r)*G(phi));


$$(u(r, \phi) = F(r) G(\phi))$$

&where [ $\{F_{r,r} = \frac{F(r) \cdot c_1}{r^2} - \frac{F_r}{r}, G_{\phi,\phi} = -G(\phi) \cdot c_1\}$ ]

> build(subs(_c[1]=omega^2, %));


$$u(r, \phi) = -C_3 \sin(\omega \phi) \cdot C_1 r^\omega + \frac{-C_3 \sin(\omega \phi) \cdot C_2}{r^\omega} \\
+ -C_4 \cos(\omega \phi) \cdot C_1 r^\omega + \frac{-C_4 \cos(\omega \phi) \cdot C_2}{r^\omega}$$


> collect(%, {sin,cos}, simplify);


$$u(r, \phi) = (-C_4 \cdot C_1 r^\omega + -C_4 \cdot C_2 r^{(-\omega)}) \cos(\omega \phi) \\
+ (-C_3 \cdot C_1 r^\omega + -C_3 \cdot C_2 r^{(-\omega)}) \sin(\omega \phi)$$


> map(factor, %);


$$u(r, \phi) = (-C_3 \sin(\omega \phi) + -C_4 \cos(\omega \phi)) (-C_1 r^\omega + -C_2 r^{(-\omega)})$$


> subs({r^omega=exp(omega*ln(r)),
> r^(-omega)=exp(-omega*ln(r))}, %);


$$u(r, \phi) = (-C_3 \sin(\omega \phi) + -C_4 \cos(\omega \phi)) (-C_1 e^{(\omega \ln(r))} + -C_2 e^{(-\omega \ln(r))})$$


> factor(convert(
>   subs({_C1=(_C1+_C2)/2, _C2=(_C1-_C2)/2}, %), trig));


$$u(r, \phi) = (-C_3 \sin(\omega \phi) + -C_4 \cos(\omega \phi)) \\
(-C_1 \cosh(\omega \ln(r)) + -C_2 \sinh(\omega \ln(r)))$$


```

Let us study the one-dimensional heat equation

$$\frac{\partial u}{\partial t} - \frac{\partial^2 u}{\partial x^2} = 0$$

more deeply. As boundary conditions we shall choose

$$u(0, t) = 0, \quad u(1, t) = 0.$$

It models the heat conduction in an infinite slab bounded by planes  $x = 0$  and  $x = 1$  in space, the boundary conditions being such that the temperature depends only on  $x$ .

```
> restart; with(PDEtools): declare(u(x,t), quiet):
> heatPDE := diff(u(x,t),t) = diff(u(x,t),[x$2]);
heatPDE :=  $u_t = u_{xx}$ 
```

First we solve the PDE by separation of variables:  $u(x, t) = F(x)G(t)$ .

```
> pdsolve(heatPDE, HINT=F(x)*G(t));
( $u = F(x) G(t)$ ) &where [ $\{F_{x,x} = -c_1 F(x), G_t = -c_1 G(t)\}$ ]
```

A more convenient choice of constant  $_c1$  is  $-c^2$ .

```
> eval(% , _c[1]=-c^2);
( $u = F(x) G(t)$ ) &where [ $\{G_t = -c^2 G(t), F_{x,x} = -c^2 F(x)\}$ ]
> build(%);
 $u = \frac{-C3 \cdot C1 \sin(cx)}{e^{(c^2 t)}} + \frac{-C3 \cdot C2 \cos(cx)}{e^{(c^2 t)}}$ 
```

A solution of this type can be rewritten as

```
> u(x,t) = A*exp(-c^2*t)*sin(c*x+beta);
 $u = A e^{(-c^2 t)} \sin(cx + \beta)$ 
> pdetest(%, heatPDE);
```

0

The condition  $u(0, t) = 0$  for all  $t$  implies  $\beta = 0$ . The condition  $u(1, t) = 0$  for all  $t$  implies  $\sin(c) = 0$ , i.e.,  $c$  is an integral multiple of  $\pi$ . So, the general solution of the boundary value problem is described as

```
> u(x,t) = Sum(A[k]*exp(-k^2*Pi^2*t)*sin(k*Pi*x),
> k=1..infinity);
 $u = \sum_{k=1}^{\infty} A_k e^{(-k^2 \pi^2 t)} \sin(k \pi x)$ 
```

For  $t = 0$ , this series, if convergent, reduces to

```
> eval(%, t=0);
 $u(x, 0) = \sum_{k=1}^{\infty} A_k \sin(k \pi x)$ 
```

Thus, the initial value  $u(x, 0)$  is represented by its Fourier sine series. Conversely, if a function  $f$  is defined on the segment  $[0, 1]$  with continuous second derivative and  $f(0) = f(1) = 0$ , then its Fourier sine series will converge uniformly for  $x \in [0, 1]$  and  $t \geq 0$ . The coefficients  $a_k$  are defined by

$$a_k = 2 \int_0^1 f(x) \sin(k\pi x) dx.$$

A few terms in the Fourier sine series are in most cases enough to give a good approximation. As an example, we consider the boundary condition

$$u(x, 0) = x(1 - x).$$

First we compute the coefficients in the Fourier sine series.

```
> A[k] = 2*Integrate(x*(1-x)*sin(k*Pi*x), x=0..1);
A_k = 2 ∫_0^1 x (1 - x) sin(k π x) dx
> value(%);
A_k = - 2 k π sin(k π) + 2 cos(k π) - 2
k³ π³
> simplify(%, assume=integer);
A_k = - 4 (-1 + (-1)ᵏ)
k³ π³
```

We use this result in the Fourier sine series and define a function that allows us to compute with only a few terms in this series.

```
> F := unapply( subs(%,
> Sum(A[k]*exp(-k^2*Pi^2*t)*sin(k*Pi*x), k=1..N)),
> x,t,N );
F := (x, t, N) → ∑_{k=1}^N (- 4 (-1 + (-1)ᵏ) e^{(-k² π² t)} sin(k π x))
k³ π³
```

The first 7 terms in the Fourier sine series suffice to approximate the boundary condition up to less than 0.001.

```
> numapprox[infnorm](F(x,0,7) - x*(1-x), x=0..1);
0.0007649617387
```

We use the first seven terms of the solution of the heat equation to plot the graph of the heat conduction when time evolves. The contours in Figure 17.30 are isotherms, i.e., lines of equal temperature.

```
> plot3d(F(x,t,7), x=0..1, t=0..0.3,
> style=patchcontour, axes=frame);
```

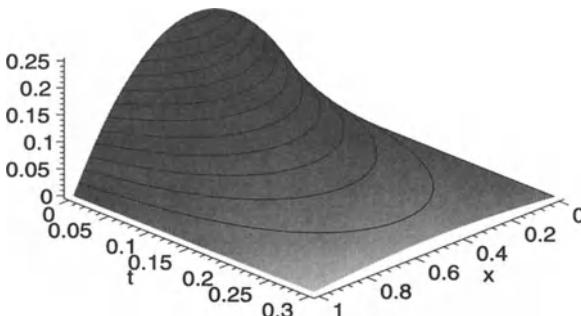


Figure 17.30. Graph of the solution of the heat equation.

Until now we only used the **pdsolve** procedure to find analytic solutions of PDEs. But with the **type=numeric** option it can also produce numerical solutions for linear PDE systems over regular regions. Let us apply this to the same boundary-value problem of the heat equation as above.

```

> heatPDE;

$$u_t = u_{x,x}$$

> BCs := {u(x,0) = x*(1-x), u(0,t), u(1,t)=0};

$$BCs := \{u(x, 0) = x(1 - x), u(0, t), u(1, t) = 0\}$$

> heatSol := pdsolve(heatPDE, BCs, type=numeric,
> timestep=0.01, spacestep=0.005);

heatSol := module()
  export plot, plot3d, animate, value, settings;
  ...
end module

```

Maple delivers the solution as a module. The module provides functions for viewing the solution as plots, animations over time, and numerical values. The graph of the heat conduction when time evolves can be obtained as follows.

```

> heatSol:-plot3d(x=0..1, t=0..0.3,
> style=patchcontour, axes=frame);

```

We omit the picture because it looks exactly the same as Figure 17.30. The heat profile at several times can be created as follows.

```

> p1 := heatSol:-plot(t=0, color=black):
> p2 := heatSol:-plot(t=0.1, color=black):
> p3 := heatSol:-plot(t=0.2, color=black):
> p4 := heatSol:-plot(t=0.3, color=black):
> plots[display]({p1,p2,p3,p4},
> title="Heat profile at t=0, 0.1, 0.2, 0.3");

```

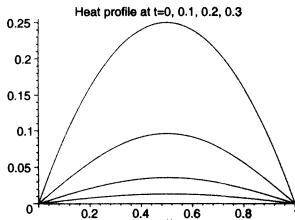


Figure 17.31. Heat profile at various times.

An animation of the temperature distribution can be obtained as follows.

```
> heatSol:=animate(u(x,t), t=0..0.3,
> frames=20, color=black, labels=["x", "u(x,t)"],
> labelfont=[TIMES,ROMAN,14]);
```

To get numerical values, create a procedure using the **value** procedure that is exported from the module.

```
> temperature := heatSol:-value();
temperature := proc() ... end proc
> temperature(0.5, 0.3);
[x = 0.5, t = 0.3, u = 0.0133268370166095690]
```

Of course there are situations in which the **pdsolve** procedure is of little or no use and where you have to assist Maple. An example is the heat conduction in an infinite rod with a point source:

$$\frac{\partial u}{\partial t} - \frac{\partial^2 u}{\partial x^2} = 0, \quad u(x, 0) = T\delta(x).$$

We apply the Fourier transform with respect to the space coordinate.

```
> heatPDE;
u_t = u_{x,x}
> with(inttrans): # load library package
> diffeqn := fourier(heatPDE, x, omega);
diffeqn := fourier_t = -omega^2 fourier(u, x, omega)
> ODE := subs(fourier(u(x,t),x,omega)=U(t), diffeqn);
ODE := U_t = -omega^2 U(t)
```

The initial condition is as follows:

```
> initval := U(0) = fourier(T*Dirac(x), x, omega);
initval := U(0) = T
```

Maple has no problem with solving this initial value problem.

```
> dsolve({ODE, initval}, U(t));
```

$$U(t) = T e^{(-\omega^2 t)}$$

We carry out the back transformation under the assumption  $t > 0$ .

```
> u(x,t) := simplify(invfourier(rhs(%), omega, x))
> assuming t>0;
```

$$u = \frac{1}{2} \frac{T e^{(-\frac{x^2}{4t})}}{\sqrt{t} \sqrt{\pi}}$$

Thus, assisting Maple, we have found a solution of the initial value problem.

And last but not least, if all fails, you can still implement some numerical scheme to solve a boundary value problem. We illustrate this by implementing a simple finite difference method for solving numerically our previous heat conduction problem

$$\frac{\partial u}{\partial t} - \frac{\partial^2 u}{\partial x^2} = 0$$

with boundary conditions

$$u(0, t) = 0, \quad u(1, t) = 0, \quad u(x, 0) = x(1 - x).$$

The basic idea of finite difference methods is to approximate (partial) derivatives by difference quotients and to replace in this manner the differential equation by a system of algebraic equations, which can be solved numerically. More examples of numerical simulation with Maple can be found in [18, 38, 74].

Firstly, we divide the strip  $[0, 1] \times [0, \infty)$  in  $x$ - $t$  space by a rectangular grid with box sizes  $h$  and  $k$ , respectively. We choose  $h = 1/m$  for some natural number  $m$  so that the point  $(1, 0)$  is a mesh point of the grid. The mesh points of the grid for which we shall approximate the temperature are  $(x, t) = (ih, jk)$  with  $i = 0, 1, 2, \dots, m$  and  $j = 0, 1, 2, \dots$ . We denote the function value  $u(x, t)$  in a mesh point  $(x, t) = (ih, jk)$  as  $u_{i,j}$ .

Secondly, we approximate the partial derivatives by finite differences. Several types of differences are possible: we shall use the so-called forward difference for the first derivative with respect to time and the central difference for the second derivative with respect to the space coordinate.

$$\left( \frac{\partial u}{\partial t} \right)_P = \frac{u_{i,j+1} - u_{i,j}}{k}, \quad \left( \frac{\partial^2 u}{\partial x^2} \right)_P = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2}.$$

The following series expansions motivate the correctness of these formulas.

```
> restart:
> series(u(x,t+k) - u(x,t), k, 2):
> convert(%, Diff);
```

$$\left( \frac{\partial}{\partial t} u(x, t) \right) k + O(k^2)$$

```
> series(u(x+h,t) - 2*u(x,t) + u(x-h,t), h, 3):
> convert(%, Diff);

$$\left( \frac{\partial^2}{\partial x^2} u(x, t) \right) h^2 + O(h^3)$$

```

In our example we get the following relationship

$$\frac{u_{i,j+1} - u_{i,j}}{k} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2},$$

which can be rewritten as

$$u_{i,j+1} = (1 - 2r)u_{i,j} + r(u_{i-1,j} + u_{i+1,j}),$$

with  $r = k/h^2$ .

The boundary conditions imply that

$$u_{0,j} = u_{m,j} = 0,$$

for  $j = 0, 1, 2, \dots$ , and that

$$u_{i,0} = f(i/m),$$

with  $f(x) = u(x, 0) = x(1 - x)$  and  $i = 0, 1, 2, \dots, m$ .

We use the above formula and conditions to determine all  $u_{i,j}$ 's in which we are interested. We start with the initial temperature for  $j = 0$ . Knowing the temperature of the mesh points at time step  $j$  we continue and determine in the  $(j + 1)^{\text{th}}$  time step for each mesh point  $(ih, (j + 1)k)$  the temperature  $u_{i,j+1}$  from the known values  $u_{i-1,j}$ ,  $u_{i,j}$ , and  $u_{i+1,j}$ .

Let us carry out this program, with the segment  $[0,1]$  subdivided into 20 equal intervals ( $m = 20$ ,  $h = 0.05$ ), with time steps of size  $k = 0.0005$ , and maximum number of time steps  $N = 600$ .

```
> m := 20: N := 600: h := 1/m: k := 0.0005: r := k/h^2:
> f := x -> x*(1-x): u := table():
> for j from 0 to N do
>   u[0,j] := 0: u[m,j] := 0
> end do:
> for i from 0 to m do
>   u[i,0] := evalf(f(i*h))
> end do:
> for j from 1 to N do
>   for i from 1 to m-1 do
>     u[i,j] := (1-2*r)*u[i,j-1] + r*(u[i-1,j-1]+u[i+1,j-1])
>   end do
> end do:
```

We plot the temperature surface with respect to space and time coordinates, showing some isotherms too.

```
> data := [seq([seq([i,10*j,u[i,10*j]], i=0..m)],
>             j=0..N/10)]:
> with(plots): # load library package
> surfdata(data, axes=frame, style=patchcontour);
```

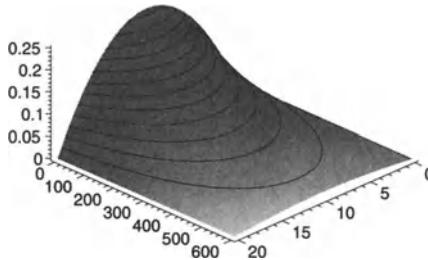


Figure 17.32. Graph of solution of the heat equation.

The application cries for an animation of the temperature of the slab with respect to the time. The Maple code to show the cooling could be as follows:

```
> figs := table():
> for k from 0 to N/10 do
>   figs[k] := listplot([seq( u[i,10*k], i=0..m)])
> end do:
> display([seq(figs[k], k=0..N/10)], insequence=true);
```

## 17.11 Lie Point Symmetries of PDEs

One of the most useful techniques for studying differential equations is the Lie symmetry method. In Maple, the `liesymm` package provides tools to apply Lie symmetry methods in the formalism described by Harrison and Estabrook [46, 123]. In this section, we shall use the package for finding the Lie symmetries of the Korteweg-de Vries equation

$$u_t + u u_x + u_{xxx} = 0,$$

using the abbreviated notation for partial derivatives.

```
> with(liesymm): # load library package
> KdV_eqn := Diff(u(t,x),t) + u(t,x)*Diff(u(t,x),x)
>      + Diff(u(t,x),[x$3])=0;
```

$$KdV\_eqn := \left(\frac{\partial}{\partial t} u(t, x)\right) + u(t, x) \left(\frac{\partial}{\partial x} u(t, x)\right) + \left(\frac{\partial^3}{\partial x^3} u(t, x)\right) = 0$$

The general idea of Lie symmetry methods for partial differential equations is the following [136, 189, 206, 213]. A *Lie point symmetry* for the PDE

$$\omega(t, x, u, u_t, u_x, u_{tt}, u_{tx}, u_{xx}, \dots) = 0$$

is a mapping

$$t \rightarrow \bar{t}(t, x, u), \quad x \rightarrow \bar{x}(t, x, u), \quad u \rightarrow \bar{u}(t, x, u)$$

such that the new variables obey the original equation. Usually, one considers only one-parameter groups of Lie point symmetries. In this case, the infinitesimal mappings are,

$$t \rightarrow t + \varepsilon\tau, \quad x \rightarrow x + \varepsilon\xi, \quad u \rightarrow u + \varepsilon\eta.$$

The PDE is unchanged if

$$X\omega = 0 \ (\text{mod } \omega = 0)$$

for the operator

$$X = \tau \partial_t + \xi \partial_x + \eta \partial_u + \dots$$

This property leads to a system of linear homogeneous PDEs for the functions  $\tau$ ,  $\xi$ , and  $\eta$ . This is called the *determining system*. In Maple, you can compute it with the procedure **determine**.

```
> eqns[1] := determine(KdV_eqn, V, u(t,x), w);
eqns1 := { $\frac{\partial}{\partial t}$  V1(t, x, u) = 3( $\frac{\partial}{\partial x}$  V2(t, x, u)) - ( $\frac{\partial^3}{\partial x^3}$  V1(t, x, u)),
 $\dots$ 
 $\frac{\partial^3}{\partial x^2 \partial u}$  V2(t, x, u) =  $\frac{\partial^3}{\partial x \partial u^2}$  V3(t, x, u)}
```

Here, the following change of notation has been used.

$$V1 = \tau, \quad V2 = \xi, \quad V3 = \eta.$$

We have omitted most of the output because we shall use the procedure **autosimp** to simplify the determining system. We are lucky: it will solve it completely (below we omit most of the output).

```
> eqns[2] := autosimp(eqns[1]);
eqns2 := {} &where{V1_2(t) = C9 +  $\frac{3}{2} t^2$  C5 + 3 C6 t,
 $\dots$ 
V2_2(t) = t C5 + C6, -t C5 - C4 - 2 C6 = 0}
```

Maple has solved the determining system of equations. But the solution contains the equation  $-t C5 - C4 - 2 C6 = 0$ , which can only be satisfied for all values of  $t$  if  $C5 = 0$  and  $C4 = -2 C6$ . When we do this substitution, we obtain the following expressions for  $V1$ ,  $V2$ , and  $V3$ .

```
> eqns := subs(C5=0,C4=-2*C6, op(2,eqns[2]));
> select(has, eqns, {V1,V2,V3});
```

$$\begin{aligned} V3(t, x, u) &= -2 u C6 + C7, \\ V1(t, x, u) &= C9 + 3 C6 t, \\ V2(t, x, u) &= x C6 + t C7 + C8 \end{aligned}$$

We were indeed lucky and have found the general solution

$$\begin{aligned} \tau &= c_9 + 3c_6 t, \\ \xi &= c_8 + c_7 t + c_6 x, \\ \eta &= c_7 - 2c_6 u, \end{aligned}$$

where  $c_6$ ,  $c_7$ ,  $c_8$ , and  $c_9$  are arbitrary constants. So, the four-dimensional symmetry algebra of the Korteweg-de Vries equation is spanned by the symmetries tabulated below

Symmetry	Meaning
$\partial_t$	time translation
$\partial_x$	space translation
$t\partial_x + \partial_u$	Galilean boost
$x\partial_x + 3t\partial_t - 2u\partial_u$	scaling

Table 17.8. Lie Point Symmetries of KdV-Equation.

The benefit of the above Lie symmetries is that if  $u = f(t, x)$  is a solution of the Korteweg-de Vries equation, so are

$$\begin{aligned} u_1 &= f(t - \varepsilon, x), \\ u_2 &= f(t, x - \varepsilon), \\ u_3 &= f(t, x - \varepsilon t) + \varepsilon, \\ u_4 &= e^{-2\varepsilon} f(e^{-3\varepsilon} t, e^{-\varepsilon} x), \end{aligned}$$

for all real  $\varepsilon$ .

Despite the above success it must be remarked that the current integration capabilities in *liesymm* for solving the determining system automatically are limited. Therefore, quite some work must still be done manually. You will notice the difference with the Lie symmetry method implemented in Maple for ordinary differential equations when you apply the procedures of the *liesymm* package to such equations. A very nice overview of other packages for determining symmetries of partial differential equations can be found in [129]. We mention the Maple package DESOLV developed by Carminati and Vu [47].

## 17.12 Exercises

- Show that  $\sqrt{a^2 - y^2} - a \ln(a + \sqrt{a^2 - y^2}) + a \ln y + x = C$  is an implicit solution of the differential equation  $y' = -\frac{y}{\sqrt{a^2 - y^2}}$ , which describes the trajectory of an object that is pulled with a rope of length  $a$  by someone who walks along the  $x$ -axis to the right.
- Solve the following ODEs with Maple. Try several methods, try to find the solutions in their simplest form, and check if Maple finds all solutions.
  - $3y^2y' + 16x = 12xy^3$
  - $y' = 2\frac{y}{x} - \left(\frac{y}{x}\right)^2$

- (c)  $xy' - y = x \tan\left(\frac{y}{x}\right)$ .
3. Some ODEs of degree larger than one can be solved with Maple. Consider the following two examples.
- Solve  $y'^2 - y^2 = 0$ .
  - Solve  $y'^2 + xy = y^2 + xy'$ .
  - Do these two examples give you any clue of how Maple attacks higher degree ODEs?
4. Solve the following ODEs with Maple and check the answers.
- $x^4y'' - (2x^2 - 1)xy' + y = 0$ .
  - $x^2y'' + 3xy' + (x^2 - 35)y = x$ .
  - $y' + xy^2 = 1$ .
5. Consider the initial value problem  $y'' - y = 0$ ,  $y(0) = 1$ ,  $y'(0) = 0$ .
- Find the solutions via the method of Laplace transforms.
  - Redo part (a) after you have given `printlevel` the value 3.
  - Repeat task (a), but now with `printlevel` equal to 33.
6. Compute the first ten terms in a Taylor series solution of the following initial value problem.  $y' = yz$ ,  $z' = xz + y$ ,  $y(0) = 1$ ,  $z(0) = 0$ .
7. Consider Airy's differential equation  $y'' + xy = 0$ .
- Find the solution for initial values  $y(0) = 1$  and  $y'(0) = 0$  via the power series method. Which terms of degree less than 30 occur in the solution?
  - What recurrence relation holds for the coefficients of the power series found in (a)?
  - Find the solution for initial values  $y(0) = 0$  and  $y'(0) = 1$  via the power series method. Which terms of degree less than 30 occur in the solution?
8. Consider Duffing's differential equation  $x'' + x + \varepsilon x^3 = \varepsilon F \cos \omega t$ . Apply the Poincaré-Lindstedt method to find an approximation of a periodic solution of the ODE which satisfies the initial values  $x(0) = A$  and  $x'(0) = 0$ . What are the results when you apply the method of multiple scales and how do both methods compare to a numerical approximation?
9. Compute the determining system for Burgers' equation,  $u_t = u_{xx} + 2u u_x$  and compute the symmetry algebra.
10. Compute the determining system for the Boltzmann equation, defined by  $u_{tx} + u_x + u^2 = 0$  and try to compute the symmetry algebra. (Hint: you may have to load the procedure `pdintegrate` of the "hidden" package `liesymm/difftools`.)

# 18

## The **LinearAlgebra** Package

In this chapter, we shall look at Maple's basic facilities for computing with matrices. Not only elementary matrix operations like addition and multiplication, but also high-level computations like determinants, inverses, eigenvalues, eigenvectors, and standard forms will pass in review. We shall give a survey of facilities that the linear algebra package, called **LinearAlgebra**, provides.

Maple contains another linear algebra package, viz., **linalg**. It is based on different data structures and is only present for backward compatibility. It has been superseded by the **LinearAlgebra** package, its subpackage **Modular** (for dense linear algebra computations in  $\mathbb{Z}[n]$ , i.e., over integers modulo the natural number  $n$ ), and the **VectorCalculus** package (for doing multivariate and vector calculus).

### 18.1 Loading the **LinearAlgebra** Package

Before you start calculating with vectors and matrices it is wise to load the **LinearAlgebra** package, which is especially designed for computations in this area.

```
> with(LinearAlgebra);  
  
[Add, Adjoint, BackwardSubstitute, BandMatrix, Basis, BezoutMatrix,  
BidiagonalForm, BilinearForm, CharacteristicMatrix,  
.....  
VectorAngle, VectorMatrixMultiply, VectorNorm,  
VectorScalarMultiply, ZeroMatrix, ZeroVector, Zip]
```

Above, we omitted most of the output. All procedures from the package are ready for use. This is already revealed when you just print one, say **HermiteForm**.

```
> print(HermiteForm);
proc(A) ... end proc
```

We have deliberately chosen this procedure, because it can illustrate how an existing function can get shadowed when loading a package. In the next instruction we load the **MatrixPolynomialAlgebra** package.

```
> with(MatrixPolynomialAlgebra);
```

```
Warning, the names HermiteForm, PopovForm and SmithForm have been
rebound
```

```
[Coeff, ColumnReducedForm, Degree, HermiteForm, Lcoeff, Ldegree,
LeftDivision, MahlerSystem, MatrixGCLD, MatrixGCRD,
MatrixLCLM, MatrixLCRM, MinimalBasis, PopovForm,
RightDivision, RowReducedForm, SmithForm, Tcoeff]
```

The warning message after loading the package reminds you that the procedure **HermiteForm**, amongst others, already exists in the Maple session and that it is shadowed by the procedure from the **MatrixPolynomialPackage** with the same name. You can check this by printing the routine and noticing that the arguments in the definition already differ.

```
> print(HermiteForm);
proc() ... end proc
```

You can get back to the previous definition of a procedure by explicitly reading it from the library, or by referring to the Maple module in which it resides, i.e., by using its full module name. For example, to get back the original definition of **HermiteForm**, you can enter the full module name **LinearAlgebra:-HermiteForm**.

```
> print(LinearAlgebra:-HermiteForm);
proc(A) ... end proc
```

Similar warnings messages for shadowed procedures you get when you load the **Modular** subpackage of the **LinearAlgebra** package.

```
> with(LinearAlgebra[Modular]):
```

```
Warning, these names have been rebound: Adjoint,
BackwardSubstitute, Basis, Determinant, ForwardSubstitute,
LUDecomposition, Multiply, Transpose
```

The most popular matrix operations are available in the **LinearAlgebra** package: multiplication and addition of matrices, row and column operations on matrices, Gaussian elimination and row reduced echelon form,

determinant, trace and inverse of a matrix, characteristic polynomial, characteristic matrix, eigenvectors and eigenvalues, and so on. Information about individual **LinearAlgebra** procedures can be obtained from the online help system. In this chapter, we shall only present a couple of examples by which the most popular matrix calculations are covered. More examples can be found in [220]. In the next chapter, we shall look at advanced applications of linear algebra. But practicing yourself will be the best way to learn about the linear algebra facilities and their usefulness.

*We assume throughout this chapter that the **LinearAlgebra** package has been loaded.*

## 18.2 Creating Vectors and Matrices

Recall from §12.4 that vectors and matrices are implemented in Maple as *arrays*. When you want to use the **LinearAlgebra** package and therefore base vectors and matrices on the data structure *rtable*, you create them by calling the procedure **Array**. This also allows indexing functions such as *symmetric* and *hermitian*. Instead of the low-level constructor **Array**, you can use the more user-friendly routines **Vector** and **Matrix**. This way of creating vectors and matrices is very convenient when you can specify the values of all entries at once through use of an index function. Table 18.1 lists options for the **Vector** and **Matrix** constructors that provide you control over the shape of a matrix or vector, the way elements are stored internally, and the data type of elements. The main reason for introducing them is efficiency when it comes to large matrices, and when you may want to use external numerical libraries or the built-in support of algorithms from the Numerical Algorithms Group (NAG).

Option	Specifying
<code>attributes = properties</code>	the associated properties
<code>datatype = type</code>	the data type of entries
<code>fill = value</code>	the default value of entries
<code>orientation = name</code>	<code>column</code> or <code>row</code> orientation of vectors
<code>readonly</code>	an immutable matrix
<code>shape = indexing function(s)</code>	one or more indexing functions
<code>storage = storage mode</code>	the way elements are stored internally

Table 18.1. Options of **Vector** and **Matrix**.

A few examples:

```
> # low-level matrix creation
> A := Array(1..2, 1..3, [[a,b,c],[d,e,f]]);
```

```


$$A := \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix}$$

> B := Array(antisymmetric, 1..2, 1..2, {(1,2)=x},
> storage=sparse);

$$B := \begin{bmatrix} 0 & x \\ -x & 0 \end{bmatrix}$$

> # high-level matrix creation
> A := Matrix(2, 3, [[a,b,c],[d,e,f]]);

$$A := \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix}$$

> B := Matrix(2, 2, shape=antisymmetric, storage=sparse):
> B[1,2] := x:
> B;

$$\begin{bmatrix} 0 & x \\ -x & 0 \end{bmatrix}$$

> MatrixOptions(B); # inspect the data structure options
shape = [skewsymmetric], datatype = anything, storage = sparse,
order = Fortran_order
> v := Vector([1,2,3]);

$$v := \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

> whattype(%);
Vector column
> v := Vector([1,2,3], orientation=row);

$$v := [1, 2, 3]$$


```

Alternatively, you can use an indexed call of **Vector**.

```

> v := Vector[row](3, fill=1);

$$v := [1, 1, 1]$$


```

The following notational shortcuts are available for creating (**rtable**-based) matrices and vectors.

- $\langle p, q \rangle$  constructs an object from  $p$  and  $q$  by *rows*
- $\langle p | q \rangle$  constructs an object from  $p$  and  $q$  by *columns*

A few examples we make things clear to you. Don't get confused by the Maple prompt  $>$ , which is not part of any command.

```

> <1 , 2 , 3>; # construct a column vector

```

```


$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

> <1 | 2 | 3>; # construct a row vector
[1, 2, 3]
> <<1,2,3> | <4,5,6>>; # construct a matrix by columns

$$\begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

> <<1|2|3>, <4|5|6>>; # construct a matrix by rows

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

> A, v;      # recall the current definitions of A and v

$$\begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix}, [1, 1, 1]$$

> <A | A>; # augment matrices column-wise

$$\begin{bmatrix} a & b & c & a & b & c \\ d & e & f & d & e & f \end{bmatrix}$$

> <A , A>; # augment matrices row-wise

$$\begin{bmatrix} a & b & c \\ d & e & f \\ a & b & c \\ d & e & f \end{bmatrix}$$

> <v | v>; # combine two row vectors
[ 1 1 1 1 1 1 ]
> <v , v>; # use two row vectors to create a matrix

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

> <v , A>; # add row at the top of the matrix

$$\begin{bmatrix} 1 & 1 & 1 \\ a & b & c \\ d & e & f \end{bmatrix}$$

> <A , v>; # add row at the bottom of the matrix

$$\begin{bmatrix} a & b & c \\ d & e & f \\ 1 & 1 & 1 \end{bmatrix}$$


```

```
> # add column at the right of the matrix
> <A | Vector(2, fill=1)>;
```

$$\begin{bmatrix} a & b & c & 1 \\ d & e & f & 1 \end{bmatrix}$$

Index functions are convenient when you want to create large matrices for which the entries are defined by some function on the indices.

```
> C := Matrix(4, 4, (i,j) -> i^(j-1));
```

$$C := \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 \\ 1 & 3 & 9 & 27 \\ 1 & 4 & 16 & 64 \end{bmatrix}$$

The matrix  $C$  is a special case of a Vandermonde matrix: you can convince yourself by using the built-in function **VandermondeMatrix**.

```
> VandermondeMatrix([1,2,3,4]);
```

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 \\ 1 & 3 & 9 & 27 \\ 1 & 4 & 16 & 64 \end{bmatrix}$$

Maple has more built-in routines for special matrices; Table 18.2 lists them. We refer to the on-line help system for detailed information.

Procedure	Matrix
<b>BezoutMatrix</b>	Bezout matrix of two polynomials
<b>CompanionMatrix</b>	companion matrix of a polynomial
<b>GivensRotationMatrix</b>	matrix of a Givens rotation
<b>HankelMatrix</b>	Hankel matrix
<b>HilbertMatrix</b>	generalized Hilbert matrix
<b>HouseholderMatrix</b>	Householder matrix
<b>IdentityMatrix</b>	identity matrix
<b>JordanBlockMatrix</b>	Jordan block matrix
<b>SylvesterMatrix</b>	Sylvester matrix of two polynomials
<b>ToeplitzMatrix</b>	symmetric Toeplitz matrix of a list
<b>UnitVector</b>	vector with all entries equal to 1
<b>VandermondeMatrix</b>	Vandermonde matrix of a list
<b>ZeroMatrix</b>	matrix with all entries equal to 0

Table 18.2. Procedures related to special vectors and matrices.

Banded matrices and diagonal matrices can be created with the procedures **BandMatrix** and **DiagonalMatrix**, respectively.

```
> BandMatrix([-1,2,1], 1, 4);
```

```


$$\begin{bmatrix} 2 & 1 & 0 & 0 \\ -1 & 2 & 1 & 0 \\ 0 & -1 & 2 & 1 \\ 0 & 0 & -1 & 2 \end{bmatrix}$$

> DiagonalMatrix(<mu, nu>, 2, 2);

$$\begin{bmatrix} \mu & 0 \\ 0 & \nu \end{bmatrix}$$

> DiagonalMatrix([%, %]);

$$\begin{bmatrix} 2 & 1 & 0 & 0 & 0 & 0 \\ -1 & 2 & 1 & 0 & 0 & 0 \\ 0 & -1 & 2 & 1 & 0 & 0 \\ 0 & 0 & -1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mu & 0 \\ 0 & 0 & 0 & 0 & 0 & \nu \end{bmatrix}$$


```

The last example shows how the procedure **DiagonalMatrix** can also be used out create a blockmatrix out of earlier defined matrices. In §18.5 we shall discuss other **LinearAlgebra** procedures that allow you to create new vectors and matrices from old ones. However, augmentation and extraction are the main facilities. Augmentation of matrices and vectors we have discussed before in examples; extraction of matrices and vectors will be discussed later on in this section.

An example of a routine to build up a matrix from another data structure is **GenerateMatrix**: it creates the coefficient matrix associated with a system of linear equations. An example:

```

> eqns := [ (c^2-1)*x + (c-1)*y = (1-c)^2,
>           (c-1)*x + (c^2-1)*y = c-1 ];
eqns := [(c2 - 1) x + (c - 1) y = (1 - c)2, (c - 1) x + (c2 - 1) y = c - 1]
> A, b := GenerateMatrix(eqns, [x,y]);
A, b :=  $\begin{bmatrix} c^2 - 1 & c - 1 \\ c - 1 & c^2 - 1 \end{bmatrix}, \begin{bmatrix} (1 - c)^2 \\ c - 1 \end{bmatrix}$ 
> A . <x,y> = b; # Matrix-vector equation

$$\begin{bmatrix} (c^2 - 1) x + (c - 1) y \\ (c - 1) x + (c^2 - 1) y \end{bmatrix} = \begin{bmatrix} (1 - c)^2 \\ c - 1 \end{bmatrix}$$

> eqns := { (c^2-1)*x + (c-1)*y = (1-c)^2,
>           (c-1)*x + (c^2-1)*y = c-1 };

```

$eqns := \{(c^2 - 1) x + (c - 1) y = (1 - c)^2, (c - 1) x + (c^2 - 1) y = c - 1\}$

Now you can use matrix algebra to solve the original system of equations. But why would you do this in the first place? Why not simply use **solve**? Look what happens if you apply this procedure to the system of equations.

```
> solve(eqns, {x,y});
{y = 2  $\frac{1}{c(c+2)}$ , x =  $\frac{-2+c^2}{c(c+2)}$ }
```

One solution! But if you look more closely at the system of equations, then you see that for  $c = 1$  all pairs  $(x, y)$  satisfy the equations, and that there does not exist a solution for  $c = 0$  and for  $c = -2$ . However, if you apply linear algebra methods and compute the Hermite normal form (row reduced echelon form over  $\mathbb{Q}(c)$ ), then the special cases become obvious.

```
> GenerateMatrix(eqns, [x,y], augmented=true);

$$\left[ \begin{array}{ccc} c^2 - 1 & c - 1 & (1 - c)^2 \\ c - 1 & c^2 - 1 & c - 1 \end{array} \right]$$

> LinearSolve(%); # the same solution as before

$$\left[ \begin{array}{c} \frac{c^2 - 2}{c(c+2)} \\ \frac{2}{c(c+2)} \end{array} \right]$$

> HermiteForm(%%, c);

$$\left[ \begin{array}{ccc} c - 1 & c^2 - 1 & c - 1 \\ 0 & c^3 + c^2 - 2c & (c + 1)(c - 1) - 1 + 2c - c^2 \end{array} \right]$$

> map(factor, %);

$$\left[ \begin{array}{ccc} c - 1 & (c + 1)(c - 1) & c - 1 \\ 0 & c(c + 2)(c - 1) & -2 + 2c \end{array} \right]$$

```

As we have said before, extracting matrices and vectors is a common way of creating such objects. The structural procedures are **SubMatrix** and **SubVector**, but you will often prefer to use the syntactical shortcuts. Examples show best how extraction of matrices and vectors can be done. For this purpose we first create a random matrix of integers.

```
> M := RandomMatrix(3, 4);
M :=  $\left[ \begin{array}{cccc} -34 & -21 & -50 & -79 \\ -62 & -56 & 30 & -71 \\ -90 & -8 & 62 & 28 \end{array} \right]$ 
```

Extraction of a single entry, say the entry at row 2 and column 3, you already know how to do.

```
> M[2,3]; # extract a single entry
```

Extracting a submatrix is not much more complicated: use a range instead of a single index for the row and column coordinate.

```
> SubMatrix(M, 2..3, 3..4); # extract submatrix
```

$$\begin{bmatrix} 30 & -71 \\ 62 & 28 \end{bmatrix}$$

```
> M[2..3, 3..4]; # extraction in shortcut-style
```

$$\begin{bmatrix} 30 & -71 \\ 62 & 28 \end{bmatrix}$$

You can count row and columns in a matrix from right to left and from bottom to top by using negative integers, just as we have explained for in §12.3 for lists.

```
> M[-2..-1, -2..-1]; # counting backwards
```

$$\begin{bmatrix} 30 & -71 \\ 62 & 28 \end{bmatrix}$$

The ranges stand for a contiguous list of entries, e.g., **M[2..3, 3..4]** stands for **M[[2,3], [3,4]]**. As a matter of fact, you can use any list for extraction of submatrices. For example, to extract the 2nd and 4th entries of the 1st and 3rd row of **M** you can enter.

```
> M[[1,3], [2,4]]; # extraction of matrix elements
```

$$\begin{bmatrix} -21 & -79 \\ -8 & 28 \end{bmatrix}$$

This is convenient when you want to swap row or columns.

```
> M[1..-1, [2,1,3..-1]]; # swap 1st and 2nd column
```

$$M := \begin{bmatrix} -21 & -34 & -50 & -79 \\ 56 & -62 & 30 & -71 \\ -8 & -90 & 62 & 28 \end{bmatrix}$$

Note that the original matrix is left unchanged.

```
> M;
```

$$\begin{bmatrix} -34 & -21 & -50 & -79 \\ -62 & -56 & 30 & -71 \\ -90 & -8 & 62 & 28 \end{bmatrix}$$

Creating a matrix that consists of a row vector or a column vector can be achieved by a range with the same beginning and end points or with a list with a single term. Below, we give three commands that extract the matrix consisting of the second row of **M**.

```
> M[2..2, 1..4]; # extraction of a 1x4 matrix
```

$$\begin{bmatrix} -62 & -56 & 30 & -71 \end{bmatrix}$$

```
> M[[2], 1..4];
```

$$\begin{bmatrix} -62 & -56 & 30 & -71 \end{bmatrix}$$

```
> M[[2], 1..-1];
```

```
[ -62 -56 30 -71 ]
> whattype(%);
```

*Matrix*

Omit the list-brackets when you want a vector being extracted.

```
> M[2, 1..-1]; # 2nd row vector
[-62, -56, 30, -71]
> whattype(%);
```

*Vector<sub>row</sub>*

We spend much time on extraction of submatrices because it also lets you understand multiple assignment of entries.

```
> M; # original matrix
[ -34 -21 -50 -79
 -62 -56 30 -71
 -90 -8 62 28 ]
> M[2..3,2..3]; # submatrix
[ -56 30
 -8 62 ]
> M[2..3,2..3] := RandomMatrix(2,2);
M2..3, 2..3 := [ -41 -7
 20 16 ]
> M; # (sub)matrix indeed changed!
[ -34 -21 -50 -79
 -62 -41 -7 -71
 -90 20 16 28 ]
> M[1..-1, 1..-1] := M[1..-1, [2,1,3..-1]]:
> M; # 1st and 2nd column of M are swapped
[ -21 -34 -50 -79
 -7 -62 -7 -71
 20 -90 16 28 ]
```

Let us continue with the survey of creating vectors and matrices. Random vector and random matrix generators are available through the procedures **RandomVector** and **RandomMatrix**, respectively. Options will give you the flexibility to determine the form of the matrix and its entries. The shape, storage mode, and data type of entries of the matrix can be specified in the **outputoptions** option. If an optional argument takes the form **generator=f** for a Maple procedure f, then f is called to generate an entry. The default function is **rand(-99..99)**, i.e., random two digit integers. The

option **density=p**, where **p** is a number between 0 and 1, can be used to specify the chance that an entry is filled with a random value. One example will do.

```
> poly := proc() randpoly(x, terms=3, degree=3) end proc;
> RandomMatrix(3, 3, generator=poly,
> outputoptions=[shape=triangular[upper]]);
```

$$\begin{bmatrix} -35 - 55x^3 - 37x & 56 + 50x^2 + 79x & -59 + 63x^3 + 57x \\ 0 & 92 - 8x^3 - 93x & 66 - 62x^2 + 77x \\ 0 & 0 & -61 - 5x^3 + 99x^2 \end{bmatrix}$$

Last but not least, you can create a matrix by reading data from a file. Consider a datafile, say **matrixdata**, in the current directory and two lines long with the following contents

```
1      2      3
4      5      6
```

With **ImportMatrix** you can import the data and then store them in matrix format. See §4.4 and §4.5 for more details about importing/exporting data and low level I/O.

```
> mat := ImportMatrix("matdata");
```

$$mat := \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

## 18.3 Vector and Matrix Arithmetic

For demonstration purposes we shall use the following matrices:

```
> A := <<a|b>, <c|d>>;
```

$$A := \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

```
> B := ToeplitzMatrix([alpha, beta], symmetric);
```

$$B := \begin{bmatrix} \alpha & \beta \\ \beta & \alpha \end{bmatrix}$$

```
> C := Matrix(3, 2, (i,j)->x^i*y^j);
```

$$C := \begin{bmatrix} xy & xy^2 \\ x^2y & x^2y^2 \\ x^3y & x^3y^2 \end{bmatrix}$$

Addition of matrices is no big deal: you can use the **+** operator.

```
> A+B;
```

$$\begin{bmatrix} a + \alpha & b + \beta \\ c + \beta & d + \alpha \end{bmatrix}$$

The `+` operator automatically sorts out that matrices are involved. Thus, matrix addition is carried out. Instead, you can use **Add** or the more dedicated procedure **MatrixAdd**.

```
> Add(A,B);

$$\begin{bmatrix} a + \alpha & b + \beta \\ c + \beta & d + \alpha \end{bmatrix}$$

> MatrixAdd(A, B);

$$\begin{bmatrix} a + \alpha & b + \beta \\ c + \beta & d + \alpha \end{bmatrix}$$

```

Linear combinations of matrices can be calculated in the usual way.

```
> 3*A - 2/7*B;

$$\begin{bmatrix} 3a - \frac{2\alpha}{7} & 3b - \frac{2\beta}{7} \\ 3c - \frac{2\beta}{7} & 3d - \frac{2\alpha}{7} \end{bmatrix}$$

```

But there is a problem when scalars are not numeric.

```
> t*A;

$$t \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

```

How can Maple know that `t` is supposed to be a scalar? The answer is simple: *you* have to inform Maple about this. Either use the **assume** facility or add assumptions to the **simplify** command.

```
> simplify(%) assuming t::scalar;

$$\begin{bmatrix} ta & tb \\ tc & td \end{bmatrix}$$

> simplify(%, symbolic);

$$\begin{bmatrix} ta & tb \\ tc & td \end{bmatrix}$$

```

You can also apply the procedure **ScalarMultiply**.

```
> ScalarMultiply(A, t);

$$\begin{bmatrix} ta & tb \\ tc & td \end{bmatrix}$$

```

Addition of scalars is also possible; the main diagonal is increased by the scalar value. But the same problem with non-numeric scalar occurs.

```
> C - 1;
```

```


$$\begin{bmatrix} xy - 1 & xy^2 \\ x^2y & x^2y^2 - 1 \\ x^3y & x^3y^2 \end{bmatrix}$$

> C := lambda;

$$-\lambda + \begin{bmatrix} xy & xy^2 \\ x^2y & x^2y^2 \\ x^3y & x^3y^2 \end{bmatrix}$$

> simplify(%) assuming lambda::scalar;

$$\begin{bmatrix} xy - \lambda & xy^2 \\ x^2y & x^2y^2 - \lambda \\ x^3y & x^3y^2 \end{bmatrix}$$

> MatrixAdd(C, IdentityMatrix(3,2), 1, -lambda);

$$\begin{bmatrix} xy - \lambda & xy^2 \\ x^2y & x^2y^2 - \lambda \\ x^3y & x^3y^2 \end{bmatrix}$$


```

You cannot use the operator `*` for matrix multiplication; this operator is reserved in Maple as the *commutative* multiplication operator.

```

> A*B;
Error, (in rtable/Product) invalid arguments

```

You must use the multiplication operator `.` or the procedures **Multiply** and **MatrixMatrixMultiply** for multiplication of matrices. Matrix multiplication is non-commutative.

```

> A.B;

$$\begin{bmatrix} a\alpha + b\beta & a\beta + b\alpha \\ c\alpha + d\beta & c\beta + d\alpha \end{bmatrix}$$

> B.A;

$$\begin{bmatrix} a\alpha + c\beta & b\alpha + d\beta \\ a\beta + c\alpha & b\beta + d\alpha \end{bmatrix}$$


```

The multiplication operator `.` can also be used for multiplication of matrices with vectors, or for inner product and dot product of vectors. But you must take care yourself whether row or column vectors are needed. The examples below show that this makes a difference

```

> v := <x,y>; w := <xi,eta>; # column vectors
> 'v'=v, w='w';

$$v = \begin{bmatrix} x \\ y \end{bmatrix}, w = \begin{bmatrix} \xi \\ \eta \end{bmatrix}$$

> A.v; # valid matrix-vector multiplication

```

```


$$\begin{bmatrix} ax + by \\ cx + dy \end{bmatrix}$$

> v.A; # v cannot be used as row vector
Error, (in LinearAlgebra:-VectorMatrixMultiply) invalid input:
LinearAlgebra:-VectorMatrixMultiply expects its 1st argument, v,
to be of type Vector[row] but received Vector[column](2, [...]),
datatype = anything, storage = rectangular, order = Fortran_order,
shape = []
> Transpose(v).A;

$$[ax + yc, xb + dy]$$

> v.w; # inner product of column vectors

$$\bar{x}\xi + \bar{y}\eta$$

> DotProduct(v,w);

$$\bar{x}\xi + \bar{y}\eta$$

> v := <x|y>; # v is now a row vector

$$v := [x, y]$$

> v.A; # valid vector-matrix product

$$[ax + yc, xb + dy]$$

> v.w; # now the dot is used for matrix product

$$\xi x + \eta y$$

> DotProduct(v,w);

$$\bar{\xi}x + \bar{\eta}y$$


```

Matrix powers can be computed with the usual exponentiation operator `^`.

```
> B3 := B^3;
```

$$B^3 := \begin{bmatrix} \%1\alpha + 2\alpha\beta^2 & \%1\beta + 2\alpha^2\beta \\ \%1\beta + 2\alpha^2\beta & \%1\alpha + 2\alpha\beta^2 \end{bmatrix}$$

$$\%1 := \alpha^2 + \beta^2$$

With the `map` procedure, you can factorize all matrix entries at the same time.

```
> factor(B3); # factorization fails
```

$$\begin{bmatrix} \%1\alpha + 2\alpha\beta^2 & \%1\beta + 2\alpha^2\beta \\ \%1\beta + 2\alpha^2\beta & \%1\alpha + 2\alpha\beta^2 \end{bmatrix}$$

$$\%1 := \alpha^2 + \beta^2$$

```
> map(factor, B3); # matrix elements are factorized
```

$$\begin{bmatrix} \alpha(\alpha^2 + 3\beta^2) & \beta(3\alpha^2 + \beta^2) \\ \beta(3\alpha^2 + \beta^2) & \alpha(\alpha^2 + 3\beta^2) \end{bmatrix}$$

Note that the original matrix **B3** is left unchanged by this mapping of the **factor** procedure. If you want this to happen, then you must use the **Map** procedure from the **LinearAlgebra** package.

```
> B3; # recall the matrix
```

$$\begin{bmatrix} \%1\alpha + 2\alpha\beta^2 & \%1\beta + 2\alpha^2\beta \\ \%1\beta + 2\alpha^2\beta & \%1\alpha + 2\alpha\beta^2 \end{bmatrix}$$

$$\%1 := \alpha^2 + \beta^2$$

```
> Map(factor, B3); # matrix has changed!
```

$$\begin{bmatrix} \alpha(\alpha^2 + 3\beta^2) & \beta(3\alpha^2 + \beta^2) \\ \beta(3\alpha^2 + \beta^2) & \alpha(\alpha^2 + 3\beta^2) \end{bmatrix}$$

Not only natural numbers are accepted by Maple as exponents; negative integral exponents can also be used, provided that the matrix is not singular.

```
> B3inverse := B3^(-1); # inverse of B3
```

$$\begin{bmatrix} \frac{\alpha(\alpha^2 + 3\beta^2)}{\%1} & -\frac{\beta(3\alpha^2 + \beta^2)}{\%1} \\ -\frac{\beta(3\alpha^2 + \beta^2)}{\%1} & \frac{\alpha(\alpha^2 + 3\beta^2)}{\%1} \end{bmatrix}$$

$$\%1 := \alpha^6 - 3\alpha^4\beta^2 + 3\alpha^2\beta^4 - \beta^6$$

```
> B3inverse . B3:
```

We have not shown the output because it is rather long. But this matrix simplifies indeed to the identity matrix.

```
> map(normal, %); # the expected answer
```

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Automatic simplification may sometimes surprise you.

```
> A^0;
```

1

```
> whattype(%);
```

*integer*

Furthermore, because the **.** operator has the same priority as the multiplication and division operators **\*** and **/**, you should be cautious about precedence of operators. Sometimes you have to use parentheses.

```
> 1/A;
```

```


$$\begin{bmatrix} \frac{d}{ad-bc} & -\frac{b}{ad-bc} \\ -\frac{c}{ad-bc} & \frac{a}{ad-bc} \end{bmatrix}$$

> A . 1/A;
Error, (in rtable/Product) invalid arguments
> A . (1/A);

$$\begin{bmatrix} \frac{ad}{ad-bc} - \frac{bc}{ad-bc} & 0 \\ 0 & \frac{ad}{ad-bc} - \frac{bc}{ad-bc} \end{bmatrix}$$

> map(normal, %);

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$


```

## 18.4 Basic Matrix Functions

The procedures **Trace** and **Determinant** compute the trace and the determinant of a matrix, respectively. Let us apply them to the following Toeplitz matrix.

```

> ToeplitzMatrix([1,2,3], 3, symmetric);

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & 2 \\ 3 & 2 & 1 \end{bmatrix}$$

> Trace(%);
3
> Determinant(%);
8

```

The row and column rank of a matrix, bases for the row and column spaces, and a basis for the kernel (nullspace) of a matrix can be easily computed.

```

> A := <<1|0|0|1>, <1|0|1|1>, <0|0|1|0>>;
A :=  $\begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$ 
> Rank(A);
2

```

```
> RowSpace(A), ColumnSpace(A);
[[1, 0, 0, 1], [0, 0, 1, 0]], [[1, 0, 0], [0, 1, 0], [-1, 0, 1]]
> NullSpace(A);
{{[0, 1, 0, 0], [-1, 0, 0, 1]}}
```

Computation of the characteristic polynomial, eigenvalues, and eigenvectors is in principle possible; but exact calculus may of course come to the deadlock of an unsolvable characteristic equation.

```
> A := <<-2|2|3>, <3|7|-8>, <10|-4|-3>>;
A := [[-2, 2, 3], [3, 7, -8], [10, -4, -3]]
> cp := CharacteristicPolynomial(A, lambda);
cp :=  $\lambda^3 - 2\lambda^2 - 97\lambda + 282$ 
```

The Cayley-Hamilton theorem states that substitution of the matrix  $A$  in its characteristic polynomial yields the zero matrix. In this particular example, the minimal polynomial of  $A$ , i.e., the monic polynomial  $f$  with lowest degree for which  $f(A) = 0$ , is the same as the characteristic polynomial. In general this is not true. The minimal polynomial can be computed in Maple with the procedure **MinimalPolynomial**. Let us check all this.

```
> eval(subs(lambda=A, cp));
[[0, 0, 0], [0, 0, 0], [0, 0, 0]]
> mp := MinimalPolynomial(A, lambda);
mp :=  $\lambda^3 - 2\lambda^2 - 97\lambda + 282$ 
```

The characteristic equation must be solved to obtain the eigenvalues of  $A$ .

```
> solve(cp);
3,  $-\frac{1}{2} + \frac{\sqrt{377}}{2}$ ,  $-\frac{1}{2} - \frac{\sqrt{377}}{2}$ 
```

The eigenvalues could have been obtained straight away with the procedure **Eigenvalues**. By default, the eigenvalues are returned as a column vector, but other output forms are available upon request.

```
> Eigenvalues(A);
```

```


$$\begin{bmatrix} 3 \\ -\frac{1}{2} + \frac{\sqrt{377}}{2} \\ -\frac{1}{2} - \frac{\sqrt{377}}{2} \end{bmatrix}$$

> Eigenvalues(A, output=list);
[3,  $-\frac{1}{2} + \frac{\sqrt{377}}{2}$ ,  $-\frac{1}{2} - \frac{\sqrt{377}}{2}$ ]

```

Eigenvectors can be computed with **Eigenvectors**. By default, this procedure returns a sequence consisting of a vector of eigenvalues and a matrix whose columns are the corresponding eigenvectors. By multiple assignment you can simultaneously assign the two elements of the sequence to separate variables. Below, the vector **v** contains the eigenvalues and the columns of the matrix **e** are the eigenvectors. More precisely, the *i*th column of **a** is an eigenvector associated with the *i*th eigenvalue of the returned vector **v**. Below, we verify one result.

```

> v, e := Eigenvectors(A);
v, e :=  $\begin{bmatrix} 3 \\ -\frac{1}{2} + \frac{\sqrt{377}}{2} \\ -\frac{1}{2} - \frac{\sqrt{377}}{2} \end{bmatrix}, \begin{bmatrix} \frac{14}{13} & -\frac{1}{2} & -\frac{1}{2} \\ \frac{31}{26} & -\frac{15}{8} - \frac{\sqrt{377}}{8} & -\frac{15}{8} + \frac{\sqrt{377}}{8} \\ 1 & 1 & 1 \end{bmatrix}$ 
> simplify(A . e[1..-1,3] - v[3]*e[1..-1,3]);

$$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$


```

Other output formats are possible.

```

> Eigenvectors(A, output=list);
[[3, 1, {{ $\begin{bmatrix} \frac{14}{13} \\ \frac{31}{26} \\ 1 \end{bmatrix}$ }}], [ $-\frac{1}{2} + \frac{\sqrt{377}}{2}$ , 1, {{ $\begin{bmatrix} -\frac{1}{2} \\ -\frac{15}{8} - \frac{\sqrt{377}}{8} \\ 1 \end{bmatrix}$ }}], [ $-\frac{1}{2} - \frac{\sqrt{377}}{2}$ , 1, {{ $\begin{bmatrix} -\frac{1}{2} \\ -\frac{15}{8} + \frac{\sqrt{377}}{8} \\ 1 \end{bmatrix}$ }}]]

```

Here, you get a sequence of lists. Each list consists of an eigenvalue, its multiplicity, and a basis of the eigenspace. By default, Maple computes the eigenvalues and eigenvectors using radicals. To compute implicitly in terms of **RootOfs**, use the **implicit** option. Later on, you can compute all values with the procedure **allvalues**. The eigenvalues can also be computed in **RootOf** notation.

```
> Eigenvalues(A, 'implicit'); # use of RootOf notation

$$\left[ \begin{array}{c} 3 \\ \text{RootOf}(-Z^2 - Z - 94, \text{index} = 1) \\ \text{RootOf}(-Z^2 - Z - 94, \text{index} = 2) \end{array} \right]$$

> v, e := Eigenvectors(A, 'implicit');
v, e :=

$$\left[ \begin{array}{c} 3 \\ \text{RootOf}(-Z^2 - Z - 94) \end{array} \right], \left[ \begin{array}{cc} \frac{14}{13} & -\frac{1}{2} \\ \frac{31}{26} & -\frac{1}{4} \text{RootOf}(-Z^2 - Z - 94) - 2 \\ 1 & 1 \end{array} \right]$$

> e[1..-1,2]; # selection of eigenvector

$$\left[ \begin{array}{c} -\frac{1}{2} \\ -\frac{1}{4} \text{RootOf}(-Z^2 - Z - 94) - 2 \\ 1 \end{array} \right]$$

> allvalues(%, 'dependent'); # 2 eigenvectors in 1 result

$$\left[ \begin{array}{c} -\frac{1}{2} \\ -\frac{15}{8} - \frac{\sqrt{377}}{8} \\ 1 \end{array} \right], \left[ \begin{array}{c} -\frac{1}{2} \\ -\frac{15}{8} + \frac{\sqrt{377}}{8} \\ 1 \end{array} \right]$$

```

Until now, most of our matrices had rational coefficients. In fact, all procedures in the **LinearAlgebra** package work for matrices over the ring of polynomials with rational coefficients. But for many procedures, coefficients may come from a wider class; complex numbers, algebraic numbers, and algebraic functions are allowed in many cases. Below is one example of a Toeplitz matrix.

```
> A := ToeplitzMatrix([sqrt(2), alpha, beta], 3, symmetric);
A := 
$$\left[ \begin{array}{ccc} \sqrt{2} & \alpha & \beta \\ \alpha & \sqrt{2} & \alpha \\ \beta & \alpha & \sqrt{2} \end{array} \right]$$

```

```

> factor(Determinant(A), sqrt(2));

$$(-\sqrt{2}\beta - 2 + 2\alpha^2)(-\sqrt{2} + \beta)$$

> CharacteristicPolynomial(A, lambda);

$$\lambda^3 - 3\sqrt{2}\lambda^2 - (-6 + \beta^2 + 2\alpha^2)\lambda - 2\sqrt{2} + 2\sqrt{2}\alpha^2 - 2\alpha^2\beta + \sqrt{2}\beta^2$$

> factor(% , sqrt(2));

$$-(\lambda\beta - \sqrt{2}\beta + 2\sqrt{2}\lambda - \lambda^2 - 2 + 2\alpha^2)(-\sqrt{2} + \beta + \lambda)$$

> Eigenvalues(A, output=list);

$$[\frac{\beta}{2} + \sqrt{2} + \frac{\sqrt{\beta^2 + 8\alpha^2}}{2}, \frac{\beta}{2} + \sqrt{2} - \frac{\sqrt{\beta^2 + 8\alpha^2}}{2}, \sqrt{2} - \beta]$$

> Eigenvectors(A);


$$\begin{bmatrix} \text{RootOf}(\%2, \text{index} = 1) \\ \text{RootOf}(\%2, \text{index} = 2) \\ \sqrt{2} - \beta \end{bmatrix},$$


$$\begin{bmatrix} 1 & 1 & 1 \\ -\frac{-R(\%2, 1) + \beta + \sqrt{2}}{\alpha} & -\frac{-R(\%2, 2) + \beta + \sqrt{2}}{\alpha} & 0 \\ 1 & 1 & 1 \end{bmatrix},$$

%1 := RootOf(_Z^2 - 2, index = 1)
%2 := _Z^2 + (-2%1 - \beta)_Z + 2 + %1\beta - 2\alpha^2

```

where we have edited the matrix of eigenvectors outside Maple for the sake of readability, by introducing  $R(a, b)$  as short notation for **RootOf**( $a, \text{index} = b$ ).

```

> Eigenvectors(A, 'implicit');
Error, (in LinearAlgebra:-LA_Main:-Eigenvectors)
radicals in matrix must be converted to RootOfs

> map(convert, A, RootOf);

$$\begin{bmatrix} \text{RootOf}(_Z^2 - 2, \text{index} = 1), \alpha, \beta \\ \alpha, \text{RootOf}(_Z^2 - 2, \text{index} = 1), \alpha \\ \beta, \alpha, \text{RootOf}(_Z^2 - 2, \text{index} = 1) \end{bmatrix}$$

> Eigenvectors(% , 'implicit');

```

$$\begin{bmatrix} \text{RootOf}(\%2, \text{index} = 1) \\ \text{RootOf}(\%2, \text{index} = 2) \\ \%1 - \beta \end{bmatrix},$$

$$\begin{bmatrix} \frac{1}{-\text{RootOf}(\%2) + \beta + \%1} & \frac{1}{-\text{RootOf}(\%2) + \beta + \%1} & -1 \\ \alpha & \alpha & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

$\%1 := \text{RootOf}(-Z^2 - 2, \text{index} = 1)$

$\%2 := -Z^2 + (-2\%1 - \beta)Z + 2 + \%1\beta - 2\alpha^2$

However, some procedures do not allow both floating-point numbers and symbolic expressions, or simply have difficulties with such mixed types.

```
> Map(evalf, A); # change of matrix entries to floats
```

$$\begin{bmatrix} 1.414213562 & \alpha & \beta \\ \alpha & 1.414213562 & \alpha \\ \beta & \alpha & 1.414213562 \end{bmatrix}$$

```
> Eigenvectors(A);
```

```
Error, (in Eigenvectors/complex_QZ/cbal) cannot determine
if this expression is true or false:
9/10*abs(Re(beta))+9/10*abs(Im(beta))+9/10*abs(Re(alpha))
+9/10*abs(Im(alpha)) < 0
```

The error message suggests that in this example Maple wants to switch to a numerical eigenvector routine, but it cannot do this as the matrix contains the symbols  $\alpha$  and  $\beta$ . A trick that works in many cases is to convert the floating-point numbers to rational numbers, do the requested computation, and evaluate the intermediate result numerically.

```
> interface(displayprecision=4):
> Map(convert, A, rational);
```

$$\begin{bmatrix} \frac{47321}{33461} & \alpha & \beta \\ \alpha & \frac{47321}{33461} & \alpha \\ \beta & \alpha & \frac{47321}{33461} \end{bmatrix}$$

```
> Eigenvectors(A, output=list);
```

$$\left[ \left[ -\beta + \frac{47321}{33461}, 1, \left\{ \left[ \begin{array}{c} -1 \\ 0 \\ 1 \end{array} \right] \right\} \right]$$

```


$$\left[ \frac{\beta}{2} + \frac{47321}{33461} + \frac{\sqrt{\%1}}{2}, 1, \left\{ \begin{array}{c} \frac{1}{-\frac{33461\beta}{2} - \frac{33461\sqrt{\%1}}{2}} \\ \frac{1}{33461\alpha} \end{array} \right\} \right],$$


$$\left[ \frac{\beta}{2} + \frac{47321}{33461} - \frac{\sqrt{\%1}}{2}, 1, \left\{ \begin{array}{c} \frac{1}{-\frac{33461\beta}{2} + \frac{33461\sqrt{\%1}}{2}} \\ \frac{1}{33461\alpha} \end{array} \right\} \right]$$


$$\%1 := \beta^2 + 8\alpha^2$$

> evalf(%);


$$\left[ \left[ -1.0000\beta + 1.4142, 1.0000, \left\{ \begin{array}{c} -1.0000 \\ 0.0000 \\ 1.0000 \end{array} \right\} \right],$$


$$\left[ 0.5000\beta + 1.4142 + 0.5000\sqrt{\%1}, 1.0000,$$


$$\left\{ \begin{array}{c} 1.0000 \\ -\frac{0.0000(16730.5000\beta - 16730.5000\sqrt{\%1})}{\alpha} \\ 1.0000 \end{array} \right\} \right],$$


$$\left[ \left[ 0.5000\beta + 1.4142 - 0.5000\sqrt{\%1}, 1.0000,$$


$$\left\{ \begin{array}{c} 1.0000 \\ -\frac{0.0000(16730.5000\beta + 16730.5000\sqrt{\%1})}{\alpha} \\ 1.0000 \end{array} \right\} \right]\right]$$


$$\%1 := \beta^2 + 8.0000\alpha^2$$


```

In Table 18.3 we give a survey of basic functions that are available in the **LinearAlgebra** package.

Procedure	Computed Object
<b>Adjoint</b>	adjoint of a matrix
<b>CharacteristicMatrix</b>	characteristic matrix
<b>CharacteristicPolynomial</b>	characteristic polynomial
<b>ConditionNumber</b>	condition number of a matrix
<b>Determinant</b>	determinant of a matrix
<b>Eigenvalues</b>	eigenvalues of a matrix
<b>Eigenvectors</b>	eigenvectors of a matrix
<b>HermitianTranspose</b>	Hermitian transpose of a matrix
<b>MatrixInverse</b>	inverse of a matrix
<b>MinimalPolynomial</b>	minimum polynomial of a matrix
<b>Minor</b>	minor of a matrix
<b>NullSpace</b>	kernel of a matrix
<b>Permanent</b>	permanent of a matrix
<b>Rank</b>	rank of a matrix
<b>SingularValues</b>	singular values of a matrix
<b>Trace</b>	trace of a matrix
<b>Transpose</b>	transpose of a matrix

Table 18.3. Basic functions in the **LinearAlgebra** package.

## 18.5 Structural Operations

Maple provides many functions for structural operations of matrices; we list them in Table 18.4

Procedure	Purpose
<b>Column</b>	extract column(s) from a matrix as vector(s)
<b>ColumnDimension</b>	determine column dimension of a matrix
<b>ColumnOperation</b>	perform elementary column operation(s)
<b>DeleteColumn</b>	delete column(s) of a matrix
<b>DeleteRow</b>	delete row(s) of a matrix
<b>Dimension</b>	determine the dimension of a matrix or vector
<b>Row</b>	extract row(s) from a matrix as vector(s)
<b>RowDimension</b>	determine row dimension of a matrix
<b>RowOperation</b>	perform elementary row operation(s)
<b>SubMatrix</b>	extract a submatrix from a matrix
<b>SubVector</b>	extract a subvector from a matrix

Table 18.4. Structural operators.

The **LinearAlgebra** package also contains functions to verify whether a matrix belongs to a certain class or is related to another matrix; see Table 18.5 below.

Procedure	Test for
<b>Equal</b>	equality of matrices
<b>IsDefinite</b>	positive (or negative) definite matrix
<b>IsOrthogonal</b>	orthogonal matrix
<b>IsSimilar</b>	similarity of matrices
<b>IsUnitary</b>	unitary matrix
<b>IsZero</b>	zero matrix

Table 18.5. Test functions for matrices.

Two examples will do. We start with a Jordan block and an identity matrix of the same dimension.

```
> J := JordanBlockMatrix([[2,4]]);
```

$$J := \begin{bmatrix} 2 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 \\ 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 2 \end{bmatrix}$$

```
> Id := IdentityMatrix(4);
```

$$Id := \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

We join these matrices horizontally.

```
> <J|Id>;
```

$$\begin{bmatrix} 2 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 2 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 & 1 \end{bmatrix}$$

We use **ReducedRowEchelonForm** to reduce this matrix to its row reduced echelon form.

```
> ReducedRowEchelonForm(%);
```

$$\begin{bmatrix} 1 & 0 & 0 & 0 & \frac{1}{2} & -\frac{1}{4} & \frac{1}{8} & -\frac{1}{16} \\ 0 & 1 & 0 & 0 & 0 & \frac{1}{2} & -\frac{1}{4} & \frac{1}{8} \\ 0 & 0 & 1 & 0 & 0 & 0 & \frac{1}{2} & -\frac{1}{4} \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & \frac{1}{2} \end{bmatrix}$$

The  $4 \times 4$  submatrix on the right should be the inverse of the original Jordan block matrix.

```
> Jinv := SubMatrix(% , [1..4] , [5..8]);
```

$$Jinv := \begin{bmatrix} \frac{1}{2} & -\frac{1}{4} & \frac{1}{8} & -\frac{1}{16} \\ 0 & \frac{1}{2} & -\frac{1}{4} & \frac{1}{8} \\ 0 & 0 & \frac{1}{2} & -\frac{1}{4} \\ 0 & 0 & 0 & \frac{1}{2} \end{bmatrix}$$

```
> Equal(Jinv, MatrixInverse(J));
```

*true*

Knowing the matrix that we started with, the following Jordan block matrix form of *Jinv* comes as no surprise.

```
> JordanForm(Jinv);
```

$$\begin{bmatrix} \frac{1}{2} & 1 & 0 & 0 \\ 0 & \frac{1}{2} & 1 & 0 \\ 0 & 0 & \frac{1}{2} & 1 \\ 0 & 0 & 0 & \frac{1}{2} \end{bmatrix}$$

The second example will demonstrate the matrix test functions. Consider the following two matrices.

```
> A := Matrix([[[(sqrt(5)+sqrt(2)*sqrt(5-sqrt(5))+1)/4,
> -sqrt(2)*sqrt(5-sqrt(5))/2],
> [sqrt(2)*sqrt(5-sqrt(5))/4,
> (sqrt(5)-sqrt(2)*sqrt(5-sqrt(5))+1)/4]]];
```

$$A := \begin{bmatrix} \frac{\sqrt{5}}{4} + \frac{\sqrt{2}\sqrt{5-\sqrt{5}}}{4} + \frac{1}{4} & -\frac{\sqrt{2}\sqrt{5-\sqrt{5}}}{2} \\ \frac{\sqrt{2}\sqrt{5-\sqrt{5}}}{4} & \frac{\sqrt{5}}{4} - \frac{\sqrt{2}\sqrt{5-\sqrt{5}}}{4} + \frac{1}{4} \end{bmatrix}$$

$$B := \begin{bmatrix} \frac{\sqrt{5}}{4} + \frac{1}{4} & -\frac{\sqrt{2}\sqrt{5-\sqrt{5}}}{4} \\ \frac{\sqrt{2}\sqrt{5-\sqrt{5}}}{4} & \frac{\sqrt{5}}{4} + \frac{1}{4} \end{bmatrix}$$

We verify that they are similar, i.e., that there exists a nonsingular matrix  $T$  such that  $B = T A T^{-1}$ .

```
> q, T := IsSimilar(A, B, output=['query', 'C']);
```

$$\begin{aligned} q, T &:= \text{true}, \\ &[1, 0] \\ &\left[ \frac{3\sqrt{2}\sqrt{5-\sqrt{5}}\sqrt{5}}{40} - \frac{1}{2} + \frac{\sqrt{2}\sqrt{5-\sqrt{5}}}{8} \right. \\ &- \frac{\sqrt{2}\sqrt{5-\sqrt{5}}(5+\sqrt{5})(\sqrt{5}+\sqrt{2}\sqrt{5-\sqrt{5}}+1)}{80}, \\ &\left. 1 - \frac{(-5+\sqrt{5})(5+\sqrt{5})}{20} \right] \end{aligned}$$

```
> T := map(simplify, T);
```

$$T := \begin{bmatrix} 1 & 0 \\ -1 & 2 \end{bmatrix}$$

```
> Equal(B, map(simplify, T.A.T^(-1)));
```

*true*

The matrix  $B$  represents the counter-clockwise rotation over the angle  $\frac{\pi}{5}$ . So it is an orthogonal matrix. The **IsOrthogonal** procedure confirms this.

```
> IsOrthogonal(B);
```

*true*

Of course you can also apply basic methods.

```
> simplify(B.Transpose(B));
```

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

## 18.6 Vector Operations

The following vector operations are available. We refer to §19.4 for examples from vector analysis.

Procedure	Computed Object
<b>Basis</b>	basis for a vector space
<b>ColumnSpace</b>	basis for the column space of a matrix
<b>CrossProduct</b>	cross product (outer product) of two vectors
<b>DotProduct</b>	dot product (inner product) of two vectors
<b>GramSchmidt</b>	Gram-Schmidt orthogonalization of vectors
<b>IntersectionBasis</b>	basis for the intersection of vector spaces
<b>MatrixNorm</b>	norm of a matrix
<b>Norm</b>	norm of a matrix or a vector
<b>Normalize</b>	normalized vector
<b>NullSpace</b>	basis for the null space
<b>RowSpace</b>	basis for the row space of a matrix
<b>SumBasis</b>	basis for the sum of vector spaces
<b>VectorAngle</b>	angle between two vectors
<b>VectorNorm</b>	norm of a vector

Table 18.6. Vector operations.

We only give a simple example of computing bases for vector spaces. Let  $U$  be the vector space spanned by the following 3 vectors

```
> u1 := <0,2,3,-1>; u2 := <0,2,7,-2>; u3 := <0,-2,1,0>;
> 'u1'=u1, 'u2'=u2, 'u3'=u3;
```

$$u1 = \begin{bmatrix} 0 \\ 2 \\ 3 \\ -1 \end{bmatrix}, u2 = \begin{bmatrix} 0 \\ 2 \\ 7 \\ -2 \end{bmatrix}, u3 = \begin{bmatrix} 0 \\ -2 \\ 1 \\ 0 \end{bmatrix}$$

and let the vector space  $V$  be spanned by

```
> v1 := <1,2,0,1>; v2 := <2,2,1,2>; 'v1'=v1, 'v2'=v2;
```

$$v1 = \begin{bmatrix} 1 \\ 2 \\ 0 \\ 1 \end{bmatrix}, v2 = \begin{bmatrix} 2 \\ 2 \\ 1 \\ 2 \end{bmatrix}$$

Below, we compute a basis for  $U$ ,  $U + V$  and  $U \cap V$ .

```
> Basis([u1,u2,u3]);
```

$$\left[ \begin{bmatrix} 0 \\ 2 \\ 3 \\ -1 \end{bmatrix}, \begin{bmatrix} 0 \\ 2 \\ 7 \\ -2 \end{bmatrix} \right]$$

```
> SumBasis([[u1,u2,u3],[v1,v2]]);  


$$\left[ \begin{bmatrix} 0 \\ 2 \\ 3 \\ -1 \end{bmatrix}, \begin{bmatrix} 0 \\ 2 \\ 7 \\ -2 \end{bmatrix}, \begin{bmatrix} 1 \\ 2 \\ 0 \\ 1 \end{bmatrix} \right]$$
  

> IntersectionBasis([[u1,u2,u3],[v1,v2]]);  


$$\left[ \begin{bmatrix} 0 \\ -2 \\ 1 \\ 0 \end{bmatrix} \right]$$

```

## 18.7 Standard Forms of Matrices

Maple provides facilities for various standard forms of matrices. They are listed in Table 18.7.

Procedure	Standard Form
<b>BidiagonalForm</b>	bidiagonal form over floats
<b>FrobeniusForm</b>	Frobenius form (rational canonical form)
<b>GaussianElimination</b>	Gaussian elimination
<b>HermiteForm</b>	Hermite normal form over univariate polynomials
<b>HessenbergForm</b>	upper Hessenberg form
<b>JordanForm</b>	Jordan form
<b>LUDecomposition</b>	LU decomposition
<b>PopovForm</b>	Popov form (polynomial echelon form) over univariate polynomials
<b>QRDecomposition</b>	QR decomposition
<b>ReducedRowEchelonForm</b>	Gauss-Jordan form (row reduced echelon form)
<b>SchurForm</b>	Schur form over floats
<b>SmithForm</b>	Smith normal form over integers and univariate polynomials
<b>TridiagonalForm</b>	tridiagonal form for real symmetric or complex hermitian matrix

Table 18.7. Test functions for matrices.

**Frobenius**, **Gausselim**, **Gaussjord**, **Hermite**, and **Smith** are equivalent but inert functions; you use them when you calculate over a finite fields or over integer modulo some natural number, i.e., when you want to carry out computations of abstract linear algebra nature.

Examples will give you a better idea of the power of the facilities. We begin with the computation of the LU decomposition of the matrix

$$A = \begin{pmatrix} 0 & -1 & -3 \\ 4 & 5 & 1 \\ 1 & 4 & -1 \end{pmatrix}.$$

We search for a lower-triangular matrix  $L$  and an upper-triangular matrix  $U$  such that  $A = P L U$  for some permutation matrix  $P$ .

```
> A := <<0|-1|-3|1>, <1|4|1|2>, <1|2|-1|3>>;
      
$$A := \begin{bmatrix} 0 & -1 & -3 & 1 \\ 1 & 4 & 1 & 2 \\ 1 & 2 & -1 & 3 \end{bmatrix}$$

> P, L, U := LUdecomposition(A, output=['P', 'L', 'U']):
> 'P' = P;
      
$$P = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

> 'L' = L;
      
$$L = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

> 'U' = U;
      
$$U = \begin{bmatrix} 1 & 4 & 1 & 2 \\ 0 & -1 & -3 & 1 \\ 0 & 0 & 4 & -1 \end{bmatrix}$$

> Equal(P.L.U, A);
      true
```

The PLU1R decomposition is achieved by using **LUdecomposition** with the options **method='RREF'** or **output=['P', 'L', 'U1', 'R']**). This factors  $U$  into  $U1 R$ , where  $U1$  is the square upper triangular factor and  $R$  is the unique reduced row-echelon form of the Matrix  $A$ . In this case,  $A = P L U1 R$ . Another construction option is **outputoptions**.

```
> P, L, U1, R := LUdecomposition(A, method='RREF',
>       outputoptions['U1']=[datatype=integer]):
> U1, R;
```

$$\begin{bmatrix} 1 & 4 & 1 \\ 0 & -1 & -3 \\ 0 & 0 & 4 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 & \frac{13}{4} \\ 0 & 1 & 0 & \frac{-1}{4} \\ 0 & 0 & 1 & \frac{-1}{4} \end{bmatrix}$$

```
> Equal(U, U1.R);
true
```

The Cholesky decomposition of a positive-definite symmetric matrix  $A$  is a lower-triangular matrix  $L$  such that  $A = LL^t$ . We let Maple compute this decomposition for the following matrix.

```
> A := <<5|-2|2>, <-2|4|1>, <2|1|2>>;
```

$$A := \begin{bmatrix} 5 & -2 & 2 \\ -2 & 4 & 1 \\ 2 & 1 & 2 \end{bmatrix}$$

```
> IsDefinite(A, 'query'='positive_definite');
true
```

```
> L := LUdecomposition(A, method='Cholesky');
```

$$L := \begin{bmatrix} \sqrt{5} & 0 & 0 \\ -\frac{2\sqrt{5}}{5} & \frac{4\sqrt{5}}{5} & 0 \\ \frac{2\sqrt{5}}{5} & \frac{9\sqrt{5}}{20} & \frac{\sqrt{3}}{4} \end{bmatrix}$$

```
> Equal(L.Transpose(L), A);
true
```

Let us compute the tridiagonal form of matrix  $A$ . We raise the information level so that you better see what will happen.

```
> infolevel[LinearAlgebra] := 1: TridiagonalForm(A);
```

```
TridiagonalForm: "calling external function"
TridiagonalForm: "NAG" hw_f08fef
```

```
[5., 2.82842712474619029, 0.]
[2.82842712474619029, 1.99999999999999822,
 -0.99999999999999856]
[0., -0.99999999999999856, 3.9999999999999912]
```

Note the many decimal numbers that appear in the floating-point number of the output. The reason is revealed in the information messages: Maple calls a NAG routine, which prefers to work with hardware floating-point numbers because of efficiency. To avoid this behavior, you must assign `UseHardwareFloats` the boolean value `false`.

```
> UseHardwareFloats := false:
> TridiagonalForm(A);
```

```
TridiagonalForm: "calling external function"
TridiagonalForm: "NAG" sw_f08fef
```

$$\begin{bmatrix} 5. & 2.828427124 & 0. \\ 2.828427124 & 1.999999998 & -0.999999999 \\ 0. & -0.999999999 & 4.000000000 \end{bmatrix}$$

In the information message you see the prefix `sw_` instead of `hw_` in front of the names of the external routines, indicating whether software or hardware floating-point number are used. Let us now change the output format and verify the answer.

```
> T, Q := TridiagonalForm(A, output=['T', 'Q']);
```

```
TridiagonalForm: "calling external function"
TridiagonalForm: "NAG" sw_f08fef
TridiagonalForm: "NAG" sw_f08fff
```

$$T, Q := \begin{bmatrix} 5. & 2.828427124 & 0. \\ 2.828427124 & 1.999999998 & -0.999999999 \\ 0. & -0.999999999 & 4.000000000 \end{bmatrix},$$

$$\begin{bmatrix} 1.0 & 0. & 0. \\ 0. & -0.707106781 & 0.7071067812 \\ 0. & 0.7071067812 & 0.7071067812 \end{bmatrix}$$

```
> map(fnormal, Q.T.Transpose(Q) - A);
```

```
Transpose: "calling external function"
HermitianTranspose: sw_MatTransRR
MatrixMatrixMultiply: "calling external function"
MatrixMatrixMultiply: "NAG" sw_f06yaf
MatrixMatrixMultiply: "calling external function"
MatrixMatrixMultiply: "NAG" sw_f06yaf
MatrixAdd: "calling external function"
MatrixAdd: "NAG" sw_f06ecf
```

$$\begin{bmatrix} 0. & 0. & 0. \\ 0. & -0. & 0. \\ 0. & 0. & 0. \end{bmatrix}$$

**LUDecomposition** offers various methods and output formats. The next example gives you an idea of the flexibility of this procedure.

```
> A := <<x|-2|2>, <-2|4|1>, <2|1|2>>;
```

$$A := \begin{bmatrix} x & -2 & 2 \\ -2 & 4 & 1 \\ 2 & 1 & 2 \end{bmatrix}$$

```
> LUDecomposition(A); # Gaussian elimination over Q(x)
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ -\frac{2}{x} & 1 & 0 \\ \frac{2}{x} & \frac{x+4}{4(x-1)} & 1 \end{bmatrix}, \begin{bmatrix} x & -2 & 2 \\ 0 & \frac{4(x-1)}{x} & \frac{x+4}{x} \\ 0 & 0 & \frac{7x-32}{4(x-1)} \end{bmatrix}$$

```
> LUDecomposition(A, method=FractionFree);
```

$$\left[ \begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array} \right], \left[ \begin{array}{ccc} 1 & 0 & 0 \\ -\frac{2}{x} & \frac{1}{x} & 0 \\ \frac{2}{x} & \frac{x+4}{x(4x-4)} & \frac{1}{4x-4} \end{array} \right], \left[ \begin{array}{ccc} x & -2 & 2 \\ 0 & 4x-4 & x+4 \\ 0 & 0 & 7x-32 \end{array} \right]$$

Next, a small change in the input: an integer is converted into a floating-point number.

```
> A := <<x|-2.0|2>, <-2|4|1>, <2|1|2>>;
```

$$A := \left[ \begin{array}{ccc} x & -2.0 & 2 \\ -2 & 4 & 1 \\ 2 & 1 & 2 \end{array} \right]$$

```
> LUDecomposition(A); # mixed types of matrix entries
```

$$\left[ \begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array} \right], \left[ \begin{array}{ccc} 1 & 0 & 0 \\ -\frac{2}{x} & 1 & 0 \\ \frac{2}{x} & \frac{x+4.}{4.x-4.} & 1 \end{array} \right], \left[ \begin{array}{ccc} x & -2.0 & 2 \\ 0 & \frac{4x-4.0}{x} & \frac{x+4}{x} \\ 0 & 0 & \frac{7.x-32.}{4.x-4.} \end{array} \right]$$

```
> LUDecomposition(A, method=FractionFree);
```

$$\left[ \begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array} \right], \left[ \begin{array}{ccc} 1 & 0 & 0 \\ -\frac{2}{x} & \frac{1}{x} & 0 \\ \frac{2}{x} & \frac{x+4.0}{x(4x-4.0)} & \frac{1}{4x-4.0} \end{array} \right],$$

$$\left[ \begin{array}{ccc} x & -2.0 & 2 \\ 0 & 4x-4.0 & x+4 \\ 0 & 0 & 7x-32.0 \end{array} \right]$$

With the option **output=NAG** and the **Cholesky** method not chosen, Maple returns the decomposition in the packed form of an expression sequence consisting of a vector followed by a matrix. The upper triangle of the matrix is the *U* factor and the strictly lower triangle is the *L* factor (with implicit ones along the diagonal). The vector is the NAG-style pivot vector. Maple can use this packed form immediately in the **LinearSolve** procedure for solving a system of linear equations described by its matrix of coefficients.

```
> LUDecomposition(A, output=NAG);
```

$$\left[ \begin{array}{c} 1 \\ 2 \\ 3 \end{array} \right], \left[ \begin{array}{ccc} x & -2.0 & 2 \\ -\frac{2}{x} & \frac{4x-4.0}{x} & \frac{x+4}{x} \\ \frac{2}{x} & \frac{x+4.}{4.x-4.} & \frac{7.x-32.}{4.x-4.} \end{array} \right]$$

Let  $A(x)$  be a matrix of polynomials over a field. Then there exist invertible matrices  $P$  and  $Q$  over this field, such that

$$A = P \operatorname{diag}(a_1(x), a_2(x), \dots, a_k(x), 0, 0, \dots, 0) Q,$$

where  $a_i$ 's are polynomials for which  $a_i$  divides  $a_{i+1}$ . The  $a_i$ 's are called invariant factors; they are unique up to multiplication by non-zero elements in the field, and  $k$  is fixed by  $A$ . The diagonal matrix is called the Smith normal form of  $A(x)$ . Let us consider the following matrix over  $\mathbb{Q}[x]$ .

```
> A := Matrix([[1,0,-1,1], [-1,x+1,-x,-x-2]
> [-1,x+1, 2*x^3+2*x^2-3*x-2, 2*x^3+2*x^2-3*x-4],
> [0,-x-1,2*x^3+2*x^2-x-1,2*x^3+2*x^2-x-1]]);
```

$$A := \begin{bmatrix} 1 & 0 & -1 & 1 \\ -1 & x+1 & -x & -x-2 \\ -1 & x+1 & 2x^3 + 2x^2 - 3x - 2 & 2x^3 + 2x^2 - 3x - 4 \\ 0 & -x-1 & 2x^3 + 2x^2 - x - 1 & 2x^3 + 2x^2 - x - 1 \end{bmatrix}$$

The Smith normal form can be computed in Maple as follows.

```
> S, U, V := SmithForm(A, output=['S', 'U', 'V']):
> 'S' = S;
```

$$S = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & x+1 & 0 \\ 0 & 0 & 0 & x^3 + x^2 - x - 1 \end{bmatrix}$$

```
> P := U^(-1);
```

$$P := \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 + 2x^3 + 2x^2 - x & -5 + 4x^2 & 4 \\ -1 & -3 + 4x^3 + 4x^2 - 3x & -9 + 8x^2 & 8 \\ 0 & -x - 1 & 1 & 0 \end{bmatrix}$$

```
> Q := V^(-1);
```

$$Q := \begin{bmatrix} 1 & 0 & -1 & 1 \\ 0 & 0 & 1 & -1 \\ 0 & -1 & 2x^2 & 2x^2 - 2 \\ 0 & 1 & -2x^2 & -2x^2 + 3 \end{bmatrix}$$

```
> Equal(map(expand, P.S.Q), A);
```

*true*

Next, we consider the matrix  $A$  over  $\mathbb{Z}_2[x]$ , i.e., we compute modulo 2.

```
> A2 := map('mod', A, 2); # create new matrix
```

$$A2 := \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & x+1 & x & x \\ 1 & x+1 & x & x \\ 0 & x+1 & x+1 & x+1 \end{bmatrix}$$

```
> SmithForm(A2, x, output=['S', 'U', 'V']); # over Q(x)
```

$$\left[ \begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & x+1 & 0 \\ 0 & 0 & 0 & 0 \end{array} \right], \left[ \begin{array}{cccc} 1 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{-1}{2} & \frac{1}{2} \\ \frac{x}{2} + \frac{1}{2} & 0 & -\frac{x}{2} - \frac{1}{2} & \frac{x}{2} - \frac{1}{2} \\ 0 & -1 & 1 & 0 \end{array} \right],$$

$$\left[ \begin{array}{cccc} 1 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 1 \end{array} \right]$$

```
> Smith(A2, x) mod 2; # over Z_2(x)
```

$$\left[ \begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & x+1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array} \right]$$

Note that this is different from computing the Smith normal form over  $\mathbb{Q}(x)$  and afterwards doing modular arithmetic. The same phenomenon holds for the other standard form reductions. Let us compute some of them using the last matrix  $A2$ .

```
> F, Q := FrobeniusForm(A2, output=['F', 'Q']); # over Q(x)
```

$$F, Q := \left[ \begin{array}{cccc} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 2x+2 \\ 0 & 1 & 0 & -4x-2 \\ 0 & 0 & 1 & 3x+3 \end{array} \right], \left[ \begin{array}{cccc} \frac{1}{2} & 1 & 1 & 2 \\ \frac{-1}{2} & 0 & 1 & 2x+2 \\ \frac{1}{2} & 0 & 1 & 2x+2 \\ 0 & 0 & 0 & 2x+2 \end{array} \right]$$

```
> Equal(map(normal, Q.F.Q^(-1)), A2);
```

*true*

```
> F := Frobenius(A2, 'Q') mod 2; # over Z_2[x]
```

$$F := \left[ \begin{array}{cccc} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & x+1 & 0 \\ 0 & 0 & 0 & 0 \end{array} \right]$$

We verify the last result over  $\mathbb{Z}_2$ .

```
> map(z -> Normal(z) mod 2, Q.F.Q^(-1));
```

$$\begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & x+1 & x & x \\ 1 & x+1 & x & x \\ 0 & x+1 & x+1 & x+1 \end{bmatrix}$$

```
> Equal(%, A2);
```

true

We compute the Hermite normal form of  $A2$  over various fields.

```
> H,U := HermiteForm(A2, x, output=['H', 'U']): # over Q[x]
```

Adjoint/univar: "input Matrix dimension" 4 4

Adjoint/univar: "coefficient bound is about" 1 "primes long"

Adjoint/univar: "reconstructing using Chinese Remainder Algorithm"

```
> 'H' = Map(expand, H);
```

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & x+1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

```
> Equal(map(expand, U.A2), H);
```

true

```
> H := Hermite(A2, x, 'U') mod 2; # over Z_2[x]
```

$$H := \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & x+1 & x+1 & x+1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

```
> Equal(map(z -> Expand(z) mod 2, U.A2), H);
```

true

We continue with Gaussian elimination and Gauss-Jordan forms over various fields, using various methods.

```
> A2; # recall the sample matrix
```

$$\begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & x+1 & x & x \\ 1 & x+1 & x & x \\ 0 & x+1 & x+1 & x+1 \end{bmatrix}$$

```
> GaussianElimination(A2); # Gaussian elimination over Q(x)
```

$$\begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & x+1 & x-1 & x-1 \\ 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

**GaussianElimination** is equivalent to the call of **LUDecomposition** with the option `output=['U']`. It is in fact the default method for **LUDecomposition**.

```
> LUDecomposition(A2, output=['U']);
```

$$\left[ \begin{array}{cccc} 1 & 0 & 1 & 1 \\ 0 & x+1 & x-1 & x-1 \\ 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & 0 \end{array} \right]$$

Compare the following two results to appreciate the method **FractionFree**.

```
> LUDecomposition(A2);
```

$$\left[ \begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{array} \right], \left[ \begin{array}{cccc} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{array} \right], \left[ \begin{array}{cccc} 1 & 0 & 1 & 1 \\ 0 & x+1 & x-1 & x-1 \\ 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & 0 \end{array} \right]$$

```
> LUDecomposition(A2, method=FractionFree);
```

$$\left[ \begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{array} \right], \left[ \begin{array}{cccc} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & \frac{1}{x+1} & 0 \\ 1 & 1 & 0 & 1 \end{array} \right], \left[ \begin{array}{cccc} 1 & 0 & 1 & 1 \\ 0 & x+1 & x-1 & x-1 \\ 0 & 0 & 2x+2 & 2x+2 \\ 0 & 0 & 0 & 0 \end{array} \right]$$

The Gauss-Jordan elimination on a matrix can be computed via the **RREF** method in **LUDecomposition**. The fourth term in the default output is the requested matrix. If **LUDecomposition** is called with the option `output='R'`, then only the row-reduced echelon form is returned.

```
> LUDecomposition(A2, method=RREF);
```

$$\left[ \begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{array} \right], \left[ \begin{array}{cccc} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{array} \right], \left[ \begin{array}{cccc} 1 & 0 & 1 & 0 \\ 0 & x+1 & x-1 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{array} \right]$$

$$\left[ \begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{array} \right]$$

```
> # Gauss-Jordan form over Z_2(x)
```

```
> LUDecomposition(A2, output='R');
```

$$\left[ \begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{array} \right]$$

```
> Gausselim(A2) mod 2; # Gaussian elimination over Z_2[x]
```

$$\begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & x+1 & x+1 & x+1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

```
> Gaussjord(A) mod 2; # Gauss-Jordan form over Z_2(x)
```

$$\begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

The **Modular** subpackage of **LinearAlgebra** also provides a procedure for LU decomposition of matrices, but it returns its answer in the packed format described before (when we discussed the **output=NAG** option of **LUDecomposition**) and at the same time it replaces the original matrix. Let us see how it works for the following matrix modulo 3.

```
> A3 := map('mod', A, 3);
```

$$A3 := \begin{bmatrix} 1 & 0 & 2 & 1 \\ 1 & x+1 & 2x & 2x+1 \\ 2 & x+1 & 2x^3 + 2x^2 + 1 & 2x^3 + 2x^2 + 2 \\ 0 & 2x+2 & 2x^3 + 2x^2 + 2x + 2 & 2x^3 + 2x^2 + 2x + 2 \end{bmatrix}$$

```
> a3 := Modular[Copy](3, A3): pv := Vector(4):
> Modular[LUDecomposition](3, a3, pv, 'det'):
> a3 := map(z -> Normal(z) mod 3, a3):
> det := Normal(det) mod 3:
> a3, pv, det;
```

$$\begin{bmatrix} 1 & 0 & 2 & 1 \\ 1 & x+1 & 2x+1 & 2x \\ 2 & 1 & 2x^3 + 2x^2 + x + 2 & 2x^3 + 2x^2 + x \\ 0 & 2 & \frac{x^3 + x^2 + 2x}{x^3 + x^2 + 2x + 1} & \frac{x^3 + x^2 + 2x + 2}{x^3 + x^2 + 2x + 1} \end{bmatrix}, \begin{bmatrix} 1 \\ 2 \\ 3 \\ 0 \end{bmatrix}$$

$$2x^4 + x^3 + 2x + 1$$

```
> U := Matrix(4,4,shape=triangular[upper], a3);
```

$$U := \begin{bmatrix} 1 & 0 & 2 & 1 \\ 0 & x+1 & 2x+1 & 2x \\ 0 & 0 & 2x^3 + 2x^2 + x + 2 & 2x^3 + 2x^2 + x \\ 0 & 0 & 0 & \frac{x^3 + x^2 + 2x + 2}{x^3 + x^2 + 2x + 1} \end{bmatrix}$$

```
> L := Matrix(4,4,shape=triangular[lower,unit], a3);
```

$$L := \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 2 & 1 & 1 & 0 \\ 0 & 2 & \frac{x^3 + x^2 + 2x}{x^3 + x^2 + 2x + 1} & 1 \end{bmatrix}$$

```

> map(z -> Normal(z) mod 3, L.U);

$$\begin{bmatrix} 1 & 0 & 2 & 1 \\ 1 & x+1 & 2x & 2x+1 \\ 2 & x+1 & 2x^3 + 2x^2 + 1 & 2x^3 + 2x^2 + 2 \\ 0 & 2x+2 & 2x^3 + 2x^2 + 2x + 2 & 2x^3 + 2x^2 + 2x + 2 \end{bmatrix}$$

> Equal(% , A3);
true

```

## 18.8 Numeric Linear Algebra

When Maple computes with numeric matrices, it has two options, viz., to use hardware floating-point numbers or to use software floating-point numbers. The advantage of using hardware floating-point numbers is that execution is more quickly than for software floating-point numbers and that external routines of compiled code from the Numerical Algorithms Group (NAG) can be used. The obvious drawback of hardware floats is that precision is fixed to machine-precision.

Software floating-point numbers can be used at any desired precision, but high precision may lead to serious performance problems. In order to do computations with large matrices, Maple allows the use of compiled NAG routines with software floats.

The first example will show that Maple decides all by itself at what precision to operate. To this end, we have raised the information level.

```

> with(LinearAlgebra):
> infolevel[LinearAlgebra] := 1:
> M1 := <<0,-2,1>|<1,-3,0>|<1,2,-1>>:
> M2 := <<0,-2.0,1>|<1,-3,0>|<1,2,-1>>:
> 'M1'=M1, 'M2'=M2;

```

$$M1 = \begin{bmatrix} 0 & 1 & 1 \\ -2 & -3 & 2 \\ 1 & 0 & -1 \end{bmatrix}, M2 = \begin{bmatrix} 0 & 1 & 1 \\ -2.0 & -3 & 2 \\ 1 & 0 & -1 \end{bmatrix}$$

The matrix **M1** only consists of integers, while the matrix **M2** is the same except for one entry that has been changed into a software floating-point number. This small change in the matrix entry makes a large difference in the computation of the matrix inverse.

```

> M1^(-1);


$$\begin{bmatrix} 1 & \frac{1}{3} & \frac{5}{3} \\ 0 & \frac{-1}{3} & \frac{-2}{3} \\ 1 & \frac{1}{3} & \frac{2}{3} \end{bmatrix}$$


> M2^(-1);

MatrixInverse: "calling external function"
MatrixInverse: "NAG"   hw_f07adf
MatrixInverse: "NAG"   hw_f07ajf


$$\begin{bmatrix} 1. & 0.33333333333333259 & 1.6666666666666652 \\ 0. & -0.3333333333333315 & -0.66666666666666630 \\ 1. & 0.3333333333333315 & 0.66666666666666630 \end{bmatrix}$$


```

In the first case, Maple applied a symbolic (exact) algorithm, while it called an external routine from the NAG library to invert the matrix containing a software float. Note that Maple did the computation using hardware floating-point numbers. To avoid this, you must assign `UseHardwareFloats` the boolean value `false`.

```

> UseHardwareFloats := false;
      UseHardwareFloats := false

> M2^(-1);

MatrixInverse: "calling external function"
MatrixInverse: "NAG"   sw_f07adf
MatrixInverse: "NAG"   sw_f07ajf


$$\begin{bmatrix} 1.000000000 & 0.3333333335 & 1.666666667 \\ -0. & -0.3333333334 & -0.6666666667 \\ 1.000000000 & 0.3333333334 & 0.6666666667 \end{bmatrix}$$


```

Maple still uses the NAG routine but does not switch to hardware floating-point arithmetic. In the information messages you see the prefix `sw_` instead of `hw_` in front of the names of the external routines.

If you always want to use hardware floats, assign `UseHardwareFloats` the value `true`. The default value of `UseHardwareFloats` is deduced. This means that Maple itself sorts out what algorithms can be applied best. It does this on the basis of the properties of the matrices involved. Let us see this in another example, viz., LU decomposition of a matrix.

```

> UseHardwareFloats := deduced;
      UseHardwareFloats := deduced

> LUDecomposition(M1); # exact arithmetic

```

```


$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -\frac{1}{2} & -\frac{3}{2} & 1 \end{bmatrix}, \begin{bmatrix} -2 & -3 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & \frac{3}{2} \end{bmatrix}$$

> LUdecomposition(M2); # using hardware floats
LUdecomposition: "calling external function"
LUdecomposition: "NAG" hw_f07adf


$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix},$$


$$\begin{bmatrix} 1.0 & 0. & 0. \\ -0.5000000000000000 & 1.0 & 0. \\ 0. & -0.66666666666666630 & 1.0 \end{bmatrix},$$


$$\begin{bmatrix} -2. & -3. & 2. \\ 0. & -1.5000000000000000 & 0. \\ 0. & 0. & 1. \end{bmatrix}$$

> LUdecomposition(M2, output=['NAG']);
LUdecomposition: "calling external function"
LUdecomposition: "NAG" hw_f07adf


$$\begin{bmatrix} 2 \\ 3 \\ 3 \end{bmatrix}, \begin{bmatrix} -2. & -3. & 2. \\ -0.5000000000000000 & -1.5000000000000000 & 0. \\ 0. & -0.66666666666666630 & 1. \end{bmatrix}$$


```

The packed output format minimizes storage allocation. It is understood by the **LinearSolve** procedure for solving systems of linear equations in matrix-style.

```

> LinearSolve([%], <1,2,3>);

LinearSolve: "calling external function"
LinearSolve: "NAG" hw_f07aef


$$\begin{bmatrix} 6.6666666666666606 \\ -2.6666666666666652 \\ 3.6666666666666652 \end{bmatrix}$$


```

Let us verify the answer by computing it in two other ways.

```

> GenerateEquations(M2, [x,y,z], <1,2,3>);
[y + z = 1, -2.0 x - 3 y + 2 z = 2, x - z = 3]
> solve({op(%)}, {x,y,z});
{z = 3.6666666667, y = -2.666666667, x = 6.666666667}
> LinearSolve(M2, <1,2,3>);

```

```

LinearSolve: "calling external function"
LinearSolve: "NAG"   hw_f07adf
LinearSolve: "NAG"   hw_f07aef


$$\begin{bmatrix} 6.66666666666666606 \\ -2.66666666666666652 \\ 3.66666666666666652 \end{bmatrix}$$


```

When storage costs really are an issue, then there are two ways to minimize them:

- create your matrices with the right properties to avoid superfluous conversions and unnecessary copies being made during computations;
- use the `inplace` feature.

The two examples below show you that the `inplace` feature allows you to store the output of a procedure directly on top of one of the inputs. The only requirements for this feature are that the output must be of the same shape and size as one of the inputs, and that the algorithm in use has been coded truly in-place. i.e., it must not just copy the result into the input object when it is done.

```

> MatrixOptions(M1); # inspect properties

      shape = [], datatype = anything, storage = rectangular,
      order = Fortran_order
> M2 := evalhf(M1); # convert into hardware-floats

      M2 :=  $\begin{bmatrix} 0. & 1. & 1. \\ -2. & -3. & 2. \\ 1. & 0. & -1. \end{bmatrix}$ 

> MatrixOptions(M2);

      shape = [], datatype = float8, storage = rectangular,
      order = Fortran_order
> LUDecomposition(M2, output=['NAG'], 'inplace');

LUDecomposition: "calling external function"
LUDecomposition: "NAG"   hw_f07adf


$$\begin{bmatrix} 2 \\ 3 \\ 3 \end{bmatrix}, \begin{bmatrix} -2. & -3. & 2. \\ -0.5000000000000000 & -1.500000000000000 & 0. \\ 0. & -0.66666666666666630 & 1. \end{bmatrix}$$


> M2; # matrix has been changed


$$\begin{bmatrix} -2. & -3. & 2. \\ -0.5000000000000000 & -1.500000000000000 & 0. \\ 0. & -0.66666666666666630 & 1. \end{bmatrix}$$


> M1; # recall the matrix

```

```


$$\begin{bmatrix} 0 & 1 & 1 \\ -2 & -3 & 2 \\ 1 & 0 & -1 \end{bmatrix}$$

> Multiply(M1, M2);

MatrixMatrixMultiply: "calling external function"
MatrixMatrixMultiply: "NAG" hw_f06yaf


$$\begin{bmatrix} -0.5000000000000000 & -2.1666666666666652 & 1. \\ 5.500000000000000 & 9.1666666666666606 & -2. \\ -2. & -2.3333333333333348 & 1. \end{bmatrix}$$

> M1; # matrix has been left unchanged


$$\begin{bmatrix} 0 & 1 & 1 \\ -2 & -3 & 2 \\ 1 & 0 & -1 \end{bmatrix}$$

> Multiply(M1, M2, 'inplace');


$$\begin{bmatrix} -0.50000000000 & -2.166666667 & 1. \\ 5.500000000 & 9.166666667 & -2. \\ -2. & -2.333333333 & 1. \end{bmatrix}$$

> M1; # matrix has changed


$$\begin{bmatrix} -0.50000000000 & -2.166666667 & 1. \\ 5.500000000 & 9.166666667 & -2. \\ -2. & -2.333333333 & 1. \end{bmatrix}$$


```

## 18.9 Exercises

1. Consider the following matrices.

$$A = \begin{pmatrix} 1 & 0 & 2 \\ 2 & -1 & 3 \\ 4 & 1 & 8 \end{pmatrix}, \quad B = \begin{pmatrix} -3 & 2 \\ 0 & 1 \\ 7 & 4 \end{pmatrix}$$

Compute:

- (a)  $A^{-1}$
- (b)  $AA^t$
- (c)  $B^t AB$
- (d)  $(2A + BB^t) A^t$

2. Compute the Wronskian of the following list of functions:  $[\cos x, \sin x, e^x]$ . Compute the determinant of this matrix. Do the same computation for the list  $[\cosh x, \sinh x, e^x]$ .
3. Create a  $5 \times 5$  matrix with entries randomly chosen as univariate polynomials with integral coefficients, 3 terms, and degree less than 5. Compute the characteristic polynomial and verify the Cayley-Hamilton theorem by substitution of the matrix in the characteristic polynomial when it is collected

in the main variable. Also compute the Horner form of the characteristic polynomial and verify the Cayley-Hamilton theorem by substitution of the matrix in the polynomial in Horner form. Compare timings.

4. Consider the matrices

$$A = \begin{pmatrix} -4 & -7 & 0 \\ 0 & 4 & 2 \\ -5 & -7 & 1 \end{pmatrix}$$

and

$$B = \begin{pmatrix} -2 & -1 & -2 \\ 2 & 2 & -2 \\ 0 & 0 & 1 \end{pmatrix}.$$

Show that  $A$  and  $B$  are similar and compute the corresponding transformation matrix.

5. Let  $a, b \in \mathbb{R}$  with  $0 \leq a \leq 1$ ,  $b^2 = 2a(1-a)$ , and

$$A = \begin{pmatrix} a & a-1 & b \\ a-1 & a & b \\ -b & -b & 2a-1 \end{pmatrix}.$$

- (a) Check with Maple that  $A$  is an orthogonal matrix with determinant equal to one.  
 (b) From (a) follows that  $A$  is a matrix that describes a rotation in the standard basis of  $\mathbb{R}^3$ . Determine the rotation axis.

6. Let  $a, b \in \mathbb{R}$  and

$$A = \begin{pmatrix} 0 & a & 1 & 0 & b \\ 1 & 0 & 0 & b & 0 \\ 0 & 1 & b & 0 & 1 \\ b & 0 & 0 & 1 & 0 \\ 0 & b & 1 & 0 & b \end{pmatrix}.$$

- (a) For what values of  $a$  and  $b$  is the matrix  $A$  singular?  
 (b) Determine the inverse of  $A$  (for those values of  $a$  and  $b$  for which  $A$  is invertible).

7. (a) Compute  $\det \begin{pmatrix} x^2 + 1 & x & 0 & 0 \\ x & x^2 + 1 & x & 0 \\ 0 & x & x^2 + 1 & x \\ 0 & 0 & x & x^2 + 1 \end{pmatrix}$ .

- (b) Compute  $\det \begin{pmatrix} x^2 + 1 & x & 0 & 0 & 0 \\ x & x^2 + 1 & x & 0 & 0 \\ 0 & x & x^2 + 1 & x & 0 \\ 0 & 0 & x & x^2 + 1 & x \\ 0 & 0 & 0 & x & x^2 + 1 \end{pmatrix}$ .

- (c) Looking at the results of (a) and (b), do you have any idea what the determinant of a general matrix of the above form is? If so, check your conjecture for a  $8 \times 8$ -matrix. If not, compute the determinants for matrices of dimension 6 and 7 to get an idea.

8. For each natural number  $n$ , the  $n \times n$ -matrix  $A_n$  is defined as

$$A_n(i, j) = \begin{cases} 0, & \text{if } i = j, \\ 1, & \text{if } i \neq j. \end{cases}$$

Carry out the following computations for  $n = 3, 4$ , and  $5$ .

- (a) Compute the determinant of  $A_n$ .
- (b) Compute the characteristic polynomial of  $A_n$ .
- (c) Determine all eigenvalues of  $A_n$  and determine for each eigenvalue a basis of the corresponding eigenspace.

9. For each natural number  $n$ , the  $n \times n$ -matrix  $A_n$  is defined as

$$A_n(i, j) = \gcd(i, j).$$

- (a) Compute the determinant of  $A_n$  for  $n = 1, 2, \dots, 15$ .
- (b) (for the mathematicians amongst us) Try to find a closed formula for the general case.

10. Compute the LU decomposition of the following matrix  $A$ , taken from [65].

$$A = \begin{pmatrix} 6 & 2 & 1 & -1 \\ 2 & 4 & 1 & 0 \\ 1 & 1 & 4 & -1 \\ -1 & 0 & -1 & 3 \end{pmatrix}$$

11. Let  $A$  be the following  $5 \times 5$  matrix over the field  $\mathbb{Z}_2$  with 2 elements.

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

Determine a transformation matrix  $T$  over  $\mathbb{Z}_2$  such that  $T^{-1}AT$  is of the form

$$A = \begin{pmatrix} B & 0 \\ 0 & C \end{pmatrix},$$

where  $B$  is a  $2 \times 2$  matrix over  $\mathbb{Z}_2$  and  $C$  is a  $3 \times 3$  matrix over  $\mathbb{Z}_2$ .

# 19

## Linear Algebra: Applications

This chapter illustrates matrix computations with five practical examples. They are:

- Kinematics of the Stanford manipulator.
- A 3-compartment model of cadmium transfer through the human body.
- Molecular-orbital Hückel theory.
- Vector calculus.
- Moore-Penrose inverse.

In all sessions we shall assume that the **LinearAlgebra** package has been loaded.

### 19.1 Kinematics of the Stanford Manipulator

The matrices below are so-called Denavit-Hartenberg matrices used in kinematics studies of robot manipulators; actually, we shall use the matrices  $A_1, A_2, \dots, A_6$  defining the Stanford Manipulator [192]. Henceforth, we shall use in the Maple session the abbreviations  $c_1 = \cos \Theta_1, c_2 = \cos \Theta_2, \dots, s_1 = \sin \Theta_1, s_2 = \sin \Theta_2, \dots$ , and  $d_2 = d\vartheta, d_3 = d\beta$ .

```

> alias(d[2]=d2, d[3]=d3
>     seq(c[i]=cos(Theta[i]), i=1..6),
>     seq(s[i]=sin(Theta[i]), i=1..6)):
> M := (a,alpha,d,theta) -> Matrix(4, 4, [[cos(theta),
> -sin(theta)*cos(alpha), sin(theta)*sin(alpha),
> a*cos(theta)], [sin(theta), cos(theta)*cos(alpha),
> -cos(theta)*sin(alpha), a*sin(theta)],
> [0, sin(alpha), cos(alpha), d], [0, 0, 0, 1]]):
> M(a, alpha, d, theta);


$$\begin{bmatrix} \cos(\theta) & -\sin(\theta)\cos(\alpha) & \sin(\theta)\sin(\alpha) & a\cos(\theta) \\ \sin(\theta) & \cos(\theta)\cos(\alpha) & -\cos(\theta)\sin(\alpha) & a\sin(\theta) \\ 0 & \sin(\alpha) & \cos(\alpha) & d \\ 0 & 0 & 0 & 1 \end{bmatrix}$$


> # link length
> a := Vector([0$6]):
> # link twist
> alpha := Vector([-Pi/2,Pi/2,0,-Pi/2,Pi/2,0]):
> # offset distance
> d := Vector(6, {(2)=d2, (3)=d3}):
> # joint angles
> theta := Vector(6, i->Theta[i]): theta[3] := 0:
> for i to 6 do
>   A[i] := M(a[i], alpha[i], d[i], theta[i])
> end do;


$$A_1 := \begin{bmatrix} c_1 & 0 & -s_1 & 0 \\ s_1 & 0 & c_1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$


$$A_2 := \begin{bmatrix} c_2 & 0 & s_2 & 0 \\ s_2 & 0 & -c_2 & 0 \\ 0 & 1 & 0 & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$


$$A_3 := \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$


$$A_4 := \begin{bmatrix} c_4 & 0 & -s_4 & 0 \\ s_4 & 0 & c_4 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$


$$A_5 := \begin{bmatrix} c_5 & 0 & s_5 & 0 \\ s_5 & 0 & -c_5 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$


```

$$A_6 := \begin{bmatrix} c_6 & -s_6 & 0 & 0 \\ s_6 & c_6 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The position and orientation of the tip of the manipulator is determined by the Denavit-Hartenberg parameters; it is the product of the matrices  $A_1, \dots, A_6$ .

```
> Tip := `.(seq(A[i], i=1..6)):
```

Let us look at the entry in the upper left corner.

```
> collect(Tip[1,1], [c[1], c[2], s[1]]);
```

$$((c_4 c_5 c_6 - s_4 s_6) c_2 - s_2 s_5 c_6) c_1 + (-s_4 c_5 c_6 - c_4 s_6) s_1$$

This is in agreement with formula 2.55 in [192]. By hand, such matrix computations take much time and are error-prone; the computer algebra system really does its work here as symbol cruncher.

An example of how computer algebra can help to solve the inverse kinematics problem, i.e., how to compute the link parameters for a given position and orientation of the tip of a robot arm, can be found in [104].

In this section, we shall study the forward kinematics problem, and more precisely, the velocities of the tip of the Stanford manipulator. Once the physical geometry of the robot arm is fixed, i.e., once  $d_2$  and  $d_3$  are fixed, the tip of the robot arm is a function of  $\theta_1, \theta_2, \theta_4, \theta_5$ , and  $\theta_6$ . We shall determine the translational velocity  $v$  and rotational velocity  $\omega$  of the tip as a function of  $\theta'_1, \theta'_2, \theta'_4, \theta'_5$ , and  $\theta'_6$ . First, we split the translation part  $T$  and rotational part  $R$  of the matrix that describe the location and orientation of the tip of the robot arm. This is easily done with the procedures **Column**, **SubVector** and **SubMatrix**.

```
> T := Column(Tip, 4);
```

$$T := \begin{bmatrix} c_1 s_2 d_3 - s_1 d_2 \\ s_1 s_2 d_3 + c_1 d_2 \\ c_2 d_3 \\ 1 \end{bmatrix}$$

```
> T := SubVector(% , [1..3]);
```

$$T := \begin{bmatrix} c_1 s_2 d_3 - s_1 d_2 \\ s_1 s_2 d_3 + c_1 d_2 \\ c_2 d_3 \end{bmatrix}$$

```
> R := SubMatrix(Tip, [1..3], [1..3]):
```

The information about the translational and rotational velocity is contained in the Jacobian matrix

$$J = \begin{pmatrix} J_1^v & J_2^v & \dots & J_5^v \\ J_1^\omega & J_2^\omega & \dots & J_5^\omega \end{pmatrix},$$

where  $J = (J_1^v \ J_2^v \ \dots \ J_5^v)$  is the Jacobian matrix of the vector function that maps the kinematic parameters  $\theta_1, \theta_2, \theta_4, \theta_5$ , and  $\theta_6$  into the translational submatrix  $T$ . The  $J_i^\omega$ 's are  $3 \times 1$  vectors defined as

$$J_i^\omega = \begin{pmatrix} \omega_{x_i} \\ \omega_{y_i} \\ \omega_{z_i} \end{pmatrix},$$

where the components can be found with respect to the kinematic parameters from the relation

$$\frac{\partial R}{\partial \theta_i} \cdot R^T = \begin{pmatrix} 0 & -\omega_{z_i} & \omega_{y_i} \\ \omega_{z_i} & 0 & -\omega_{x_i} \\ -\omega_{y_i} & \omega_{x_i} & 0 \end{pmatrix}.$$

First, we shall consider the translational part. In the **VectorCalculus** package the procedure **Jacobian** already exists for computing the Jacobian matrix of a vector valued function.

```
> thetas := [seq(Theta[i], i=[1,2,4,5,6])];
> with(VectorCalculus, Jacobian);
> Jacobian(T, thetas);
```

```
Error, (in VectorCalculus:-Jacobian) the number of functions must
equal the number of variable names
```

Alas, no success. As the error message says, the length of the vector and the number of variables must match in the current implementation. A trick is to augment the vector with trailing zeros, then to apply the **Jacobian** procedure, and finally to take the required submatrix.

```
> Jacobian(Vector([T,0,0]), thetas);


$$\begin{bmatrix} -s_1 s_2 d_3 - c_1 d_2 & c_1 c_2 d_3 & 0 & 0 & 0 \\ c_1 s_2 d_3 - s_1 d_2 & s_1 c_2 d_3 & 0 & 0 & 0 \\ 0 & -s_2 d_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$


> Jv := %[1..3, 1..-1]; # get submatrix


$$Jv := \begin{bmatrix} -s_1 s_2 d_3 - c_1 d_2 & c_1 c_2 d_3 & 0 & 0 & 0 \\ c_1 s_2 d_3 - s_1 d_2 & s_1 c_2 d_3 & 0 & 0 & 0 \\ 0 & -s_2 d_3 & 0 & 0 & 0 \end{bmatrix}$$

```

An alternative is to create the Jacobian matrix from basic principles.

```
> Jv := Matrix(3,5, (i,j) -> diff(T[i], thetas[j]));
```

$$Jv := \begin{bmatrix} -s_1 s_2 d_3 - c_1 d_2 & c_1 c_2 d_3 & 0 & 0 & 0 \\ c_1 s_2 d_3 - s_1 d_2 & s_1 c_2 d_3 & 0 & 0 & 0 \\ 0 & -s_2 d_3 & 0 & 0 & 0 \end{bmatrix}$$

Next, we shall consider the rotational velocity of the tip. We start with computing the derivatives  $\frac{\partial R}{\partial \theta_i}$  by mapping the procedure **diff** to all matrix coefficients.

```
> R1 := map(diff, R, Theta[1]):  
> R2 := map(diff, R, Theta[2]):  
> R3 := map(diff, R, Theta[4]):  
> R4 := map(diff, R, Theta[5]):  
> R5 := map(diff, R, Theta[6]):
```

The transpose of a matrix can be computed with — to no-one's surprise — **Transpose**.

```
> Rtranspose := Transpose(R):
```

Now we have all ingredients for computing the matrices  $J_1^\omega, J_2^\omega, \dots, J_5^\omega$ . As an example, we show the results for  $J_1^\omega$  (in row notation).

```
> R1 . Rtranspose: omega := simplify(%);
```

$$\omega := \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

```
> Jomega[1] := Vector[row]([omega[3,2], -omega[3,1],  
> omega[2,1]]);
```

$$Jomega_1 := [0, 0, 1]$$

All other vectors may be computed in a loop; and finally we can concatenate them into the requested submatrix of  $J$ .

```
> for i from 2 to 5 do  
>   R||i . Rtranspose:  
>   omega := simplify(%):  
>   Jomega[i] := Vector[row]([omega[3,2], -omega[3,1],  
>     omega[2,1]]):  
>   print(evaln(Jomega[i])=Jomega[i])  
> end do:
```

$$Jomega_2 = [-s_1, c_1, 0]$$

$$Jomega_3 = [s_2 c_1, s_2 s_1, c_2]$$

$$Jomega_4 = [-c_1 c_2 s_4 - s_1 c_4, -s_1 c_2 s_4 + c_1 c_4, s_2 s_4]$$

$$Jomega_5 = [s_5 c_1 c_2 c_4 - s_5 s_1 s_4 + c_1 s_2 c_5, s_5 s_1 c_2 c_4 + s_5 c_1 s_4 + s_1 s_2 c_5,  
-s_2 c_4 s_5 + c_2 c_5]$$

```
> Jomega := Transpose(<seq(Jomega[i], i=1..5)>);
```

*Jomega* :=

$$\begin{bmatrix} 0 & -s_1 & s_2 c_1 & -c_1 c_2 s_4 - s_1 c_4 & s_5 c_1 c_2 c_4 - s_5 s_1 s_4 + c_1 s_2 c_5 \\ 0 & c_1 & s_2 s_1 & -s_1 c_2 s_4 + c_1 c_4 & s_5 s_1 c_2 c_4 + s_5 c_1 s_4 + s_1 s_2 c_5 \\ 1 & 0 & c_2 & s_2 s_4 & -s_2 c_4 s_5 + c_2 c_5 \end{bmatrix}$$

The translational and rotational part of  $J$  can be placed below each other in one matrix.

>  $J := \langle J_v, J_{omega} \rangle;$

$$J := \begin{bmatrix} -s_1 s_2 d_3 - c_1 d_2, c_1 c_2 d_3, 0, 0, 0 \\ c_1 s_2 d_3 - s_1 d_2, s_1 c_2 d_3, 0, 0, 0 \\ 0, -s_2 d_3, 0, 0, 0 \\ 0, -s_1, s_2 c_1, -c_1 c_2 s_4 - s_1 c_4, s_5 c_1 c_2 c_4 - s_5 s_1 s_4 + c_1 s_2 c_5 \\ 0, c_1, s_2 s_1, -s_1 c_2 s_4 + c_1 c_4, s_5 s_1 c_2 c_4 + s_5 c_1 s_4 + s_1 s_2 c_5 \\ 1, 0, c_2, s_2 s_4, -s_2 c_4 s_5 + c_2 c_5 \end{bmatrix}$$

Consider the translational part of the Jacobian matrix.

> ' $J_v$ '=Jv;

$$J_v = \begin{bmatrix} -s_1 s_2 d_3 - c_1 d_2 & c_1 c_2 d_3 & 0 & 0 & 0 \\ c_1 s_2 d_3 - s_1 d_2 & s_1 c_2 d_3 & 0 & 0 & 0 \\ 0 & -s_2 d_3 & 0 & 0 & 0 \end{bmatrix}$$

The maximum rank is equal to two. If for some kinematic parameters the maximum rank is not reached, then we say that the manipulator is in a singular state. Singular states are forbidden configurations of the robot arm. Let us try to find such parameter values. What happens when  $s_2 = 0$  (i.e., when  $\theta_2 \in \{0, \pi\}$ )?

>  $\text{subs}(s[2]=0, J_v);$

$$\begin{bmatrix} -c_1 d_2 & c_1 c_2 d_3 & 0 & 0 & 0 \\ -s_1 d_2 & s_1 c_2 d_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

> %[1..2, 1..2]; # SubMatrix

$$\begin{bmatrix} -c_1 d_2 & c_1 c_2 d_3 \\ -s_1 d_2 & s_1 c_2 d_3 \end{bmatrix}$$

The determinant of this submatrix can be computed with **Determinant**.

> **Determinant**(%);

$$0$$

So, the rank is less than two and the manipulator is in a singular state. When  $s_2 \neq 0$  we can use elementary row operations to inspect the row space. For example, we can add  $c_2 c_1 s_2$  times the third row to the first row with **RowOperation**.

> **RowOperation**(Jv, [1,3], c[2]\*c[1]/s[2]);

$$\begin{bmatrix} -s_1 s_2 d_3 - c_1 d_2 & 0 & 0 & 0 \\ c_1 s_2 d_3 - s_1 d_2 & 0 & 0 & 0 \\ 0 & -s_2 d_3 & 0 & 0 \end{bmatrix}$$

The rank is less than two if and only if the first column is equal to zero.

```
> {[1,1]=0, [2,1]=0};  
 { -s1 s2 d3 - c1 d2 = 0, c1 s2 d3 - s1 d2 = 0 }  
> solve(%, {s[1], c[1]});  
 {s1 = 0, c1 = 0}
```

This can only be true when both  $\cos\Theta_1$  and  $\sin\Theta_1$  are equal to zero, which is by definition impossible. So, no additional singular states of the manipulator have been detected.

## 19.2 A 3-Compartment Model of Cadmium Transfer

The next example comes from a case study in linear system theory. In [33, 126], structural identifiability of several time-invariant, continuous-time, compartmental models that describe the transfer of cadmium through the human body has been studied. Here, we shall only consider the 3-compartment model shown in Figure 19.1.

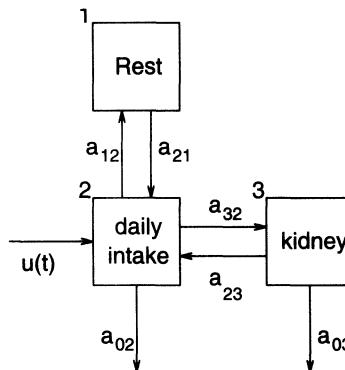


Figure 19.1. Cadmium transfer in the human body.

The corresponding mathematical model is a system of differential equations.

$$\begin{aligned}\frac{dx(t)}{dt} &= Ax(t) + Bu(t) \\ y(t) &= Cx(t)\end{aligned}$$

where

$$A = \begin{pmatrix} -a_{21} & a_{12} & 0 \\ a_{21} & -(a_{02} + a_{12} + a_{32}) & a_{23} \\ 0 & a_{32} & -(a_{03} + a_{23}) \end{pmatrix}$$

and

$$B = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \quad C = \begin{pmatrix} 0 & 0 & c_3 \\ 0 & a_{02} & a_{03} \end{pmatrix}.$$

All parameters are nonnegative. Two measurements of cadmium concentrations are done: measurements in the kidney are described by

$$y_1(t) = c_3 x_3(t)$$

and measurements in urine are described by

$$y_2(t) = a_{02} x_2(t) + a_{03} x_3(t).$$

The parameter  $c_3$  is supposed to be known and positive.

In general, a time-invariant, continuous-time, linear  $n$ -compartment model  $M(\theta)$  that depends on parameters  $\theta$  is described by

$$M(\theta) : \begin{cases} \dot{x}(t) = A(\theta) \cdot x(t) + B(\theta) \cdot u(t), & x(t_0) = x_0, \\ y(t) = C(\theta) \cdot x(t) + D(\theta) \cdot u(t), \end{cases}$$

where  $A(\theta)$  is an  $n \times n$  matrix,  $B(\theta)$  is an  $n \times m$  matrix,  $C(\theta)$  is an  $k \times n$  matrix,  $D(\theta)$  is an  $k \times m$  matrix,  $x = (x_1, x_2, \dots, x_n)^T$  is the state vector in  $\mathbb{R}_+^n$ ,  $u = (u_1, u_2, \dots, u_m)^T$  is the input vector in  $\mathbb{R}_+^m$ ,  $y = (y_1, y_2, \dots, y_k)^T$  is the output vector in  $\mathbb{R}_+^k$ , and  $\theta = (\theta_1, \dots, \theta_r)$  is the parameter vector. Here,  $\mathbb{R}_+$  denotes the set of nonnegative real numbers. The coefficients in the system matrix  $A(\theta)$  satisfy

$$A_{ij} \geq 0 \text{ if } i \neq j, \quad A_{ii} \leq 0, \quad \text{and} \quad \sum_j A_{ji} \leq 0.$$

The coefficients in the input distribution matrix  $B(\theta)$ , the output connection matrix  $C(\theta)$ , and  $D(\theta)$  are all nonnegative real numbers. The input and output are related through the so-called *external behavior* of the compartmental system that is defined as

$$y(t) = C(\theta) e^{(t-t_0)A(\theta)} x_0 + \int_{t_0}^t W_\theta(t-\tau) u(\tau) d\tau,$$

where

$$W_\theta(t) = C(\theta) e^{tA(\theta)} B(\theta) + D(\theta) \delta(t).$$

$W_\theta$  is called the *impulse response function* of the compartmental system. It is completely characterized by the so-called *Markov parameter matrices*

$M_k(\theta)$ ,  $k = 0, 1, 2, \dots$  defined as

$$\begin{aligned} M_0(\theta) &= D \\ M_k(\theta) &= \frac{d^{k-1}}{dt^{k-1}} W_\theta(t) \Big|_{x=0} \\ &= C(\theta) A(\theta)^{k-1} B(\theta), \quad \text{for } k = 1, 2, \dots \end{aligned}$$

Henceforth, we shall assume  $D(\theta) = 0$ .

Three notions in system theory are relevant in the study of theoretical (a priori) identifiability: *controllability*, *observability*, and *structural identifiability*. For a short introduction and the application of computer algebra, the interested reader is referred to [161, 198]. The following criteria hold:

**Kalman's criterion for controllability.** *The linear system is controllable iff the controllability matrix*

$$M_C = \left( B \mid AB \mid A^2B \mid \cdots \mid A^{n-rb}B \right),$$

where  $rb = \text{rank}(B)$ , has maximum rank  $n$ .

**Kalman's criterion for observability.** *The linear system is observable iff the observability matrix*

$$M_O = \begin{pmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-rc} \end{pmatrix},$$

where  $rc = \text{rank}(C)$ , has maximum rank  $n$ .

**Remark:** You may always take  $rb = rc = 1$  in the above criteria (e.g., if you do not know or cannot compute the rank of matrix  $C$ ).

Now, the main question in parameter identifiability is roughly stated as follows.

“Is it in principle possible, given the observations of input and output, to uniquely determine the parameter values?”

Recall that the set of common zeros of a set of polynomials is called an algebraic set. We shall use the following definitions of structural identifiability.

A model  $M(\theta)$  is *structurally locally identifiable at  $\theta$*  if there is an open neighborhood  $\Omega$  around  $\theta$  such that there is no  $\theta' \in \Omega$  different from  $\theta$  with the same external behavior for all identification experiments  $(t_0, x_0, u)$ .

A model  $M(\theta)$  is *structurally locally identifiable* if it is structurally locally identifiable for all  $\theta$  outside an algebraic set.

A model  $M(\theta)$  is *structurally globally identifiable* at  $\theta$  if there is no  $\theta'$  different from  $\theta$  with the same external behavior for all identification experiments  $(t_0, x_0, u)$ .

A model  $M(\theta)$  is *structurally globally identifiable* if it is structurally globally identifiable for all  $\theta$  outside an algebraic set.

In our model for cadmium transfer in the human body we have assumed that the system is stable and that the time horizon is relatively long compared to the dynamics of the system. So, the effect of the initial condition may be neglected. Thus, the input  $u(t)$  and the output  $y(t)$  are related by the convolution  $y = W_\theta * u$ . Below we mention three methods to determine structural identifiability.

**Markov parameter matrix approach to identifiability.** Use the Markov parameter matrices  $M_1(\theta), M_2(\theta), \dots, M_{2n}(\theta)$  to build up the matrix  $M_\theta$  in the following way:

$$M_\theta = \begin{pmatrix} C(\theta) B(\theta) \\ C(\theta) A(\theta) B(\theta) \\ C(\theta) A(\theta)^2 B(\theta) \\ \vdots \\ C(\theta) A(\theta)^{2n-1} B(\theta) \end{pmatrix}.$$

The linear system is structurally globally identifiable iff for all  $\theta$  outside an algebraic set  $M_{\theta'} = M_\theta$  implies  $\theta' = \theta$ . The linear system is structurally locally identifiable iff the  $2nkm \times r$  Jacobian matrix

$$\begin{pmatrix} \frac{\partial M_1}{\partial \theta_1} & \dots & \frac{\partial M_1}{\partial \theta_r} \\ \vdots & & \vdots \\ \frac{\partial M_{2n}}{\partial \theta_1} & \dots & \frac{\partial M_{2n}}{\partial \theta_r} \end{pmatrix}$$

has maximum rank for all  $\theta$  outside an algebraic set.

**Transfer function approach.** Define the transfer function matrix  $H_\theta(s)$  as

$$H_\theta(s) = C(\theta) [sI - A(\theta)]^{-1} B(\theta).$$

The linear system is structurally globally identifiable iff, for all  $\theta$  outside an algebraic set,  $H_{\theta'}(s) = H_\theta(s)$  implies  $\theta' = \theta$ . The linear compartmental system is structurally locally identifiable iff the  $(2n - 1)km \times r$  Jacobian matrix of the mapping of  $\theta$  into the coefficients of the numerator and denominator of matrix elements of  $H(s, \theta)$  has maximum rank  $r$  for all  $\theta$  outside an algebraic set.

**WARNING:** The very popular *similarity transformation approach* only holds for general time-invariant, continuous-time, finite-dimensional, linear systems, where state, input, and output vectors are from  $\mathbb{R}$ -spaces

and  $A$ ,  $B$ ,  $C$ , and  $D$  are matrices with coefficients in  $\mathbb{R}$ . The similarity approach is of limited use in compartmental systems, where state, input, and output vectors are positive vectors. The following criterion is the best you can get in this approach.

**Similarity-transformation method.** *Let the compartmental system be structurally controllable and structurally observable for all parameters outside an algebraic set, and let the system have less than four compartments. Under these conditions the compartmental system is structurally globally identifiable at  $\theta$  iff for all  $\theta'$  and for all nonsingular  $T$  the system of equations*

$$\begin{aligned} T A(\theta') &= A(\theta) T \\ T B(\theta') &= B(\theta) \\ C(\theta') &= C(\theta) T \end{aligned}$$

*has a unique solution  $(I, \theta)$ . If the system of equations has a finite set of solutions  $(T, \theta)$ , then the model is structurally locally identifiable at  $\theta$ .*

For systems with four or more compartments the above criterion is sufficient but not necessary. This limits its usefulness enormously.

In all the above criteria, computer algebra is the appropriate computational tool. We shall apply the criteria to our example of cadmium transfer.

First, we introduce with the **macro** facility some shortcuts for various indexed names  $a_{ij}$ ,  $b_{ij}$  and  $t_{ij}$ . This is done in a somewhat tricky way because of the evaluation rules of arguments of **macro**.

```
> sequence :=  
>   seq(seq( a||i||j = a[i,j], i=0..3 ), j=0..3 ),  
>   seq(seq( b||i||j = b[i,j], i=0..3 ), j=0..3 ),  
>   seq(seq( t||i||j = t[i,j], i=0..3 ), j=0..3 ),  
>   c3 = c[3]:  
>   eval('macro(S)', S=sequence):
```

Next, we introduce matrices that describe the compartmental system. We use the bracket notation.

```
> A := <<-a21 | a12 | 0>, <a21 | -(a02+a12+a32) | a23>,  
> <0 | a32 | -(a03+a23)>>;
```

$$A := \begin{bmatrix} -a_{2,1} & a_{1,2} & 0 \\ a_{2,1} & -a_{0,2} - a_{1,2} - a_{3,2} & a_{2,3} \\ 0 & a_{3,2} & -a_{0,3} - a_{2,3} \end{bmatrix}$$

```
> B := <<0,1,0>>;
```

$$B := \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

```
> C := <<0 | 0 | c3> , <0 | a02 | a03>>;
```

$$C := \begin{bmatrix} 0 & 0 & c_3 \\ 0 & a_{0,2} & a_{0,3} \end{bmatrix}$$

```
> AB := A.B; A2B := A.AB;
```

```
> MC := <B | AB | A2B>;
```

$$MC := \begin{bmatrix} 0 & a_{1,2} & -a_{2,1}a_{1,2} + a_{1,2}\%1 \\ 1 & \%1 & a_{2,1}a_{1,2} + \%1^2 + a_{2,3}a_{3,2} \\ 0 & a_{3,2} & a_{3,2}\%1 + (-a_{0,3} - a_{2,3})a_{3,2} \end{bmatrix}$$

$$\%1 := -a_{0,2} - a_{1,2} - a_{3,2}$$

It is tempting to compute the rank of the controllability matrix  $M_C$  with the Maple procedure **Rank**.

```
> Rank(MC);
```

3

But this gives only the generic rank for the matrix considered as a matrix with coefficients in the quotient field  $\mathbb{Q}(a_{12}, a_{21}, \dots, a_{32})$ . This is not what we wanted! The row reduced echelon form of the matrix with coefficients in this quotient field equal to the identity matrix.

```
> ReducedRowEchelonForm(MC);
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

We must be more careful and first do a fraction free Gaussian elimination, carried out by the procedure **GaussianElimination** or, if you want to go the safe way, by the **LUDecomposition** procedure with the **FractionFree** method explicitly mentioned. Notice the subtle difference between the following elimination results. As a matter of fact, the procedure **GaussianElimination** is equivalent to calling **LUDecomposition** with the options **method=GaussianElimination** and **output='U'**.

```
> GaussianElimination(MC);
```

$$\begin{aligned} & [1, -a_{0,2} - a_{1,2} - a_{3,2}, a_{2,1}a_{1,2} + a_{0,2}^2 + 2a_{0,2}a_{1,2} + 2a_{0,2}a_{3,2} \\ & + a_{1,2}^2 + 2a_{1,2}a_{3,2} + a_{3,2}^2 + a_{2,3}a_{3,2}] \\ & [0, a_{1,2}, -a_{2,1}a_{1,2} - a_{0,2}a_{1,2} - a_{1,2}^2 - a_{1,2}a_{3,2}] \\ & [0, 0, a_{3,2}(-a_{0,3} - a_{2,3} + a_{2,1})] \end{aligned}$$

```
> LUDecomposition(MC, method=FractionFree);
```

```


$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & \frac{a_{3,2}}{a_{1,2}} & \frac{1}{a_{1,2}} \end{bmatrix},$$


$$\left[ 1, -a_{0,2} - a_{1,2} - a_{3,2}, a_{2,1} a_{1,2} + a_{0,2}^2 + 2 a_{0,2} a_{1,2} + 2 a_{0,2} a_{3,2} \right.$$


$$\left. + a_{1,2}^2 + 2 a_{1,2} a_{3,2} + a_{3,2}^2 + a_{2,3} a_{3,2} \right]$$


$$\left[ 0, a_{1,2}, -a_{2,1} a_{1,2} - a_{0,2} a_{1,2} - a_{1,2}^2 - a_{1,2} a_{3,2} \right]$$


$$\left[ 0, 0, -a_{1,2} a_{3,2} a_{0,3} - a_{1,2} a_{2,3} a_{3,2} + a_{3,2} a_{2,1} a_{1,2} \right]$$

> map(factor, %[3]);

$$\left[ 1, -a_{0,2} - a_{1,2} - a_{3,2}, a_{2,1} a_{1,2} + a_{0,2}^2 + 2 a_{0,2} a_{1,2} + 2 a_{0,2} a_{3,2} \right.$$


$$\left. + a_{1,2}^2 + 2 a_{1,2} a_{3,2} + a_{3,2}^2 + a_{2,3} a_{3,2} \right]$$


$$\left[ 0, a_{1,2}, -a_{1,2} (a_{2,1} + a_{0,2} + a_{1,2} + a_{3,2}) \right]$$


$$\left[ 0, 0, a_{1,2} a_{3,2} (-a_{0,3} - a_{2,3} + a_{2,1}) \right]$$


```

It follows immediately that the system is controllable iff  $a_{12} \neq 0$ ,  $a_{32} \neq 0$ , and  $a_{21} \neq a_{03} + a_{23}$ . However, recall that Maple considers the matrix coefficients as elements in  $\mathbb{Q}(a_{12}, a_{21}, \dots, a_{32})$  so that, at each step of the fraction free Gaussian elimination, it is possible that in each row the previous pivot is divided out or that the gcd of the polynomials in the row are divided out. The latter operation is not implemented in the procedure, and the former operation of dividing out previous pivots is of no harm as these pivots must be nonzero anyway in order to reach the maximum rank. If you don't want to rely on these implementation issues, then you can use the following equivalent controllability criterion.

**Criterion for controllability.** *The linear system is controllable iff for the controllability matrix  $M_C$  holds  $\det(M_C M_C^T) \neq 0$  (or  $\det(M_C) \neq 0$  when  $M_C$  is a square matrix).*

```

> factor(Determinant(MC));

$$-a_{1,2} a_{3,2} (-a_{0,3} - a_{2,3} + a_{2,1})$$


```

But, in general this criterion will give trickier conditions.

Observability can be checked in the same way. We shall assume  $a_{02} \neq 0$  so that the matrix  $C$  has rank 2.

```

> CA := C.A;
> MO := <C, CA>;

```

```


$$MO :=$$


$$[0, 0, c_3]$$


$$[0, a_{0,2}, a_{0,3}]$$


$$[0, c_3 a_{3,2}, c_3 (-a_{0,3} - a_{2,3})]$$


$$\left[ a_{0,2} a_{2,1}, a_{0,2} (-a_{0,2} - a_{1,2} - a_{3,2}) + a_{3,2} a_{0,3}, \right.$$


$$\left. a_{0,2} a_{2,3} + a_{0,3} (-a_{0,3} - a_{2,3}) \right]$$

> map(factor, LUdecomposition(MO, method=FractionFree,
> output='U'));

```

$$\left[ a_{0,2} a_{2,1}, -a_{0,2}^2 - a_{0,2} a_{1,2} - a_{0,2} a_{3,2} + a_{3,2} a_{0,3}, \right.$$

$$\left. a_{0,2} a_{2,3} - a_{0,3}^2 - a_{0,3} a_{2,3} \right]$$

$$[0, a_{0,2}^2 a_{2,1}, a_{0,3} a_{0,2} a_{2,1}]$$

$$[0, 0, -a_{2,1} a_{0,2} c_3 (a_{0,2} a_{0,3} + a_{0,2} a_{2,3} + a_{3,2} a_{0,3})]$$

$$[0, 0, 0]$$

So, the system is observable if  $a_{02} \neq 0$  and  $a_{21} \neq 0$ .

In our example, the next criterion for observability leads to more difficult, but equivalent conditions.

**Criterion for observability.** *The linear system is observable iff for the observability matrix  $M_O$  holds  $\det(M_O^T M_O) \neq 0$  (or  $\det(M_O) \neq 0$  when  $M_O$  is a square matrix).*

```

> factor(Determinant(Transpose(MO).MO));

```

$$a_{2,1}^2 a_{0,2}^2 c_3^2 (2 a_{0,2} a_{3,2} a_{0,3}^2 + 2 a_{0,2}^2 a_{0,3} a_{2,3} + a_{3,2}^2 a_{0,3}^2 + a_{0,2}^2 a_{2,3}^2 + c_3^2 a_{3,2}^2 + a_{0,2}^2 a_{0,3}^2 + a_{0,2}^2 + 2 a_{0,2} a_{3,2} a_{0,3} a_{2,3})$$

It looks as if an extra condition follows from the fourth factor. But when you consider it as a second degree polynomial in  $a_{32}$  and compute the discriminant, then you come to the conclusion that this term is always negative.

```

> collect(op(4,%), a32);

```

$$(c_3^2 + a_{0,3}^2) a_{3,2}^2 + (2 a_{0,2} a_{0,3}^2 + 2 a_{0,2} a_{0,3} a_{2,3}) a_{3,2} + 2 a_{0,2}^2 a_{0,3} a_{2,3} + a_{0,2}^2 + a_{0,2}^2 a_{2,3}^2 + a_{0,2}^2 a_{0,3}^2$$

$$> discrim(% , a32);$$

$$-4 (a_{0,3}^2 + c_3^2 a_{0,3}^2 + 2 c_3^2 a_{0,3} a_{2,3} + c_3^2 a_{2,3}^2 + c_3^2) a_{0,2}^2$$

$$> collect(% , c3, factor);$$

$$-4 a_{0,2}^2 (2 a_{0,3} a_{2,3} + a_{2,3}^2 + a_{0,3}^2 + 1) c_3^2 - 4 a_{0,2}^2 a_{0,3}^2$$

```

> applyrule(expand((a[2,3]+a[0,3])^2) =
>      (a[2,3]+a[0,3])^2, %);
      -4 a0,2^2 ((a0,3 + a2,3)^2 + 1) c3^2 - 4 a0,2^2 a0,3^2

```

Next, we shall study structural local identifiability via the Markov parameter matrix. First, we must introduce the matrix and its Jacobian.

```

> n := RowDimension(A);
      n := 3
> for i from 0 to 2*n-1 do ca||i||b := C.A^i.B end do:
> J := <seq(ca||i||b, i=0..2*n-1)>;
> J12 := map(diff, J, a12); J21 := map(diff, J, a21);
> J23 := map(diff, J, a23); J32 := map(diff, J, a32);
> J02 := map(diff, J, a02); J03 := map(diff, J, a03);
> JM := <J21 | J12 | J02 | J23 | J32 | J03>;
> LUDecomposition(JM, method=FractionFree, output='U'):
> SubMatrix(%,[6..6],[1..6]):
> map(factor, %);
> SubMatrix(%,[6..6],[1..6]);

```

$$\begin{bmatrix} 0, 0, 0, 0, 0, \\ c_3^2 a_{0,2}^2 a_{1,2} a_{3,2}^2 (-a_{0,3} - a_{2,3} + a_{2,1}) (a_{2,1} - a_{2,3} - a_{0,3} + a_{1,2}) \end{bmatrix}$$

From the sixth row of the matrix it follows easily that the system is structurally locally identifiable if and only if  $a_{12}$ ,  $a_{02}$ , and  $a_{32}$  are nonzero,  $a_{21} \neq a_{03} + a_{23}$ , and  $a_{12} + a_{21} \neq a_{03} + a_{23}$ . We shall also study structural global identifiability via the transfer function method.

```

> H := C . (ScalarMatrix(s,n,n)-A)^(-1) . B:
> simplify(denom(H[1,1])*H) / collect(denom(H[1,1]), s);

      \left[ \frac{c_3 (a_{2,1} + s) a_{3,2}}{(a_{2,1} + s) (a_{0,2} a_{0,3} + a_{0,2} a_{2,3} + a_{0,2} s + a_{3,2} a_{0,3})} \right] /
(s^3 + (a_{0,3} + a_{3,2} + a_{1,2} + a_{0,2} + a_{2,1} + a_{2,3}) s^2 + (a_{0,2} a_{2,1} + a_{0,3} a_{1,2}
+ a_{0,2} a_{0,3} + a_{3,2} a_{0,3} + a_{2,1} a_{0,3} + a_{0,2} a_{2,3} + a_{2,1} a_{2,3} + a_{3,2} a_{2,1}
+ a_{1,2} a_{2,3}) s + a_{2,1} a_{3,2} a_{0,3} + a_{2,1} a_{0,2} a_{0,3} + a_{2,1} a_{0,2} a_{2,3}) /
> collect(numer(H[1,1]), s);

      c_3 a_{3,2} s + c_3 a_{2,1} a_{3,2}
> collect(numer(H[2,1]), s);

      a_{0,2} s^2 + (a_{0,2} a_{2,1} + a_{0,2} a_{0,3} + a_{0,2} a_{2,3} + a_{3,2} a_{0,3}) s
      + a_{2,1} (a_{0,2} a_{0,3} + a_{0,2} a_{2,3} + a_{3,2} a_{0,3})

```

According to the transfer function criterion, structural global identifiability is equivalent to the uniqueness of the solution of the system of equations that is generated as follows.

```

> cfs1 := {coeffs(expand(numer(H[1,1])), s)};
cfs1 := {c3 a3,2, c3 a2,1 a3,2}
> cfs2 := {coeffs(expand(numer(H[2,1])), s)};
cfs2 := {a0,2, a0,2 a2,1 + a0,2 a0,3 + a0,2 a2,3 + a3,2 a0,3,
          a2,1 a3,2 a0,3 + a2,1 a0,2 a0,3 + a2,1 a0,2 a2,3}
> cfs3 := {coeffs(expand(denom(H[1,1])), s)} minus {1};
cfs3 := {a2,1 a3,2 a0,3 + a2,1 a0,2 a0,3 + a2,1 a0,2 a2,3,
          a0,3 + a3,2 + a1,2 + a0,2 + a2,1 + a2,3, a0,2 a2,1 + a0,3 a1,2 + a0,2 a0,3
          + a3,2 a0,3 + a2,1 a0,3 + a0,2 a2,3 + a2,1 a2,3 + a3,2 a2,1 + a1,2 a2,3}
> cfs := 'union'(cfs||(1..3));
> eqns := map(x -> x = subs(a12=b12, a21=b21, a23=b23,
> a32=b32, a23=b23, a03=b03, a02=b02, x), cfs);

eqns := {a0,2 = b0,2, a0,2 a2,1 + a0,2 a0,3 + a0,2 a2,3 + a3,2 a0,3 =
b0,2 b2,1 + b0,2 b0,3 + b0,2 b2,3 + b3,2 b0,3, c3 a3,2 = c3 b3,2, ...
.....
b0,3 + b0,2 b2,3
..., c3 a2,1 a3,2 = c3 b2,1 b3,2}

```

We omitted some equations in the output.

```

> vars := {b12,b21,b23,b32,b02,b03}:
> solve(eqns, vars);

```

$$\begin{aligned} & \{b_{0,3} = a_{0,3}, b_{2,3} = a_{2,3}, b_{1,2} = a_{1,2}, b_{3,2} = a_{3,2}, b_{0,2} = a_{0,2}, b_{2,1} = a_{2,1}\}, \\ & \left\{ b_{0,3} = -\frac{-a_{0,2}a_{0,3} - a_{3,2}a_{0,3} - a_{0,2}a_{2,3} + a_{0,2}a_{2,1} + a_{0,2}a_{1,2}}{a_{3,2}}, b_{2,3} = \right. \\ & \left. -\frac{-a_{0,2}a_{0,3} - a_{3,2}a_{0,3} - a_{0,2}a_{2,3} + a_{0,2}a_{2,1} + a_{0,2}a_{1,2} + a_{3,2}a_{2,1} + a_{1,2}a_{3,2}}{a_{3,2}} \right. \\ & , b_{1,2} = a_{0,3} + a_{2,3} - a_{2,1}, b_{3,2} = a_{3,2}, b_{0,2} = a_{0,2}, b_{2,1} = a_{2,1} \} \end{aligned}$$

There are two solutions. Thus, the system is not structurally globally identifiable; it is only structurally locally identifiable.

The last result will be checked by the similarity-transformation method. First we introduce the matrices  $AA$ ,  $BB$ , and  $CC$  corresponding to the same model  $M$  but with parameter values  $b_{12}, b_{21}, \dots$  instead of  $a_{12}, a_{21}, \dots$

```

> AA := subs(a12=b12, a21=b21, a23=b23, a32=b32,
> a23=b23, a03=b03, a02=b02, A);

```

$$AA := \begin{bmatrix} -b_{2,1} & b_{1,2} & 0 \\ b_{2,1} & -b_{0,2} - b_{1,2} - b_{3,2} & b_{2,3} \\ 0 & b_{3,2} & -b_{0,3} - b_{2,3} \end{bmatrix}$$

```

> BB := copy(B);
> CC := subs(a02=b02, a03=b03, C);


$$CC := \begin{bmatrix} 0 & 0 & c_3 \\ 0 & b_{0,2} & b_{0,3} \end{bmatrix}$$

> T := Matrix(3, 3, (i,j)->t[i,j]);

$$T := \begin{bmatrix} t_{1,1} & t_{1,2} & t_{1,3} \\ t_{2,1} & t_{2,2} & t_{2,3} \\ t_{3,1} & t_{3,2} & t_{3,3} \end{bmatrix}$$


```

From the matrix operation

```

> T.BB - B;

$$\begin{bmatrix} t_{1,2} \\ t_{2,2} - 1 \\ t_{3,2} \end{bmatrix}$$


```

it follows that  $t_{12} = t_{32} = 0$  and  $t_{22} = 1$ . Then the matrix equation  $CC - C \cdot T = 0$  looks as follows.

```

> t12 := 0: t32 := 0: t22 := 1:
> map(factor, CC - C.T);


$$\begin{bmatrix} -c_3 t_{3,1} & 0 & -c_3 (-1 + t_{3,3}) \\ -a_{0,2} t_{2,1} - a_{0,3} t_{3,1} & b_{0,2} - a_{0,2} & b_{0,3} - a_{0,2} t_{2,3} - a_{0,3} t_{3,3} \end{bmatrix}$$


```

Because  $c_3 \neq 0$ , it follows immediately that  $t_{31} = 0$ ,  $t_{33} = 1$ , and  $b_{02} = a_{02}$ .

```

> t31 := 0: t33 := 1: b02 := a02: CC - C.T;

$$\begin{bmatrix} 0 & 0 & 0 \\ -a_{0,2} t_{2,1} & b_{0,2} - a_{0,2} & b_{0,3} - a_{0,2} t_{2,3} - a_{0,3} \end{bmatrix}$$


```

If  $a_{02} \neq 0$  (as is the case when the system is observable), then  $t_{21} = 0$ , and we have one equation left.

```

> t21 := 0: eqn1 := (CC - C.T)[2,3];
eqn1 := b_{0,3} - a_{0,2} t_{2,3} - a_{0,3}

```

Now we derive the non-trivial equations obtained from the similarity transformation of the  $A$  matrix.

```

> T.AA - A.T:
> eqn2 := map(op, convert(%, 'listlist')):
> eqns := {eqn1, op(eqn2)} minus {0};

```

```

eqns := {b0,3 - a0,2 t2,3 - a0,3, -t1,1 b2,1 + a2,1 t1,1, b2,1 - a2,1 t1,1,
b3,2 - a3,2, t1,1 b1,2 + t1,3 b3,2 - a1,2,
t1,3 (-b0,3 - b2,3) + a2,1 t1,3 - a1,2 t2,3,
-b1,2 - b3,2 + t2,3 b3,2 + a1,2 + a3,2,
b2,3 + t2,3 (-b0,3 - b2,3) - a2,1 t1,3 - (-a0,2 - a1,2 - a3,2) t2,3 - a2,3,
-b0,3 - b2,3 - a3,2 t2,3 + a0,3 + a2,3}
> solve(eqns, {b12,b32,b23,b03,b21,t11,t13,t23});
{b0,3 = a0,3, t1,3 = 0, t2,3 = 0, b2,3 = a2,3, b1,2 = a1,2, b3,2 = a3,2,
b2,1 = a2,1, t1,1 = 1}, {b0,3 =

$$-\frac{-a_{0,2}a_{0,3} - a_{3,2}a_{0,3} - a_{0,2}a_{2,3} + a_{0,2}a_{2,1} + a_{0,2}a_{1,2}}{a_{3,2}}, t_{1,3} =$$


$$\frac{-a_{0,3} - a_{2,3} + a_{2,1} + a_{1,2}}{a_{3,2}}, t_{2,3} = -\frac{-a_{0,3} - a_{2,3} + a_{2,1} + a_{1,2}}{a_{3,2}}, b_{2,3} =$$


$$\frac{-a_{0,2}a_{0,3} - a_{3,2}a_{0,3} - a_{0,2}a_{2,3} + a_{0,2}a_{2,1} + a_{0,2}a_{1,2} + a_{3,2}a_{2,1} + a_{1,2}a_{3,2}}{a_{3,2}},$$

b_{1,2} = a_{0,3} + a_{2,3} - a_{2,1}, b_{3,2} = a_{3,2}, b_{2,1} = a_{2,1}, t_{1,1} = 1}

```

There are two distinct solutions, which proves again that the system is not structurally globally identifiable; it is only locally identifiable.

For more complicated systems, the analysis of controllability, observability, and structural identifiability via the outlined methods will not be as easy as above. Appropriate polynomial equation solvers like elimination, characteristic sets, or Gröbner basis methods must be used to find the answers. But at least the computational work is done by the computer algebra system with its linear algebra package, equation solvers, and so on.

### 19.3 Molecular-Orbital Hückel Theory

The next example, molecular-orbital Hückel theory for the computation of  $\pi$ -electron energies and electronic charge distributions in molecules, comes from quantum chemistry. This method can be found in any book on quantum chemistry, e.g., in [32]; here we only sketch the method.

The particular molecule we shall use is azulene  $C_{10}H_{10}$ , a skeleton of ten carbon atoms linked by so-called  $\sigma$ -bonds as shown in Figure 19.2.

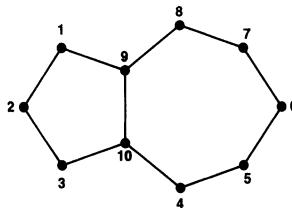


Figure 19.2. Azulene skeleton.

Each  $\sigma$ -bond contains 2 electrons; what remains are 10 electrons, which are called  $\pi$ -electrons. These  $\pi$ -electrons form  $\pi$ -bonds, which are constructed from the  $2p_z$  carbon orbitals. We number each carbon atom and denote its  $2p_z$  orbital as  $\phi_i$ . The molecular orbitals are constructed from these atomic orbitals as linear combinations  $\psi = c_1\phi_1 + c_2\phi_2 + \dots + c_{10}\phi_{10}$ . The coefficients  $c_1, c_2, \dots, c_{10}$  are determined by a variational method such that the energy is as low as possible. This leads to a generalized eigenvalue problem:  $(H - ES)C = 0$ , where  $H$  is the Hamiltonian matrix with matrix elements  $\langle \phi_i | H | \phi_j \rangle$ ,  $S$  is the overlap matrix with coefficients  $\langle \phi_i | \phi_j \rangle$ , and  $C$  is a generalized eigenvector with generalized eigenvalue  $E$ . The following assumptions are made in Hückel theory.

- All matrix coefficients  $H_{ii}$  are equal, and are given the symbol  $\alpha$ .
- All matrix coefficients  $H_{ij}$  are equal when atoms  $i$  and  $j$  are neighbors, and are denoted as  $\beta$ . When atoms  $i$  and  $j$  are not directly bonded, then  $H_{ij} = 0$ .
- Overlap is neglected, i.e.,  $S$  is the identity matrix.

Under these assumptions we get an ordinary eigenvalue problem. The symmetric matrix  $H$ , which is called the *Hückel matrix*, for azulene is as follows.

$$\begin{pmatrix} \alpha & \beta & 0 & 0 & 0 & 0 & 0 & 0 & \beta & 0 \\ \beta & \alpha & \beta & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \beta & \alpha & 0 & 0 & 0 & 0 & 0 & 0 & \beta \\ 0 & 0 & 0 & \alpha & \beta & 0 & 0 & 0 & 0 & \beta \\ 0 & 0 & 0 & \beta & \alpha & \beta & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \beta & \alpha & \beta & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \beta & \alpha & \beta & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \beta & \alpha & \beta & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \beta & \alpha & \beta \\ \beta & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \beta & \alpha \end{pmatrix}$$

Instead of the eigenvalue problem for the Hückel matrix, we shall solve the eigenvalue problem for the corresponding topological matrix.

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

The relation between an eigenvalue  $x$  of the topological matrix and the eigenvalue  $E$  is simple, viz..  $E = \alpha + \beta x$ ; the corresponding eigenvectors are the same.

Let us start computing. First, we tabulate the directly bonded carbon atom pairs.

```
> 'number of C-atoms' := 10: macro(n='number of C-atoms')
> Azulene_skeleton := 
>   [[1,2], [1,9], [2,3], [3,10], [4,5], [4,10],
>    [4,10], [5,6], [6,7], [7,8], [8,9], [9,10]]:
> Hueckel_matrix := Matrix(n,n):
> for i from 1 to nops(Azulene_skeleton) do
>   Hueckel_matrix[op(Azulene_skeleton[i])] := beta
> end do:
> Hueckel_matrix := ScalarMatrix(alpha,n,n)
>   + Hueckel_matrix + Transpose(Hueckel_matrix):
> topological_matrix := subs(alpha=0, beta=1,
>   Hueckel_matrix):
```

The characteristic matrix and the characteristic polynomial of the matrix can be computed with the procedures **CharacteristicMatrix** and **CharacteristicPolynomial**, respectively.

```
> factor(CharacteristicPolynomial(topological_matrix, x));

$$(x^4 + x^3 - 3x^2 - x + 1)(x^6 - x^5 - 7x^4 + 5x^3 + 13x^2 - 6x - 4)$$

> CharacteristicMatrix(topological_matrix, x);
```

$$\begin{bmatrix} -x & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & -x & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -x & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -x & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & -x & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -x & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -x & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -x & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -x & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & -x \end{bmatrix}$$

```
> factor(Determinant(%));  

(x^4 + x^3 - 3 x^2 - x + 1) (x^6 - x^5 - 7 x^4 + 5 x^3 + 13 x^2 - 6 x - 4)
```

Eigenvalues and eigenvectors can be computed with **Eigenvalues** and **Eigenvectors**, respectively. The **implicit** option can be used if **RootOf** expressions are all you need.

```
> Eigenvalues(topological_matrix);
```

$$\left[ \begin{array}{l} -\frac{1}{4} + \frac{\sqrt{5}}{4} + \frac{\sqrt{22 - 2\sqrt{5}}}{4} \\ -\frac{1}{4} + \frac{\sqrt{5}}{4} - \frac{\sqrt{22 - 2\sqrt{5}}}{4} \\ -\frac{1}{4} - \frac{\sqrt{5}}{4} + \frac{\sqrt{22 + 2\sqrt{5}}}{4} \\ -\frac{1}{4} - \frac{\sqrt{5}}{4} - \frac{\sqrt{22 + 2\sqrt{5}}}{4} \\ \text{RootOf}(\%1, \text{index} = 1) \\ \text{RootOf}(\%1, \text{index} = 2) \\ \text{RootOf}(\%1, \text{index} = 3) \\ \text{RootOf}(\%1, \text{index} = 4) \\ \text{RootOf}(\%1, \text{index} = 5) \\ \text{RootOf}(\%1, \text{index} = 6) \end{array} \right]$$

$$\%1 := -Z^6 - Z^5 - 7 Z^4 + 5 Z^3 + 13 Z^2 - 6 Z - 4$$

```
> Eigenvalues(topological_matrix, 'implicit');
```

$$\left[ \begin{array}{l} \text{RootOf}(\%1, \text{index} = 1) \\ \text{RootOf}(\%1, \text{index} = 2) \\ \text{RootOf}(\%1, \text{index} = 3) \\ \text{RootOf}(\%1, \text{index} = 4) \\ \text{RootOf}(\%2, \text{index} = 1) \\ \text{RootOf}(\%2, \text{index} = 2) \\ \text{RootOf}(\%2, \text{index} = 3) \\ \text{RootOf}(\%2, \text{index} = 4) \\ \text{RootOf}(\%2, \text{index} = 5) \\ \text{RootOf}(\%2, \text{index} = 6) \end{array} \right]$$

$$\%1 := -Z^4 + Z^3 - 3 Z^2 - Z + 1$$

$$\%2 := -Z^6 - Z^5 - 7 Z^4 + 5 Z^3 + 13 Z^2 - 6 Z - 4$$

Multiple assignment is convenient in solving eigenvalue problems.

```
> (v,e) := Eigenvectors(topological_matrix, 'implicit');
```

$$v, e := \begin{bmatrix} \%1 \\ \%2 \\ \vdots \end{bmatrix}, \begin{bmatrix} \vdots \\ \%1 \\ \%2 \\ \vdots \end{bmatrix}$$

$$\%1 := \text{RootOf}(-Z^4 + -Z^3 - 3\cdot Z^2 - -Z + 1)$$

$$\%2 := \text{RootOf}(-Z^6 - -Z^5 - 7\cdot Z^4 + 5\cdot Z^3 + 13\cdot Z^2 - 6\cdot Z - 4)$$

We have omitted the matrix with eigenvectors as columns. Let us consider the first element in the above vector of eigenvalues and the corresponding first column of the matrix built up from eigenvectors.

```
> v[1]; # eigenvalue
```

$$\text{RootOf}(-Z^4 + -Z^3 - 3\cdot Z^2 - -Z + 1)$$

```
> e[1..-1,1]; # eigenvector
```

$$\begin{bmatrix} -3 + 2\%1^2 - 2\%1 + \%1^3 \\ 0 \\ -\%1^3 - 2\%1^2 + 3 + 2\%1 \\ -1 \\ -3\%1 + \%1^2 + \%1^3 - 1 \\ 0 \\ 3\%1 - \%1^2 - \%1^3 + 1 \\ 1 \\ \%1^2 - 2\%1 + \%1^3 - 1 \\ -\%1^2 + 2\%1 - \%1^3 + 1 \end{bmatrix}$$

$$\%1 := \text{RootOf}(-Z^4 + -Z^3 - 3\cdot Z^2 - -Z + 1)$$

You should actually consider the above object as a description of four eigenvectors:  $\%1$  can take four values, but you must always choose the same value in one vector. Below is the one that corresponds with the eigenvalue  $-\frac{1}{4} + \frac{1}{4}\sqrt{5} + \frac{1}{4}\sqrt{22 - 2\sqrt{5}}$ .

```
> allvalues(%, 'dependent')[1];
```

$$\begin{bmatrix} -\frac{5}{2} + 2\%1^2 - \frac{\sqrt{5}}{2} - \frac{\sqrt{22 - 2\sqrt{5}}}{2} + \%1^3 \\ 0 \\ -\%1^3 - 2\%1^2 + \frac{5}{2} + \frac{\sqrt{5}}{2} + \frac{\sqrt{22 - 2\sqrt{5}}}{2} \\ -1 \\ -\frac{1}{4} - \frac{3\sqrt{5}}{4} - \frac{3\sqrt{22 - 2\sqrt{5}}}{4} + \%1^2 + \%1^3 \end{bmatrix}$$

$$\begin{bmatrix}
 0 \\
 \frac{1}{4} + \frac{3\sqrt{5}}{4} + \frac{3\sqrt{22-2\sqrt{5}}}{4} - \%1^2 - \%1^3 \\
 1 \\
 \%1^2 - \frac{1}{2} - \frac{\sqrt{5}}{2} - \frac{\sqrt{22-2\sqrt{5}}}{2} + \%1^3 \\
 - \%1^2 + \frac{1}{2} + \frac{\sqrt{5}}{2} + \frac{\sqrt{22-2\sqrt{5}}}{2} - \%1^3 \\
 \%1 := -\frac{1}{4} + \frac{\sqrt{5}}{4} + \frac{\sqrt{22-2\sqrt{5}}}{4}
 \end{bmatrix}$$

```

> allvalues(v[1], 'dependent')[1];

```

$$-\frac{1}{4} + \frac{\sqrt{5}}{4} + \frac{\sqrt{22-2\sqrt{5}}}{4}$$

Singular values of a matrix  $M$ , which are defined as the square roots of the eigenvalues of  $MM^T$ , can be computed by **SingularValues**. Numerical eigenvalues and eigenvectors can also be obtained by **Eigenvalues**. Below, we use the option **output=list** to specify that the eigenvalues must be placed in a list. This option can also be used in the **Eigenvectors** procedure. Then it will return a list of lists. The first component of each sublist is the eigenvalue; the second component is the multiplicity of the eigenvalue; the third component is a set of eigenvectors that form a basis of the eigenspace. Other output options are available, as can be found in the on-line help system.

```

> interface(displayprecision=5):
> eigenvalues :=
>   Eigenvalues(evalf(topological_matrix), output=list);

eigenvalues := [2.31028 + 0.00000 I, -2.09529 + 0.00000 I,
-1.86921 + 0.00000 I, -1.57922 + 0.00000 I, 1.65157 + 0.00000 I,
1.35567 + 0.00000 I, 0.88698 + 0.00000 I, 0.47726 + 0.00000 I,
-0.73764 + 0.00000 I, -0.40039 + 0.00000 I]

> eigenvalues := sort(map(Re, eigenvalues));

eigenvalues := [-2.09529, -1.86921, -1.57922, -0.73764, -0.40039,
0.47726, 0.88698, 1.35567, 1.65157, 2.31028]

```

Let us check the lowest eigenvector.

```

> (v,e) := Eigenvectors(evalf(topological_matrix)):
> (v,e) := map(Re,v), map(Re,e):
> vlist := convert(v, list);

```

```

vlist := [2.31028, -2.09529, -1.86921, -1.57922, 1.65157, 1.35567,
0.88698, 0.47726, -0.73764, -0.40039]
> sorted_vlist := sort(vlist):
> v1 := sorted_vlist[1]; # lowest eigenvalue
v1 := -2.09529
> member(v1, vlist, 'indx'):
> e1 := map(fnormal, e[1..-1,indx]): Transpose(e1);

[0.25908, -0.00000, -0.25908, -0.33550, 0.16012, 0.00000, -0.16012,
0.33550, -0.54285, 0.54285]
> map(fnormal, (topological_matrix - v1).e1):
> Transpose(%);

[-0.00000, 0.00000, 0.00000, 0.00000, -0.00000, 0.00000, 0.00000,
-0.00000, 0.00000, -0.00000]

```

We end this example with a numerical description of the  $\pi$ -electron energy and the  $\pi$ -electron density function. When the orthonormalized molecular orbitals  $\psi_i$ 's are occupied by  $n_i$   $\pi$ -electrons, then the  $\pi$ -electron density  $q_j$  on the carbon atom labeled  $j$  is defined as

$$q_j = \sum_{i=0}^{10} n_i c_{ji}^2.$$

We shall concentrate on the state with lowest  $\pi$ -electron energy.

```

> for i to n do
>   vals[i] := sorted_vlist[i]:
>   member(vals[i], vlist, 'indx'):
>   vecs[i] := map(fnormal, e[1..-1,indx]):
>   vecs[i] := Normalize(vecs[i], Euclidean)
> end do:
> pi_electron_energy := sum('occupation_numbers[i]
>   * (alpha + vals[i]*beta)', 'i'=1..n);
pi_electron_energy := 10\alpha + 13.36352\beta

```

In this case, no eigenvalues with multiplicities greater than one occur, otherwise we would have applied Gram-Schmidt orthogonalization (via the procedure **GramSchmidt** in the **LinearAlgebra** package). Above, we only have normalized the eigenvectors with the **Normalize** procedure.

We are ready to compute the  $\pi$ -electron density.

```

> electron_density := seq(sum('occupation_numbers[i] *
>   (vecs[i][j])^2', 'i'=1..n), j=1..n);

```

```

electron_density := 1.17288, 1.04660, 1.17288, 0.85495, 0.98645, 0.87000,
0.98645, 0.85495, 1.02743, 1.02743

```

For chemists, this simple model explains stereospecificity of substitution. Electrophilic substitutions (like substitution by a chlorine atom) are most likely to take place at positions of highest electron density. In the case of azulene, this is position 1 or 3. Nucleophilic substitution (e.g., by a methyl group) is most likely to take place at positions of lowest electron density, i.e., on positions 4 and 8.

## 19.4 Vector Calculus

**Gradient**, **Divergence**, **Laplacian**, and **Curl** are some of the vector calculus procedures present in the **VectorCalculus** package and they work well for various orthogonal curvilinear coordinate systems. The examples of this section will give you an idea of possibilities. Throughout this section we shall assume that the **VectorCalculus** package is always loaded and that the interface variable **showassumed** has value 0 (so that variables with assumptions are not displayed with a tilde).

```
> restart; interface(showassumed=0): with(VectorCalculus);
> alias(f=f(x,y,z), g=g(x,y,z), h=h(x,y,z)): vars := x,y,z
> Gradient(f, [vars]);
```

$$\left(\frac{\partial}{\partial x} f\right) \bar{e}_x + \left(\frac{\partial}{\partial y} f\right) \bar{e}_y + \left(\frac{\partial}{\partial z} f\right) \bar{e}_z$$

You can switch to list-notation and use the procedure **attributes** to get information about a particular expression.

```
> convert(%, list);
[ $\frac{\partial}{\partial x} f$ ,  $\frac{\partial}{\partial y} f$ ,  $\frac{\partial}{\partial z} f$ ]
> attributes(%);
vectorfield, coords = cartesianx, y, z
```

Use **SetCoordinates** to set a particular system of curvilinear coordinates.

```
> SetCoordinates('cartesian' [vars]);
cartesianx, y, z
```

An advantage of declaring the coordinate system is that you do not have to specify the variables in the calls of many vector analysis procedures anymore.

```
> Gradient(f);

$$\left(\frac{\partial}{\partial x} f\right) \bar{e}_x + \left(\frac{\partial}{\partial y} f\right) \bar{e}_y + \left(\frac{\partial}{\partial z} f\right) \bar{e}_z
> F := VectorField(<f,g,h>);
F := f \bar{e}_x + g \bar{e}_y + h \bar{e}_z
> Divergence(F);

$$\left(\frac{\partial}{\partial x} f\right) + \left(\frac{\partial}{\partial y} g\right) + \left(\frac{\partial}{\partial z} h\right)$$$$

```

```

> Laplacian(f);

$$\left( \frac{\partial^2}{\partial x^2} f \right) + \left( \frac{\partial^2}{\partial y^2} f \right) + \left( \frac{\partial^2}{\partial z^2} f \right)$$

> Jacobian([f,g,h], [vars]);

$$\begin{bmatrix} \frac{\partial}{\partial x} f & \frac{\partial}{\partial y} f & \frac{\partial}{\partial z} f \\ \frac{\partial}{\partial x} g & \frac{\partial}{\partial y} g & \frac{\partial}{\partial z} g \\ \frac{\partial}{\partial x} h & \frac{\partial}{\partial y} h & \frac{\partial}{\partial z} h \end{bmatrix}$$

> Hessian(f, [vars]);

$$\begin{bmatrix} \frac{\partial^2}{\partial x^2} f & \frac{\partial^2}{\partial y \partial x} f & \frac{\partial^2}{\partial z \partial x} f \\ \frac{\partial^2}{\partial y \partial x} f & \frac{\partial^2}{\partial y^2} f & \frac{\partial^2}{\partial z \partial y} f \\ \frac{\partial^2}{\partial z \partial x} f & \frac{\partial^2}{\partial z \partial y} f & \frac{\partial^2}{\partial z^2} f \end{bmatrix}$$

> Curl(F);

$$\left( \left( \frac{\partial}{\partial y} h \right) - \left( \frac{\partial}{\partial z} g \right) \right) \bar{e}_x + \left( \left( \frac{\partial}{\partial z} f \right) - \left( \frac{\partial}{\partial x} h \right) \right) \bar{e}_y + \left( \left( \frac{\partial}{\partial x} g \right) - \left( \frac{\partial}{\partial y} f \right) \right) \bar{e}_z$$

> Divergence(%);

$$0$$

> Curl(Gradient(f, [vars]));

$$0 \bar{e}_x$$


```

The last two statements are well-known properties.

Tables 19.1 and 19.2 list the main orthogonal curvilinear coordinate systems for which Maple has built-in facilities to express gradient, divergence, laplacian, and curl. Enter **?changecoords** for a complete list and explanation of the predefined coordinate systems. The definitions of the coordinate transformations can be found in [182, 212].

Coordinates	Definition $x, y, z$	Maple Name
rectangular	$(u, v)$	cartesian
polar	$(u \cos v, u \sin v)$	polar
bipolar	$\left( \frac{\sinh v}{\cosh v - \cos u}, \frac{\sin u}{\cosh v - \cos u} \right)$	bipolar
elliptic	$(\cosh u \cos v, \sinh u \sin v)$	elliptic
parabolic	$\left( \frac{u^2 - v^2}{2}, uv \right)$	parabolic
hyperbolic	$(\sqrt{u + \sqrt{u^2 + v^2}}, \sqrt{-u + \sqrt{u^2 + v^2}})$	hyperbolic

Table 19.1. Some predefined 2D orthogonal curvilinear coordinates.

Coordinates	Definition	Maple Name
rectangular	$(u, v, w)$	cartesian
circular-cylinder	$(u \cos v, u \sin v, w)$	cylindrical
elliptic-cylinder	$(a \cosh u \cos v,$ $a \cosh u \sin v,$ $w)$	ellcylindrical
parabolic-cylinder	$(u v \cos w,$ $u v \sin w,$ $\frac{u^2 - v^2}{2})$	paraboloidal
hyperbolic-cylinder	$(\sqrt{u + \sqrt{u^2 + v^2}},$ $\sqrt{v + \sqrt{u^2 + v^2}},$ $w)$	hypercylindrical
spherical	$(u \cos v \sin w,$ $u \sin v \sin w,$ $u \cos v)$	spherical
prolate spheroidal	$(a \sinh u \sin v \cos w$ $a \sinh u \sin v \sin w,$ $a \cosh u \cos v)$	prolatespheroidal
oblate spheroidal	$(a \cosh u \sin v \cos w,$ $a \cosh u \sin v \sin w,$ $a \sinh u \cos v)$	oblatespheroidal
ellipsoidal	$(\frac{uvw}{ab},$ $\frac{1}{b} \sqrt{\frac{(u^2 - b^2)(v^2 - b^2)(b^2 - w^2)}{a^2 - b^2}},$ $\frac{1}{a} \sqrt{\frac{(u^2 - a^2)(a^2 - v^2)(a^2 - w^2)}{a^2 - b^2}})$	ellipsoidal

Table 19.2. Some predefined 3D orthogonal curvilinear coordinates.

As an example, we shall show how things work for prolate spheroidal coordinates that are defined in Maple as

$$\begin{aligned}x &= a \sinh u \sin v \cos w, \\y &= a \sinh u \sin v \sin w, \\z &= a \cosh u \cos v.\end{aligned}$$

The coordinate transformation to prolate spheroidal coordinates can be visualized by **plots[coordplot3d]**.

```
> plots[coordplot3d](prolatespheroidal);
```

The graph is shown in Figure 19.3.

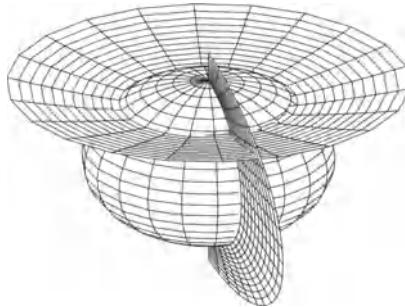


Figure 19.3. Graph of the coordinate transformation to prolate spheroidal coordinates.

The gradient, curl, divergence, and laplacian can be computed easily by setting the coordinate system first via the **SetCoordinates** procedure. Obviously, **GetCoordinateParameters** and **SetCoordinateParameters** are for getting and setting the values of the parameters in a system of curvilinear coordinates.

```

> alias(f=f(u,v,w), g=g(u,v,w), h=h(u,v,w)):
> SetCoordinates('prolatespheroidal'[u,v,w]);

          prolatespheroidalu, v, w

> GetCoordinateParameters(); # default values
           1, 1/2, 1/3

> assume(a>0):
> SetCoordinateParameters(a,b,c):
> Gradient(f);

          
$$\frac{\frac{\partial}{\partial u} f}{\sqrt{-\cos(v)^2 + \cosh(u)^2} a} \bar{e}_u + \frac{\frac{\partial}{\partial v} f}{\sqrt{-\cos(v)^2 + \cosh(u)^2} a} \bar{e}_v +$$

          
$$\frac{\frac{\partial}{\partial w} f}{\sqrt{1 - \cos(v)^2} \sqrt{\cosh(u)^2 - 1} a} \bar{e}_w$$


> subs({cos(v)^2=1-sin(v)^2, cosh(u)^2=1+sinh(u)^2}, %);

          
$$\frac{\frac{\partial}{\partial u} f}{\sqrt{\sin(v)^2 + \sinh(u)^2} a} e?u + \frac{\frac{\partial}{\partial v} f}{\sqrt{\sin(v)^2 + \sinh(u)^2} a} e?v +$$

          
$$\frac{\frac{\partial}{\partial w} f}{\sqrt{\sin(v)^2} \sqrt{\sinh(u)^2} a} e?w$$


```

Below, we compute the Jacobian matrix  $J$  of the coordinate transformation to prolatespheroidal coordinates and use it to compute the metric

tensor  $g$  (equal to  $J^T J$ ), the scale factors, and the volume element in this new coordinate system.

```

> J := Jacobian([a*sinh(u)*sin(v)*cos(w),
> a*sinh(u)*sin(v)*sin(w),
> a*cosh(u)*cos(v)], [u,v,w]);
J :=
[a cosh(u) sin(v) cos(w), a sinh(u) cos(v) cos(w), -a sinh(u) sin(v) sin(w)]
[a cosh(u) sin(v) sin(w), a sinh(u) cos(v) sin(w), a sinh(u) sin(v) cos(w)]
[a sinh(u) cos(v), -a cosh(u) sin(v), 0]
> g := map(simplify, LinearAlgebra[Transpose](J).J:
> g := subs({cos(v)^2=1-sin(v)^2, cosh(u)^2=1+sinh(u)^2},%);

g :=

$$\begin{bmatrix} a^2 (\sin(v)^2 + \sinh(u)^2) & 0 & 0 \\ 0 & a^2 (\sin(v)^2 + \sinh(u)^2) & 0 \\ 0 & 0 & a^2 \sinh(u)^2 \sin(v)^2 \end{bmatrix}$$

```

The diagonal matrix elements are the scale factors. Their product is the volume element in the new coordinate system.

```

> scalefactors := seq(sqrt(g[i,i]), i=1..3);

scalefactors :=
a sqrt(sin(v)^2 + sinh(u)^2), a sqrt(sin(v)^2 + sinh(u)^2), a sqrt(sinh(u)^2 sin(v)^2)
> dV := '*'(scalefactors); # the volume element
dV := a^3 (sin(v)^2 + sinh(u)^2) sqrt(sinh(u)^2 sin(v)^2)
```

Prolate spheroidal coordinates are used to compute the overlap integrals between Slater-Zener type atomic orbitals (q.v., [32]). In this context, these coordinates are denoted by  $\xi$ ,  $\eta$ , and  $\phi$  and are defined as

$$\begin{aligned} x &= a \sqrt{(\xi^2 - 1)(1 - \eta^2)} \cos \phi, \\ y &= a \sqrt{(\xi^2 - 1)(1 - \eta^2)} \sin \phi, \\ z &= a \xi \eta, \end{aligned}$$

where

$$a > 0, 1 \leq \xi < \infty, -1 \leq \eta \leq 1, 0 \leq \phi < 2\pi.$$

You can add this new definition with the procedure **AddCoordinates**.

```

> assume(a>0, xi>=1, -1<=eta, eta<=1, 0<=phi, phi<=2*Pi):
> AddCoordinates( myprolatespheroidal[xi,eta,phi],
> [a*sqrt((xi^2-1)*(1-eta^2))*cos(phi),
> a*sqrt((xi^2-1)*(1-eta^2))*sin(phi),
> a*xi*eta] ):
> SetCoordinates(myprolatespheroidal[xi,eta,phi]):
> alias(f=f(xi,eta,phi)): Gradient(f);
```

$$\begin{aligned} & \frac{\sqrt{\xi^2 - 1} \left( \frac{\partial}{\partial \xi} f \right)}{a \sqrt{\xi^2 - \eta^2}} \bar{e}_\xi + \frac{\sqrt{-\eta^2 + 1} \left( \frac{\partial}{\partial \eta} f \right)}{a \sqrt{\xi^2 - \eta^2}} \bar{e}_\eta + \\ & \frac{\frac{\partial}{\partial \phi} f}{a \sqrt{\xi^2 - 1} \sqrt{-\eta^2 + 1}} \bar{e}_\phi \end{aligned}$$

The Jacobian matrix of this coordinate system is as follows:

```
> J := Jacobian(
>   [a*sqrt((xi^2-1)*(1-eta^2))*cos(phi),
>    a*sqrt((xi^2-1)*(1-eta^2))*sin(phi),
>    a*xi*eta], [xi,eta,phi]);
```

$$J := \begin{bmatrix} \frac{a \cos(\phi) \xi (-\eta^2 + 1)}{\sqrt{\%1}} & -\frac{a \cos(\phi) (\xi^2 - 1) \eta}{\sqrt{\%1}} & -a \sqrt{\%1} \sin(\phi) \\ \frac{a \sin(\phi) \xi (-\eta^2 + 1)}{\sqrt{\%1}} & -\frac{a \sin(\phi) (\xi^2 - 1) \eta}{\sqrt{\%1}} & a \sqrt{\%1} \cos(\phi) \\ a \eta & a \xi & 0 \end{bmatrix}$$

$$\%1 := (\xi^2 - 1) (-\eta^2 + 1)$$

We use this matrix to compute the metric tensor, the scale factors, and the volume element.

```
> g := map(simplify, LinearAlgebra[Transpose](J).J;
```

$$g := \begin{bmatrix} \frac{a^2 (\xi^2 - \eta^2)}{\xi^2 - 1} & 0 & 0 \\ 0 & -\frac{a^2 (\xi^2 - \eta^2)}{\eta^2 - 1} & 0 \\ 0 & 0 & -a^2 (\xi^2 - 1) (\eta^2 - 1) \end{bmatrix}$$

```
> scalefactors := seq(sqrt(g[i,i]), i=1..3);
```

$$scalefactors := a \sqrt{\frac{\xi^2 - \eta^2}{\xi^2 - 1}}, a \sqrt{\frac{\xi^2 - \eta^2}{-\eta^2 + 1}}, a \sqrt{(\xi^2 - 1) (-\eta^2 + 1)}$$

```
> dV := `*`(scalefactors); # the volume element
```

$$dV := a^3 \sqrt{\frac{\xi^2 - \eta^2}{\xi^2 - 1}} \sqrt{\frac{\xi^2 - \eta^2}{-\eta^2 + 1}} \sqrt{(\xi^2 - 1) (-\eta^2 + 1)}$$

```
> simplify(%);
```

$$(\xi^2 - \eta^2) a^3$$

```
> factor(%);
```

$$(\xi - \eta) (\xi + \eta) a^3$$

Maple will now use the definition and the scale factors to compute the gradient, laplacian, etc. But sometimes a lot of simplification must be carried out to get results in proper shape.

```

> alias(f=f(xi,eta,phi)):
> Laplacian(f);


$$\left( 2 a \xi \left( \frac{\partial}{\partial \xi} f \right) + a (\xi^2 - 1) \left( \frac{\partial^2}{\partial \xi^2} f \right) - 2 a \eta \left( \frac{\partial}{\partial \eta} f \right) + a (-\eta^2 + 1) \left( \frac{\partial^2}{\partial \eta^2} f \right) \right. \\ \left. + \frac{(\xi^2 - \eta^2) a \left( \frac{\partial^2}{\partial \phi^2} f \right)}{(\xi^2 - 1) (-\eta^2 + 1)} \right) / ((\xi^2 - \eta^2) a^3)

> collect(%, [diff(f,xi$2), diff(f,eta$2), diff(f,xi),
>           diff(f,eta)], distributed, simplify);


$$\frac{(\xi^2 - 1) \left( \frac{\partial^2}{\partial \xi^2} f \right)}{\%1 a^2} - \frac{(\eta^2 - 1) \left( \frac{\partial^2}{\partial \eta^2} f \right)}{\%1 a^2} + \frac{2 \xi \left( \frac{\partial}{\partial \xi} f \right)}{\%1 a^2} - \frac{2 \eta \left( \frac{\partial}{\partial \eta} f \right)}{\%1 a^2} \\ - \frac{\frac{\partial^2}{\partial \phi^2} f}{a^2 (\xi^2 - 1) (\eta^2 - 1)}
\%1 := \xi^2 - \eta^2$$$$

```

## 19.5 Moore-Penrose Inverse

In this book, we have not really considered Maple as a programming language allowing for extension of the built-in facilities. For this we refer to books like [180, 181, 241]. But sometimes programming is a big word. For example, the *Moore-Penrose inverse*  $A^+$  of a matrix  $A$  can be computed via a limit-definition [7],

$$A^+ = \lim_{x \rightarrow 0} ((A^T A + x^2 I)^{-1} A^T),$$

if the limit exists. With all the tools of the **LinearAlgebra** package and your knowledge of Maple, you are already able to extend Maple. Actually, there is no need to implement the Moore-Penrose or generalized inverse of a matrix, because the procedure **MatrixInverse** does the job already. We use this procedure to verify our examples.

```

> Pinv := A -> map(limit,
>   MatrixInverse(Transpose(A).A +
>     ScalarMatrix(x^2, ColumnDimension(A), ColumnDimension(A))) +
>     . Transpose(A), x=0):
> M := RandomMatrix(3,2);

```

```


$$M := \begin{bmatrix} -50 & -79 \\ 30 & -71 \\ 62 & 28 \end{bmatrix}$$

> MPinv(M);

$$\left[ \begin{array}{ccc} -40297 & 76807 & 162131 \\ \hline 9345121 & 9345121 & 18690242 \\ -98619 & -155251 & -2205 \\ \hline 18690242 & 18690242 & 9345121 \end{array} \right]$$

> Equal(%, MatrixInverse(M));
true
> poly := proc() Randpoly(2,y) mod 2 end proc:
> M := RandomMatrix(3, 2, generator=poly);

$$M := \begin{bmatrix} y^2 + 1 & y^2 + y \\ y^2 + 1 & y^2 + y \\ y^2 & y^2 \end{bmatrix}$$

> MPinv(M);

$$\left[ \begin{array}{ccc} -\frac{1}{2(y-1)} & -\frac{1}{2(y-1)} & \frac{y+1}{y(y-1)} \\ \hline \frac{1}{2(y-1)} & \frac{1}{2(y-1)} & -\frac{y^2+1}{y^2(y-1)} \end{array} \right]$$

> Equal(%, MatrixInverse(M));
true

```

## 19.6 Exercises

1. In [193], the following Denavit-Hartenberg parameters for a PUMA robot arm can be found.

<i>Joint</i>	$\alpha$	$\theta$	$d$	$a$
1	$-90^\circ$	$\theta_1$	0	0
2	$0^\circ$	$\theta_2$	0	$a_2$
3	$90^\circ$	$\theta_3$	$d_3$	$a_3$
4	$-90^\circ$	$\theta_4$	$d_4$	0
5	$90^\circ$	$\theta_5$	0	0
6	$0^\circ$	$\theta_6$	0	0

Determine the position and orientation of the tip of this robot arm as a function of the kinematic parameters. Also describe the translational and rotational velocity of the tip.

2. In [221], the following Denavit-Hartenberg parameters for the IRb-6 robot manipulator can be found.

Joint	$\alpha$	$\theta$	$d$	$a$
1	$90^\circ$	$\theta_1$	$d_1$	0
2	$0^\circ$	$\theta_2$	0	$a_2$
3	$0^\circ$	$\theta_3$	0	$a_3$
4	$90^\circ$	$\theta_4$	0	0
5	$0^\circ$	$\theta_5$	$d_5$	0

Determine the position and orientation of the tip of this robot arm as a function of the kinematic parameters. Also describe the translational and rotational velocity of the tip.

3. Consider the 4-compartment model for cadmium transfer in the human body shown in Figure 19.4.

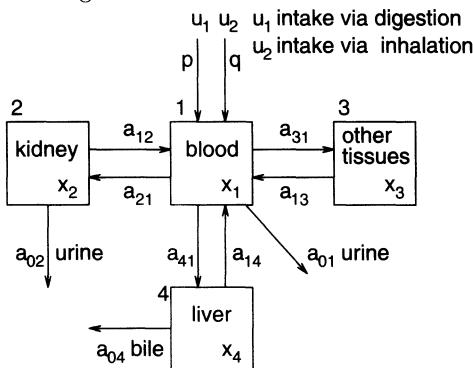


Figure 19.4. Cadmium transfer in the human body.

The corresponding mathematical equations are

$$\begin{aligned}\dot{x}(t) &= Ax(t) + Bu(t), \quad x(0) = x_0, \\ y(t) &= Cx(t),\end{aligned}$$

where

$$\begin{aligned}A &= \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & 0 & 0 \\ a_{31} & 0 & a_{33} & 0 \\ a_{41} & 0 & 0 & a_{44} \end{pmatrix}, \\ B &= \begin{pmatrix} p & q \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}, \\ C &= \begin{pmatrix} c_{11} & c_{12} & 0 & 0 \\ 0 & c_{22} & 0 & 0 \\ 0 & 0 & 0 & c_{44} \end{pmatrix},\end{aligned}$$

with  $a_{11} = -(a_{01} + a_{21} + a_{31} + a_{41})$ ,  $a_{22} = -(a_{02} + a_{12})$ ,  $a_{33} = -a_{13}$ ,  $a_{44} = -(a_{04} + a_{14})$ ,  $c_{11} = a_{01}$ , and  $c_{12} = a_{02}$ . The parameters  $p$ ,  $q$ ,  $c_{22}$ , and  $c_{44}$  are supposed to be known. Use Maple to prove that this model is structurally globally identifiable.

4. The ln-tan-cylinder coordinates are defined as

$$\begin{aligned}x &= \frac{a}{\pi} \ln \left( \frac{\sinh^2 \eta + \sin^2 \psi}{\sinh^2 \eta + \cos^2 \psi} \right), \\y &= \frac{2a}{\pi} \arctan \left( \frac{\sinh 2\eta}{\sin 2\psi} \right), \\z &= z,\end{aligned}$$

where

$$a > 0, -1 \leq \eta \leq 1, 0 \leq \psi < 2\pi.$$

Determine the scale factors of these orthogonal curvilinear coordinates and express the laplacian operator in these coordinates.

5. Using the molecular-orbital Hückel theory, compute the  $\pi$ -electron energy levels of benzene. Determine also the charge distribution of the state with lowest energy.

# 20

## A Bird's-Eye View of Gröbner Bases

In this expository chapter we give a short introduction to Gröbner basis theory and its main applications. It is a long write-up of the overview paper [127]. Quite a few books [3, 16, 37, 66, 85, 91, 98, 156] and overview articles [5, 35, 36, 61, 105, 175] on Gröbner basis theory already exist. This one differs in style and in choice of examples. The style is concrete: examples illustrate the main techniques and the use of a computer algebra system, in our case Maple, is not shunned. Most examples are taken from the science literature and illustrate the Gröbner basis techniques on real applications.

The chapter is organized as follows: in the first two sections, we describe the problem of solving polynomial equations, we discuss two heuristic solution methods, and we present the main examples, which are worked out in later sections. Hereafter, we give a gentle introduction to the basics of Gröbner basis theory. Finally, we present the main properties of Gröbner bases in the form of recipes without proof and apply them if possible to one of the examples in the introduction. Some of these examples have been treated before in this book, but will be repeated here to make the chapter self-contained.

### 20.1 Introduction

Many problems in science and engineering lead to systems of equations of the type

$$\begin{aligned} f_1(x_1, x_2, \dots, x_n) &= 0 \\ f_2(x_1, x_2, \dots, x_n) &= 0 \\ &\vdots \\ f_m(x_1, x_2, \dots, x_n) &= 0 \end{aligned}$$

where  $f_1, f_2, \dots, f_m$  are polynomials in  $n$  unknowns. Then the main task is to solve the system of polynomial equations, i.e., to find the set  $V(f_1, \dots, f_m)$  of common zeros. Some problem classes with examples from existing literature are:

**Steady state analysis of an ODE.** Given an explicit system of ODEs with polynomial right-hand sides, the steady state is described by vanishing left-hand sides. To this category belong

- NOONBURG [186]: Modeling of neural networks.

$$\begin{aligned} cx + xy^2 + xz^2 - 1 &= 0 \\ cy + yx^2 + yz^2 - 1 &= 0 \\ cz + zx^2 + zy^2 - 1 &= 0 \end{aligned} \quad (20.1)$$

where  $c$  is a parameter taking only rational values.

- FEINBERG [80]: Modeling of a chemical reaction system of type

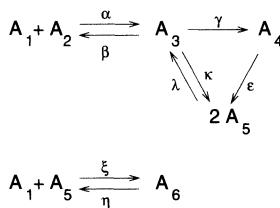


Figure 20.1. A chemical reaction mechanism.

The steady state solutions are determined by

$$\begin{aligned} -\alpha c_1 c_2 + \beta c_3 - \xi c_1 c_5 + \eta c_6 &= 0 \\ -\alpha c_1 c_2 + \beta c_3 &= 0 \\ \alpha c_1 c_2 - (\beta + \gamma + \kappa) c_3 + \lambda c_5^2 &= 0 \\ \gamma c_3 - \epsilon c_4 &= 0 \\ 2\kappa c_3 + 2\epsilon c_4 - 2\lambda c_5^2 + \eta c_6 - \xi c_1 c_5 &= 0 \\ \xi c_1 c_5 - \eta c_6 &= 0 \end{aligned} \quad (20.2)$$

**Geometrical descriptions.** Think of variables representing coordinates, distances, and angles (embedded in trigonometric functions).

- VAN DER BLIJ [22], LEVELT [166]: Molecular structure of cyclohexane. It is a special case of a regular hexagon in 3-space with sides of equal length (see Figure 20.2).

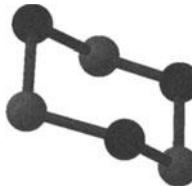


Figure 20.2. A cyclohexane configuration.

Let  $x, y, z$  represent the squares of the lengths of the “long diagonals” of the cyclic structure,  $a$  be the square of the length of a side, and let  $b$  be the length of the “short diagonals”. Consider the determinant

$$f(a, b, x, y) = \begin{vmatrix} 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & a & b & x & b \\ 1 & a & 0 & a & b & y \\ 1 & b & a & 0 & a & b \\ 1 & x & b & a & 0 & a \\ 1 & b & y & b & a & 0 \end{vmatrix}$$

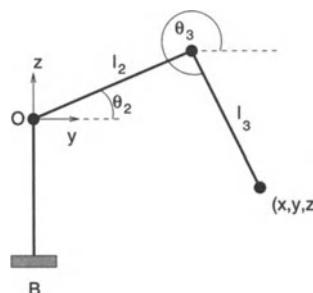
For cyclohexane (bond angle about  $110.4^\circ$ ) you may approximate  $a = 1, b = 8/3$ . The system of polynomial equations for the general cyclic structure is

$$f(a, b, x, y) = 0, \quad f(a, b, y, z) = 0, \quad f(a, b, z, x) = 0$$

and the main task is to find the real solutions.

- GONZÁLEZ-LÓPEZ/RECIO [104]: Inverse kinematics of the ROMIN robot (see Figure 20.3).

$$\begin{aligned} -\sin \theta_1(l_2 \cos \theta_2 + l_3 \cos \theta_3) - x &= 0 \\ \cos \theta_1(l_2 \cos \theta_2 + l_3 \cos \theta_3) - y &= 0 \\ l_2 \sin \theta_2 + l_3 \sin \theta_3 - z &= 0 \end{aligned} \tag{20.3}$$

Figure 20.3. projection of ROMIN robot on  $yz$ -plane.

The task is to find for fixed lengths  $l_2$  and  $l_3$  of robot arms and for a given triple  $(a, b, c)$  the allowed values of joint angles  $\theta_1$ ,  $\theta_2$  and  $\theta_3$ . The equations are polynomial in terms of sines and cosines of the joint angles  $\theta_1$ ,  $\theta_2$ ,  $\theta_3$ , augmented with trigonometric identities such as  $\cos^2 \theta_1 + \sin^2 \theta_1 = 1$ .

- HECK [125]: Geodesy.

The relationship between the geocentric Cartesian coordinates  $x$ ,  $y$ , and  $z$  of a point P on or near the surface of the earth the geodetic coordinates  $h$  (height),  $\lambda$  (longitude), and  $\phi$  (latitude) of its Helmert's projection on the geocentric reference ellipsoid is

$$\begin{aligned} x &= (N + h) \cos \phi \cos \lambda \\ y &= (N + h) \cos \phi \sin \lambda \\ z &= (N(1 - e^2) + h) \sin \phi \\ N &= \frac{a}{\sqrt{1 - e^2 \sin^2 \phi}} \\ e &= \sqrt{\frac{a^2 - b^2}{a^2}} \end{aligned} \quad (20.4)$$

Here,  $N$  is the prime vertical radius of curvature,  $e$  is the eccentricity of the reference ellipsoid, and  $a$  and  $b$  are the semi-major and semi-minor axes of the reference ellipsoid, respectively. The problem is to solve the above nonlinear system of equations in the unknowns  $h$ ,  $\lambda$ , and  $\phi$  for given  $x$ ,  $y$ , and  $z$ .

**Truncated power series.** An ansatz of a truncated power series substituted into an equation and the vanishing of coefficients of corresponding powers often gives a solvable system of polynomial equations.

- K. FORSMAN [83]: Frequencies of periodic solutions of the van der Pol equation

$$y'' - a(1 - by^2)y' + y = 0$$

through the method of harmonic balancing. The ansatz

$$y(t) = \sum_{k=-3}^3 c_k e^{-ik\omega t}$$

leads to

$$\begin{aligned} -c_1 \omega^2 + abc_1^2 c_{3i} \omega + c_1 &= 0 \\ 2abc_1 c_{3r}^2 \omega + abc_1^2 c_{3r} \omega + 2abc_1 c_{3i}^2 \omega + abc_1^3 \omega - ac_1 \omega &= 0 \\ -9c_{3r} \omega^2 + 3abc_{3i} c_{3r}^2 \omega + 3abc_{3i}^2 \omega + 6abc_1^2 c_{3i} \omega - 3ac_{3i} \omega + c_{3r} &= 0 \\ -3abc_{3r}^3 \omega - 9c_{3i} \omega^2 - 3abc_{3i}^2 c_{3r} \omega - 6abc_1^2 c_{3r} \omega + 3ac_{3r} \omega &= 0 \end{aligned}$$

$$-abc_1^3\omega + c_{3i} = 0 \quad (20.5)$$

where  $c_3 = c_{3r} + c_{3i}$  and the equations  $c_0 = c_2 = 0$  are left out.

**Structural identifiability of compartmental systems.** Basic properties of compartmental systems such as reachability, observability, and identifiability are often translated into properties of systems of polynomial equations.

- HECK [126]: Structural identifiability of a 3-compartment model of cadmium transfer through the human body. The model is shown in Figure 20.4 below.

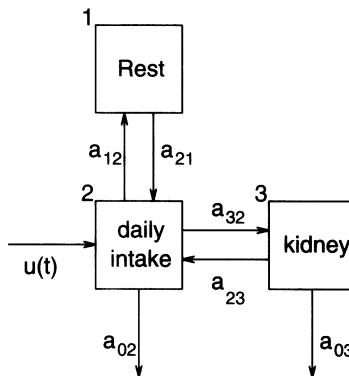


Figure 20.4. Cadmium transfer in the human body.

The transfer-function method leads to the following polynomials in  $a_{02}, a_{03}, a_{12}, a_{21}, a_{23}, a_{32}$ :

$$\begin{aligned}
 f_1 &= a_{32} \\
 f_2 &= a_{21}a_{32} \\
 f_3 &= a_{02} \\
 f_4 &= a_{02}a_{03} + a_{02}a_{21} + a_{02}a_{23} + a_{03}a_{32} \\
 f_5 &= a_{02}a_{03}a_{21} + a_{02}a_{21}a_{23} + a_{03}a_{21}a_{32} \\
 f_6 &= a_{02} + a_{03} + a_{12} + a_{21} + a_{23} + a_{32} \\
 f_7 &= a_{02}a_{03} + a_{02}a_{21} + a_{02}a_{23} + a_{03}a_{12} + a_{03}a_{21} \\
 &\quad + a_{03}a_{32} + a_{12}a_{23} + a_{21}a_{23} + a_{21}a_{32} \quad (20.6)
 \end{aligned}$$

Identifiability testing is equivalent to solving the polynomial equations

$$f_k(a_{02}, a_{03}, a_{12}, a_{21}, a_{23}, a_{32}) = f_k(b_{02}, b_{03}, b_{12}, b_{21}, b_{23}, b_{32})$$

for  $k = 1, 2, \dots, 7$  and where we consider the  $a$ 's as unknowns and the  $b$ 's as formal parameters.

Not only finding the exact solutions of systems of polynomial equations is interesting. Without solving the system of equations, you can also answer questions such as

- Is the system solvable?
- Are there a finite number of solutions and, if so, how many?
- Do equivalent systems of equations exist that give more insight to their solutions?

Gröbner basis theory is a systematic approach to answering such questions and to solving polynomial equations. In subsequent sections we shall use the above examples and others to illustrate the theory and its main applications.

## 20.2 Elementary Solution Methods

Let us consider

$$\begin{aligned} f_1 &= x - y - z \\ f_2 &= x + y - z^2 \\ f_3 &= x^2 + y^2 - 1 \end{aligned} \tag{20.7}$$

and let us try to find the set  $V(f_1, f_2, f_3)$  of common zeros.

### 20.2.1 Heuristic Method

A heuristic approach may go as follows:

$$f_1 + f_2 = 2x - z - z^2 \text{ and } f_2 - f_1 = 2y + z - z^2$$

$$\text{So, } x = \frac{1}{2}(z^2 + z) \text{ and } y = \frac{1}{2}(z^2 - z).$$

$$\text{Substitution in } f_3 \text{ gives the polynomial } \frac{1}{2}z^4 + \frac{1}{2}z^2 - 1.$$

Its solutions are  $z = 1$ ,  $z = -1$ ,  $z = \sqrt{2}i$ , and  $z = -\sqrt{2}i$ .

### 20.2.2 Gaussian Elimination-Like Method

*Step 1.* We choose  $x$  in the first polynomial as the term most suitable for eliminating terms in the other polynomials.

$$\begin{aligned} g_1 &= f_1 = x - y - z \\ g_2 &= f_2 - f_1 = 2y - z^2 + z \\ g_3 &= f_3 - (x + y + z)f_1 = 2y^2 + 2yz + z^2 - 1 \end{aligned}$$

*Step 2.* We choose the term  $2y$  in  $g_2$  as the term most suitable for eliminating terms containing  $y$  in the other polynomials.

$$\begin{aligned} h_1 &= 2g_1 + g_2 = 2x - z^2 - z \\ h_2 &= g_2 = 2y - z^2 + z \\ h_3 &= g_3 - (2y + z^2 + z)g_2 = z^4 + z^2 - 2 \end{aligned}$$

Then

$$V(f_1, f_2, f_3) = V(g_1, g_2, g_3) = V(h_1, h_2, h_3),$$

i.e., the above system of polynomials has the same set of common zeros as the original system, but the upper triangular form implies that it can be solved more easily.

### 20.2.3 Conclusion

The above methods have in common that they replace the original polynomials by nice polynomials that have the same solution set:

$$V(x - y - z, x + y - z^2, x^2 + y^2 - 1) = V(2x - z^2 - z, 2y - z^2 + z, z^4 + z^2 - 2).$$

“Nice” means here that the set of common zeros can be more easily computed from the new polynomials than from the original ones.

Let us go back to the Gaussian elimination-like method. At each step, new polynomials  $h$  are formed from pairs of old ones  $f, g$  by  $h = \alpha f + \beta g$ , where  $\alpha$  is a non-zero scalar and  $\beta$  a polynomial. The set of common zeros of  $f$  and  $g$  is the same as the set of common zeros of  $h$  and  $g$ . The set  $I(f, g)$  of all linear combinations  $\alpha f + \beta g$ , where  $\alpha$  and  $\beta$  are polynomials, is called the ideal generated by  $f$  and  $g$ . The set of common zeros of the ideal  $I(f, g)$  equals the set of common zeros of  $f$  and  $g$ , i.e.,  $V(f, g) = V(I(f, g))$ . So, all we do in the Gaussian elimination-like method is to choose at each step  $f, g, \alpha$ , and  $\beta$  clever enough such that in the end the new polynomials form a triangular system that can be easily solved.

The Gröbner basis is another nice set of generators of an ideal. Actually, the generators computed above form a Gröbner basis. In the next section we shall introduce Gröbner bases and give the basic algorithm for computing them.

## 20.3 Basics of the Gröbner Basis Method

In this section, the mathematical ingredients of Gröbner bases will be introduced step by step. But first a few words about the terminology used, since this is not universally agreed on. Let  $k$  be a field and let  $A = k[x_1, \dots, x_n]$  be the ring of polynomials in  $x_1, x_2, \dots, x_n$ . An element  $x_1^{i_1} \cdots x_n^{i_n}$  in  $A$

is called a *monomial* and an element  $cx_1^{i_1} \cdots x_n^{i_n}$  with  $c \in k \setminus \{0\}$  is called a *term*. In other words, a monomial is a power product and a term is a coefficient times a power product. So, every power product is a term (with coefficient 1), but a term is not necessarily a power product. We shall also always assume that the different terms in a polynomial have different power products (so we never write  $4x^3y^2$  as  $3x^3y^2 + x^3y^2$ ).

### 20.3.1 Term Ordering

In the Gaussian elimination-like method of the previous section we chose the pure lexicographic ordering on the set of variables that can be described as “first  $x$ , then  $y$ , and then  $z$ ”, and we sorted the terms in polynomials according to this ordering. In Gröbner basis theory, the starting point is also the choice of a *term ordering*. Although we speak of a term ordering, it is in fact an ordering on monomials. Since a polynomial is a finite linear combination (with coefficients in the ground field) of monomials, an ordering on the monomials will enable us to arrange the terms in a nonzero polynomial unambiguously in descending order of its monomials. For multivariate polynomials in  $x_1, x_2, \dots, x_n$  the *pure lexicographic ordering* with  $x_n \prec x_{n-1} \prec \dots \prec x_2 \prec x_1$  is the ordering determined by

$$x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n} \prec x_1^{j_1} x_2^{j_2} \cdots x_n^{j_n} \text{ if and only if,}$$

there exists an  $l \in \{1, \dots, n\}$ , such that  $i_k = j_k$  for all  $k < l$  and  $i_l < j_l$ .

In other words, for two monomials  $m$  and  $\tilde{m}$  holds that  $m \prec \tilde{m}$  if and only if the first variable with different exponents in  $m$  and  $\tilde{m}$  has lower exponent in  $m$ . For example, in the pure lexicographic ordering of three variables with  $z \prec y \prec x$  we have

$$\begin{aligned} 1 &\prec z \prec z^2 \prec \dots \prec y \prec yz \prec yz^2 \prec \dots \\ &\prec y^2 \prec y^2z \prec y^2z^2 \prec \dots \prec x \prec xz \prec xz^2 \prec \dots \\ &\prec xy \prec xy^2 \prec \dots \prec x^2 \prec \dots \end{aligned}$$

Four other frequently used orderings are the *graded lexicographic ordering*, the *graded reverse lexicographic ordering*, the *weighted reverse lexicographic ordering*, and the *elimination ordering*. The graded lexicographic ordering is defined by

$$\begin{aligned} m = x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n} \prec x_1^{j_1} x_2^{j_2} \cdots x_n^{j_n} = \tilde{m} &\text{ if and only if} \\ \text{either } \deg(m) = \sum_{k=1}^n i_k &< \sum_{k=1}^n j_k = \deg(\tilde{m}) \\ \text{or } \deg(m) = \deg(\tilde{m}) &\text{ and there exist an } l \in \{1, \dots, n\}, \\ &\text{such that } i_k = j_k \text{ for all } k < l \text{ and } i_l < j_l. \end{aligned}$$

In this ordering, the total degree of a term (i.e., the sum of the exponents of the variables) is the most important thing and terms of equal total

degree are ordered using the lexicographic ordering. In three variables with  $z \prec y \prec x$ :

$$\begin{aligned} 1 &\prec z \prec y \prec x \\ &\prec z^2 \prec yz \prec y^2 \prec xz \prec xy \prec x^2 \\ &\prec z^3 \prec y^2z \prec yz^2 \prec y^3 \prec xz^2 \\ &\prec xyz \prec xy^2 \prec x^2z \prec x^2y \prec x^3 \prec \dots \end{aligned}$$

The graded reverse lexicographic ordering is defined by

$$\begin{aligned} m = x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n} &\prec x_1^{j_1} x_2^{j_2} \cdots x_n^{j_n} = \tilde{m} \text{ if and only if} \\ \text{either } \deg(m) = \sum_{k=1}^n i_k &< \sum_{k=1}^n j_k = \deg(\tilde{m}) \\ \text{or } \deg(m) = \deg(\tilde{m}) \text{ and there exist an } l \in \{1, \dots, n\}, \\ &\text{such that } i_k = j_k \text{ for all } k > l \text{ and } i_l < j_l. \end{aligned}$$

In other words, in this ordering  $m \prec \tilde{m}$  if and only if  $\deg(m) < \deg(\tilde{m})$  or  $\deg(m) = \deg(\tilde{m})$  and the last variable with different exponents in the monomials  $m$  and  $\tilde{m}$  has higher exponent in  $m$ . In three variables with  $z \prec y \prec x$ :

$$\begin{aligned} 1 &\prec z \prec y \prec x \\ &\prec z^2 \prec yz \prec xz \prec y^2 \prec xy \prec x^2 \\ &\prec z^3 \prec yz^2 \prec xz^2 \prec y^2z \prec xyz \\ &\prec x^2z \prec y^3 \prec xy^2 \prec x^2y \prec x^3 \prec \dots \end{aligned}$$

The weighted reverse lexicographic ordering with respect to a vector  $(w_1, \dots, w_n)$  in  $\mathbb{Z}^n$  is defined by

$$\begin{aligned} m = x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n} &\prec x_1^{j_1} x_2^{j_2} \cdots x_n^{j_n} = \tilde{m} \text{ if and only if} \\ \text{either } \text{wdeg}(m) = \sum_{k=1}^n w_k i_k &< \sum_{k=1}^n w_k j_k = \text{wdeg}(\tilde{m}) \\ \text{or } \text{wdeg}(m) = \text{wdeg}(\tilde{m}) \text{ and there exist an } l \in \{1, \dots, n\}, \\ &\text{such that } i_k = j_k \text{ for all } k > l \text{ and } i_l < j_l. \end{aligned}$$

In other words, first order the terms with respect to the weighted degree and then use the reverse lexicographic ordering as tie-breaking algorithm. An example: in three variables with  $z \prec y \prec x$  with weight vector  $(3, 2, 1)$  we have

$$\begin{aligned} 1 &\prec z \prec z^2 \prec y \prec z^3 \prec yz \\ &\prec x \prec yz^2 \prec xz \prec y^2 \prec z^2x \\ &\prec y^2z \prec xy \prec xyz \prec y^3 \\ &\prec x^2 \prec x^2z \prec xy^2 \prec x^2y \prec x^3 \prec \dots \end{aligned}$$

The  $i^{\text{th}}$  elimination ordering is the weighted reverse lexicographic ordering with respect to some vector  $(1, \dots, 1, 0, \dots, 0)$ , where there are  $i$  1's and  $n - i$  0's. In this ordering, a monomial that contains one of  $x_1, \dots, x_i$  is always greater than a monomial that involves only  $x_{i+1}, \dots, x_n$ . As its name suggests, elimination orderings play a role in elimination theory, to eliminate the variables  $x_1, \dots, x_i$ . An example: in three variables with  $z \prec y \prec x$  with weight vector  $(1, 0, 0)$  we have

$$\begin{aligned} 1 &\prec z \prec y \prec z^2 \prec yz \\ &\prec y^2 \prec z^3 \prec yz^2 \prec y^2z \prec y^3 \\ &\prec x \prec xz \prec xy \prec xz^2 \prec xyz \\ &\prec xy^2 \prec x^2 \prec x^2z \prec x^2y \prec x^3 \prec \dots \end{aligned}$$

### A Maple intermezzo

Let us verify the above example term orderings. This way you will see how to use built-in term orderings and how to introduce any new term ordering in Maple.

First define the algebra over which the term ordering under creation is to be defined. Below, we use an algebra of polynomials in three unknowns over the rational field and introduce it in the following way.

```
> with(Ore_algebra): # load the Ore algebra package
> A := poly_algebra(x,y,z):
```

Next, we generate the list of all terms on  $x, y, z$  of degree less than or equal to 3.

```
> L:=[op(subs(u=1, convert(map(expand,
> series(1/((1-x*u)*(1-y*u)*(1-z*u)), u, 4)), polynom))));
```

$$L := [1, x, y, z, x^2, yx, y^2, zx, zy, z^2, x^3, \\ yx^2, y^2x, y^3, zx^2, zyx, zy^2, z^2x, z^2y, z^3]$$

We sort the monomials with respect to different term orderings. First we choose the lexicographic ordering, then the graded reverse lexicographic ordering, and finally a weighted reverse lexicographic ordering. Each ordering is introduced via the **termorder** command in the **Groebner** package. The procedure **testorder** allows us to compare two terms with respect to some give ordering.

```
> with(Groebner):
> T := termorder(A, plex(x,y,z)):
> sort(L, (t1,t2)->testorder(t1,t2,T));
```

$$[1, z, z^2, z^3, y, zy, z^2y, y^2, zy^2, y^3, x, \\ zx, z^2x, yx, zyx, y^2x, x^2, zx^2, yx^2, x^3]$$

```
> T := termorder(A, tdeg(x,y,z)):
> sort(L, (t1,t2)->testorder(t1,t2,T));
```

```

[1, z, y, x, z2, zy, zx, y2, yx, x2, z3,
 z2y, z2x, zy2, zyx, zx2, y3, y2x, yx2, x3]
> T := termorder(A, wdeg([3,2,1],[x,y,z])):
> sort(L, (t1,t2)->testorder(t1,t2,T));
[1, z, z2, y, z3, zy, x, z2y, zx, y2, z2x,
 zy2, yx, zyx, y3, x2, zx2, y2x, yx2, x3]

```

The graded lexicographic ordering is not built-in in Maple, but it can be introduced in the following way. First, define a procedure that compares two terms: it should return true if and only if the first argument is less than the second argument. Below, we implement the routine in such a way that it can be used for any number of variables: the trick is that we specify the variables in a global variable `_vars`.

```

> _vars := [x,y,z]:
> prec := proc(t1, t2)
>   local L1, L2, L, k, n;
>   n := nops(_vars):
>   L1 := [0$ n]: L2 := [0$ n]:
>   if degree(t1, _vars) < degree(t2, _vars) then
>     return true
>   elif degree(t1) = degree(t2) then
>     for k to nops(_vars) do
>       L1[k] := degree(t1, _vars[k]):
>       L2[k] := degree(t2, _vars[k])
>     end do:
>     L := L1-L2: L := subs(0=NULL, L);
>     if L[1]<0 then return true end if
>   end if:
>   return false
> end proc:
> T := termorder(A, user(prec, _vars)):
> Lgrlex1 := sort(L, (t1,t2)->testorder(t1,t2,T));

```

```

Lgrlex1 := [1, z, y, x, z2, zy, y2, zx, yx, x2,
z3, z2y, zy2, y3, z2x, zyx, y2x, zx2, yx2, x3]

```

Alternatively you can define a so-called *matrix term ordering*. Let

$$w_1 = (w_{11}, w_{12}, \dots, w_{1n}), \dots, w_r = (w_{r1}, w_{r2}, \dots, w_{rn}) \in \mathbb{Z}^n$$

be linearly independent row vectors defining a matrix  $W$ . We define an order  $\prec$  in  $\mathbb{Z}^r$  as follows:  $(i_1, \dots, i_r) \prec (j_1, \dots, j_r)$  if and only if the first  $i_k, j_k$  from the left which are different satisfy  $i_k < j_k$  (this is just the lexicographic ordering on  $\mathbb{Z}^r$ ). Now the matrix ordering in  $K[x_1, \dots, x_n]$  with respect to the matrix  $W$ , denoted by  $\prec_W$ , is as follows:

$$m = x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n} \prec_W x_1^{j_1} x_2^{j_2} \cdots x_n^{j_n} = \tilde{m} \text{ if and only if } \\ (\text{mdeg}(m) \cdot w_1, \dots, \text{mdeg}(m) \cdot w_r) \prec (\text{mdeg}(\tilde{m}) \cdot w_1, \dots, \text{mdeg}(\tilde{m}) \cdot w_r),$$

where the multi-degree is defined as  $\text{mdeg}(x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n}) = (i_1, \dots, i_n)$  and  $\cdot$  denotes the usual dot product in vectors in  $\mathbb{Z}^n$ . If we insist on the precedence relation  $1 \prec_w x_k$ , for all  $k$ , then all coordinates of  $w_1$  must be nonnegative. If  $w_1 = (1, \dots, 1)$ , then we have an ordering that is graded with respect to total degree. For example, the graded lexicographic ordering is the matrix term ordering defined by the following matrix

$$\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & \cdots & 0 & 1 & 0 \end{pmatrix}$$

Let us verify this for three variables:

```
> M := Matrix([[1,1,1],[1,0,0],[0,1,0]]);
```

$$M := \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Maple expects (strangely enough) a list of lists to define a matrix ordering.

```
> M := convert(M, listlist);
```

$$M := [[1, 1, 1], [1, 0, 0], [0, 1, 0]]$$

```
> T := termorder(A, 'matrix'(M, _vars));
> Lgrlex2 := sort(L, (t1,t2)->testorder(t1,t2,T));
```

$$\begin{aligned} Lgrlex2 := [1, z, y, x, z^2, zy, y^2, zx, yx, x^2, \\ z^3, z^2y, zy^2, y^3, z^2x, zyx, y^2x, zx^2, yx^2, x^3] \end{aligned}$$

We compare the two lists  $Lgrlex1$  and  $Lgrlex2$  and show that they are equal.

```
>{op(zip(testeq, Lgrlex1, Lgrlex2))};
```

$$\{ \text{true} \}$$

Note that the above matrix is not the only one that defines the graded lexicographic ordering: the following  $n \times n$ -matrix, with orthogonal row vectors,

$$\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ (n-1) & -1 & -1 & \cdots & -1 \\ 0 & (n-2) & -1 & \cdots & -1 \\ \vdots & & \ddots & & \vdots \\ 0 & \cdots & 0 & 1 & -1 \end{pmatrix}$$

also defines the graded lexicographic ordering. We illustrate this for the case of three variables:

```
> M := Matrix([[1,1,1],[2,-1,-1],[0,1,-1]]);
```

```


$$M := \begin{bmatrix} 1 & 1 & 1 \\ 2 & -1 & -1 \\ 0 & 1 & -1 \end{bmatrix}$$


> M := convert(M, listlist):
> T := termorder(A, 'matrix'(M, _vars) ):
> Lgrlex3 := sort(L, (t1,t2)->testorder(t1,t2,T));

Lgrlex3 := [1, z, y, x, z^2, zy, y^2, zx, yx, x^2,
z^3, z^2y, zy^2, y^3, z^2x, zyx, y^2x, zx^2, yx^2, x^3]
> {op(zip(testeq, Lgrlex2 , Lgrlex3))};

{true}

```

Robbiano [204] has shown that *every* term ordering can be realized as a matrix ordering with pairwise orthogonal row vectors.

One kind of elimination ordering is in Maple denoted by `lexdeg`. Below is the elimination ordering suitable for eliminating the variable  $x$ :

```

> T := termorder(A, lexdeg([x],[y,z])):
> sort(L, (t1,t2)->testorder(t1,t2,T));

[1, z, y, z^2, zy, y^2, z^3, z^2y, zy^2, y^3, x,
zx, yx, z^2x, zyx, y^2x, x^2, zx^2, yx^2, x^3]

```

You see that all terms that do not involve  $x$  appear first in this ordered list. Maple can give you a description of this ordering as a matrix term ordering:

```

> 'Groebner/term_order_to_matrix'(lexdeg([x],[y,z]));
matrix \left( \left[ \begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{array} \right], [x, y, z] \right)

```

The same ordering is defined by the following matrix of pairwise orthogonal row vectors:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & -1 \end{pmatrix}$$

Other term orderings are possible in Gröbner basis theory. The ordering  $\prec$  only has to be *admissible*, i.e., be a total ordering that satisfies

- (i)  $1 \prec m$  for every monomial  $m \neq 1$
- (ii)  $m_1 \prec m_2 \implies m_1 \cdot m \prec m_2 \cdot m$  for all monomials  $m_1, m_2, m$

To each nonzero polynomial  $f$  we can associate the *leading term*,  $\text{lt}(f)$ , as the term that is maximal among those in  $f$  according to the chosen term ordering. The *leading coefficient*,  $\text{lc}(f)$ , is defined as the coefficient of

the leading term of  $f$ . The leading term of  $f$  is the product of the leading coefficient of  $f$  and the *leading monomial*,  $\text{lm}(f)$ , of  $f$ :  $\text{lt}(f) = \text{lc}(f) \cdot \text{lm}(f)$ . An example in the pure lexicographic ordering  $z \prec y \prec x$ :

$$\text{lt}(2y - z^2 + z) = 2y, \quad \text{lc}(2y - z^2 + z) = 2, \quad \text{lm}(2y - z^2 + z) = y.$$

The names of the Maple procedures in the **Gröbner** package for obtaining the leading term and leading monomial may surprise you: they are **leadmon** and **leadterm**, respectively. This is because the software developers use a different terminology and call a power product a term and a coefficient times a power product a monomial. Just opposite to what we did and most mathematicians would do. The Maple procedure to get the leading coefficient is **leadcoeff**.

### 20.3.2 Polynomial Reduction and Normal Form

For nonzero polynomials  $f, g$  and polynomial  $\tilde{f}$  we say that  $f$  *reduces to*  $\tilde{f}$  modulo  $g$  and denote it by  $f \rightarrow_g \tilde{f}$  if there exists a term  $t$  in  $f$  that is divisible by the leading term of  $g$  and  $\tilde{f} = f - \frac{t}{\text{lt}(g)} \cdot g$ . Admissibility of the term ordering  $\prec$  guarantees that if the terms in  $f$  and  $\tilde{f}$  are ordered from high to low terms, then the first terms in which these polynomials differ are  $t$  in  $f$  and some lower term in  $\tilde{f}$ .

Some examples in the pure lexicographic ordering  $z \prec y \prec x$ :

$$x + y - z^2 \rightarrow_{x-y-z} 2y - z^2 - z$$

$$\begin{aligned} x^2 + y^2 - 1 &\rightarrow_{x-y-z} x^2 + y^2 - 1 - \frac{x^2}{x}(x - y - z) \\ &= xy + xz + y^2 - 1 \end{aligned}$$

$$\begin{aligned} xy + xz + y^2 - 1 &\rightarrow_{x-y-z} xz + 2y^2 + yz - 1 \\ &\rightarrow_{x-y-z} 2y^2 + 2yz + z^2 - 1 \end{aligned}$$

In the last example, after all the above steps, a so-called *normal form* of  $x^2 + y^2 - 1$  with respect to  $x - y - z$  is obtained and no further reductions are possible. We denote it by

$$\text{normalf}(x^2 + y^2 - 1, \{x - y - z\}) = 2y^2 + 2yz + z^2 - 1.$$

The two reductions shown above form exactly the first step in the Gaussian elimination-like method in the example of the previous section.

Let  $G = \{g_1, g_2, \dots, g_m\}$  be a set of polynomials. A polynomial  $f$  *reduces to*  $\tilde{f}$  modulo  $G$  if there exists a polynomial  $g_i$  in  $G$  such that  $f \rightarrow_{g_i} \tilde{f}$ . A *normal form*  $\text{normalf}(f, G)$  of  $f$  with respect to  $G$  is a polynomial obtained

after a finite number of reductions which contains no term anymore that is divisible by leading terms of polynomials of  $G$ . The normal form is in general not unique: consider the pure lexicographic ordering of two variables  $x$  and  $y$  such that  $y \prec x$ . Let  $g_1 = x^2y - 1$ ,  $g_2 = xy^2 - 1$  and  $f = x^2y^2$ . Then  $f \rightarrow_{g_1} = f - yg_1 = y$ , and  $f \rightarrow_{g_2} = f - xg_2 = x$ . After each step, no more reductions are possible. Both  $x$  and  $y$  are normal forms of  $f$  with respect to  $\{g_1, g_2\}$ .

Let us verify the above computations with Maple. First, the set of polynomials with which we started the previous section to explain the Gaussian elimination-like solution method.

```
> f1 := x - y - z;
> f2 := x + y - z^2;
> f3 := x^2 + y^2 - 1;
```

In the first step, we reduced the polynomials  $f_2$  en  $f_3$  with  $f_1$  in the pure lexicographic ordering with  $z \prec y \prec x$ . The Maple procedure that does the job is (of course) called **reduce**.

```
> g2 := reduce(f2, [f1], plex(x,y,z));
g2 := 2y - z^2 + z
> g3 := reduce(f3, [f1], plex(x,y,z));
g3 := 2y^2 - 1 + 2yz + z^2
```

In the next step we multiplied  $g_3$  by 2 and we reduced the result with  $g_2$ . This is an example of a pseudo-division of two multivariate polynomials. In general, **reduce** returns the remainder of the full pseudo-division of a polynomial by a list of polynomials with respect to a given term order. More precisely, the process of reduction of a polynomial  $f$  with a list of polynomials  $f_1, \dots, f_s$ , all say in  $K[x_1, \dots, x_n]$ , returns a remainder  $r \in K[x_1, \dots, x_n]$ , such that:

- $uf = u_1f_1 + \dots + u_sf_s + r$ , where  $u$  and the  $u_j$ 's are polynomials in  $K[x_1, \dots, x_n]$ ;
- no leading term of  $f_1, \dots, f_k$  divides any term of  $r$ .

When a fourth optional argument to **reduce** is provided, the value  $u$  is put in the corresponding name.

```
> h3 := reduce(g3, [g2], plex(x,y,z), 's');
h3 := -2 + z^2 + z^4
> s;
```

2

This is the difference between the procedures **reduce** and **normalf**, which computes a normal form of a polynomial with respect to a list of polynomials in a given term ordering.

```
> normalf(g3, [g2], plex(x,y,z));
          -1 +  $\frac{1}{2} z^2 + \frac{1}{2} z^4$ 
```

Let us verify that a normal form does not have to be unique. The same example as in the text before:

```
> g1 := x^2*y-1;
> g2 := x*y^2-1;
> f := x^2*y^2;
> normalf(f, [g1,g2], plex(x,y));
          y
> normalf(f, [g2,g1], plex(x,y));
          x
```

### 20.3.3 Characterization of a Gröbner Basis

The first two characterizations of a Gröbner basis are only of theoretical interest:

*G is a Gröbner basis (with respect to an admissible ordering) if and only if normal forms modulo G are unique, i.e., for all polynomials f, g, and h holds: if g = normalf(f, G) and h = normalf(f, G), then g = h.*

Alternatively,

*G is a Gröbner basis if and only if normalf(g, G) = 0 for all g in the ideal generated by G.*

Essentially, these characterizations are not much more than definitions of a Gröbner basis. They do not give any clue of how to verify the property of normal forms or how to compute such a basis. For this we have to introduce a concept that is missing in the Gaussian elimination-like method, viz., the extension of the system of polynomials by *S-polynomials*: for two polynomials f and g, the *S-polynomial* spoly(f, g) is defined by

$$\text{spoly}(f, g) = \frac{L}{\text{lt}(f)} \cdot f - \frac{L}{\text{lt}(g)} \cdot g,$$

where  $L = \text{lcm}(\text{lm}(f), \text{lm}(g))$ , i.e., L is the least common multiple of the leading monomial of f and the leading monomial of g. Alternatively, we could define  $\text{spoly}(f, g) = \alpha \cdot f - \beta \cdot g$ , where  $\alpha$  and  $\beta$  are chosen such that the leading terms cancel in the difference and the degree of  $a b$  is minimal. Two examples in the pure lexicographic ordering with  $y \prec x$ :

- For  $f = x - y - z$  and  $g = x^2 + y^2 - 1$  we have  
 $\text{spoly}(f, g) = x \cdot f - g = -xy - xz - y^2 + 1.$
- For  $f = x^2y - 1$  and  $g = yx^2 - 1$  we have  $\text{spoly}(f, g) = y \cdot f - x \cdot g = x - y.$

If you want to verify this in Maple you can use the procedure **spoly** of the **Gröbner** package.

```
> with(Groebner, spoly):
> spoly(x-y-z, x^2+y^2-1, plex(x,y));

$$-z x - y x + 1 - y^2$$

> spoly(x^2*y-1, x*y^2-1, plex(x,y));

$$-y + x$$

```

An algorithmic characterization of a Gröbner basis is the following.

*A finite set  $G$  of polynomials is a Gröbner basis if and only if  $\text{normalf}(\text{spoly}(f, g), G) = 0$  for all pairs  $(f, g)$  in  $G$ .*

It forms the basis of the following Maple procedure to test whether a list of polynomials forms a Gröbner basis with respect to some given term ordering or not.

```
> isGroebnerBasis := proc(polys::list(polynomial),
>                         tord::{ShortTermOrder, TermOrder})
>   local i,j,h,n;
>   with(Groebner, spoly, normalf):
>   n := nops(polys);
>   for i to n do
>     for j to n do
>       # compute normal form of S-polynomial
>       h := normalf(
>         spoly(polys[i], polys[j], tord), polys, tord);
>       if h <> 0 then return false end if
>     end do;
>   end do;
>   return true
> end proc:
```

Let us apply the procedure to some cases that are related with the example of the previous section.

```
> sys := [x-y-z, x+y-z^2, x^2+y^2-1];

$$\text{sys} := [x - y - z, x + y - z^2, x^2 + y^2 - 1]$$

> isGroebnerBasis(sys, plex(x,y,z));

$$\text{false}$$

> sys := [op(sys), 2*y+z-z^2, z^4+z^2-2];

$$\text{sys} := [x - y - z, x + y - z^2, x^2 + y^2 - 1, 2 y + z - z^2, z^4 + z^2 - 2]$$

> isGroebnerBasis(sys, plex(x,y,z));

$$\text{true}$$

```

### 20.3.4 The Buchberger Algorithm

Buchberger [34] invented the following algorithm to compute a Gröbner basis  $GB$  for a given set  $GB$  of multivariate polynomials computes for a given finite set  $G$  of polynomials such that the common zeros of  $GB$  and  $G$  are the same.

```

procedure groebnerBasis( $G$ )
   $GB \leftarrow G$ 
   $B \leftarrow \{(f, g) | f, g \in G, f \neq g\}$ 
  while  $B \neq \emptyset$  do
     $(f, g) \leftarrow$  select a pair in  $B$ 
     $B \leftarrow B \setminus \{(f, g)\}$ 
     $h \leftarrow \text{normalf( spoly}(f, g), GB)$ 
    if  $h \neq 0$ 
      then  $GB \leftarrow GB \cup \{h\}$ 
       $B \leftarrow B \cup \{(f, h) | f \in GB\}$ 
    end if
  end do
  return( $GB$ )
end procedure

```

Let us see how the Buchberger algorithm goes for our example

$$f_1 = x - y - z, \quad f_2 = x + y - z^2, \quad f_3 = x^2 + y^2 - 1$$

We choose the pure lexicographic ordering with  $z \prec y \prec x$ .

- (i)  $GB \leftarrow \{f_1, f_2, f_3\};$   
 $B \leftarrow \{(f_1, f_2), (f_1, f_3), (f_2, f_3)\};$   
select  $(f_1, f_2)$ ;  $B \leftarrow \{(f_1, f_3), (f_2, f_3)\};$   
 $\text{spoly}(f_1, f_2) = -2y - z + z^2$  is already in normal form;  
 $f_4 \leftarrow -2y - z + z^2;$   
 $GB \leftarrow \{f_1, f_2, f_3, f_4\};$   
 $B \leftarrow B \cup \{(f_1, f_4), (f_2, f_4), (f_3, f_4)\}.$
- (ii) select  $(f_1, f_3)$ ;  $B \leftarrow B \setminus \{(f_1, f_3)\};$   
 $\text{spoly}(f_1, f_3) = -xy - xz - y^2 + 1 \xrightarrow{\text{normalform}}_{GB} -\frac{1}{2}z^4 - \frac{1}{2}z^2 + 1;$   
 $f_5 \leftarrow -\frac{1}{2}z^4 - \frac{1}{2}z^2 + 1;$   
 $GB \leftarrow \{f_1, f_2, f_3, f_4, f_5\};$   
 $B \leftarrow B \cup \{f_1, f_5\}, \dots, (f_4, f_5)\}.$
- (iii) select  $(f_2, f_3)$ ;  $B \leftarrow B \setminus \{(f_2, f_3)\};$   
 $\text{spoly}(f_2, f_3) = xy - xz^2 - y^2 + 1 \xrightarrow{\text{normalform}}_{GB} 0.$
- (iv) select  $(f_1, f_4)$ ;  $B \leftarrow B \setminus \{(f_1, f_4)\};$   
 $\text{spoly}(f_1, f_4) = 2y^2 + 2yz + xz - xz^2 \xrightarrow{\text{normalform}}_{GB} 0.$
- (v) all remaining pairs have S-polynomials whose normal form with respect to  $GB$  equals zero.

- (vi)  $GB = \{f_1, \dots, f_5\} = \{x - y - z, x + y - z^2, x^2 + y^2 - 1, -2y - z + z^2, -\frac{1}{2}z^4 - \frac{1}{2}z^2 + 1\}$  is a Gröbner basis.

The following Maple code implements the Buchberger algorithm:

```
> groebnerBasis := proc(polys::list(polynom),
>                      tord::{ShortTermOrder,TermOrder})
>   local B, GB, p, h, i, j, f;
>   with(Groebner,normalf, spoly);
>   # B <-- { (f,g) | f, g in G }
>   B:=[seq(seq([polys[i], polys[j]], i=1..j-1),
>           j=2..nops(polys))];
>   GB:= polys; # GB <-- G
>   while not B=[] do
>     p:= B[1];      # select a pair
>     B:= B[2..-1]; # update B
>     # compute normal form of S-polynomial
>     h:= normalf(spoly(p[1], p[2], tord)), GB, tord);
>     if h <> 0 then
>       GB:= [op(GB), h]; # GB <-- GB union {h}
>       B:= [ op(B), seq( [f,h], f=GB) ]; # update B
>     end if;
>   end do;
>   GB
> end proc:
```

It is an almost direct translation of the pseudo code given before into the Maple language. Let us use it to compute the same Gröbner basis as given above.

```
> groebnerBasis([x-y-z, x+y-z^2, x^2+y^2-1], plex(x,y,z));
[x - y - z, x + y - z^2, x^2 + y^2 - 1, -2y - z + z^2, 1 -  $\frac{1}{2}z^2 - \frac{1}{2}z^4]$ 
```

The Gröbner basis is not unique:  $\{f_1, f_4, f_5\} = \{x - y - z, -2y - z + z^2, -\frac{1}{2}z^4 - \frac{1}{2}z^2 + 1\}$  is also a Gröbner basis. A *reduced Gröbner basis*  $G$  is a basis such that for all  $g$  in  $G$ :  $g = \text{normalf}(g, G \setminus \{g\})$  and  $\text{lc}(g)=1$ . This basis is in fact unique. It is created from the computed basis in the above Buchberger algorithm by the following two steps:

*Step 1. Creating a minimal Gröbner basis*

```
for g in GB do
  if there exists a p in GB \ {g} such that lt(p) divides lt(g)
  then remove g from GB
  else g ←  $\frac{1}{\text{lc}(g)} g$ 
  end if
end do
```

Now we have removed all superfluous polynomials and made them monic, i.e., all leading coefficients are equal to 1. Unfortunately a minimal Gröbner basis does not have to be unique. To achieve this, we must ensure that no

leading term of a polynomial in the basis divides any monomial in the remaining polynomials. This is done in the second step: suppose we have a minimal Gröbner basis  $GB'$ , then we compute the normal form of each  $g \in GB'$  with respect to  $GB \setminus \{g\}$

*Step 2. Computing normal forms*

**for**  $g$  in  $GB$  **do**

$g \leftarrow \text{normalf}\left(g, GB \setminus \{g\}\right)$

**end do**

In our example:  $f_2$ , and  $f_3$  are removed in step 1 and we get the minimal basis  $[f_1, -\frac{1}{2}f_4, -2f_5] = [x - y - z, y + \frac{1}{2}z - \frac{1}{2}z^2, z^4 + z^2 - 2]$ . In the second step,  $f_1$  is replaced by  $f_6 \stackrel{\text{def}}{=} \text{normalf}(f_1, \{-\frac{1}{2}f_4, -2f_5\}) = x - \frac{1}{2}z^2 - \frac{1}{2}z$ ; So, finally we get that  $\{f_6, -\frac{1}{2}f_4, -2f_5\} = \{x - \frac{1}{2}z - \frac{1}{2}z^2, y + \frac{1}{2}z - \frac{1}{2}z^2, z^4 + z^2 - 2\}$  is the reduced Gröbner basis.

The procedure **gbasis** in the **Gröbner** package computes the reduced Gröbner basis, but after this computation it takes the primitive parts of all polynomials in the basis and it sorts them in ascending order of leading monomials. The following Maple session illustrates this. Let us first verify with our own procedure whether the above reduced basis is indeed a Gröbner basis.

```
> sys := [x-1/2*z-1/2*z^2, y+1/2*z-1/2*z^2, -2+z^2+z^4];
      sys := [x -  $\frac{1}{2}z - \frac{1}{2}z^2$ , y +  $\frac{1}{2}z - \frac{1}{2}z^2$ , -2 +  $z^2 + z^4$ ]
> groebnerBasis(sys, plex(x,y,z));
      [ $x - \frac{1}{2}z - \frac{1}{2}z^2$ , y +  $\frac{1}{2}z - \frac{1}{2}z^2$ , -2 +  $z^2 + z^4$ ]
> map(primpart, %);
      [2x - z -  $z^2$ , 2y + z -  $z^2$ , -2 +  $z^2 + z^4$ ]
```

This corresponds with the Gröbner basis computed by the **gbasis** procedure in the **Gröbner** package.

```
> Groebner[gbasis]([x-y-z, x+y-z^2, x^2+y^2-1], plex(x,y,z));
      [-2 +  $z^2 + z^4$ , 2y + z -  $z^2$ , 2x - z -  $z^2$ ]
```

### 20.3.5 Improvements of Buchberger's Algorithm

The basic Buchberger algorithm from the previous subsection leaves a lot of freedom in the computational process:

- at the beginning a particular admissible ordering is chosen;

- at each step of the while-loop, a pair of polynomials in the current basis is selected.
- normal form reduction is in general not unique, so there may be different reduction steps leading to different normal forms.

The choices made in the computational process are of great influence on the performance of the algorithm, not on the final output, but on the complexity of the algorithm: number of pairs to process, growth of the coefficients, in general time and space complexity. The computational complexity makes it often difficult to compute the reduced Gröbner basis for even small problems. This limits, in practice, the scope of application of the theory. Therefore, a lot of effort has been and is still being put into adapting the algorithm towards efficiency. We mention the main improvements and other observations:

#### *Choice of admissible ordering*

Experience shows that the total degree inverse lexicographic ordering is most efficient ordering for Buchberger's algorithm. For solving systems of equations however, the pure lexicographic ordering is more useful. Luckily, for zero-dimensional systems, i.e., polynomial systems with a finite set of solutions, the Faugère-Gianni-Lazard-Mora method (abbreviated as FGLM-method [78]) can be applied to convert Gröbner bases with respect to any ordering by linear algebra methods into Gröbner bases with respect to the pure lexicographic ordering.

#### *Selection of pairs for the reduction process*

Two well-known, heuristic strategies are:

- *normal strategy* [35]. Choose a pair  $(f, g)$  such that the least common multiple of the leading terms  $\text{lt}(f)$  and  $\text{lt}(g)$  is minimal in the current ordering. In case of a ties, use another strategy to break the ties.
- *sugar strategy* [102]. The pairs are ordered with respect to a phantom degree called "sugar", priority is given to the pair with lowest sugar, and ties are broken with some algorithm.

The sugar strategy appears to be the winning strategy in many cases. In Maple, you can specify the selection strategy through the environment variable `_EnvGroebnerPairSelectionStrategy` as string values "`normal`" and "`sugar`".

#### *Avoidance of useless computations of S-polynomials*

The computation of an S-polynomial and the reduction to a normal form is a time- and space-consuming step that must possibly be avoided in the algorithm. The following criteria are used to avoid computations:

- *Criterion 1:* If polynomials  $f$  and  $g$  have disjoint leading terms, i.e., if  $\text{lt}(f)$  and  $\text{lt}(g)$  have no variables in common (in other words,

$\gcd(\text{lt}(f), \text{lt}(g)) = 1$ , then  $\text{spoly}(f, g)$  reduces to zero with respect to  $\{f, g\}$ . So, for such pairs the computation of the S-polynomial and the reduction to normal form can be skipped in the Buchberger algorithm.

- *Criterion 2:* If there are elements  $p$ ,  $f$ , and  $g$  in the current basis  $GB$  such that  $\text{lt}(p)$  divides the least common multiple of  $\text{lt}(f)$  and  $\text{lt}(g)$ , and if the pairs  $(f, p)$  and  $(g, p)$  have already been dealt with, i.e.,  $(f, p) \notin B$  and  $(g, p) \notin B$ , then the pair  $(f, g)$  can be discarded.

#### *Use of polynomials in normal form.*

In the computation of the reduced Gröbner basis via Buchberger's algorithm we brought the polynomials in the Gröbner basis to normal form with respect to each other only as a final step. Experience shows that it is better to do this at every step in the algorithm.

#### *Removal of superfluous polynomials.*

In the computation of the reduced Gröbner basis via Buchberger's algorithm we removed superfluous polynomials in the Gröbner basis after it has been computed. It is possible to remove superfluous polynomials in the intermediate bases  $GB$  in the following way: for  $g, g' \in GB, g \neq g'$ , if  $\text{lt}(g)$  divides  $\text{lt}(g')$ , then  $g'$  can be expressed by  $g$  and the S-polynomial  $\text{spoly}(g, g')$ . After you have reduced  $\text{spoly}(g, g')$  into normal form,  $g'$  as superfluous and all pairs  $(g', g'')$  can be deleted from  $B$ . Although  $g'$  is discarded you can still use it in the normal form algorithm.

#### *Normal form calculation.*

As we have seen normal form computation in general does not give a unique result: in each reduction step one has to choose the term to be eliminated and different choices may lead to different normal forms. A good strategy in the Buchberger algorithm is always to eliminate the largest term that can be eliminated in the normal form algorithm.

The following pseudo code is a better Buchberger algorithm that uses the normal selection strategy and brings the polynomials in the initial basis in normal form with respect to each other.

```

procedure groebnerBasis( $G$ )
   $GB \leftarrow \text{reduceAll}(G)$ 
   $B \leftarrow \{(f, g) \mid f, g \in G \text{ with nondisjoint leading terms}, f \neq g\}$ 
  while  $B \neq \emptyset$  do
     $(f, g) \leftarrow \text{select a pair in } B$ 
    with minimal least common multiple of leading terms
     $B \leftarrow B \setminus \{(f, g)\}$ 
    if there exists no  $p \in GB$  such that  $(f, p) \notin B, (g, p) \notin B$ ,
      and  $\text{lt}(p)$  divides  $\text{lcm}(\text{lt}(f), \text{lt}(g))$ 
    then  $h \leftarrow \text{normalf}(\text{spoly}(f, g), GB)$ 
    if  $h \neq 0$ 

```

```

then  $GB \leftarrow GB \cup \{h\}$ 
       $B \leftarrow B \cup \{(f, h) \mid f \in GB\}$ 
    end if
  end if
end do
return( $GB$ )
end procedure

procedure reduceAll( $F$ )
   $F \leftarrow \bigcup_{f \in F \setminus \{0\}} \text{makeMonic}(f)$ 
  while  $\text{normalf}\left(f, F \setminus \{f\}\right) \neq f$  for some  $f \in F$  do
     $F \leftarrow F \setminus \{f\}$ 
     $f \leftarrow \text{normalf}(f, F)$ 
    if  $f \neq 0$  then  $F \leftarrow F \cup \{f\}$  end if
  end do
  return( $F$ )
end procedure

```

These improvements have already a drastic effect on the computation of the reduced Gröbner basis of our previous example  $\{x - y - z, x + y - z^2, x^2 + y^2 - 1\}$  with respect to the pure lexicographic ordering  $z \prec y \prec x$ . It turns out that bringing the original polynomials into monic normal form with respect to each other already produces the requested Gröbner basis  $\{x - \frac{1}{2}z^2 - \frac{1}{2}z, y - \frac{1}{2}z^2 - \frac{1}{2}z, z^4 + z^2 - z\}$ .

Many of the above improvements of the basic Buchberger algorithm have been incorporated in the Maple package **Groebner**. Furthermore, this package provides some utility functions that make life for non-experts more easy. In the applications section we shall see the most important ones. One further improvement of Buchberger's algorithm, present in Maple, should be mentioned: a factorization version of the algorithm. In the applications section we shall discuss this further.

## 20.4 Properties and Applications of Gröbner Bases

We shall present the main properties of Gröbner bases in the form of recipes without proof and apply them if possible to one the examples in the introduction. The coefficient field is almost always assumed to be an algebraically closed field with characteristic zero, say the complex numbers. When we speak about a Gröbner basis we shall always mean the unique reduced Gröbner basis. We shall always assume that the **Groebner** package has been loaded via the command **with(Groebner)**. The main procedure in this package is of course **gbasis** to compute the Gröbner basis. In the examples in this section and the next ones we shall encounter many procedures of the **Groebner** package.

### 20.4.1 Equivalence of Systems of Polynomial Equations

The system of equations  $f_1(x_1, \dots, x_n) = 0, \dots, f_m(x_1, \dots, x_n) = 0$  has the same solutions as the system of polynomial equations arising from the Gröbner basis of  $f_1, \dots, f_m$  with respect to any term ordering.

In other words:

The common zeros of the polynomials  $f_1, \dots, f_m$  coincide with those of any Gröbner basis of  $f_1, \dots, f_m$  (with respect to any term ordering).

This is the main property. It can also be formulated as follows:

Two sets of polynomials generate the same ideal if and only if their Gröbner bases are equal (any term ordering may be chosen).

The next Maple session shows the computation of the Gröbner basis with respect to a pure lexicographic ordering for the system of polynomial equations (20.2), which describes the steady state of the chemical reaction shown in the introduction. A technical detail before introducing the set of polynomials: because some Maple procedure have difficulties with indexed names and interpret `gamma` as Euler-Mascheroni's number, we are not going to use the parameter  $\gamma$  and concentrations  $c_1, \dots, c_6$  directly. Via the `alias` procedure we can use names like `g` and `c1, ..., c6`, and still have nice display of formulas.

```
> alias(gamma=g, seq(c[k]=c||k, k=1..6));
           $\gamma, c_1, c_2, c_3, c_4, c_5, c_6$ 
> polys := [-alpha*c1*c2+beta*c3-xi*c1*c5+eta*c6,
   >           -alpha*c1*c2+beta*c3, alpha*c1*c2-(beta+g+kappa)*c3
   >           +lambda*c5^2, g*c3-epsilon*c4, 2*kappa*c3+2*epsilon*c4
   >           -2*lambda*c5^2+eta*c6-xi*c1*c5, xi*c1*c5-eta*c6];
polys := [- $\alpha c_1 c_2 + \beta c_3 - \xi c_1 c_5 + \eta c_6, -\alpha c_1 c_2 + \beta c_3,$ 
           $\alpha c_1 c_2 - (\beta + \gamma + \kappa) c_3 + \lambda c_5^2, \gamma c_3 - \varepsilon c_4,$ 
           $2 \kappa c_3 + 2 \varepsilon c_4 - 2 \lambda c_5^2 + \eta c_6 - \xi c_1 c_5, \xi c_1 c_5 - \eta c_6]$ 
> reactionConstants := alpha, beta, g, epsilon, kappa,
>                      lambda, eta, xi;
reactionConstants :=  $\alpha, \beta, \gamma, \varepsilon, \kappa, \lambda, \eta, \xi$ 
> concentrations := c||(1..6);
concentrations :=  $c_1, c_2, c_3, c_4, c_5, c_6$ 
> vars := reactionConstants, concentrations:
> gbasis(polys, plex(vars));
[ $\eta c_6 - \xi c_1 c_5, \varepsilon c_4 + \kappa c_3 - \lambda c_5^2, \gamma c_3 + \kappa c_3 - \lambda c_5^2, \alpha c_1 c_2 - \beta c_3]$ 
```

The original set of polynomials and the Gröbner basis have the same common zeroes. But they more easily computed with the Gröbner basis.

```
> solve({op(%)}, {c1,c2,c3,c4});
{c1 =  $\frac{\eta c_6}{\xi c_5}$ , c4 =  $\frac{\lambda c_5^2 \gamma}{\varepsilon(\gamma + \kappa)}$ , c3 =  $\frac{\lambda c_5^2}{\gamma + \kappa}$ , c2 =  $\frac{\beta \xi c_5^3 \lambda}{\alpha \eta c_6 (\gamma + \kappa)}$ }
```

The conclusion is that the chemical reaction system has a two-dimensional solution space of positive steady states.

### 20.4.2 Dimension, Hilbert Series and Hilbert Polynomial

The conclusion at the end of the previous section was that the chemical reaction system under study has a two-dimensional solution space of positive steady states. This, we could also have found by the **hilbertdim** procedure. To find this dimension of the solution space, we could have used another term ordering and we could have considered the reaction constants as parameters (simply by not mentioning them in the list of variables).

```
> hilbertdim(polys, tdeg(concentrations));
2
```

Less information, but also less computational effort.

Readers with interest in algebraic geometry may continue with this section and learn about other methods of computing with Maple the dimension of an affine variety. Others may proceed with studying themes in other subsections.

Here, we start with computing the Hilbert series of the ideal generated by the polynomials (with the reaction constants as parameters) using the graded reverse lexicographic ordering.

```
> hilbertseries(polys, tdeg(concentrations), t);

$$\frac{t^2 + 3t + 1}{(-1 + t)^2}$$

```

The *Hilbert series* of an ideal  $I$  in  $K[x_1, \dots, x_n]$  is defined as the series

$$\sum_{j=0}^{\infty} (d_j - d_{j-1}) t^j,$$

where  $d_j$  is the dimension of the vector space of the remainders with respect to the ideal of the polynomials of degree less than or equal to  $j$ . The function  $j \mapsto d_j$  is called the *affine Hilbert function* of  $I$  and is denoted by  ${}^aHF_I$ . The Hilbert series is a rational function in  $t$  of normal form  $P(t)/(1-t)^d$ , where  $d$  is the dimension of the affine variety defined by  $I$ . Let us compute the first few terms of the Hilbert series of our example:

```
> series(% , t);
1 + 5t + 10t^2 + 15t^3 + 20t^4 + 25t^5 + O(t^6)
```

There exists an integer  $\delta$  and a polynomial  $HP$  such that

$$HP(j) = {}^a\text{HF}_I(j) - {}^a\text{HF}_I(j - 1),$$

for all  $j \geq \delta$ . The Maple procedure **hilbertpoly** computes this polynomial  $HP$ .

```
> hilbertpoly(polys, tdeg(concentrations), i);
      5 i
```

This implies that there also exists a polynomial, called the *affine Hilbert polynomial* and denoted by  ${}^a\text{HP}_I$ , such that  ${}^a\text{HP}_I(j) = {}^a\text{HF}_I(j)$  for sufficiently large  $j$ . Although its name suggest differently, the Maple procedure **hilbertpoly** does not compute the affine Hilbert polynomial, but instead the backward difference of the affine Hilbert polynomial. The smallest  $\delta$  such that  ${}^a\text{HP}_I(j) = {}^a\text{HF}_I(j)$  for all  $j \geq \delta$  is called the *index of regularity* of the ideal  $I$ . The dimension theorem states the following.

*The dimension of the affine variety defined by  $I$  is equal to the degree of the affine Hilbert polynomial (for an algebraically closed ground field  $k$ ).*

The affine Hilbert polynomial can in fact be computed in Maple with the help of the **hilbertpoly** procedure. But for this, we must first introduce the *homogenization of the ideal* with respect to an extra indeterminate. Let  $f$  be a polynomial in  $K[x_1, \dots, x_n]$  of total degree  $d$ , and let  $x_0$  be a new variable. The *homogenization*  $f^h$  of the polynomial  $f$  with respect to  $x_0$  is then defined by  $f^h = x_0^d f(\frac{x_1}{x_0}, \dots, \frac{x_n}{x_0}) \in K[x_0, x_1, \dots, x_n]$ . Note that  $f^h$  is a homogeneous polynomial, i.e., all terms have the same total degree. There is an opposite operation to homogenization: if the polynomial  $(x_0, x_1, \dots, x_n) \in K[x_0, x_1, \dots, x_n]$  is homogeneous, then the polynomial  $f^a = f(1, x_1, \dots, x_n)$  is the dehomogenization of  $f$  with respect to  $x_0$ . Let  $I$  be an ideal in  $K[x_1, \dots, x_n]$ . We define the *homogenization of the ideal*  $I$  with respect to  $x_0$  to be the ideal  $I^h = \{ f^h \mid f \in I \} \subset K[x_0, x_1, \dots, x_n]$ . If  $I$  is generated by the polynomials  $f_1, \dots, f_s$ , then the ideal generated by the homogenized polynomials  $f_1^h, \dots, f_s^h$  is a subset of the homogenization  $I^h$ ; it may in fact be a strict subset. Gröbner basis theory helps us to find a finite generating set for the ideal  $I^h$ .

Let  $\prec$  be an admissible term ordering in  $K[x_1, \dots, x_n]$ , then an admissible term ordering  $\prec_h$  in  $K[x_0, x_1, \dots, x_n]$  is called a *good extension* of  $\prec$  if the leading monomial of  $f^h$  with respect to  $\prec_h$  is equal to the leading monomial of  $f$  with respect to  $\prec$ , for all polynomial  $f$  in  $K[x_1, \dots, x_n]$ . Note that a graded ordering  $\prec$  on  $K[x_1, \dots, x_n]$  can always be extended to a good extension  $\prec_h$  as follows

$$x_0^{i_0} x_1^{i_1} \cdots x_n^{i_n} \prec_h x_0^{j_0} x_1^{j_1} \cdots x_n^{j_n} \text{ if and only if}$$

$$x_1^{i_1} \cdots x_n^{i_n} \prec x_1^{j_1} \cdots x_n^{j_n} \text{ or } i_1 = j_1, \dots, i_n = j_n, i_0 < j_0.$$

Under this ordering  $x_0 \prec_h x_i$  for all  $i \geq 1$ . The most important property of the ordering  $\prec_h$  is given in the following theorem.

**Theorem.** *Let  $I$  be an ideal in  $K[x_1, \dots, x_n]$  and let  $I^h$  be its homogenization in  $K[x_0, x_1, \dots, x_n]$ . Let  $\prec_h$  be a good extension of the term ordering  $\prec$  on  $K[x_1, \dots, x_n]$ . If  $G = \{g_1, \dots, g_r\}$  is a Gröbner basis of  $I$  with respect to  $\prec$ , then  $G^h = \{g_1^h, \dots, g_r^h\}$  is the Gröbner basis of  $I^h$  with respect to  $\prec_h$ .*

It forms the basis of the algorithm for computing the projective closure of an affine variety over an algebraically closed field. We shall however concentrate on the theory of computing the Hilbert function, Hilbert polynomial and Hilbert series of  $I$ . Let  $K[x_0, x_1, \dots, x_n]_j$  denote the set of polynomials of total degree  $j$  and the zero polynomial; it forms a vector space over the field  $K$  with dimension  $\binom{n+j}{j}$ . Since  $I^h$  is a homogeneous ideal in  $R = K[x_0, x_1, \dots, x_n]$ , we have that  $I_j^h = I^h \cap R_j$ , which stands for the set of homogeneous polynomials in  $I^h$  of total degree  $j$  and the zero polynomial, is a vector subspace of  $R_j$ . The (projective) Hilbert function  $HF_{I^h}$  of the ideal  $I^h$  is defined by

$$HF_{I^h}(j) = \dim(R_j / I_j^h).$$

We have put the word projective between brackets because the commonly used terminology in algebraic geometry leaves this word out.  $HF_{I^h}(j)$  is equal to the number of monomials of degree  $j$  in  $R$  that are not in  $I^h$ . By the Hilbert-Serre Theorem, the Hilbert function can be encoded in the power series

$$\sum_{j=0}^{\infty} HF_{I^h}(j) t^j,$$

which is rational of type  $P(t) / (1 - t)^d$ . This is called the Hilbert series of  $I^h$ . For large  $j$ ,  $HF_{I^h}(j)$  is a polynomial in  $j$  of degree  $d - 1$ , called the *Hilbert polynomial*  $HP_{I^h}$  of  $I^h$ . The following relation between affine and projective Hilbert functions and between affine and projective Hilbert polynomials can be proved: for all  $j \geq 0$ ,

$${}^a HF_I(j) = HF_{I^h}(j) = {}^a HF_{I^h}(j) - {}^a HF_{I^h}(j - 1)$$

and

$${}^a HP_I(j) = HP_{I^h}(j) = {}^a HP_{I^h}(j) - {}^a HP_{I^h}(j - 1)$$

From this last statement follows that the Maple procedure **hilbertpoly** applied to the homogenization  $I^h$  of the ideal  $I$  computes in fact the affine Hilbert polynomial of  $I$  (which is equal to the projective Hilbert polynomial of  $I^h$ ).

Let us see how all fits together in the example we are working on. First we compute the Gröbner basis of the ideal generated by our polynomials.

```
> gbasis(polys, tdeg(concentrations));
[ $\gamma c_3 - \varepsilon c_4, \gamma \lambda c_5^2 - \gamma \varepsilon c_4 - \kappa \varepsilon c_4, -\gamma \alpha c_2 \eta c_6 + \xi \beta c_5 \varepsilon c_4, \xi c_1 c_5 - \eta c_6,$ 
 $\xi \varepsilon c_1 c_4 \gamma + \xi \varepsilon c_1 c_4 \kappa - \gamma \lambda \eta c_6 c_5, \gamma \alpha c_1 c_2 - \beta \varepsilon c_4,$ 
 $\gamma^2 \lambda \alpha \eta c_6 c_5 c_2 - \xi \beta \varepsilon^2 c_4^2 \gamma - \xi \beta \varepsilon^2 c_4^2 \kappa,$ 
 $\gamma^3 \lambda \alpha^2 \eta^2 c_6^2 c_2^2 - \xi^2 \beta^2 \varepsilon^3 c_4^3 \gamma - \xi^2 \beta^2 \varepsilon^3 c_4^3 \kappa]$ 
```

We need a procedure to homogenize the elements of the Gröbner basis.

```
> makeHomogeneous := proc(f::polynom, v::set(name), t::name)
>   local d, h, vv;
>   d := degree(f,v):
>   vv := seq(v[i]/t, i=1..nops(v)):
>   h := unapply(f,op(v)):
>   expand(t^d*h(vv))
> end:
```

The homogenized polynomials are:

```
> homopolys := map(makeHomogeneous, %%, {concentrations}, w);
homopolys := [ $\gamma c_3 - \varepsilon c_4, \gamma \lambda c_5^2 - w \gamma \varepsilon c_4 - w \kappa \varepsilon c_4,$ 
 $-\gamma \alpha c_2 \eta c_6 + \xi \beta c_5 \varepsilon c_4, \xi c_1 c_5 - w \eta c_6,$ 
 $\xi \varepsilon c_1 c_4 \gamma + \xi \varepsilon c_1 c_4 \kappa - \gamma \lambda \eta c_6 c_5, \gamma \alpha c_1 c_2 - w \beta \varepsilon c_4,$ 
 $\gamma^2 \lambda \alpha \eta c_6 c_5 c_2 - w \xi \beta \varepsilon^2 c_4^2 \gamma - w \xi \beta \varepsilon^2 c_4^2 \kappa,$ 
 $\gamma^3 \lambda \alpha^2 \eta^2 c_6^2 c_2^2 - w \xi^2 \beta^2 \varepsilon^3 c_4^3 \gamma - w \xi^2 \beta^2 \varepsilon^3 c_4^3 \kappa]$ 
```

These polynomials form a basis of the homogenization of the ideal. It is easy to check that the homogenization of the original polynomials does not form a generating set of the homogenization of the ideal. We can compute the Hilbert series of the homogenization of the ideal.

```
> hilbertseries(homopolys, tdeg(concentrations,w), t);

$$-\frac{t^2 + 3t + 1}{(-1 + t)^3}$$

```

The first elements of the series are as follow:

```
> series(% ,t );

$$1 + 6t + 16t^2 + 31t^3 + 51t^4 + 76t^5 + O(t^6)$$

```

We compute the (projective) Hilbert polynomial:

```
> hilbertpoly(homopolys, tdeg(concentrations,w), i);

$$1 + \frac{5}{2} i^2 + \frac{5}{2} i$$

```

Its degree equals the dimension of the solution space of our problem. The backward difference of this polynomial gives indeed the previous result of applying the **hilbertpoly** procedure on the original set of polynomials.

```
> expand(% - eval(%, i=i-1));
```

5 i

In a later subsection we shall discuss how to count the number of solutions of a system of polynomial equations, once the solution space is known to be a finite set. The following recipe could be used, too.

*Let  $I$  be the ideal generated by polynomials  $f_1, \dots, f_m$  in  $K[x_1, \dots, x_n]$  (over an algebraically closed field  $k$ ) and suppose that the Hilbert series of the homogenization  $I^h$  of  $I$  in  $K[x_0, x_1, \dots, x_n]$  equals the rational expression  $P(t) / (1-t)$ . Then the system of equations  $f_1 = 0, \dots, f_n = 0$  has a finite number of solutions and this number (counted with multiplicities) is equal to  $P(1)$ , where  $P$  coincides with the Hilbert series of  $I$ .*

Let us use this criterion to check whether the system (20.1) of the introductory section that describes the steady state of an ODE associated to a neural network has a finite number of solutions in case the system is considered in three unknowns  $x$ ,  $y$ , and  $z$ , and the ground field equals  $\mathbb{C}(c)$ , and let us compute the number of solutions.

```
> polys := [c*x+x*y^2+x*z^2-1, c*y+y*x^2+y*z^2-1,
   >           c*z+z*x^2+z*y^2-1];
polys := [c x + x y2 + x z2 - 1, c y + y x2 + y z2 - 1, c z + z x2 + z y2 - 1]
> hilbertdim(polys, tdeg(x,y,z));
0
```

From the last result we know that the solution space is zero-dimensional. Let us compute the Hilbert series of the ideal generated by the polynomials.

```
> hilbertseries(polys, tdeg(x,y,z), t);
4 t4 + 7 t3 + 6 t2 + 3 t + 1
```

The number of solutions is obtained by substitution of  $t = 1$ .

```
> subs(t=1,%);
```

21

### 20.4.3 Solvability of Polynomial Equations

*The system of equations  $f_1(x_1, \dots, x_n) = 0, \dots, f_m(x_1, \dots, x_n) = 0$  is solvable if and only if the Gröbner basis of the set of polynomials  $f_1, \dots, f_m$  is not equal to  $\{1\}$ .*

For example, this criterion allows to show that the system of equations  $x + xy^2 - 1 = 0$ ,  $x^2y + y - 1 = 0$ ,  $x^2 + y^2 - 1 = 0$  has no solutions.

```
> gbasis([x+x*y^2-1, x^2*y+y-1, x^2+y^2-1], tdeg(x,y));
[1]
```

The procedure **is\_solvable** uses this method to verify whether a system is solvable or not.

```
> is_solvable([x+x*y^2-1, x^2*y+y-1, x^2+y^2-1], {x,y,z});
false
```

Let us apply this criterion in the following graph theoretical application. An undirected graph  $G = (V, E)$ , with vertices  $V$  and edges  $E$ , is  $k$ -colorable if there exists a mapping  $f : V \rightarrow \{1, 2, \dots, k\}$  such that every pair of adjacent vertices are assigned different colors. The minimum number of colors in a coloring of  $G$  is called the chromatic number of  $G$  and is denoted by  $\chi(G)$ . The famous 4-color theorem states that  $\chi(G) = 4$  for any planar graph  $G$ . We shall apply Gröbner basis theory to prove that the Petersen graph has chromatic number 3. To this end, we shall transcribe the problem of  $k$ -colorability of a graph into a problem of solvability of some system of polynomial equations.

Let  $\omega$  be the  $k$ -th root of unity in the field of complex numbers and let  $1, \omega, \omega^2, \dots, \omega^{k-1}$  represent the  $k$  colors. For each vertex  $v_i$  in  $V$ , associate a variable  $x_i$  and introduce the following equation into the system:

$$x_i^k - 1 = 0.$$

This equation corresponds with the statement that the vertex  $v_i$  takes a color in  $\{1, \omega, \dots, \omega^{k-1}\}$ . From

$$x_i^k - x_j^k = (x_i - x_j)(x_i^{k-1} + x_i^{k-2}x_j + \dots + x_i x_j^{k-2} + x_j^{k-1})$$

follows that the condition that each pair of adjacent vertices  $[v_i, v_j] \in E$  are assigned distinct colors can be expressed by the following set of equations:

$$x_i^{k-1} + x_i^{k-2}x_j + \dots + x_i x_j^{k-2} + x_j^{k-1} = 0, \quad \text{where } [v_i, v_j] \in E.$$

The criterion is now the following.

*The graph  $G = (V, E)$  is  $k$ -colorable if and only if the constructed system of  $\#V + \#E$  polynomials equations in  $\#V$  unknowns is solvable.*

As promised, we shall prove that the Petersen graph is 3-colorable and actually has chromatic number 3. The vertices of the Petersen graph can be numbered as follows:

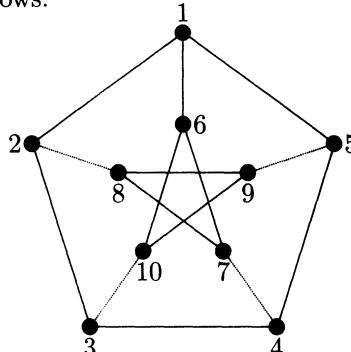


Figure 20.5. Petersen graph with numbering of vertices.

A 3-coloring with the colors **red**, **green**, and **blue** may look as follows:

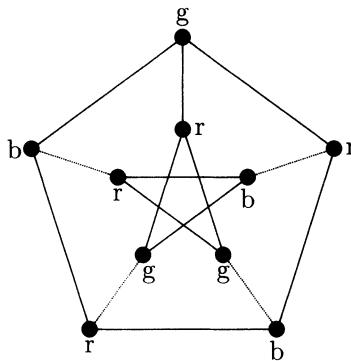


Figure 20.6. Petersen graph with some coloring.

The following Maple session proves that the Petersen graph has chromatic number 3. First, we load the **networks** package in Maple so that we can apply graph theoretical algorithms with the computer algebra system. For example, we can introduce very easily the Petersen graph and draw it.

```
> with(networks):
> G:=petersen():
> draw(G);
```

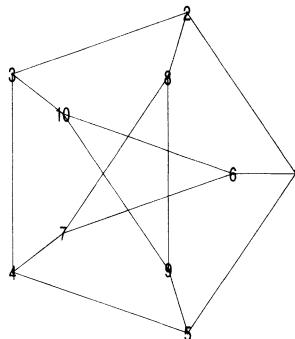


Figure 20.7. Petersen graph as drawn by Maple.

As you see in the above picture, there are 10 vertices and 15 edges.

```
> vertices(G);
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
> n := nops(%);
n := 10
> edges(G);
{e1, e2, e11, e12, e13, e14, e15, e3, e4, e5, e6, e7, e8, e9, e10}
```

```
> ends(G);

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

The next step is to build the system of polynomials that allows us to verify 3-colorability via the criterion of solvability of the system of polynomial equations.

```
> vars := { x||(1..n) };

vars := {x1, x7, x8, x9, x10, x2, x3, x4, x5, x6}

> polys := {seq(v^3-1, v=vars)} union
> {seq(cat(x,e[1])^2+cat(x,e[1])*cat(x,e[2])
> +cat(x,e[2])^2, e=ends(G))};

polys := {x4^2 + x4 x7 + x7^2, x5^2 + x5 x9 + x9^2, x7^3 - 1,
x8^2 + x8 x9 + x9^2, x1^2 + x1 x6 + x6^2, x4^2 + x4 x5 + x5^2,
x7^2 + x7 x8 + x8^2, x1^2 + x1 x5 + x5^2, x3^2 + x3 x4 + x4^2, x1^3 - 1,
x8^3 - 1, x9^3 - 1, x10^3 - 1, x2^3 - 1, x3^3 - 1, x4^3 - 1, x5^3 - 1,
x6^3 - 1, x1^2 + x1 x2 + x2^2, x6^2 + x6 x10 + x10^2, x6^2 + x6 x7 + x7^2,
x9^2 + x9 x10 + x10^2, x2^2 + x2 x3 + x3^2, x2^2 + x2 x8 + x8^2,
x3^2 + x3 x10 + x10^2}
> is_solvable(polys, vars);

true
```

Conclusion: the Petersen graph is 3-colorable. Next, we show by the same technique that the graph is not 2-colorable.

```
> polys := {seq(v^2-1, v=vars)} union
> {seq(cat(x,e[1])+cat(x,e[2]), e=ends(G))};

polys := {x1^2 - 1, x7^2 - 1, x8^2 - 1, x9^2 - 1, x10^2 - 1, x2^2 - 1, x3^2 - 1,
x4^2 - 1, x6^2 - 1, x5 + x9, x1 + x5, x1 + x6, x5^2 - 1, x2 + x3, x6 + x7,
x6 + x10, x3 + x4, x8 + x9, x1 + x2, x3 + x10, x4 + x7, x5 + x4,
x9 + x10, x7 + x8, x2 + x8}
> is_solvable(polys, vars);

false
```

Conclusion: the Petersen graph is not 2-colorable. In combination with the previous result, we have verified that the Petersen graph has chromatic number 3. This result could also have been found by computing the so-called chromatic polynomial. The value of this polynomial when one substitutes a natural number  $k$  gives the number of proper vertex-colorings of the graph using  $k$  colors.

```
> chrompoly(G, lambda);
```

```


$$\begin{aligned} & \lambda(\lambda - 1)(\lambda - 2) \\ & (\lambda^7 - 12\lambda^6 + 67\lambda^5 - 230\lambda^4 + 529\lambda^3 - 814\lambda^2 + 775\lambda - 352) \\ > \text{eval}(\%, \text{lambda}=3); & 120 \\ > \text{eval}(\%%, \text{lambda}=2); & 0 \end{aligned}$$


```

So, there exist 120 distinct 3-colorings of the Petersen graph and no 2-coloring.

#### 20.4.4 Finite Solvability of Polynomial Equations

The system of equations  $f_1(x_1, \dots, x_n) = 0, \dots, f_m(x_1, \dots, x_n) = 0$  has a finite number of solutions if and only if any Gröbner basis of the polynomials  $f_1, \dots, f_m$  has the following property: For every variable  $x_i$ , there exists a polynomial such that its leading term with respect to the chosen term ordering is a power of  $x_i$ .

Let us use this criterion to check whether system (20.1), which describes the steady state of an ODE associated to a neural network, has finite solutions.

```

> polys := [c*x+x*y^2+x*z^2-1, c*y+y*x^2+y*z^2-1,
>           c*z+z*x^2+z*y^2-1];
polys := [cx + xy2 + xz2 - 1, cy + yx2 + yz2 - 1, cz + zx2 + zy2 - 1]
> gbasis(polys, plex(c,x,y,z));
[ -yz3 + z + zy3 - y, -xz3 + z + zx3 - x, x3y - y3x - x + y,
  cz + zx2 + zy2 - 1, cy + x2y + yz2 - 1, cx + xy2 + xz2 - 1]

```

Note that no power of  $c$  appears in the grönber basis. From the criterion it follows that there is no finite set of solutions. The procedure **is\_finite** automates checking of finite solvability of polynomial equations via the above criterion.

```

> is_finite(polys, {c,x,y,z});
false

```

Let us use this procedure to check whether a finite number of solutions exists when we consider  $c$  as a parameter, i.e., when in the Gröbner basis computation coefficients are considered as rational functions in  $c$ .

```

> is_finite(polys, {x,y,z});
true

```

### 20.4.5 Counting of Finite Solutions

Suppose that the polynomials  $f_1(x_1, \dots, x_n), \dots, f_m(x_1, \dots, x_n)$  have a finite number of common zeros. Then, the number of solutions (counted with multiplicities and solutions at infinity) is equal to the cardinality of the set of monomials that are no multiples of the leading monomials of the polynomials in any Gröbner basis (any term ordering may be chosen).

We apply this criterion on the previous example, with  $c$  considered as a parameter. First we compute the leading monomials in the Gröbner basis with respect to the total degree inverse lexicographic ordering for which  $z \prec y \prec x$ .

```
> polys := [c*x+x*y^2+x*z^2-1, c*y+y*x^2+y*z^2-1,
>           c*z+z*x^2+z*y^2-1]:
> GB := gbasis(polys, tdeg(x,y,z));
```

$$\begin{aligned} GB := & [cz + z^2 + zy^2 - 1, cx + xy^2 + xz^2 - 1, cy + yx^2 + yz^2 - 1, \\ & cy^2 + 2z^2y^2 - y - cx^2 + x + cz^2 - z, zy^3 - yz^3 - y + z, cy^4 - cz^4 \\ & - y^3 + zy^2 - 2xz^2 - yz^2 + z^3 + c^2y^2 - c^2z^2 - cx - cy + cz + 1, \\ & cx^4 - cz^4 - x^3 - zy^2 - xz^2 - 2yz^2 + z^3 + c^2x^2 - c^2z^2 - cx - cy \\ & + 2, 2cz^5 + 4z^3x + 4yz^3 - 2z^4 + 3c^2z^3 - cx^2 - cy^2 + 3cxz \\ & + 3cyz - 2cz^2 + x + y - 5z + c^3z - c^2, \\ & 2yz^4 + cy^3 + 2cyz^2 + xy - y^2 + yz - 2z^2 + c^2y - c, \\ & 2z^4x + cx^3 + 2cxz^2 - x^2 + xy + zx - 2z^2 + c^2x - c, \\ & 2xyz^3 - yz + cxyz + xy - zx] \end{aligned}$$

```
> map(f->leadmon(f, tdeg(x,y,z))[2], GB);
```

$$[zx^2, xy^2, yx^2, z^2y^2, zy^3, y^4, x^4, z^5, yz^4, z^4x, xy^3]$$

So, the set of monomials that are no multiples of these leading monomials equals

$$\{1, z, y, x, z^2, yz, xz, y^2, xy, x^2, z^3, yz^2, xz^2, y^2z, xyz, y^3, x^3, z^4, yz^3, xz^3, xyz^2\}$$

and has cardinality 21. So, according to the above criterion there are 21 finite solutions.

In the above example we used the total degree ordering because the pure lexicographic ordering would take much more computing time and computer resources. For the curious reader, below are the results on a PC running Windows 2000, with a 1.7 Ghz Pentium 4 processor having 512MB main memory.

```
> settimel := time(): setbytes := kernelopts(bytesalloc):
> GBlex := gbasis(polys, plex(x,y,z)):
```

```

> cpu_time := (time()-settime)*seconds;
          cpu_time := 217.568 seconds
> memory_allocated := evalf(kernelopts(bytesalloc)
>   -setbytes)/1024/1024*Mbytes, 5;
          memory_allocated := 18.5609 Mbytes

```

In the pure lexicographic ordering with  $z \prec y \prec x$ , the leading monomials in the Gröbner basis are

```

> map(f->leadmon(f, plex(x,y,z))[2], GBlex);
          [z^14, y z^7, y^2, x]

```

So  $\{1, z, \dots, z^{13}, y, yz, \dots, yz^6\}$  is the set of monomials that are no multiples of the leading monomials. The cardinality of this set is clearly equal to 21.

You can compute the Gröbner basis in the pure lexicographic ordering much faster in Maple with the FGLM-method. First, we define three procedures, say **NFproc**, **FDproc**, and **TMproc**, in view of a call to the **fglm** procedure from the **Groebner** package: **NFproc** is a procedure to compute normal forms, **FDproc** is a procedure to find dependencies between these normal forms, and **TMproc** is used to decide when to stop the algorithm.

```

> NFproc:=proc(t,NF,TOrd)
>   NF[t]:=normalf(t,GB,tdeg(x,y,z))
> end proc:
> FDproc:=proc(M,NF)
>   local ind_lst,term,eta,sol,zero,sys,rel;
>   ind_lst:=map(op,[indices(NF)]);
>   for term in M do
>     ind_lst:=remove(divide,ind_lst,term)
>   end do;
>   zero := expand(normal(add(eta[term]*NF[term],
>     term=ind_lst)));
>   sys := {coeffs(zero, {x,y,z})};
>   sol:=[solve(sys,{seq(eta[term],term=ind_lst))});
>   rel := subs(sol[1],add(eta[term]*term,term=ind_lst));
>   'if'(rel=0,FAIL,collect(primpart(normal(subs(map(n->n=1,
>     map2(op,1,select(evalb,sol[1]))),rel)),{x,y,z}),
>     {x,y,z},distributed,factor))
> end proc:
> TMproc:=proc(border,monoideal,TOrd)
>   border<>[]
> end proc:

```

Next, we define the term ordering and compute the Gröbner basis.

```

> with(Ore_algebra):
> A:=poly_algebra(x,y,z):
> T := termorder(A, plex(x,y,z)):
> settime := time():

```

```

> FGLM:=[fglm(NFproc,FDproc,TMproc,T)]:
> GBplex := map(op,[entries(FGLM[2])]):
> cpu_time := (time()-settime)*seconds;

```

$$\text{cpu\_time} := 5.789 \text{ seconds}$$

Recall the leading monomials in this basis.

```

> map(f->leadmon(f, plex(x,y,z))[2], GBplex);

```

$$[x, y^2, yz^7, z^{14}]$$

### 20.4.6 Converting a System of Polynomial Equations into Triangular Form

If a system of equations  $f_1(x_1, \dots, x_n) = 0, \dots, f_m(x_1, \dots, x_n) = 0$  has a finite number of solutions then the Gröbner basis of the polynomials  $\{f_1, \dots, f_n\}$  with respect to the pure lexicographic ordering  $x_1 \succ x_2 \succ \dots \succ x_n$  has the following upper triangular structure:

$$\begin{array}{cccc}
g_1(x_1, & x_2, & x_3, & \dots, & x_n) \\
& \vdots & & & \\
g_p(x_1, & x_2, & x_3, & \dots, & x_n) \\
g_{p+1}(x_2, & & x_3, & \dots, & x_n) \\
& \vdots & & & \\
g_q(x_2, & x_3, & \dots, & x_n) \\
g_{q+1}(x_3, & \dots, & x_n) \\
& \vdots & & & \\
& & & & g_r(x_n) \\
& & & \vdots & \\
& & & & g_s(x_n)
\end{array}$$

**Shape lemma.** (cf. [15])

If a system of equations  $f_1(x_1, \dots, x_n) = 0, \dots, f_m(x_1, \dots, x_n) = 0$  has a finite number of solutions, then the Gröbner basis of the polynomials  $\{f_1, \dots, f_n\}$  with respect to the pure lexicographic ordering  $x_1 \succ x_2 \succ \dots \succ x_n$  has the following structure, under some suitable assumptions verified in most of the cases.

$$\left\{ x_1 - g_1(x_n), \quad x_2 - g_2(x_n), \quad \dots, \quad x_{n-1} - g_{n-1}(x_n), \quad g_n(x_n) \right\},$$

where each  $g_i$  is a univariate polynomial.

The Gröbner basis of system (20.7) with respect to the pure lexicographic ordering  $x \succ y \succ z$  has been computed in §20.3.4:  $\{x - \frac{1}{2}z^2 - \frac{1}{2}z, y - \frac{1}{2}z^2 - \frac{1}{2}z, z^4 + z^2 - z\}$ . It clearly satisfies the shape lemma.

As another illustration of the shape lemma we look at the polynomials (20.6), which define a system of polynomial equations that must be solved to verify structural identifiability of the given 3-compartment model for cadmium transfer in the human body. The following Maple session does the work via the Gröbner basis method.

First we introduce short-hand notation for variables and create the polynomials.

```
> sequence :=  
>   seq(seq(cat(a,i,j)=a[i,j], i=0..3), j=0..3),  
>   seq(seq(cat(b,i,j)=b[i,j], i=0..3), j=0..3),  
>   seq(seq(cat(t,i,j)=t[i,j], i=0..3), j=0..3), c3=c[3]:  
> (eval@subs)(S=sequence, 'macro(S')):  
> f[1] := a32;  
> f[2] := a21*a32;  
> f[3] := a02;  
> f[4] := a02*a03 + a02*a21+a02*a23 + a03*a32;  
> f[5] := a02*a03*a21 + a02*a21*a23 + a03*a21*a32;  
> f[6] := a02 + a03 + a12 + a21 + a23 + a32;  
> f[7] := a02*a03 + a02*a21 + a02*a23 + a03*a12 + a03*a21  
      + a03*a32 + a12*a23 + a21*a23 + a21*a32;
```

$$f_1 := a_{3,2}$$

$$f_2 := a_{2,1} a_{3,2}$$

$$f_3 := a_{0,2}$$

$$f_4 := a_{0,2} a_{0,3} + a_{0,2} a_{2,1} + a_{0,2} a_{2,3} + a_{0,3} a_{3,2}$$

$$f_5 := a_{0,2} a_{0,3} a_{2,1} + a_{0,2} a_{2,1} a_{2,3} + a_{0,3} a_{2,1} a_{3,2}$$

$$f_6 := a_{0,2} + a_{0,3} + a_{1,2} + a_{2,1} + a_{2,3} + a_{3,2}$$

$$f_7 := a_{0,2} a_{0,3} + a_{0,2} a_{2,1} + a_{0,2} a_{2,3} + a_{0,3} a_{1,2} + a_{0,3} a_{2,1} + a_{0,3} a_{3,2} \\ + a_{1,2} a_{2,3} + a_{2,1} a_{2,3} + a_{2,1} a_{3,2}$$

Identifiability testing is equivalent to solving the polynomial equations

$$f_k(a_{02}, a_{03}, a_{12}, a_{21}, a_{23}, a_{32}) - f_k(b_{02}, b_{03}, b_{12}, b_{21}, b_{23}, b_{32}) = 0$$

for  $k = 1, 2, \dots, 7$  and where we consider the  $a$ 's as unknowns and the  $b$ 's as formal parameters. Below, we create this list of polynomials and compute its Gröbner basis with respect to the pure lexicographic ordering  $a_{12} \succ a_{21} \succ a_{23} \succ a_{32} \succ a_{02} \succ a_{03}$ .

```
> polys := [seq(subs({a12=b12, a21=b21, a23=b23, a32=b32,  
>   a23=b23, a03=b03, a02=b02}, f[i]) - f[i], i=1..7)]:  
> vars := a12, a21, a23, a32, a02, a03:  
> gbasis(polys, plex(vars));
```

$$\begin{aligned}
& [a_{0,3}^2 b_{3,2} - b_{0,2} b_{0,3} b_{2,1} - b_{0,2} a_{0,3} b_{0,3} + b_{0,2} a_{0,3} b_{2,1} - b_{0,2} a_{0,3} b_{2,3} \\
& + b_{0,3}^2 b_{0,2} + b_{0,3}^2 b_{3,2} + b_{0,3} b_{0,2} b_{2,3} + b_{0,2} a_{0,3} b_{1,2} \\
& - b_{0,2} b_{0,3} b_{1,2} - 2 a_{0,3} b_{3,2} b_{0,3}, -b_{0,2} + a_{0,2}, -b_{3,2} + a_{3,2}, \\
& a_{2,3} b_{0,2} + a_{0,3} b_{3,2} + a_{0,3} b_{0,2} - b_{0,2} b_{0,3} - b_{0,2} b_{2,3} - b_{0,3} b_{3,2}, \\
& a_{2,1} - b_{2,1}, b_{0,2} a_{1,2} - a_{0,3} b_{3,2} - b_{0,2} b_{1,2} + b_{0,3} b_{3,2}]
\end{aligned}$$

This Gröbner basis has the form described by the shape lemma. It follows that

- the 3-compartment model is not structurally identifiable because there are two solutions and not just the trivial solution.
- the parameters  $a_{02}$ ,  $a_{21}$ , and  $a_{32}$  are identifiable because in each solution they are equal to  $b_{02}$ ,  $b_{21}$ ,  $b_{32}$ , respectively.

So, the Gröbner basis method not only determines whether the compartmental model as a whole is structurally identifiable, but it also gives, in case of non-identifiability, information on which variables are identifiable and which not.

#### 20.4.7 Finding a Univariate Polynomial

In the previous subsection we have seen that in the case of systems of polynomial equations with finite solutions, the system can be brought into triangular form with respect to the pure lexicographic ordering or under some assumptions into the form described by the shape lemma. The last polynomial is univariate in the lowest variable.

More often it is possible that the system of polynomials can be brought into a triangular form in which the last polynomial is univariate in the lowest variable in the chosen term ordering. We have already seen this for the example from geodesy with the equations (20.4) in our ultrashort introduction to the Gröbner basis method in §16.5. Here, we shall illustrate the method of finding a univariate polynomial once more, viz., for the system (20.5). This allows us to approximate frequencies of periodic solutions of the van der Pol equation

$$y'' - a(1 - by^2)y' + y = 0.$$

We assume that the following aliases have been introduced and that the defining polynomials have been entered in Maple.

```
> alias(c[1]=c1, c[3,r]=c3r, c[3,i]=c3i, omega=w):
> sys;
```

$$\begin{aligned}
& [-c_1 \omega^2 + a b c_1^2 c_{3,i} \omega + c_1, 2 a b c_1 c_{3,r}^2 \omega + a b c_1^2 c_{3,r} \omega \\
& + 2 a b c_1 c_{3,i}^2 \omega + a b c_1^3 \omega - a c_1 \omega, -9 c_{3,r} \omega^2 \\
& + 3 a b c_{3,i} c_{3,r}^2 \omega + 3 a b c_{3,i}^3 \omega + 6 a b c_1^2 c_{3,i} \omega - 3 a c_{3,i} \omega]
\end{aligned}$$

$$+ c_{3,r} - 3ab c_{3,r}^3 \omega - 9c_{3,i} \omega^2 - 3ab c_{3,i}^2 c_{3,r} \omega \\ - 6ab c_{1}^2 c_{3,r} \omega + 3ac_{3,r} \omega - ab c_1^3 \omega + c_{3,i}]$$

To make computations simpler we discard solutions with  $c_1 = 0$  and divide the first polynomial by  $c_1$  and the second polynomial by  $ac_1\omega$ .

```
> sys[1] := expand(sys[1]/c1):
> sys[2] := expand(sys[2]/(a*c1*w)):
> sys;
```

$$[-\omega^2 + c_1 ab c_{3,i} \omega + 1, 2bc_{3,r}^2 + c_1 bc_{3,r} + 2bc_{3,i}^2 + c_1^2 b - 1, \\ -9c_{3,r} \omega^2 + 3ab c_{3,i} c_{3,r}^2 \omega + 3ab c_{3,i}^3 \omega + 6ab c_1^2 c_{3,i} \omega \\ - 3ac_{3,i} \omega + c_{3,r}, -3ab c_{3,r}^3 \omega - 9c_{3,i} \omega^2 - 3ab c_{3,i}^2 c_{3,r} \omega \\ - 6ab c_1^2 c_{3,r} \omega + 3ac_{3,r} \omega - ab c_1^3 \omega + c_{3,i}]$$

We choose the pure lexicographic ordering with  $c_{3r} > c_{3i} > c_1 > \omega$  and we compute the Gröbner basis with respect to this term ordering. We are interested in the first polynomial in this basis.

```
> gsys := gbasis(sys, plex(c3r,c3i,c1,w)):
> collect(gsys[1], w);
```

$$-1 + 3249\omega^{10} + (549a^2 - 6213)\omega^8 + (-567a^2 + 3754)\omega^6 \\ + (-842 + 159a^2)\omega^4 + (-13a^2 + 53)\omega^2$$

It is a polynomial in  $\omega^2$  of 5<sup>th</sup> degree with one real solution for any positive  $a$ . The **univpoly** procedure of the **Gröbner** package gives the same result.

```
> collect(univpoly(w, sys, {c3r,c3i,c1,w}), w);
```

$$-1 + 3249\omega^{10} + (549a^2 - 6213)\omega^8 + (-567a^2 + 3754)\omega^6 \\ + (-842 + 159a^2)\omega^4 + (-13a^2 + 53)\omega^2$$

The Taylor series of  $\omega^2$  about  $a = 0$  can easily be computed.

```
> subs(w=sqrt(W), %):
> w^2 = series(RootOf(% ,W), a, 8);
```

$$\omega^2 = 1 - \frac{1}{8}a^2 + \frac{3}{256}a^4 + \frac{1}{512}a^6 + O(a^8)$$

### 20.4.8 Decomposition of Ideals

Suppose that during a Gröbner basis computation in the intermediate set  $GB$  of generators there appears a polynomial  $g$  that factorizes into two polynomials, say  $g = g_1 \cdot g_2$ . Then  $g$  vanishes if either  $g_1$  or  $g_2$  vanishes. A common zero of  $GB$  is either a common zero of  $GB_1 = GB \cup \{g_1\}$  or of  $GB_2 = GB \cup \{g_2\}$ . The Gröbner basis for  $GB_1$  and  $GB_2$  can be computed separately. This decomposition of the Gröbner basis computation makes the algorithm more efficient and it gives more insight in the structure of the problem.

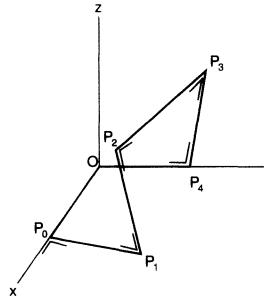
The procedure **gsolve** from Maple's **Groebner** package inspects all intermediate polynomials in the Buchberger algorithm with respect to factorization and if possible forks the computation. Let us illustrate this first on system (20.1):

```
> polys := [c*x+x*y^2+x*z^2-1, c*y+y*x^2+y*z^2-1,
>           c*z+z*x^2+z*y^2-1];
polys := [cx + xy2 + xz2 - 1, cy + yx2 + yz2 - 1, cz + zx2 + zy2 - 1]
> gsolve(polys, [x,y,z]);
{[[2cz4 - 2z3 + c2z2 + 2cz + 1, -2c2z3 - c3z - 2z + 2y - c2 + 4cz2,
-2c2z3 + 4cz2 - c3z - c2 - 2z + 2x], plex(x, y, z),
{-c + 4z2 - 2cz3 - zc2, -1 + 2z3 + cz}], [
[z3 + cz + 1, y2 + zy + z2 + c, x + y + z], plex(x, y, z),
{-zy, -1 - yz2, yz2 - 2}], [
[-1 + 2z3 + cz, -z + y, -z + x], plex(x, y, z), {2z2 + c}], [
[2z4 + 3cz2 - z + c2, -z + y, cx + 2z3 + 2cz - 1], plex(x, y, z),
{-1 + 2z3 + cz, 2z2 + c, z3 + cz + 1}], [
[2z4 + 3cz2 - z + c2, 2z3 + 2cz + cy - 1, -z + x], plex(x, y, z),
{-3c3 - 2z3 - 2c2z2 - 3cz + 1, -2z3 - 3cz + 1}]}
```

You get a list of new systems of polynomials whose common zeros are the solutions of the original system. The idea is that the common zeros of the subsystems can be found more easily than the common zero of the original system as a whole. Let us have a look at one subsystem, viz.,  $\{[-1 + 2z^3 + cz, -z + y, -z + x], \text{plex}(x, y, z), \{2z^2 + c\}\}$ . This means that for the solutions of the system  $\{[-1 + 2z^3 + cz, -z + y, -z + x]\}$  the condition holds that  $2z^2 + c$  does not vanish, i.e.,  $z \neq \frac{1}{2}\sqrt{-c}$ . An empty set as third argument in the subsystem means that there are no conditions.

One more remark: in the **gsolve** command we have put the variables in a list, indicating that Maple should try to maintain the given ordering as much as possible. Here it was successful in doing so, but there is no guarantee that Maple always succeeds.

An example where the decomposition of the Gröbner basis computation gives more insight is the following study of the conformational geometry of cyclohexane. Actually we treat the more general case of a closed linkage system  $M_6$  in 3-space with sides of equal length. More precisely, any pair of subsequent edges is at a fixed angle, but the plane through these edges is not constrained. Figure 20.8 shows  $M_6$  for right joint angles.

Figure 20.8. Closed linkage system  $M_6$ .

Let  $x, y, z$  represent the squares of the lengths of the “long diagonals” of the cyclic structure,  $a$  be the square of the length of a side, and let  $b$  be the square of length of the “short diagonals”. The geometric restrictions can be formulated in algebraic terms as vanishing of the following three determinants.

```
> Matrix([[0,1,1,1,1,1], [1,0,a,b,x,b], [1,a,0,a,b,y],
>          [1,b,a,0,a,b], [1,x,b,a,0,a], [1,b,y,b,a,0]]);
```

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & a & b & x & b \\ 1 & a & 0 & a & b & y \\ 1 & b & a & 0 & a & b \\ 1 & x & b & a & 0 & a \\ 1 & b & y & b & a & 0 \end{bmatrix}$$

```
> f[1] := collect(LinearAlgebra[Determinant](%), 
> [x,y,z], distributed, factor);
```

$$\begin{aligned} f_1 := & (b+a)(7b-a)(b-a)^2 + (2b+2a)xy^2 + (2b+2a)yx^2 \\ & - (b-a)^2y^2 - (b-a)^2x^2 + (-4ba - 4a^2 - 4b^2)xy - x^2y^2 \\ & - 2(b-a)^3x - 2(b-a)^3y \end{aligned}$$

Cyclic permutation gives the other two defining polynomials.

```
> f[2] := subs({x=y, y=z}, f[1]): 
> f[3] := subs({y=z, z=x}, f[2]):
```

Of course there are restrictions on values of  $a, b, x, y$ , and  $z$ :  $0 \leq b \leq 4a$  and  $x, y, z$  are between  $\frac{(b-a)^2}{a}$  and  $a+2b$ . For cyclohexane you can approximately take  $a = 1$ ,  $b = \frac{8}{3}$ . Let us now compute the decomposed Gröbner basis for the general case.

```
> sys := gsolve([f[1], f[2], f[3]], [z, y, x]): 
> nops(sys);
```

There are 8 sets of Gröbner bases. The last six describe discrete geometries as special cases. For example,

```
> sys[3];
```

$$[[-a - b + x, y - b - a, -a + z - b], \text{plex}(z, y, x), \{\}]$$

This is the special case that  $x = y = z = a + b$ .

```
> sys[7];
```

$$[[-a - b + x, y - b - a, az - b^2 + 2ba - a^2], \text{plex}(z, y, x), \{\}]$$

This is the special case that  $x = y = a + b$  and  $z = \frac{(b-a)^2}{a}$ . The second system does not have a finite set of solutions as can be seen from its form.

```
> sys[2];
```

$$\begin{aligned} & [-7b^4 + a^4 - 2a^3x + 8b^3a + b^2y^2 + 2b^3y + 6b^2a^2 + 2b^3x - 8ba^3 \\ & + x^2y^2 + x^2b^2 + x^2a^2 + a^2y^2 - 2a^3y + 4axby + 6a^2xb \\ & - 2ax^2y^2 + 4a^2xy - 6ax^2b - 6b^2ya - 2bx^2y^2 + 4b^2xy \\ & - 2b^2ay^2 + 6ba^2y - 2x^2ba - 2x^2by - 2x^2ya, 2b^3 + b^2y \\ & - 6ab^2 + b^2z + 4xb^2 + 6a^2b - 2bxy - 2x^2b - 2bxz - 2bya \\ & + 4xba - 2ba^2z + yx^2 - 2ax^2 + 4a^2x + zx^2 + za^2 - 2axy \\ & + a^2y - 2a^3 - 2xz^a, 3b^2 - 2by - 2zb + 6ba - 2bx - 2ax \\ & + zx + xy + zy - 2az - 2ya + 3a^2], \text{plex}(z, y, x), \\ & \{a^2 - 2ax - 8ba + 7b^2 - 4bx + x^2, -a + b + x, a + b - x\}] \end{aligned}$$

The solution set will depend on one parameter and describes the situation where there is one degree of freedom. This freedom comes actually from the symmetry of the molecule. Once this is broken, there are only at most 16 discrete configurations. In order to get a better view on the situation, we introduce symmetric coordinates

$$s = x + y + z, \quad t = xy + yz + zx, \quad p = xyz.$$

We add these defining polynomials to the second basis and compute the Gröbner basis with respect to the ordering  $z \succ y \succ x \succ p \succ t \succ s$ .

```
> polys := [op(sys[2][1]), s-x-y-z, t-x*y-y*z-z*x, p-x*y*z];
> gb := gbasis(polys, plex(z,y,x,p,t,s));
```

$$\begin{aligned} & gb := [3a^2 + 6ba + 3b^2 + t - 2bs - 2as, \\ & p + 6ab^2 - 6a^2b + 2a^3 - 2b^3 + 2bas - a^2s - b^2s, -x^2s + x^3 \\ & + 6ab^2 - 6a^2b + 2a^3 - 2b^3 - 3a^2x + 2axs + 2bxz - 6xba \\ & - 3xb^2 + 2bas - a^2s - b^2s, \\ & -3b^2 + 2bs - 6ba + x^2 + y^2 - ys - 3a^2 + 2as - xs + xy, \\ & -s + x + y + z] \end{aligned}$$

From the first two basis elements we can express  $p$  and  $t$  in terms of  $s$ .

```
> solve({gb[1], gb[2]}, {p,t}):
> map(collect, %, s, factor);
```

$$\{t = (2b + 2a)s - 3(b + a)^2, p = (b - a)^2 s + 2(b - a)^3\}$$

The third basis element gives  $x$  as a root of a third degree polynomial in  $s$ .

```
> collect(gb[3], [x,s], recursive, factor);
```

$$x^3 - x^2 s + ((2b + 2a)s - 3(b + a)^2)x - (b - a)^2 s - 2(b - a)^3$$

Once  $x$  is known in terms of  $s$  then the first two basis elements can be used to determine  $y$  and  $z$ .

#### 20.4.9 An Example From Robotics

The study of the inverse kinematics of the ROMIN manipulator [104] shall be used to give an idea how helpful Gröbner bases are for this kind of work. The system of equations is as follows (20.3).

$$\begin{aligned} -\sin \theta_1(l_2 \cos \theta_2 + l_3 \cos \theta_3) - x &= 0 \\ \cos \theta_1(l_2 \cos \theta_2 + l_3 \cos \theta_3) - y &= 0 \\ l_2 \sin \theta_2 + l_3 \sin \theta_3 - z &= 0 \end{aligned}$$

where  $l_2$  and  $l_3$  denote the length of the first and second arm and the robot  $\theta_1, \theta_2, \theta_3$  represent rotation angles around the base and the robot arms. With these equations, the angles should be computed given a position  $(x, y, z)$  of the tip of the robot. Actually we are satisfied if we can find the cosines or sines of these angles. There are two ways of converting the equations into polynomial form.

- A rational parameterization of trigonometric functions is used:

$$\cos \theta_i = \frac{1 - t_i^2}{1 + t_i^2}, \quad \sin \theta_i = \frac{2t_i}{1 + t_i^2},$$

for  $i = 1, 2, 3$ . When you use these rational expressions make sure that you multiply the equations with products of  $(1 + t_1^2)$ ,  $(1 + t_2^2)$ , and  $(1 + t_3^2)$  so that polynomial equations arise. In our case, we get three polynomials in  $l_2, l_3, x, y, z, t_1, t_2, t_3$  from which we could solve  $t_1, t_2$ , and  $t_3$  by computing the decomposed Gröbner basis with respect to lexicographic ordering  $x \succ y \succ z \succ t_3 \succ t_2 \succ t_1$ . Drawback of this method is that joint angles of 180 degrees are not possible in this parameterization and joint angles close to 180 degrees give awkwardly large numerical values.

- The cosines and sines are considered as variables and trigonometric relations are added in the format of polynomial equations. For the ROMIN manipulator you get:

$$\begin{aligned} -s_1(l_2 c_2 + l_3 c_3) - x &= 0 \\ c_1(l_2 c_2 + l_3 c_3) - y &= 0 \\ l_2 s_2 + l_3 s_3 - z &= 0 \end{aligned}$$

$$\begin{aligned} c_1^2 + s_1^2 - 1 &= 0 \\ c_2^2 + s_2^2 - 1 &= 0 \\ c_4^2 + s_3^2 - 1 &= 0 \end{aligned} \quad (20.8)$$

where  $s_i = \sin \theta_i$ ,  $c_i = \cos \theta_i$  for  $i = 1, 2, 3$ . The set of defining polynomials can now be converted into a Gröbner basis with respect to the pure lexicographic ordering  $s_1 \succ c_1 \succ s_2 \succ c_2 \succ s_3 \succ c_3$ . We consider  $l_2, l_3, x, y, z$  as parameters.

```
> alias(s[1]=s1, s[2]=s2, s[3]=s3, c[1]=c1, c[2]=c2,
>       c[3]=c3, l[1]=l1, l[2]=l2, l[3]=l3):
> polys := [-s1*(l2*c2+l3*c3)-x, c1*(l2*c2+l3*c3)-y,
>            l2*s2+l3*s3-z, c1^2+s1^2-1, c2^2+s2^2-1, c3^2+s3^2-1,
>            c3^2+s3^2-1];
```

```
polys := [-s1 (l2 c2 + l3 c3) - x, c1 (l2 c2 + l3 c3) - y,
           l2 s2 + l3 s3 - z, c1^2 + s1^2 - 1, c2^2 + s2^2 - 1, c3^2 + s3^2 - 1]
> gbasis(polys, plex(c3,s3,c2,s2,c1,s1));
```

$$\begin{aligned} & [x^2 s_1^2 - x^2 + y^2 s_1^2, y s_1 + c_1 x, 4 y^2 l_2^2 s_2^2 + 4 z^2 l_2^2 s_2^2 \\ & + 4 l_2^2 x^2 s_2^2 - 4 l_2 s_2 z^3 - 4 l_2^3 s_2 z - 4 l_2 x^2 s_2 z + 4 l_2 s_2 z l_3^2 \\ & - 4 l_2 s_2 z y^2 + 2 x^2 y^2 + 2 z^2 l_2^2 - 2 y^2 l_3^2 - 2 l_2^2 l_3^2 + 2 y^2 z^2 \\ & - 2 z^2 l_3^2 + x^4 - 2 y^2 l_2^2 + z^4 + l_2^4 + l_3^4 + 2 x^2 z^2 - 2 l_2^2 x^2 \\ & - 2 l_3^2 x^2 + y^4, 2 l_2 c_2 x - 2 l_2 z s_2 s_1 + s_1 z^2 + s_1 l_2^2 - s_1 l_3^2 \\ & + x^2 s_1 + s_1 y^2, l_2 s_2 + l_3 s_3 - z, 2 l_3 c_3 x + 2 l_2 z s_2 s_1 \\ & + x^2 s_1 + s_1 y^2 - s_1 z^2 - s_1 l_2^2 + s_1 l_3^2] \end{aligned}$$

The Gröbner basis is in triangular form. So, in principle the inverse kinematics problem is solved. However, the key problem is whether for numerical values of the parameters the above basis stays a Gröbner basis. If so, everything is ok. In the example we are considering, if we choose  $l_2 \neq 0, l_3 \neq 0, x^2 + y^2 \neq 0$ , then the above set is still a Gröbner basis. This specialization problem can be avoided by using so-called comprehensive Gröbner bases [232].

#### 20.4.10 Implicitization of Parametric Objects

Consider the parametric equations

$$\begin{aligned} x_1 &= f_1(t_1, t_2, \dots, t_m) \\ x_2 &= f_2(t_1, t_2, \dots, t_m) \\ &\vdots \\ x_n &= f_n(t_1, t_2, \dots, t_m) \end{aligned}$$

where  $f_1, f_2, \dots, f_n$  are polynomials in  $m$  unknowns. The question is to eliminate the  $t$ 's and find polynomial equations in  $x_1, \dots, x_n$  that define the parametric object. More precisely, we search for polynomials  $g_1, \dots, g_k$  in the unknowns  $x_1, \dots, x_n$  such that for all  $a_1, \dots, a_n$ :

$$g_1(a_1, \dots, a_n) = \dots = g_k(a_1, \dots, a_n) = 0 \text{ if and only if} \\ a_1 = f_1(b_1, \dots, b_m), \dots, a_n = f_n(b_1, \dots, b_m) \text{ for some } b_1, \dots, b_m.$$

### Implicitization algorithm

Take the pure lexicographic ordering determined by  $t_1 \succ \dots \succ t_m \succ x_1 \succ \dots \succ x_n$  and compute the Gröbner basis  $GB$  of  $\{x_1 - f_1(t_1, \dots, t_m), \dots, x_n - f_n(t_1, \dots, t_m)\}$ . Then  $\{g_1, \dots, g_k\} = \{g \in GB \mid g \text{ is a polynomial in } x_1, \dots, x_n \text{ only}\}$ .

An example: Enneper's minimal surface defined by

$$x = \frac{1}{2}s - \frac{1}{6}s^3 + \frac{1}{2}st^2, y = -\frac{1}{2}t + \frac{1}{6}t^3 - \frac{1}{2}s^2t, z = \frac{1}{2}s^2 - \frac{1}{2}t^2.$$

The graph of this surface is shown in Figure 20.9.

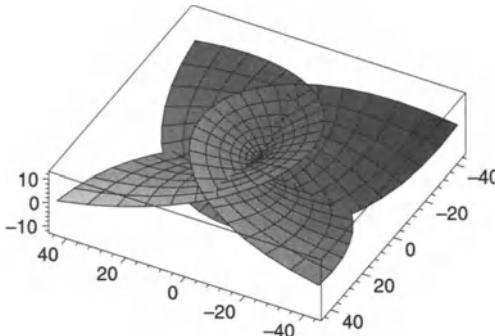


Figure 20.9. Enneper's minimal surface.

```
> polys := [x-s/2+s^3/6-s*t^2/2, y+t/2-t^3/6+t*s^2/2,
>           z-s^2/2+t^2/2];
polys := [x -  $\frac{1}{2}s + \frac{1}{6}s^3 - \frac{1}{2}st^2$ , y +  $\frac{1}{2}t - \frac{1}{6}t^3 + \frac{1}{2}ts^2$ , z -  $\frac{s^2}{2} + \frac{t^2}{2}$ ]
```

Next we compute the Gröbner basis with respect to the pure lexicographic ordering defined by  $s \succ t \succ x \succ y \succ z$ . We do not show the result here but after quite some computing time a Gröbner basis is obtained that contains one polynomial in the coordinates  $x, y, z$  only. This is the one shown below.

```
> GB := gbasis(polys, plex(s,t,x,y,z));
> collect(GB[1], z, factor);
```

$$\begin{aligned} & -1024z^9 + 4608z^7 + 3456(x-y)(x+y)z^6 \\ & + (15552y^2 + 15552x^2 - 5184)z^5 + 12960(x-y)(x+y)z^4 \\ & + (4860y^4 + 25272y^2x^2 - 3888x^2 + 4860x^4 - 3888y^2)z^3 \end{aligned}$$

$$+ 8748 (x - y) (x + y) (x^2 + y^2) z^2 - 729 (x - y)^2 (x + y)^2 z \\ + 1458 (x - y)^3 (x + y)^3$$

### 20.4.11 Invertibility of Polynomial Mappings

The Jacobian conjecture states that a polynomial mapping has an inverse which is itself a polynomial mapping if and only if the determinant of the jacobian of the mapping is nonzero. In an attempt of proving the conjecture the following criterion of invertibility has been found [76]:

*Let  $f_1, \dots, f_n$  be the coordinate functions of a polynomial mapping in the variables  $x_1, \dots, x_n$ . Let  $y_1, \dots, y_n$  be new indeterminates and let  $\prec$  be an admissible ordering such that  $y_1 \prec y_2 \prec \dots \prec y_n \prec x_1 \prec x_2 \prec \dots \prec x_n$ . Then the mapping is invertible if and only if the Gröbner basis of  $y_1 - f_1, y_2 - f_2, \dots, y_n - f_n$  has the form  $x_1 - g_1, x_2 - g_2, \dots, x_n - g_n$ , where  $g_1, g_2, \dots, g_n$  are the coordinate functions of the inverse mapping.*

We take the same example as in §1.4

```
(x,y,z)  →  (x^4 + 2(y+z)x^3 + (y+z)^2x^2 + (y+1)x + y^2 + yz,
              x^3 + (y+z)x^2 + y, x + y + z).
> F := [x^4+2*(y+z)*x^3+(y+z)^2*x^2+(y+1)*x+y^2+y*z,
>        x^3+(y+z)*x^2+y, x+y+z];
F := [x^4 + 2 (y + z) x^3 + (y + z)^2 x^2 + (y + 1) x
      + y^2 + y z, x^3 + (y + z) x^2 + y, x + y + z]
> gbasis([X-F[1], Y-F[2], Z-F[3]], plex(x,y,z,X,Y,Z));
[X - Z X^2 + 2 Z^2 X Y - Z^3 Y^2 + X - Z + z - Z Y,
 - Y + y + Z X^2 - 2 Z^2 X Y + Z^3 Y^2, x - X + Z Y]
```

So, the mapping has inverse

```
(X,Y,Z)  →  (X - ZY, Y - X^2Z + 2XYZ^2 - Y^2Z^3,
              - X - Y + Z + YZ + X^2Z - 2XYZ^2 + Y^2Z^3).
```

It turns out that verification of the result by composition of mappings takes more time then the Gröbner basis computation of the inverse mapping.

### 20.4.12 Simplification of Expressions

For a reduced monic Gröbner basis, the normal form of a polynomial is unique: it is a “canonical form” in the sense that two polynomials are equivalent when their normal forms are equal. This can be used to simplify mathematical expressions with respect to polynomial relations. In §14.7 you can find an example of simplification with respect to polynomial side

relation, originating from the Dutch Mathematics Olympiad of 1991. Here we shall show how to rewrite a symmetric polynomial in three variables in terms of elementary symmetric polynomials.

The elementary symmetric polynomials in  $x, y, z$  are defined by  $\sigma_1 = x + y + z$ ,  $\sigma_2 = xy + yz + zx$ , and  $\sigma_3 = xyz$ . Rewrite the symmetric polynomial  $f = xy^2 + xz^2 + yx^2 + yz^2 + zx^2 + zy^2$  in terms of the  $\sigma_1, \sigma_2, \sigma_3$ .

```
> alias(sigma[1]=s1, sigma[2]=s2, sigma[3]=s3):
> siderels := [s1=x+y+z, s2=x*y+y*z+z*x, s3=x*y*z];
siderels := [ $\sigma_1 = x + y + z$ ,  $\sigma_2 = xy + yz + zx$ ,  $\sigma_3 = xyz$ ]
> polys := map(lhs-rhs, siderels);
polys := [ $\sigma_1 - x - y - z$ ,  $\sigma_2 - xy - yz - zx$ ,  $\sigma_3 - xyz$ ]
```

Next, we compute the Gröbner basis of these polynomials with respect to the pure lexicographic ordering  $x \succ y \succ z \succ \sigma_1 \succ \sigma_2 \succ \sigma_3$ .

```
> G := gbasis(polys, plex(x,y,z,s1,s2,s3));
G := [ $z\sigma_2 - \sigma_3 - z^2\sigma_1 + z^3$ ,  $\sigma_2 - y\sigma_1 + y^2 - z\sigma_1 + yz + z^2$ ,
        $- \sigma_1 + x + y + z$ ]
```

The normal form of  $f$  indeed turns out to be a polynomial in  $\sigma_1, \sigma_2, \sigma_3$ .

```
> f := x*y^2+x*z^2+y*x^2+y*z^2+z*x^2+z*y^2;
f :=  $y^2x + z^2x + yx^2 + yz^2 + zx^2 + y^2z$ 
> normalf(f, G, plex(x,y,z,s1,s2,s3));
 $\sigma_1\sigma_2 - 3\sigma_3$ 
```

In Maple, this method is actually carried out when simplification with respect to polynomial side relations is requested.

```
> simplify(f, siderels, [x,y,z,s1,s2,s3]);
 $\sigma_1\sigma_2 - 3\sigma_3$ 
```

### 20.4.13 Working over General Algebras

The **Groebner** package supports various ground fields for doing polynomial calculus. The general scheme of work is:

- define the algebra  $A$  of polynomials together with its ground field;
- define the term ordering  $T$  on the algebra  $A$ ;
- do the Gröbner basis calculation with respect to the chosen term ordering.

We shall illustrate this scheme by two examples, one with ground field  $\mathbb{Z}_2$  and another with the algebra  $A = \mathbb{Q}(\sqrt{2}, w)[x, y, z]$ . We consider the following polynomials, which have been extensively studied in the second section of this chapter over the field of rational numbers .

```
> polys := [x-y-z, x+y-z^2, x^2+y^2-1];
polys := [x - y - z, x + y - z2, x2 + y2 - 1]
```

Its Gröbner basis with respect to the pure lexicographic ordering with  $x \succ y \succ z$  is

```
> gbasis(polys, plex(x,y,z));
[-2 + z4 + z2, 2y + z - z2, 2x - z2 - z]
```

For computations over the finite field  $\mathbb{Z}_p$ , for some prime number  $p$ , you must declare the corresponding algebra of polynomials. For the ground field  $\mathbb{Z}_2$ , you do this in the following way.

```
> with(Ore_algebra): # load the package
> A := poly_algebra(x,y,z,characteristic=2):
> polys := A[normalizer](polys);
```

```
polys := [x + y + z, z2 + x + y, 1 + x2 + y2]
```

Next, define the term ordering and compute the Gröbner basis.

```
> T := termorder(A, plex(x,y,z)):
> gbasis(polys, T);
```

```
[z + 1, x + y + 1]
```

The common zeros  $(0, 1, 1)$  and  $(1, 0, 1)$  of the polynomials over  $\mathbb{Z}_2$  can easily be recognized. The **msolve** procedure would have found the same result.

```
> msolve(convert(polys, set), 2);
{z = 1, x = 0, y = 1}, {z = 1, y = 0, x = 1}
```

As second example, we change the system of polynomials a bit and introduce the radical  $\sqrt{2}$  and a parameter  $w$ .

```
> polys := [x-y-sqrt(2)*z, x+w*y-z^2, x^2+y^2-w];
polys := [x - y - √2z, x + w y - z2, x2 + y2 - w]
```

The Gröbner basis for the algebra of polynomials in the unknowns  $w, x, y, z$  over the algebraic field extension  $\mathbb{Q}(\sqrt{2})$  can be computed in the following way.

```
> gbasis(polys, plex(w,x,y,z));
[2y3 + 2y2√2z + 2yz2 + y - z2 + √2z, x - y - √2z,
w - 2y2 - 2y√2z - 2z2]
```

In case you want to consider  $w$  as a parameter, i.e., compute in the algebra  $A = \mathbb{Q}(\sqrt{2}, w)[x, y, z]$ , you may try the following shortcut in which Maple guesses your intentions.

```
> gb := gbasis(polys, plex(x,y,z));
gb := [2z^4 - 2sqrt(2)z^3 + 2wsqrt(2)z^3 + 2z^2w^2 + 2z^2 - w - 2w^2 - w^3,
y + sqrt(2)z + wy - z^2, x + xw - z^2 - wsqrt(2)z]
```

It is in general more safe to use the following long format. First, introduce the radical number as a parameter  $r$  satisfying the relation  $r^2 = 2$  and inform Maple that the unknown  $w$  is used as a rational indeterminate.

```
> polys := [x-y-r*z, x+w*y-z^2, x^2+y^2-w];
polys := [x - y - r z, x + w y - z^2, x^2 + y^2 - w]
> A := poly_algebra(w,x,y,z,r, alg_relations={r^2=2},
> rational=w):
```

Next, compute the Gröbner basis over the previously defined algebra in the pure lexicographic term ordering.

```
> T := termorder(A, plex(x,y,z));
> gbasis(polys, T);
```

```
[-w - 2w^2 - w^3 + 2z^4 - 2z^3r + 2z^3wr + 2z^2w^2 + 2z^2,
wy - z^2 + y + rz, -z^2 + x + xw - wrz]
```

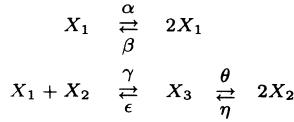
Finally, you can replace the parameter  $r$  by the radical number  $\sqrt{2}$ . The result is the same as the one computed before.

```
> eval(% , r=sqrt(2));
[2z^4 - 2sqrt(2)z^3 + 2wsqrt(2)z^3 + 2z^2w^2 + 2z^2 - w - 2w^2 - w^3,
y + sqrt(2)z + wy - z^2, x + xw - z^2 - wsqrt(2)z]
> % - gb;
[0, 0, 0]
```

## 20.5 Exercises

- Let  $f = x^2y^2 - w^2 + w$ ,  $f_1 = x - y^2w$ ,  $f_2 = y - zw$ ,  $f_3 = z - w^3$ , and  $f_4 = w^3 - w$ . Compute the normal form of  $f$  with respect to the pure lexicographic ordering with  $x \succ y \succ z \succ w$ . What is the result with respect to the pure lexicographic ordering with  $w \succ z \succ y \succ x$ ?
- Find the common zeros of the polynomials  $xyz - w$ ,  $yzw - x$ ,  $zwx - y$ , and  $wxy - z$  using the Gröbner basis method.

3. Consider the following chemical reaction mechanism

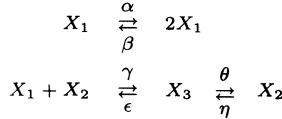


whose steady state is described by the polynomial equations

$$\begin{aligned} 0 &= -\beta c_1^2 - \gamma c_1 c_2 + \alpha c_1 + \epsilon c_3 \\ 0 &= -\gamma c_1 c_2 - 2\eta c_2^2 + \epsilon c_3 + 2\theta c_3 \\ 0 &= \gamma c_1 c_2 + \eta c_2^2 - \epsilon c_3 - \theta c_3 \end{aligned}$$

Show that this chemical reaction system has one positive steady state.

A small change in the reaction mechanism has a big effect: the steady states of



are described by the polynomial equations

$$\begin{aligned} 0 &= -\beta c_1^2 - \gamma c_1 c_2 + \alpha c_1 + \epsilon c_3 \\ 0 &= -\gamma c_1 c_2 - \eta c_2 + \epsilon c_3 + 2\theta c_3 \\ 0 &= \gamma c_1 c_2 + \eta c_2 - \epsilon c_3 - \theta c_3 \end{aligned}$$

Show that this reaction mechanism has a one-dimensional solution space of positive steady states.

4. Answer the following questions first without actually solving the system of polynomial equations. Is the given system of equations is solvable? Has it a finite number of solutions? If so, determine the number of solutions (counting with multiplicities). If not, determine the dimension of the solution space. Verify the answers by explicitly solving the systems of equations.
  - (i)  $[x^2 - 2x + 5, xy^2 + y * z^3, 3y^2 - 8z^3]$
  - (ii)  $[x^2 z^2 + x^3, xz^4 + 2x^2 z^2 + x^3, y^2 z - 2yz^2 + z^3]$
5. The method of Langrange multipliers tells that if a critical point (maximum or minimum) of a function  $f(x_1, \dots, x_n)$  subject to the condition  $g(x_1, \dots, x_n)$  exists, it is attained at a point where  $\frac{\partial}{\partial x_i}(f + \lambda g) = 0$  for some  $\lambda$ . Use this method to find the radius of the largest sphere inscribable in the ellipsoid  $x^2 + 2y^2 + 3z^2 = 4$ . What is the radius of the smallest sphere containing the ellipsoid?
6. Show that the graph known as the tetrahedron is 4-colorable, but not 3-colorable. Show that the dodecahedron is 3-colorable, but not 2-colorable.
7. The tangent surface of the twisted cubic in  $\mathbb{C}^3$  is given by the parameterization  $((s+t, s(s+2t), s^2(s+3t)))$ . Prove that it is also given by the equation  $-4x^3z + 3x^2y^2 - 4y^3 + 6xyz - z^2 = 0$ . Does the same result hold in  $\mathbb{R}^3$ ?

# References

- [1] S.A. Abramov and K.Uy. Kvashenko. Fast Algorithms to Search for the Rational Solutions of Linear Differential Equations with Polynomial Coefficients. In S.M. Watt, editor, *Proceedings of ISSAC '91*, pages 267–270. ACM Press, 1991.
- [2] A. Abramowitz and I. Stegun. *Handbook of Mathematical Functions*. Dover Publishers, 1970.
- [3] W.W. Adams and P. Loustaunau. *An Introduction to Gröbner Bases*. Graduate Studies in Mathematics, Vol. 3, Oxford University Press, 1994.
- [4] J.M. Aguirregabiria, A. Hernández, and M. Rivas. Are we careful enough when using computer algebra? *Computers in Physics*, 8:56–61, 1994.
- [5] I.A. Ajwa, Z. Liu, and P. Wang. Gröbner Bases Algorithm. Technical report, ICM Technical Reports (ICM-199502-00), Kent State University, Kent, Ohio, 1995.
- [6] A.G. Akritas. *Elements of Computer Algebra*. John Wiley & Sons, 1989.
- [7] A.E. Albert. *Regression and the Moore-Penrose Pseudoinverse*. Academic Press, 1972.
- [8] C.M. Andersen and J.F. Geer. Power Series Expansions for the Frequency and Period of the Limit Cycle of the van der Pol Equation. *SIAM J. Appl. Math.*, 42:678–693, 1982.
- [9] A. Armando and C. Ballarin. Maple’s Evaluation Process as Constraint Contextual Rewriting. In B. Mourrain, editor, *Proceedings of ISSAC 2001*, pages 32–37. ACM Press, 2001.
- [10] T. Banchoff. Differential Geometry and Computer Graphics. In W. Jäger, J. Moser and R. Remmert, editors, *Perspectives in Mathematics*, pages 43–60. Birkhäuser Verlag, 1984.
- [11] D. Barton and J.P. Fitch. Applications of Algebraic Manipulation Programs in Physics. *Rep. Prog. Phys.*, 35:235–314, 1972.
- [12] D. Barton and J.P. Fitch. CAMAL: the Cambridge Algebra System. *SIGSAM Bull.*, 8(3):17–23, 1974.
- [13] D. Barton and R. Zippel. Polynomial Decomposition Algorithms. *J. Symbolic Computation*, 1(2):159–168, 1985.
- [14] D. Barton, I.M. Willers, and R.V.M. Zahar. Taylor Series Methods for Ordinary Differential Equations — An Evaluation. In J.R. Rice, editor, *Mathematical Software*, pages 369–389. Academic Press, 1972.

- [15] E. Becker, M.G. Marinari, T. Mora, and C. Traverso. The shape of the Shape Lemma. In J. von zur Gathen and M. Giesbrecht, editors, *Proceedings of ISSAC '94*, pages 129–133. ACM Press, 1994.
- [16] T. Becker, V. Weispfenning, and H. Kredel. *Gröbner Bases*. Springer Verlag, 1993.
- [17] B. Beckerman and G. Labahn. A fast and numerically stable Euclidean-like algorithm for detecting relatively prime numerical polynomials. *J. Symbolic Computation*, 26(6):691–714, 1998.
- [18] A.I. Beltzer. *Engineering Analysis*. Academic Press, 1995.
- [19] A. Berdnikov. Private communications. RIACA, 1994.
- [20] F. Bergeron. Surprising Mathematics Using a Computer Algebra System. *J. Symbolic Computation*, 15(3):365–370, 1993.
- [21] E.R. Berlekamp. Factoring Polynomials over Large Finite Fields. *Math. Comp.*, 24:713–715, 1970.
- [22] F. v.d. Blij. Rekenen met ellebogen: computer algebra in dienst van de meetkunde (in Dutch). In A.W. Grootendorst, editor, *Vacantiecursus 1994*, pages 50–76. CWI Syllabus 36, 1994.
- [23] W. Bosma and J. Cannon. *Handbook of Magma Functions*. Univ. of Sydney, 1995.
- [24] W. Bosma, J. Cannon, C. Playoust, and A. Steel. *Solving Problems with Magma*. Univ. of Sydney, 199b.
- [25] W. Bosma, J. Cannon, and C. Playoust. The Magma Algebra System I, The User Language. *J. Symbolic Computation*, 24(3):235–265, 1997.
- [26] A. Boyle and B.F. Caviness. Future Directions for Research in Symbolic Computation. SIAM Reports on Issues in the Mathematical Sciences, 1990.
- [27] M. Bronstein, J.H. Davenport, and B.M. Trager. Symbolic Integration is Algorithmic. Tutorial, Computers and Mathematics 1989, MIT, 1989.
- [28] M. Bronstein. Integration of Elementary Functions. *J. Symbolic Computation*, 9(2):117–174, 1990.
- [29] M. Bronstein. Linear Ordinary Differential Equations: breaking through the order 2 barrier. In P. Wang, editor, *Proceedings of ISSAC '92*, pages 42–48. ACM Press, 1992.
- [30] M. Bronstein and B. Salvy. Full Partial Fraction Decomposition of Rational Functions. In M. Bronstein, editor, *Proceedings of ISSAC '93*, pages 157–160. ACM Press, 1993.
- [31] M. Bronstein. *Symbolic Integration I – Transcedental Functions*. Springer Verlag, 1997.
- [32] D.A. Brown. *Quantum Chemistry*. Penguin Books, 1972.
- [33] C. de Bruijn. De structurele identificeerbaarheid en het schatbaar zijn van de modelparameters van het compartimentele model voor de verspreiding van cadmium in het menselijk lichaam. Technical report, RIVM (in Dutch), 1990.
- [34] B. Buchberger. *An Algorithm for Finding a Basis for a Residue Class Ring of a Zero-Dimensional Polynomial Ideal*. PhD thesis, Univ. of Innsbruck, Innsbruck, Austria, 1965.

- [35] B. Buchberger. Gröbner Bases: An Algorithmic Method in Polynomial Ideal Theory. In N.K. Bose, editor, *Progress, Directions and Open Problems in Multidimensional Systems Theory*, pages 184–232. Reidel Publishing Company, 1985.
- [36] B. Buchberger. Applications of Gröbner Bases in Non-linear Computational Geometry. In J.R. Rice, editor, *Mathematical Aspects of Scientific Software*, IMA Volume in Mathematics and its Applications 14, pages 59–87. Springer Verlag, 1988.
- [37] B. Buchberger and F. Winkler (Eds.). *Gröbner Bases and Applications*. London Mathematical Society Lecture Note Series 251. Cambridge University Press, 1998.
- [38] R.L. Burden and J.D. Faires. *Numerical Analysis*. Brooks/Cole Publishing Company, sixth edition, 1997.
- [39] G. Butler and J. Cannon. The Design of Cayley - A Language for Modern Algebra. In A. Miola, editor, *Design and Implementation of Symbolic Computation Systems*, Lecture Notes in Computer Science 429, pages 10–19. Springer Verlag, 1990.
- [40] P.F. Byrd and M.D. Friedman. *Handbook of Elliptic Integrals for Engineers and Physicists*, volume LXVII of *Die Grundlagen der Mathematischen Wissenschaften*. Springer Verlag, 1971.
- [41] J. Calmet and J.A. van Hulzen. Computer Algebra Applications. In B. Buchberger, G.E. Collins and R. Loos, editors, *Computer Algebra — Symbolic and Algebraic Computation*, pages 245–258. Springer Verlag, 1983.
- [42] J. Cannon and W. Bosma. *Cayley Quick Reference Guide*. Univ. of Sydney, 1991.
- [43] J. Cannon and C. Playoust. *An Introduction to Algebraic Programming in Magma*. Univ. of Sydney, 1996.
- [44] D.G. Cantor and H. Zassenhaus. A New Algorithm over Large Finite Fields. *Math. Comp.*, 36:587–592, 1981.
- [45] A. Capanni, G. Niesi, and L. Robbiano. *CoCoA, a system for doing Computations in Commutative Algebra*. Available via anonymous ftp from: [cocoa.dima.unige.it](http://cocoa.dima.unige.it).
- [46] J. Carminati, J.S. Devitt, and G.J. Fee. Isogroup of Differential Equations Using Algebraic Computing. *J. Symbolic Computation*, 14(1):103–120, 1992.
- [47] J. Carminati and K. Vu. Symbolic Computation and Differential Equations: Lie Symmetries. *J. Symbolic Computation*, 29(1):95–116, 2000.
- [48] B.W. Char, K.O. Geddes, W.M. Gentleman, and G.H. Gonnet. The Design of Maple: A Compact, Portable, and Powerful Computer Algebra System. In J.A. van Hulzen, editor, *Computer Algebra — (Proceedings of EUROS-CAL '83)*, Lecture Notes in Computer Science 162, pages 101–115. Springer Verlag, 1983.
- [49] B.W. Char, K.O. Geddes, and G.H. Gonnet. GCDHEU: Heuristic GCD Algorithm Based on Integer GCD Computation. *J. Symbolic Computation*, 7(1):31–48, 1989.

- [50] E.S. Cheb-Terrab, L.G.S. Duarte and L.A.C.P. da Mota. Computer Algebra Solving of First Order ODEs Using Symmetry Methods. *Computer Physics Communications*, 101:254–267, 1997.
- [51] E.S. Cheb-Terrab, L.G.S. Duarte and L.A.C.P. da Mota. Computer Algebra Solving of Second Order ODEs Using Symmetry Methods. *Computer Physics Communications*, 108:90–112, 1998.
- [52] E.S. Cheb-Terrab and A.D. Roche. Symmetries and First Order ODE Patterns. *Computer Physics Communications*, 113:239–260, 1998.
- [53] E.S. Cheb-Terrab and A.D. Roche. Integrating Factors for Second Order ODEs. *J. Symbolic Computation*, 27(5):501–519, 1999.
- [54] E.S. Cheb-Terrab and T. Kolokolnikov. First order ODEs, Symmetries and Linear Transformations. submitted to *European Journal of Applied Mathematics*; available at e-print archive arXiv:math-ph/0007023, July 2000.
- [55] P.L. Chebyshev. Sur l'intégration des différentielles qui contiennent une racine carrée d'un polynôme du troisième ou du quatrième degré. In *Oeuvres de P.L. Tchebychef*, volume I, pages 171–200. Chelsea, 1957.
- [56] S.M. Christensen. Resources for Computer Algebra. *Computers in Physics*, 8:308–315, 1994.
- [57] A.M. Cohen. Computer Algebra, Theory and Practice. *Nieuw Arch. voor Wisk.*, IV(7):215–230, 1989.
- [58] A.M. Cohen and G.C.M. Ruitenburg. Generating Functions and Lie Groups. In A.M. Cohen, editor, *Computational Aspects of Lie Group Representations and Related Topics*, CWI Tract 84, pages 19–28. CWI, 1991.
- [59] A.M. Cohen, R.L. Griess Jr., and B. Lisser. The Group  $L(2,61)$  embeds in the Lie Group of type  $E_8$ . *Comm. Algebra*, 21:1889–1907, 1993.
- [60] A.M. Cohen, J.H. Davenport, and A.J.P. Heck. An Overview of Computer Algebra. In *Computer Algebra for Industry: Problem Solving in Practice*, pages 1–52. John Wiley & Sons, 1993.
- [61] A.M. Cohen. Gröbner Bases, an Introduction. In A.M. Cohen, H. Cuypers, and H. Sterk, editors, *Some Tapas of Computer Algebra. Algorithms and Computation in mathematics*, Vol. 4, Springer Verlag, 1998.
- [62] J.W. Cooley and J.W. Tukey. An Algorithm for the Machine Calculation of Complex Fourier Series. *Math. Computation*, 19:297–301, 1965.
- [63] R.M. Corless, G.H. Gonnet, D.E.G. Hare, and D.J. Jeffrey. On Lambert's W Function. Preprint in Maple Share Library, 1993.
- [64] R.M. Corless. Simplification and the Assume Facility. *Maple Technical Newsletter*, 1(1):24–31, 1994.
- [65] R.M. Corless and K. El-Sawy. Solution of Banded Linear Systems in Maple Using LU Factorization. In R.J. Lopez, editor, *Maple V: Mathematics and Its Applications*, pages 219–227. Birkhäuser Verlag, 1994.
- [66] D. Cox, J. Little, and D. O'Shea. *Ideals, Varieties, and Algorithms*. Springer Verlag, 1992.

- [67] M. Daberkow et al. KANT V4. *J. Symbolic Computation*, 24(3–4):267–283, 1997.
- [68] J.H. Davenport. *On the Integration of Algebraic Functions*. Lecture Notes in Computer Science 102. Springer Verlag, 1981.
- [69] J.H. Davenport, Y. Siret, and E. Tournier. *Computer algebra: systems and algorithms for algebraic computation*. Academic Press, 1988.
- [70] J.H. Davenport and B.M. Trager. Scratchpad's View of Algebra I: Basic Commutative Algebra. In A. Miola, editor, *Design and Implementation of Symbolic Computation Systems*, Lecture Notes in Computer Science 429, pages 40–54. Springer Verlag, 1990.
- [71] J.H. Davenport. The Axiom System. In *Proceedings of NAGUA '91*. NAG Ltd., 1991.
- [72] J. Dieudonné. *Treatise on Analysis*, volume 10-III of *Pure and Applied Mathematics*. Academic Press, 1972.
- [73] A. Dingle and R. Fateman. Branch Cuts in Computer Algebra. In J. von zur Gathen and M. Giesbrecht, editors, *Proceedings of ISSAC '94*, pages 250–257. ACM Press, 1994.
- [74] R.H. Enns and G.C. McGuire. *Nonlinear Physics with Maple*. Birkhäuser Verlag, 1997.
- [75] W.H. Enright. The Relative Efficiency of Alternative Defect Control Schemes for High Order Continuous Runge-Kutta Formulas. Technical Report 252/91, University of Toronto, Dept. of Computer Science, June 1991.
- [76] A. v.d. Essen. *Polynomial Automorphisms and the Jacobian Conjecture*. Progress in Mathematics, Vol. 190, Birkhäuser Verlag, 2000.
- [77] R.J. Fateman. Advances and Trends in the Design and Construction of Algebraic Manipulation Systems. In S. Watanabe and M. Nagata, editor, *Proceedings of ISSAC '90*, pages 60–67. ACM Press, 1990.
- [78] J.C. Faugère, P. Gianni, D. Lazard, and T. Mora. Efficient Computation of Zero-Dimensional Gröbner Bases by Change of Ordering. *J. Symbolic Computation*, 16(4):329–344, 1993.
- [79] E. Fehlberg. Klassische Runge-Kutta Formeln vierter und niedriger Ordnung mit Schrittweiten Kontrolle und ihre Anwendungen in Wärmeleitungsprobleme. *Computing*, 6:61–71, 1970.
- [80] M. Feinberg. Reaction Network Structure, Multiple Steady States, and Sustained Composition Oscillations. In *Modelling of Chemical Reaction Systems*, Springer Series in Chemical Physics 18, pages 56–68. Springer Verlag, 1981.
- [81] R.P. Feynman, R.B. Leighton, and M. Sands. *The Feynman Lectures on Physics*, volume II. Addison-Wesley, 1964. fourth printing, pp. 22-8.
- [82] J. Fitch. Solving Algebraic Problems with REDUCE. *J. Symbolic Computation*, 1(2):211–228, 1985.
- [83] K. Forsman and T. Glad. Constructive Algebraic Geometry in Nonlinear Control. In *Proc. 29th CDC*, volume 5, pages 2825–2827, Honolulu, Hawaii, 1990. IEEE CSS.

- [84] I. Foster and S. Taylor. *Strand — New Concepts in Parallel Programming*. Prentice-Hall, 1989.
- [85] R. Fröberg. *An Introduction to Gröbner Bases*. John Wiley & Sons, 1997.
- [86] I. Frick. *SHEEP User's Manual*. Univ. of Stockholm, 1977.
- [87] F.N. Fritsch, R.E. Shafer, and W.P. Crowley. Solution of the Transcendental Equation  $we^w = \chi$ . *Comm. ACM.*, 16:123–124, 1973.
- [88] B. Fuchssteiner et al. *MuPAD User's Manual; MuPAD Version 1.2.2*. Wiley–Teubner, 1996.
- [89] W. Gander, J. Hřebíček and S. Bartoň. The Tractrix and Similar Curves. In W. Gander and J. Hřebíček, editors, *Solving Problems in Scientific Computing Using Maple and Matlab*, pages 1–14. Springer Verlag, 1995.
- [90] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.2*, 2000. (<http://www.gap-system.org>).
- [91] J. Von zur Gathen, J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 1999.
- [92] C.W. Gear. *Numerical Initial Value Problems in Ordinary Differential Equations*. Prentice-Hall, 1971.
- [93] K.O. Geddes. Numerical Integration in a Symbolic Context. In B. Char, editor, *Proceedings of SYMSAC '86*, pages 185–191. ACM Press, 1986.
- [94] K.O. Geddes and G.H. Gonnet. A New Algorithm for Computing Symbolic Limits Using Hierarchical Series. In P. Gianni, editor, *Symbolic and Algebraic Computation, Proceedings ISSAC '88*, Lecture Notes in Computer Science 358, pages 490–495. Springer Verlag, 1989.
- [95] K.O. Geddes and T.C. Scott. Recipes for Classes of Definite Integrals Involving Exponentials and Logarithms. In E. Kaltofen and S.M. Watt, editors, *Computers and Mathematics 1989*, pages 192–201. Academic Press, 1989.
- [96] K.O. Geddes and L.Y. Stefanus. On the Risch-Norman integration method and its implementation in Maple. In G.H. Gonnet, editor, *Proceedings of ISSAC '89*, pages 212–217. ACM Press, 1989.
- [97] K.O. Geddes. Numerical Integration using Symbolic Analysis. *Maple Technical Newsletter*, 6:8–17, 1991.
- [98] K.O. Geddes, S.R. Czapor and G. Labahn. *Algorithms for Computer Algebra*. Kluwer Academic Publishers, 1992.
- [99] K.O. Geddes. A Package for Numerical Approximation. *Maple Technical Newsletter*, 10:28–36, 1993.
- [100] J. Gerhard et al. *MuPAD Tutorial*. Springer Verlag, 2000.
- [101] A. Giovini and G. Niesi. CoCoA: A User-Friendly System for Commutative Algebra. In A. Miola, editor, *Design and Implementation of Symbolic Computation Systems*, Lecture Notes in Computer Science 429, pages 20–29. Springer Verlag, 1990.
- [102] A. Giovinni, T. Mora, G. Niesi, L. Robbiano, and C. Traverso. One sugar cube, please, or selection strategies in the Buchberger algorithm. In S.M. Watt, editor. *Proceedings of ISSAC '91*, pages 49–54. ACM Press, 1991.

- [103] G.H. Gonnet, D.W. Gruntz, and L Bernardin. Computer Algebra Systems. In A. Ralston et al, editors, *Encyclopedia of Computer Science*. fourth edition, pages 287–301. Nature Publishing Group, 2000.
- [104] M.J. González-López and T. Recio. The ROMIN inverse geometric model and the dynamic evaluation method. In A.M. Cohen, editor, *Computer Algebra for Industry: Problem Solving in Practice*, pages 117–141. John Wiley & Sons, 1993.
- [105] L. Gonzalez-Lopez. Symbolic Recipes for Polynomial Systems Solving. In A.M. Cohen, H. Cuypers, and H. Sterk, editors, *Some Tapas of Computer Algebra*. Algorithms and Computation in mathematics, Vol. 4, Springer Verlag, 1998.
- [106] R.W. Gosper. Decision Procedure for Indefinite Hypergeometric Summation. *Proc. Natl. Acad. Sci. USA*, 75:40–42, 1978.
- [107] X. Gourdon and B. Salvy. Computing one million digits of  $\sqrt{2}$ . *Maple Technical Newsletter*, 10:66–71, 1993.
- [108] J. Grabmeier, E. Kaltofen, and V. Weispfenning (eds.). *Handbook of Computer Algebra: Foundation, Applications, Systems*. Springer Verlag, 2002.
- [109] I.S. Gradshteyn and I.M. Ryzhik. *Table of Integrals, Series and Products*. Academic Press, fifth edition, 1994.
- [110] D.R. Grayson and M.E. Stillman. Macaulay 2, a software system for research in algebraic geometry. <http://www.math.uiuc.edu/Macaulay2>.
- [111] G. Pfister G.-M. Greuel and H. Schönemann. SINGULAR 2.0. A Computer Algebra System for Polynomial Computations, Center for Computer Algebra, University of Kaiserslautern, 2001. <http://www.singular.uni-kl.de>.
- [112] G.-M. Greuel and G. Pfister. *A Singular Introduction to Commutative Algebra*. Springer Verlag, 2002.
- [113] A. Griewank. On Automatic Differentiation. In M. Iri and K. Tanabe, editors, *Mathematical Programming*, pages 83–107. Kluwer Academic Publishers, 1989.
- [114] A. Griewank. The Chain Rule Revisited in Scientific Computing. *SIAM News*, pages 20–21 (part I), 8–9, 24 (part II), May (part I), July (part II) 1991.
- [115] A. Griewank and G.F. Corliss. Automatic Differentiation of Algorithms: Theory, Implementation and Application. In *Proceedings in Applied Mathematics 53*. SIAM, Philadelphia, 1991.
- [116] E. Groswald. *Bessel Polynomials*. Lecture Notes in Mathematics 698. Springer Verlag, 1980.
- [117] J. Grotendorst. A Maple package for transforming series, sequences, and functions. *Comp. Phys. Comm.*, 67:325–342, 1991.
- [118] D.W. Gruntz. *On Computing Limits in a Symbolic Manipulation System*. PhD thesis, ETH 11432, Zürich, 1996.
- [119] J. Gutiérrez, T. Recio, and C. Ruiz de Velasco. Polynomial decomposition algorithm of almost quadratic complexity. In T. Mora, editor, *Proceedings*

- of AAECC-6, Lecture Notes in Computer Science 357. Springer Verlag, 1989.
- [120] J. Gutiérrez and T. Recio. A practical implementation of two rational decomposition algorithms. In P. Wang, editor, *Proceedings of ISSAC '92*, pages 152–157. ACM Press, 1992.
  - [121] J. Gutiérrez and T. Recio. Rational Function Decomposition and Gröbner Basis in the Parametrization of Plane Curves. In *Proceedings of LATIN '92, São Paulo, Brazil*, pages 239–245. Springer Verlag, 1992. Springer Lecture Notes of Comput. Sci. 583.
  - [122] D. Harper, C. Wooff, and D. Hodgkinson. *A Guide to Computer Algebra Systems*. John Wiley & Sons, 1991.
  - [123] B.K. Harrison and F.B. Estabrook. Geometric Approach to Invariance Groups and Solutions of Partial Differential Equations. *J. Math. Phys.*, 12:653–665, 1971.
  - [124] A.C. Hearn. *REDUCE User's Manual*. The Rand Corporation, Santa Monica, California, 1987.
  - [125] A.J.P. Heck. Transformation between Geocentric and Geodetic Coordinates. In A.M. Cohen, editor, *Computer Algebra for Industry: Problem Solving in Practice*, pages 203–219. John Wiley & Sons, 1993.
  - [126] A.J.P. Heck. Computer Algebra: A Tool in Identifiability Testing. In A.M. Cohen, L. van Gastel and S. Verdun Lunel, editors, *Computer Algebra for Industry 2: Problem Solving in Practice*, pages 267–289. John Wiley & Sons, 1995.
  - [127] A.J.P. Heck. Bird's-Eye View of Gröbner Bases. *Nucl. Instr. and Meth. in Phys. Res.*, A 389:16–21, 1997.
  - [128] A.J.P. Heck. FORM for Pedestrians. <http://www.nikhef.nl/~form>.
  - [129] W. Hereman. Symbolic Software for Lie Symmetry Analysis. In N.H. Ibragimov, editor, *CRC Handbook of Lie Group Analysis of Differential Equations, Vol. 3: New Trends in Theoretical Developments and Computational Methods, chap. 13*. CRC Press, 1995.
  - [130] A.C. Hindmarsh. ODEPACK: a Systemized Collection of ODE Solvers. In R. Stepleman, editor, *Numerical Methods for Scientific Computation*. North-Holland, 1983.
  - [131] M. van Hoeij. Factoring polynomials and the knapsack problem. accepted for publication in *Journal of Number Theory*, 2001.
  - [132] C. Hollinger and P. Serf. SIMATH - a computer algebra system. In A. Pethö, M. Pohst, H. Williams and H. Zimmer, editors, *Computational Number Theory*, pages 331–342. de Gruyter, 1991.
  - [133] L. Hornfeldt. *STENSOR Reference Manual*. Univ. of Stockholm, 1988.
  - [134] E. Horowitz. Algorithms for Partial Fraction Decomposition and Rational Integration. In S.R. Petrick, editor, *Proceedings of SYMSAM '71*, pages 441–457. ACM Press, 1971.
  - [135] J.A. van Hulzen and J. Calmet. Computer Algebra Systems. In B. Buchberger, G.E. Collins and R. Loos, editors, *Computer Algebra — Symbolic and Algebraic Computation*, pages 221–244. Springer Verlag, 1983.

- [136] P.E. Hydon. *Symmetry Methods for Differential Equations: A Beginner's Guide*. Cambridge Texts in Applied Mathematics. Cambridge University Press, 2000.
- [137] N.H. Ibragimov. Sophus Lie and Harmony in Mathematical Physics, on the 150th Anniversary of his Birth. *The Mathematical Intelligencer*, 16:20–28, 1994.
- [138] A. Ivic. *The Riemann Zeta Function*. John Wiley & Sons, 1985.
- [139] R.D. Jenks, R.S. Sutor, S.M. Watt. Scratchpad II: An abstract Datatype System for Mathematical Computing. In J.R. Rice, editor, *Mathematical Aspects of Scientific Software*, IMA Volume in Mathematics and its Applications 14, pages 157–182. Springer Verlag, 1988.
- [140] Richard D. Jenks and Robert S. Sutor. *AXIOM, The Scientific Computation System*. Springer Verlag, 1992.
- [141] W. Kahan. Branch Cuts for Complex Elementary Functions or Much Ado About Nothing's Sign Bit. In A. Iserles and M.J.D. Powell, editors, *The State of the Art in Numerical Analysis*, pages 65–212. Clarendon Press, 1987.
- [142] E. Kaltofen. Polynomial Factorization. In B. Buchberger, G. Collins and R. Loos, editors, *Computer Algebra — Symbolic and Algebraic Computation*, pages 95–114. Springer Verlag, 1983.
- [143] E. Kaltofen. Sparse Hensel Lifting. In B.F. Caviness, editor, *Proceedings of EUROCAL '85*, Lecture Notes in Computer Science 204, Vol. 2, pages 4–17. Springer Verlag, 1985.
- [144] E. Kaltofen. Polynomial Factorization 1982–1986. In D.V. Chudnovsky and R.D. Jenks, editors, *Computers & Mathematics*, Lecture Notes in Pure and Applied Mathematics 125, pages 285–309. Marcel Dekker, 1990.
- [145] E. Kaltofen. Polynomial Factorization 1987–1991. In *Proceedings of LATIN '92, São Paulo, Brazil*, pages 294–313. Springer Verlag, 1992. Springer Lecture Notes of Comput. Sci. 583.
- [146] M. Klerer and F. Grossman. Error Rates in Tables of Indefinite Integrals. *Industrial Mathematics*, 18. 1968.
- [147] E. Kamke. *Differentialgleichungen: Lösungsmethoden und Lösungen*. Clessea Publishing Co, New York, 1959.
- [148] D.E. Knuth. *The Art of Computer Programming, Vol. II, Seminumerical Algorithms*. Addison-Wesley, second edition, 1981.
- [149] D.E. Knuth. Problem E3335. *Amer. Math. Monthly*, 96:525, 1989.
- [150] H.-P. Ko. Geometry Theorem Proving by Decomposition of Quasi-Algebraic Sets: An Application of the Ritt-Wu Principle. In D.Kapur and J.L. Mundy, editors, *Geometrical Reasoning*, pages 95–122. MIT Press, 1989.
- [151] W. Koepf. Algorithms for the Indefinite and Definite Summation. Technical Report, Preprint SC 94-33, Konrad-Zuse-Zentrum Berlin (ZIB), December 1994.
- [152] W. Koepf. Summation in Maple. *MapleTech*, 3(2):26–33, 1996.

- [153] J. Kovacic. An algorithm for solving second order homogeneous differential equations. *J. Symbolic Computation*, 2(1):3–43, 1986.
- [154] A. Kovács. Computer Algebra: Impact and Perspectives. *Nieuw Archief voor Wiskunde*, 17(1):29–56, 1999.
- [155] D. Kraft. Modeling and Simulating Robots in Maple. *Maple Technical Newsletter*, 1(2):39–47, 1994.
- [156] M. Kreuzer and L. Robbiano. *Computational Commutative Algebra 1*. Springer Verlag, 2000.
- [157] B. Kutzler. *Introduction to Derive for Windows*. Austria, 1996.
- [158] G. Labahn. Solving Linear Differential Equations in Maple. *Maple Technical Newsletter*, 2(1):20–28, 1995.
- [159] L. Lamport. *LaTeX, A Document Preparation System*. Addison-Wesley, 1988.
- [160] D. Lazard and R. Rioboo. Integration of Rational Functions: Rational Computation of the Logarithmic Part. *J. Symbolic Computation*, 9(2):113–116, 1990.
- [161] Y. Lecourtier and A. Raksanyi. The Testing of Structural Properties Through Symbolic Computation. In E. Walter, editor, *Identifiability of Parametric Models*. Pergamon Press, 1987.
- [162] M.A. van Leeuwen, A.M. Cohen, and B. Lisser. *LiE: A Package for Lie Group Computations*. CAN Expertise Centre, 1992.
- [163] A. Lehtonen. The Klein Bottle. *The Mathematica Journal*, 1(3):65, 1991.
- [164] A.K. Lenstra. Factoring Polynomials over Algebraic Number Fields. In J.A. van Hulzen, editor, *Computer Algebra — (Proceedings of EURO-CAL '83)*, Lecture Notes in Computer Science 162, pages 245–254. Springer Verlag, 1983.
- [165] A.H.M. Levelt. Various Problems Solved by Computer Algebra. In A.M. Cohen, L. van Gastel and S. Verdun Lunel, editors, *Computer Algebra for Industry 2: Problem Solving in Practice*, pages 43–59. John Wiley & Sons, 1995.
- [166] A.H.M. Levelt. The Cycloheptane Molecule: A Challenge to Computer Algebra. In W.W. Küchlin, editor, *ISSAC '97*. ACM Press, 1997. available from [www-math.sci.kun.nl/math/medewerkers/ahml/other.htm](http://www-math.sci.kun.nl/math/medewerkers/ahml/other.htm).
- [167] B. Lisser. Kostant's Conjecture. *Maple Technical Newsletter*, pages 29–34, 1994. Special Issue: “Maple in Mathematics and the Sciences”.
- [168] R. Loos. Computing in Algebraic Extensions. In B. Buchberger, G.E. Collins and R. Loos, editors, *Computer Algebra — Symbolic and Algebraic Computation*, pages 173–187. Springer Verlag, 1983.
- [169] J. v.d. Lune, H.J.J. te Riele, and D.T. Winter. On the zeros of the Riemann zeta function. *Math. Comp.*, 46:667–681, 1986.
- [170] M. MacCallum and J. Skea. SHEEP, a computer algebra system for general relativity. In M. MacCallum, J. Skea, J. McCrea and R. McLenaghan, editors, *Algebraic Computing in General Relativity*, pages 1–172. Clarendon Press, 1994.

- [171] M.A.H. MacCallum. Using Computer Algebra to Solve Ordinary Differential Equations. In A.M. Cohen, L. van Gastel and S. Verduyn Lunel, editors, *Computer Algebra for Industry 2: Problem Solving in Practice*, pages 19–41. John Wiley & Sons, 1995.
- [172] MACSYMA User's Guide. *MACSYMA User's Guide, System Reference Manual, and Mathematics Reference Manual*. Macsyma, Inc., 1992.
- [173] *Maple 8 Learning Guide*. Toronto: Waterloo Maple Inc., 2002.
- [174] MathSource. <http://www.mathsource.com>.
- [175] H. Melenk. Solving Polynomial Equation Systemss by Gröbner Type Methods. *CWI Quarterly*, 3:121–136, 1990.
- [176] M. Mignotte. *Mathematics for Computer Algebra*. Springer Verlag, 1992.
- [177] B. Mishra. *Algorithmic Algebra*. Springer Verlag, 1993.
- [178] R. Moenck. On computing closed forms for summation. In *Proceedings MACSYMA User's Conference*, pages 225–236, 1977.
- [179] M.B. Monagan. Tips for Maple Users. *Maple Technical Newsletter*, 1(2):11–13, 1994.
- [180] M.B. Monagan et al. *Maple 8 Advanced Programming Guide*. Toronto: Waterloo Maple Inc., 2002.
- [181] M.B. Monagan et al. *Maple 8 Introductory Programming Guide*. Toronto: Waterloo Maple Inc., 2002.
- [182] P. Moon and D.E. Spencer. *Field theory handbook: including coordinate systems, differential equations and their solutions*. Springer Verlag, 1961.
- [183] J. Moses. Symbolic Integration: The Stormy Decade. *Comm. ACM.*, 14:548–560, 1971.
- [184] A.H. Nayfeh and D.T. Mook. *Nonlinear Oscillations*. John Wiley & Sons, 1979.
- [185] B. Noble and M.A. Hussain. Multiple Scaling and a Related Expansion Method, with Applications. Report BICOM 87/7, Brunel Univ., Uxbridge, England, June 1987.
- [186] V.W. Noonburg. A neural network modeled by an adaptive Lotka-Volterra system. *SIAM J. Appl. Math.*, 49:1779–1792, 1989.
- [187] A.M. Odlyzko. Analytic Computations in Number Theory. In W. Gautschi, editor, *Mathematics of Computation 1943–1993: a Half-Century of Computational Mathematics*, pages 451–463. Proceedings of Symposia in Applied Mathematics, American Mathematical Society, 1994.
- [188] G.J. Oldenborgh. An Introduction to FORM. Univ. of Leiden, <http://www.lorentz-leidenuniv.nl/form/form.html>.
- [189] P.J. Olver. *Applications of Lie Groups to Differential Equations*. Springer Verlag, 1986.
- [190] The PARI-Group. *PARI/GP, Version 2.1.1*, 2000. available from <http://www.parigp-home.de/>.
- [191] M.K. Paul. A Note on Computation of Geodetic Coordinates from Geocentric (Cartesian) Coordinates. *Bull. Géodésique*, 108:135–139, 1973.

- [192] R.P. Paul. *Robot Manipulators: Mathematics, Programming and Control*. MIT Press, 1981.
- [193] R.P. Paul. Kinematic Control Equations for Simple Manipulators. In C. Lee, R. Gonzalez and K. Fu, editors, *Tutorial on Robotics*, pages 66–72. IEEE Computer Society Press, 1983.
- [194] R. Pavelle. Problems sent to the USENET sci.math.symbolic bulletin board. Archived in the REDUCE network library, 1989.
- [195] P.P. Petrushev and V.A. Popov. *Rational Approximation of Real Functions*. Cambridge University Press, 1987.
- [196] M.E. Pohst. *Computational Algebraic Number Theory*, volume 21 of *DMV Seminar*. Birkhäuser Verlag, 1993.
- [197] P. Paule and V. Strehl. Symbolic Summation – Some Recent Developments. in J. Fleischer, J. Grabmeier, F. Hehl, and W. Küchlin, editors, *Computer Algebra in Science and Engineering – Algorithms, Systems, and Applications*. pages 138–162 World Scientific, Singapore, 1995.
- [198] A. Raksanyi, Y. Lecourtier, E. Walter, and A. Venot. Identifiability and Distinguishability Testing Via Computer Algebra. *Math. Biosciences*, 77:245–266, 1985.
- [199] R.H. Rand. *Computer Algebra in Applied Mathematics: An Introduction to MACSYMA*. Research Notes in Mathematics 94. Pitman Publishing, 1984.
- [200] R.H. Rand and D. Armbruster. *Perturbation Methods, Bifurcation Theory and Computer Algebra*. Applied Mathematical Sciences 65. Springer Verlag, 1987.
- [201] E.Ya. Remez. Sur un procédé convergent d'approximation successives pour déterminer les polynômes d'approximation. *Comptes Rendues*, 193:2063–2065, 1934.
- [202] E.Ya. Remez. Sur le calcul effectif des polynômes d'approximation de Tschebyscheff. *Comptes Rendues*, 199:337–340, 1934.
- [203] R.H. Risch. The problem of integration in finite terms. *Trans. AMS*, 139:167–189, 1969.
- [204] L. Robbiano. On the theory of graded structures. *J. Symbolic Computation*, 2(2):139–170, 1986.
- [205] M. Rothstein. *Aspects of Symbolic Integration and Simplification of Exponential and Primitive Functions*. PhD thesis, Univ. of Wisconsin, Madison, 1976.
- [206] F.W. Schwarz. Symmetries of Differential Equations: From Sophus Lie to Computer Algebra. *SIAM Review*, 30:450–481, 1988.
- [207] T.C. Scott, Y.B. Band, and K.O. Geddes. Recipes for Solving Broad Classes of Definite Integrals and Applications. *Maple Technical Newsletter*, 10:19–27, 1993.
- [208] W. Schreiner. A Parallel Computer Algebra System Based on Maple and Java. Technical Report 99-08, RISC-Linz, Johannes Kepler University, Linz, Austria, 1999.

- [209] K. Siegl. Parallelizing Algorithms for Symbolic Computation Using ||MAPLE||. Technical Report 93-08, RISC-Linz, Johannes Kepler University, Linz, Austria, 1993. Published in: 4th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming, San Diego, CA, May 19–21, 1993.
- [210] T.J. Smedley. *Fast Methods for Computation with Algebraic Numbers*. PhD thesis, Univ. of Waterloo, 1990.
- [211] B.K. Spearman and K.S. Williams. Characterization of Solvable Quintics  $x^5 + ax + b$ . *Amer. Math. Monthly*, 101:986–992, 1994.
- [212] M.R. Spiegel. *Mathematical handbook of formulas and tables*. McGraw-Hill, 1968.
- [213] H. Stephani. *Differential Equations: Their Solution Using symmetries*. Cambridge University Press, 1993.
- [214] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis*. Text in Applied Mathematics 12. Springer Verlag, second edition, 1993.
- [215] D. Stoutemyer. Crimes and Misdemeanors in the Computer Algebra Trade. *Notices of the AMS*, 38:778–785, 1991.
- [216] D. Stoutemyer. *Derive User Manual*. Soft Warehouse, Inc., Honolulu, Hawaii, 1994. seventh edition.
- [217] V. Strehl. Binomial Sums and Identities. *Maple Technical Newsletter*, 10:37–49, 1993.
- [218] H. Strubbe. Manual for SCHOONSCHIP. *Comput. Phys. Commun.*, 8:1–30, 1974.
- [219] R.S. Sutor (ed.). The Scratchpad II Computer Algebra System Interactive Environment Users Guide. Technical report, IBM Thomas J. Watson Research Center, Yorktown Heights, 1988.
- [220] F. Szabo. *Linear Algebra: An Introduction Using Maple*. Harcourt Academic Press, 2002.
- [221] T. Szkodny. Modelling of Kinematics of the IRb-6 Manipulator. *Computers Math. Applic.*, 29:77–94, 1995.
- [222] R.G. Tobey. *Algorithms for Antidifferentiation of Rational Functions*. PhD thesis, Univ. of Wisconsin, Madison, 1967.
- [223] B. Trager. Algebraic Factoring and Rational Function Integration. In R.D Jenks, editor, *Proceedings of SYMSAC '76*, pages 219–226. ACM Press, 1976.
- [224] B. Trager. *Integration of Algebraic Functions*. PhD thesis, MIT, 1984.
- [225] J.A.M. Vermaseren. *Symbolic Manipulation with FORM*. CAN Expertise Centre, 1991.
- [226] J.A.M. Vermaseren. Symbolic Heroics. *CAN Newsletter*, 11:57–58, 1993.
- [227] J.A.M. Vermaseren. New Features of FORM. arXiv.org e-Print archive, math-ph/0010025, 2000.
- [228] D. Wang. An Implementation of the Characteristic Set Method in Maple. Technical Report 91-25, RISC-Linz, 1991.

- [229] T. Weibel and G.H. Gonnet. An Algebra of Properties. In *Proceedings of ISSAC '91*, pages 352–359. ACM Press, 1991.
- [230] T. Weibel and G.H. Gonnet. An Algebra of Properties. Technical report 158, Informatik ETH-Zürich, 1991.
- [231] T. Weibel and G.H. Gonnet. An Assume facility for CAS, with a sample implementation for Maple. In J. Fitch, editor, *Design and Implementation of Symbolic Computation Systems*, Lecture Notes in Computer Science 721, pages 95–103. Springer Verlag, 1992.
- [232] V. Weispfenning. Comprehensive Gröbner Bases. *J. Symbolic Computation*, 14(1):1–29, 1993.
- [233] E.J. Weniger. Nonlinear Sequence Transformations for the Acceleration of Convergence and the Summation of Divergent Series. *Comp. Phys. Reports*, 10:189–371, 1989.
- [234] M.J. Wester (ed.). *Computer Algebra Systems: A Practical Guide*. John Wiley & Sons, 1999.
- [235] H.S. Wilf and D. Zeilberger. Rational functions certify combinatorial identities. *J. Amer. Math. Soc.*, 3:147–158, 1990.
- [236] H.S. Wilf and D. Zeilberger. Towards computerized proofs of identities. *Bull. of the Amer. Math. Soc.*, 23:77–83, 1990.
- [237] F. Winkler. Computer Algebra: Problems and Developments. In A.M. Cohen, L. van Gastel and S. Verdun Lunel, editors, *Computer Algebra for Industry 2: Problem Solving in Practice*, pages 1–18. John Wiley & Sons, 1995.
- [238] S. Wolfram. *The Mathematica Book*. Cambridge University Press, fourth edition, 1999.
- [239] C. Wooff and D. Hodgkinson. *MuMath: A Microcomputer Algebra System*. Academic Press, 1987.
- [240] P.E.S. Wormer and F. de Groot. The Potential Energy Surface of Triplet  $H_3^+$ : A Representation in Hyperspherical Coordinates. *J. Chem. Phys.*, 1989:2344, 90.
- [241] F.J. Wright. *Computing with Maple*. CRC Press, 2001.
- [242] Computer Algebra WWW servers, good starting points. SymbolicNet, [www.SymbolicNet.org/](http://www.SymbolicNet.org/); GDR MEDICIS, [medicis.polytechnique.fr/](http://medicis.polytechnique.fr/); RISC-Linz, [info.risc.uni-linz.ac.at/](http://info.risc.uni-linz.ac.at/); Computer Algebra Fachgruppe, [www.uni-karlsruhe.de/~CAIS/](http://www.uni-karlsruhe.de/~CAIS/).
- [243] S.Y. Yan. Primality Testing of Large Numbers in Maple. *Computers Math. Applic.*, 29:1–8, 1995.
- [244] R. Zippel. Rational Function Decomposition. In S.M. Watt, editor, *Proceedings of ISSAC '91*, pages 1–6. ACM Press, 1991.
- [245] R. Zippel. *Effective Polynomial Computation*. Kluwer Academic Publishers, 1993.

# Index

**Remark:** page numbers in *italics* are prime references.

- ! (factorial), 34, 362, 378
- !! (double factorial), 34
- + (sum), 34, 84
- $\wedge$  (power), 34, 84, 96, 633
- / (slash), division, 34
  - symbol in file names, 80
- \* as field length, 121
  - as product, 34, 84
  - as symbol, 116
- \*\* (power), 34, 84, 96
- \* vs. .., 631
- | (vertical bar), column separator in matrix, 622, 623
- || (double vertical bar), concatenation operator, 82, 84, 292
- $\leq$  (less than or equal), 84
- $\neq$  (unequal), 84
- = (equal), 84, 87
- > as prompt, 6, 34, 107
  - as relational operator, 84
- , (comma), row separator in matrix, 622, 623
- . (matrix multiplication operator), 307, 315, 631–634
- dot product of vectors, 632
- .. (ellipsis operator), 84, 292
- .m (file extension), internal format file extension, 107–110
- .mapleinit (initialization file), 109
- .mws (worksheet extension), Maple worksheet file extension, 106
- ; (semicolon), 6, 34, 86, 87, 96
- : (colon), 9, 34, 86, 87, 96, 98
- :: (double colon), association operator, 86, 87, 201
- :- (module member selection, 87, 102, 104, 325, 388, 611, 612, 620
- := (assignment operator), 66, 71, 87
- ? (question mark), literal, 122
- ? (help), 36
- ? (usage), 39
- ? (example), 39
- ?changecoords, 688
- ?debugger, 98
- ?dilog, 37
- ?dsolve,algorithms, 523
- ?dsolve,numeric,BVP, 577
- ?dsolve,numeric,IVP, 577
- ?dsolve,references, 523
- ?DESol, 538
- ?environment, 79

**?evalf,int**, 240  
**?help**, 40  
**?ifactor**, 37  
**?index**, 40  
**?index,expression**, 40  
**?index,function**, 40, 41  
**?index,misc**, 40, 41  
**?index,module**, 40  
**?index,packages**, 40, 41, 104  
**?index,procedure**, 40  
**?inifcns**, 49, 189  
**?interface**, 136  
**?kernelopts**, 134  
**?LinearAlgebra**, 41  
**?LinearAlgebra,Norm**,  
    **?LinearAlgebra:-Norm**,  
    **?LinearAlgebra[Norm]**, 41  
**?LinearAlgebra,TridiagonalForm**,  
    41  
**?Norm**, 41  
**?numeric\_overview**, 53  
**?odeadvisor,linear\_ODEs**, 532  
**?odeadvisor,types**, 531  
**?plot,coords**, 472  
**?plot,device**, 406  
**?plot,options**, 417  
**?plot,polar**, 36  
**?plot,structure**, 422  
**?plot3d,coords**, 472  
**?plot3d,option**, 442  
**?property**, 344  
**?repository,management**, 111, 395  
**?reserved**, 78  
**?RootOf,indexed**, 55  
**?rtable**, 324  
**?rtable,printf**, 120  
**?share,address**, 105  
**?share,contrib**, 105  
**?structuredview**, 308  
**?TridiagonalForm**, 41  
**?type,statement**, 40  
**?type,surface**, 40, 83  
**?updates**, 40  
`[]`, empty list, 295  
    selection operator, 291, 293, 296,  
        316  
`#` (sharp symbol), 9, 97  
`{ }`, empty set, 292  
    set notation, 292  
    -> (arrow operator), 12, 193–195,  
        204, 208, 210  
`@` (at sign), 210  
`@@` (double at sign), 210  
`$` (sequence operator), 214, 291  
`&`\* (multiplication operator of arrays),  
        304, 315  
`_` (underscore), 54, 77, 78, 80, 562  
`_Envadditionally`, 339, 340  
`_EnvAllSolutions`, 488, 489  
`_Envdiffopdomain`, 558  
`_EnvExplicit`, 80, 105, 484, 499, 502,  
        504, 509  
`_EnvFormal`, 260  
`_EnvGroebnerPairSelectionStrategy`,  
        717  
`_Envignum0`, 60, 61, 80  
`_EnvTry`, 341  
`_MaxSols`, 489  
`_SolutionsMayBeLost`, 485  
`%` (percentage symbol), 17, 46, 79, 83,  
        101, 208  
    in labels, 135, 482  
`%%` (double percent), 46, 79, 83  
`%%%` (triple percent), 46, 79, 83  
`\` (backslash), 35, 42, 97, 122  
`\\` (raw backslash), 122  
`\"` (raw double quote), 122  
`\'` (raw single quote), 122  
`\?` (raw question mark), 122  
`\b` (raw backspace), 122  
`\f` (raw formfeed), 122  
`\n` (raw newline), 122  
`\r` (carriage return), 122  
`\t` (raw horizontal tabulate), 122  
`\v` (raw vertical tabulate), 122  
`"` (double quote), 81–83  
``` see left quotes  
`'` see apostrophes  
`~` (tilde), as home directory, 108  
    in names of variables, 16, 89, 90,  
        92, 334  
 $\gamma$ , see gamma  
 $\infty$ , see infinity  
 $\pi$ , see Pi  
2D coordinate systems, 415, 416, 472,  
        688  
2D graphics, 403–440  
2D plot, general format, 404

- objects, 418, 419, 422, 423
- options, 407–418
- scaling, 407, 473
- specialties, 426–436
- styles, 410–412, 417, 475
- vertical range of, 407
- view of, **view**, 407, 408, 474
- window, 404
- 3D coordinate systems, 453, 454, 472, 689
- 3D graphics, 441–458
- 3D plot, general format, 441
  - objects, 448, 449
  - options, 442–447, 472–477
  - specialties, 452–458
  - styles, 442, 476, 477
- abbreviations, in **solve**, 483, 484
  - introducing, **macro**, 49, 102, 103, 163, 210, 327, 494, 673
- aborting a computation, 35
- about**, 20, 61, 62, 89–93, 338–341, 344
- Abramov's algorithm for solving rational ODEs, 531
- Abramov's summation method, 256, 257
- abs**, 50, 58, 59, 334, 336
- absolute factorization, **AFactor**, 180
- absolute value, **abs**, 50, 58, 334
- accents, acute, 67, 68, 71, 75, 76, 83, 87, 143, 202, 213, 291, 318, 440
  - grave, 80, 81, 97
- acute accents, 67, 68, 71, 75, 76, 83, 87, 143, 202, 213, 291, 318, 440
- adapting Maple code, 92, 93, 275, 276, 286, 287, 317, 358, 391–396
- adaptive**, plot option, 420, 474
- adaptive double-exponential quadrature method, 32, 239
- adaptive Newton-Cotes method, 32, 239
- adaptive plotting, 410, 419–421, 474
- adaptive sampling, in 2D plotting, 419–421
- add**, 255
- Add**, 630, 631
- AddCoordinates**, 691
- addcoords**, 588, 589
- addition of, elements to lists, 297
- elements to sets, 317
- functions, +, 210
- matrices, **MatrixAdd**, **Add**, 630, 631
- numbers, +, 34
- properties, **addproperty**, 338, 344, 345
- rows and columns to a matrix, 623, 624, 642, 666, 668, 674, 675
- additionally**, 89, 90, 338, 340
- additive functions, 391
- addproperty**, 338, 344, 345
- addressof**, 174
- addtable**, 246, 247
- Adjoint**, 641
  - adjoint of a matrix, **Adjoint**, 641
- admissible term ordering, 709, 717, 722
  - good extension of, 722
- advantages, of computer algebra, 11–23
- aerodynamics, 512–514
- AFactor**, 180
- Airy equation, 618
- algebraic functions, 80, 177–179, 229, 230, 233, 503
- algebraic geometry, 3
- algebraic numbers, 53–57, 80, 177, 178, 487, 636
  - conversion between radicals and **RootOfs**, 55
  - simplification of, 53–55
- algebraic sets, 671
- algebraic substitution, **algsubs**, 166–168, 381
  - Algol68**, 11
  - algsubs**, 166–168, 381
    - exact** (option), 167
    - remainder** (option), 167
    - vs. **subs**, 166, 168
- alias**, 55, 62, 91, 102, 103, 494, 525, 720, 734, 740, 743
  - vs. **macro**, 103
- allvalues**, 55, 484, 497, 637
  - dependent** (option), 497, 637
  - independent** (option), 497
- ambientlight**, plot option, 446, 475

**AMBIENTLIGHT**, graphics primitive, 452  
**analysis** of variance, **anova**, 104  
**anames**, 69, 70, 72  
**and**, 84  
**AndProp**, 343, 344  
angle between vectors, **VectorAngle**, 645  
angle of view, **orientation**, 443–445, 476  
anharmonic oscillator, 578–580  
**animate**, 469, 612  
**animate3d**, 469, 471  
**animatecurve**, 470  
animation, 469–472, 612  
  of a curve, 470  
  of a family of algebraic curves, 470  
  of a sine wave, 469  
  of a solution of a PDE, 612  
  of a 3D rotation, 472  
animation options,  
  **frames**, 477  
  **framescaling**,  
    **nonuniform**, 477  
    **uniform**, 477  
  **insequence**, 470, 471, 477  
anonymous functions, 70, 210, 211  
**anova**, subpackage of **stats**, 104  
antiderivatives, 225–234  
**antihermitean**, special indexing, 310  
antilogarithm, *see* exponential  
  function  
**antisymmetric**, special indexing, 310, 311, 313, 622  
antisymmetric matrix, 310, 313, 622  
apostrophes, 67, 68, 71, 75, 76, 83, 87, 143, 202, 213, 291, 318, 440  
**APPEND**, 116–118  
appending, to files, **appendto**, 107, 112, 132  
  to lists, 297  
**appendto**, 107, 112, 132  
applying functions to, all matrix  
  elements, **Map**, **map**, 320, 633, 639, 653  
all operands of an expression, **map**, 171  
lists, **map map2**, 210, 211  
  specific parts of expressions,  
    **applyop**, 170, 171, 211  
**applyop**, 170, 171, 210  
**applyrule**, 56, 166, 172, 677  
approximate vs. exact computation, 8, 46, 51  
approximation of functions,  
  **numapprox** package, 10, 149, 265, 267, 273, 276–281, 610  
approximations, algebraic, **series**, 265–276,  
  numerical, **evalf**, 46–53  
arbitrary precision numbers, *see*  
  floating-point numbers  
**arc**, 424  
**arccos**, 50  
**arccosh**, 50  
**arccot**, 50  
**arccoth**, 50  
**arccsc**, 50  
**arccsch**, 50  
**arcsec**, 50  
**arcsech**, 50  
**arcsin**, 50  
**arcsinh**, 50  
**arctan**, 5, 9, 50, 354, 369  
**arctanh**, 50  
**AreCollinear**, 438  
**args**, argument sequence, 290  
**argument**, 58  
argument of a complex number,  
  **argument**, 58  
arguments, constraints on types of, 201–203  
  default values of, 203  
  functions with variable number of, 290  
  in arrow operators, 194  
  optional, 432  
  preventing evaluation of, 75, 76, 440  
  sequence of, **args**, 289, 290  
arithmetic, approximate, 8, 46, 48, 51  
  at arbitrary precision, 8, 48  
  at hardware precision, 51, 450, 656–658  
  at user-defined precision, 8, 48, 656–658  
complex, 58–62

exact, 8, 41  
 exact vs. approximate, 8, 46, 51  
 floating-point, 7, 8, 46, 48, 53,  
     656–658  
 hardware vs. software floats, 25, 52,  
     449, 648, 649, 656, 657  
 modular, 44, 45, 144, 514, 648–660  
 of algebraic numbers, 53, 55  
 of complex numbers, 58–62  
 of floating-point numbers, 7, 8,  
     46–48, 53, 656–658  
 of fractions, 41  
 of integers, 41  
 of matrices, 629–634, 656–658  
 of radicals, 46, 53  
 of rational numbers, 41  
 of vectors, 629–634  
 over algebraic function fields, 178  
 over finite fields, 45, 144, 514, 648,  
     651–656  
 arithmetic operations, addition, +, 34  
 division, /, 34  
 double factorial, !!, 34  
 exponentiation, ^, 34  
 factorial, !, 34  
 multiplication, \*, 34  
 precedence of, 34, 633, 634  
 arranging terms, *see* **sort**  
**array**, 300, 301  
**Array**, 300, 305, 311, 621, 622  
**ArrayDims**, 303  
**ArrayNumDims**, 303  
**ArrayNumElems**, 303  
 arrays, 300–315, 621, 622  
     copying, **copy**, 321, 326  
     creation of, **Array**, 300, 305, 311,  
         621, 622  
     creation of, **array**, 300, 301  
     displaying, **lprint**, 304  
     evaluation of, **eval**, 304, 320, 321,  
         629  
     graphics, **display**, 427, 445, 470,  
         471, 477  
     multidimensional, 300, 301  
     multiplication of, &\*, 304, 315  
     reading from data files,  
         **readdata**, 113, 114, 315  
     with special indexing, 309–315  
**arrow**, 424, 425  
 arrow operator, ->, 12, 193–195, 204,  
     208, 210  
 ASCII characters, 106, 109, 113, 118,  
     290  
**assemble**, 174  
**assign**, 66, 71, 72, 90, 91, 493  
**assigned**, 69–71  
 assigned names, **anames**, 69, 70, 72  
 assignment, of solutions of (systems  
     of) equations, **assign**, 493  
 assignment operator (:=), 66, 71, 87  
 assignments, 66, 71  
     functional, 207  
     of indexed names, 316  
     to list elements, 297  
     vs. substitutions, 66, 67  
     with assumptions, 90–92, 337  
 assisting Maple, 242, 243, 251–255,  
     275, 276, 285–287, 536, 602  
**assume**, 19, 61, 89–92, 197, 217,  
     334–339, 341, 347–349, 356,  
     387, 437, 690  
     vs. **assuming**, 338  
 assume facility, basic commands of,  
     338  
     basics of, 338–341  
     implementation of, 344–348  
     need for, 333–338  
     peculiarities of, 89–92, 348, 349  
     restrictions of, 341  
**assuming**, 15, 18, 19, 61, 78, 198,  
     249–252, 254, 286, 334–338,  
     354, 356, 359–361, 364, 367,  
     369, 374, 380, 387  
     vs. **assume**, 338  
 assuming, negative numbers, 334  
     positive numbers, 334  
     real context, 334, 336, 354, 356,  
         372, 374, 386–388, 484  
 assumptions, 286, 333–353  
     assigning variables with, 90–92, 337  
     forgetting, 335  
     getting information about,  
         **about**, 20, 61, 62, 89–93,  
         338–341, 344  
     renaming variables under, 16, 92,  
         334  
     saving variables with, 348, 349  
 asterisk, \*, as field length, 121

- as product, 34, 84
- as symbol, 116
- asymp**, 274, 516
- asymptotic series, 274
- at sign, double, **@@**, 210
  - single, **@**, 210
- atomic orbitals, 682
- attributes**, option of **Matrix** and **Vector**, 621
- attributes**, 88, 89
- attributes, defining, **setAttribute**, 88, 89, 125, 331, 462
  - of expressions, 88, 89
- augmentation of matrices, 623, 624, 642, 666, 668, 674, 675
- augmented**, option of **GenerateMatrix**, 626
- automatic differentiation, **D**, 220–224
- automatic loading of library files, 101, 108
- automatic simplification, 7, 8, 62, 354–356
  - difficulties with, 355
- autosimp**, 616
- avoid**, option of **solve**, 510, 511
- axes**, plot option, 443, 472
- AXES**, graphics primitive, 452
- axesfont**, plot option, 408, 410, 427, 472
- AXESTYLE**, graphics primitive, 418, 422, 423
- AXESTICKS**, graphics primitive, 422
- AXIOM**, 5, 25
- azulene, 680, 681, 687
- back quotes, 80, 81, 97
- background lighting, **ambientlight**, 446, 475
- backslash, **\**, 35, 42, 97, 122
- backspace, **\b**, 122
- band**, special indexing, 310
- banded matrix, **band**, 310
  - BandMatrix**, 624
- BandMatrix**, 624
- bar charts, 462–464, 466
- base of natural logarithm, 49, 78
- Basis**, 645
- basis, for a vector space, **Basis**, 645
  - for column space of matrix, **ColumnSpace**, 635, 645
  - for intersection of vector spaces, **IntersectionBasis**, 645, 646
  - for nullspace (kernel) of matrix, **NullSpace**, 635, 641, 645
  - for row space of matrix, **RowSpace**, 635, 645
  - for sum of vector spaces, **SumBasis**, 645, 646
  - of polynomial ideal, Gröbner basis, **gbasis**, 168, 383, 384, 501, 502, 505, 507, 716, 720
- batch files, 107–112, 395
- Berlekamp’s factorization algorithm, 178
- Bernoulli numbers, 256
- Bernoulli polynomials, 256
- Bessel equation, 533, 535, 536
- Bessel function, **BesselII**, **BesselJ**, **BesselK**, **BesselY**, 50, 242–246, 409, 410, 534, 535
  - expansion of, 362
- Bessel polynomial, 11, 25
- BesselII**, 50, 534, 535
- BesselJ**, 50, 242–246, 362, 409, 410
- BesselK**, 50, 534, 535
- BesselY**, 50
- Beta**, 203
- beta function, **Beta**, 203
- Bezout matrix, **BezoutMatrix**, 624
- BezoutMatrix**, 624
- bidirectional limits, **limit**, 285
- big O, order symbol, 266, 267, 269
- BINARY**, 117
- binary files, 117–119
- binomial**, 50
- binomial coefficient, **binomial**, 50
- binomials, expansion of, 361
- bipolar coordinates, 472, 688
- blanks, 9, 35, 80
- block matrix, consisting of Jordan blocks, **JordanBlockMatrix**, 624, 642
  - using **DiagonalMatrix**, 625
- bmp**, graphics output, 406
- Bode plots, 430–432
- Boltzmann equation, 618
- Boolean algebra, 343

- Boolean expressions, 84
  - evaluation of, 198
- Boolean values, FAIL, 48, 340
  - false, 48, 340
  - true, 48, 340
- bottom of fraction, denominator, denom, 147, 161, 177, 184, 254, 382, 677, 678
- BottomProp**, 343, 350, 352
- bound variables, *see* assigned variables
- boundary conditions in differential equations, *see* boundary value problem
- boundary value problem, 528, 529, 566, 577, 579, 580, 609–611, 613, 614
  - derivatives in, 529
- box**, axesstyle option, 412–414, 433–436, 442, 443, 446, 448–455, 462, 463, 465, 472, 530, 574, 585
  - plot symbol, 411, 473
- boxplot**, 411, 465,
- braces, 292, 294, 328
- bracket notation of matrices and vectors, 622, 623
- brackets, curly, 292, 294, 328
  - round, 34, 194
  - square, 294, 296, 328
- branches**, 190
  - branches, of Lambert's *W* function, 190, 489
    - of multiple-valued functions, branches, 58, 59, 190, 489
- Bronstein's integration algorithm, 531
- b-spline approximation, 469
- BSplineCurve**, 469
- build**, 602, 606–609
- built-in Maple procedures, getting a list of, 40
- built-in mathematical functions, a list of, 49, 50
- Burgers' equation, 618
- button, Exit, 35
  - File, 35
  - Full-Text Search, 40
  - Help, 36
  - Stop, 35
- Topic Search**, 39
- bytes-used messages, 133, 134
  - controlling frequency of, 134
- bytesalloc**, kernel option, 134
- bytesused**, kernel option, 134
- C**, 129
- C code generation, C, 129
  - options of, 129
    - declare, 129
    - optimize, 129
    - output, 129
- C language, 4, 16, 29, 47
- cadmium transfer, 669–680, 701
- calculation, algebraic, 2
  - exact vs. numerical, 8, 46, 51
  - numerical, 1, 8, 46, 48, 51, 450, 656–658
  - symbolic, 2
- calculus, differential, 213–224
  - integral, 225–263
  - polynomial, 139–152, 175–188
  - vector, 687–693
- CAMAL, 3
- camera position, in graphics, orientation, 443–445, 476
- Canadian flag, 458
- canoni**, 548, 553, 560
- canonical form, general, 181, 182
  - of polynomials, 139, 384, 710–712
  - of rational numbers, 45
- canonical simplifier, 182
- Cantor-Zassenhaus factorization algorithm, 178
- carbon atom, 680, 681
- caret (^), 96
- carriage return, Return key, 34, 122
- Cartesian coordinates, 406, 443, 453, 455, 472, 506, 687–689
- Cartesian form of a complex number, 326, 328
- Cartesian product, 301
- case clauses, 195, 196
- case sensitivity, 77
- casesplit**, 603–605
- cat**, 82
- Catalan**, 48
- Catalan's number, Catalan, 48
- catenating, lists, 297

- sequences, 290
- sets, **union**, 293
- strings, 82
- catenoid, 455, 479
- Cayley, 3, 4
- Cayley-Hamilton theorem, 635, 660, 661
- CBS, 126, 459
- celestial mechanics, 3, 283
- Cesaro summation, 260
- cfrac**, 87
- change of coordinate system, **changecoords**, 415, 416, 424
  - in differential equations 586–589
  - in plot command, 453, 454
  - in vector analysis, 688–693
- change of variables, in integration, 252
  - in differential equations, **dchange**, 549, 553, 586–590, 608
  - in graphics, **changecoords**, 415, 416, 424
  - in ODEs, **Dchangevar**, 586, 588
  - in PDEs, **PDEchangecoords**, 586–589, 591
  - in vector analysis, 687–693
- changecoords**, 415, 416, 424
- changing default plot options, **setoptions**, 416
  - setoptions3d**, 447
- changing parts, of lists, 297–299
  - of expressions, 166–171
- characteristic, equations, 635
  - matrix, **CharacteristicMatrix**, 641, 682
  - polynomial, **CharacteristicPolynomial**, 635, 638, 641, 682
- characteristic sets, **charsets**, 105, 508
  - csolve**, 105
- characters, allowed in names, 77
  - ASCII, 106, 109, 113, 118, 290
  - escape, 121
  - Greek, 16, 77
  - special, 80, 81
- charge density, 336, 337
- charge distribution, 434, 582, 583
- CharacteristicMatrix**, 641, 682
- CharacteristicPolynomial**, 635, 638, 641, 682
- charsets** package, 105
  - csolve**, 105
  - chebdeg**, 278
  - chebmult**, 278
  - chebpade**, 10, 278
  - chebsort**, 278
  - chebyshev**, 273, 278
- Chebyshev, approximation, 273, 278, 279
  - integral, 232
  - polynomial, **ChebyshevT**, 274, 278, 279, 288
  - series expansion, **chebyshev**, 273, 278, 279
- ChebyshevT**, 274, 278, 279, 288
- Chebyshev-Padé approximation, **chebpade**, 10, 278, 279
- checking solutions of, determining system, **syntest**, 546, 552, 560
  - equations, 481, 482, 496, 497, 502, 503
- ordinary differential equations, **odetest**, 524, 525, 537, 538
- partial differential equations, **pdetest**, 603, 604, 606, 607, 609
- chemical reaction, mechanism, 698, 720, 721, 724, 746
  - stoichiometry of, 519
- chlorine atom, 687
- Cholesky, method of **LUDecomposition**, 648
- Cholesky decomposition, 648
- choose**, 312, 394
- choosing elements, *see* selection
- chromatic number, 726
- chrompoly**, 728
- circle**, plot symbol, 410, 429, 460, 461, 473, 568, 580
- circle**, geometry command, 436
- Circle Theorem of Apollonius, 436
- circular definitions, 68, 69, 204–208
- circular functions, *see* trigonometric functions
- circumcircle**, 437

- circumscribing circle, **circumcircle**, 437  
 cleaning up expressions, *see* simplification  
 clearing, remember tables, *see* **forget**  
 memory, **gc**, 134  
 variables, **evaln**, 69, 71, 72, 75, 76, 318  
 Clenshaw-Curtis quadrature, 32, 239  
**close**, 117, 118  
 closing, of files, **close**, **fclose**, 117  
     of streams, **close**, **pclose**, 117, 118  
**CoCoA**, 3  
 code generation, 16, 127–133  
     in C, C, 129  
     in FORTRAN, **Fortran**, 16, 127, 128, 131  
     in Java, **Java**, 129–131  
**codegen** package,  
     **cost**, 128, 148, 149, 182, 183  
     **intrep2maple**, 110  
     **JACOBIAN**, 131  
     **optimize**, 128, 129  
     vs. **CodeGeneration** package, 131  
**CodeGeneration** package, 16, 127–133  
     vs. **codegen** package, 131  
 code optimization, 16, 128, 129  
**coeff**, 30, 141, 146, 270, 381, 545, 591–594, 596–598  
**CoefficientList**, 142  
 coefficients, of a series, **coeftayl**, 271  
**coeffs**, option of **dsolve**, 522  
**coeffs**, 141, 142, 146, 513, 678  
**coeftayl**, 271, 540, 541  
 cofactor of a matrix, **Minor**, 32, 641  
 collation, *see* **sort**  
**collect**, 18, 142, 146, 185–187, 210, 211, 214, 216, 244, 294, 362, 493, 495, 507, 508, 531, 541, 545, 551, 565, 570–572, 587, 591–598, 608, 665, 676, 677, 693  
 collected form, of polynomials, 139–142, 182, 186  
 collection of terms, *see* **collect**  
 colon, double, 86, 87, 201  
     single, 9, 34, 86, 87, 96, 98  
**color**, plot option, 10, 406, 412–415, 421–424, 426–428, 434, 435, 438, 445, 448, 449, 452, 458, 460, 461, 463, 468, 469, 471, 472, 475, 574, 575, 578, 580, 585, 611, 612  
 color charts, cube, **RGB**, 414  
     disk, **HUE**, 415  
     normal, **RGB**, 413  
     wheel, **HUE**, 414  
 color models, 413–415  
     **HSV**, 413, 414  
     **HUE**, 414, 415  
     **RGB**, 413, 414  
 colorability of graphs, 726–729  
 coloring of graphs, 726–729  
**COLOUR**, graphics primitive, 418, 419, 422, 448, 452  
**Column**, 641, 665  
**column** (vector notation), 621  
**ColumnDimension**, 641, 693  
**ColumnOperation**, 641  
 column rank, of matrix, **Rank**, 634, 641, 674  
**ColumnSpace**, 635, 645  
 column space, of matrix,  
     **ColumnSpace**, 635, 645  
**combinat** package, **choose**, 312, 394  
     **powerset**, 293  
**combine**, 364–370  
     **arctan**, 369  
     **exp**, 366, 367, 589  
     **ln**, 242, 367, 368, 513  
     **power**, 368  
     **radical**, 369  
     **symbolic**, 364–369  
     **trig**, 275, 365, 366, 380, 529, 592, 598  
     validity of, 364  
**combine**, 364–370, 380, 592  
 combining, arctangents, **arctan**, 369  
     exponential mappings, **exp**, 366, 367, 589  
     expressions, 364–370  
     hyperbolic functions, **trig**, 365, 366  
     logarithms, **ln**, 242, 367, 368, 513  
     matrices and vectors, 623, 624, 642, 666, 668, 674, 675  
     polylogarithms, 369  
     powers, **power**, 368

- radicals, **radical**, 369
- rational expressions, 370
- square roots, 364, 369
- sums, 364, 365
- trigonometric functions, **trig**, 275, 365, 366, 380, 529, 592, 598
- combining plots into, *see also display*
  - a graphics array, **display**, 427, 445, 470, 471, 477
  - a single picture, 426, 427, 431, 435, 445, 454, 461, 462, 470
  - an animation, 471
- command line editor, 35, 96
- commands, across several lines, 34
  - alphabetical listing of built-in, **?index,function**, 40
  - correcting, 35, 96
  - executing, 34
  - interrupting, 35
  - timing execution of, **time**, 44
- comments, **#**, 9, 97
- common plot options, *see plot options*, common
- common subexpressions, sharing of, 45, 155
- commutative algebra, 3
- compacting expressions, **optimize**, 128
- CompanionMatrix**, 624
- compartimental models, 669–680, 695
  - four-compartment model of cadmium transfer, 695
  - three-compartment model of cadmium transfer, 669, 670, 701, 733, 734
- complements, of sets, **minus**, 293
- complex**, **fsolve**, 510
  - partial fraction decomposition, 149
- complex arithmetic, 58–62
- complex conjugate, **conjugate**, 58
- complex functions, 58
  - branches of, 53, 58, 190, 355, 484, 489
  - plotting of, **complexplot**, 432
    - complexplot3d**, 455
- complex numbers and expressions, absolute values of, **abs**, 50, 58, 59, 334, 336
- arguments of, **argument**, 58
- arithmetic of, 58–62
- automatic simplification of, 58
- conjugates of, **conjugate**, 58
- evaluation of, **evalc**, 59–62, 208, 326, 327, 376, 430, 431, 447
- imaginary parts of, **Im**, 58, 249, 336, 340, 356, 484
- real parts of, **Re**, 58, 249, 336, 340
- complex roots of polynomials, 487, 510
- complex sign function, **csgn**, 60, 251, 334–336, 386, 388, 526
- complex square root of  $-1$ , **I**, 58, 62, 78
- complexplot**, 432
- complexplot3d**, 455
- compoly**, 385
- compose**, 283
- composite**, property, 343, 350
- composite data types, 289–332
- composition, of functions, **@**, 210, 217
  - of polynomials, **compoly**, 25, 385
  - of formal power series, **compose**, 283
- composition operator, **@**, 210, 217
- compound statements, 34, 35
- CompSeq**, 209
- computation, *see calculation*
- computer algebra, advantages of, 11–23
  - characterization of, 1, 2, 11
  - limitations of, 23–29
- computer algebra systems,
  - general purpose, 3–5
  - special purpose, 3
- computing time, 44, 133
- concatenation operator (**||**), 82, 84, 292
- concatenating, strings, **cat**, **||**, 82, 84, 292
  - lists, 297
- matrices, 623, 642, 668, 674, 675
- sequences, 290
- sets, **union**, 293
- condition number, **ConditionNumber**, 641
- conditionals, *see case clauses*
- ConditionNumber**, 641
- conformal**, 432, 433

conformal mappings, 432, 433  
 conformal plots, **conformal**, 432, 433  
**confrac**, 87, 149  
**confracform**, 149, 273, 277  
**conjugate**, 58  
**constant**, 310  
 constant functions, 217  
 constants, boolean, 48, 340  
     defining, 48, 49  
     Catalan's number, **Catalan**, 48  
     Euler-Mascheroni number, **gamma**,  
         16, 48  
     infinity,  $\infty$ , 48  
     of integration, 226  
     Pi,  $\pi$ , 48, 77, 155  
**constrained**, scaling option, 407, 473  
 contagious floats, 47  
 continuation character ( $\backslash$ ), 35, 42, 97  
 continued fraction form,  
     **confrac**, option of **convert**, 148,  
         149  
     **confracform**, 149, 273, 277  
 continued fractions of,  
     rational functions, 148, 149, 273,  
         277  
     rational numbers, 87  
**continuous**, option of **integrate**, 235  
**contour**, plotstyle option, 442, 447,  
     475, 477  
 contour integration, 253, 254  
 contour lines, in 3D-plots, 447, 454,  
     475, 476  
 contour plots, 435, 436  
**contourplot**, 435  
**contourplot3d**, 459  
**contours**, plot option, 434, 435, 447,  
     451, 455, 475  
 Control-c, 35  
 controllability, of linear systems, 671,  
     675, 676  
 controllability matrix, 671, 675, 676  
 conversion between,  
     composite data types, 328–331  
     factorials, binomials, and gamma  
         functions, 378  
     series and polynomials, 270, 273,  
         274  
 special functions and standard  
     functions, 378  
     trigonometric, hyperbolic, and  
         exponential functions, 375–377  
     types, 87  
 conversion routines, writing your  
     own, 328, 485  
**convert**, 328–331, 375–378  
     ‘\*’, 514  
     ‘and’, 497, 498  
     **Array**, 329  
     **array**, 303, 314, 329–331  
     **binary**, 87  
     **binomial**, 258, 378  
     **bytes**, 118  
     **cartesian**, 328  
     **confrac**, 87, 149, 273  
     D, 224  
     **diff**, 219, 557, 595, 597–599  
     **Diff**, 613, 614  
     **exp**, 375, 376  
     **expln**, 376  
     **expincos**, 377  
     **factorial**, 378  
     **float**, 46, 47  
     **fraction**, 87  
     **fullparfrac**, 150  
     **GAMMA**, 378  
     **Heaviside**, 249  
     **horner**, 148, 182  
     **hypergeom**, 378  
     **hypergeometric**, 535  
     **list**, 247–249, 329–331, 685, 687  
     **listlist**, 124, 330, 331, 467, 468,  
         677, 679  
     **ln**, 376  
     **matrix**, 115, 467  
     **name**, 396, 435, 445, 463  
     **noIndexedRootOf**, 486, 487  
     **parfrac**, 149–151  
     **piecewise**, 200  
     **PLOT3Doptions**, 452  
     **polar**, 327  
     **polynom**, 201, 273, 274, 276, 561,  
         565, 566  
     **procedure**, 209  
     **radical**, 8, 50, 55, 57, 179, 180,  
         185, 378, 486, 498  
     **ratpoly**, 273, 274  
     **RootOf**, 55, 185, 232, 638  
     **set**, 295, 329–331, 379

**signum**, 334  
**sincos**, 250, 377, 380  
**sqrfree**, 180  
**StandardFunctions**, 378, 536  
**surd**, 53  
**tan**, 377, 380  
**trig**, 28, 376, 605, 608  
**units**, *amu*, 49  
**units**, *miles*, 319  
**vector**, 315, 330  
**Vector[column]**, 330  
**Whittaker**, 535  
convolutions, 248  
coordinate mapping, 13–16, 689–693  
coordinate systems, in 2D graphics, 415, 416  
in 3D graphics, 453, 454  
orthogonal curvilinear, 687–689  
setting you own, **addcoords**, 588  
    **changecoords**, 415, 416, 424  
    **SetCoordinates**, 687, 690, 691  
coordinates, bipolar, 688  
    Cartesian, 406, 443, 453, 455, 472, 506, 687–689  
    changing, **addcoords**, 588  
        **changecoords**, 415, 416, 424  
        **SetCoordinates**, 687, 690, 691  
    circular-cylinder, 453, 473, 689  
defining you own, **addcoords**, 588  
    **changecoords**, 415, 416, 424  
    **SetCoordinates**, 687, 690, 691  
ellipsoidal, 689  
elliptic, 415, 688  
elliptic-cylinder, 689  
hyperbolic, 688  
hyperbolic-cylinder, 689  
hyperspherical, 13, 14  
oblate spheroidal, 689  
orthogonal curvilinear, 687–689  
parabolic, 688  
parabolic-cylinder, 689  
polar, 415, 688  
prolate spheroidal, 689–693  
rectangular, 688, 689  
spherical, 453, 689  
**coordplot3d**, 689  
**coords**, 406, 414–416, 453, 455, 458, 472  
**copy**, 321, 679  
**Copy**, 655  
Coriolis term, 571  
correction of input, 35, 96  
correctness of simplification, 8, 61, 354, 355  
**cos**, 50  
**cosh**, 50  
**cost**, 128, 148, 149, 182, 183  
**cot**, 50  
**coth**, 50  
**coulditbe**, 338, 340  
**cputime**, kernel option, 134  
criteria, removing elements based on, **remove**, 210, 211, 294, 599  
selecting elements based on, **select**, 70, 210, 211, 294, 587, 599, 603, 616  
selecting solutions based on, **select**, 484, 498  
slitting elements based on, **selectremove**, 294  
critical line for Riemann  $\zeta$ -function, 59  
**cross**, plot symbol, 473  
cross product, **CrossProduct**, 645  
**CrossProduct**, 645  
**csc**, 50  
**csch**, 50  
**csgn**, 60, 251, 334–336, 386, 388, 526  
**csolve**, 105  
cubic equations, 481, 482  
**cuboid**, 414  
**currentdir**, 111, 137, 395, 396  
**Curl**, 687, 688  
**curve**, 424, 425  
curve fitting, **BsplineCurve**, 469  
    fit package, 104, 468  
    **leastsquare**, 468  
    **LeastSquares**, 469  
**CurveFitting** package, 469  
    **BsplineCurve**, 469  
    **Interactive**, 469  
    **LeastSquares**, 469  
curves, color of, *see* **color**  
    dashing of, *see* **linestyle**  
    thickness of, *see* **thickness**  
plotting 3D, **spacecurve**, 452, 574, 575  
points to sample on, *see* **numpoints**

- styles of, *see linestyle*  
**CURVES**, graphics primitive, 422, 423  
 curvilinear coordinate system,  
     687–689  
 customization of Maple, 92, 93,  
     133–136, 275, 276, 286, 287,  
     317, 358  
 cyclohexane, 698, 699  
 cyclotomic polynomials, 152  
 cylinder, plot of a, 453, 454  
 cylindrical coordinates, 453, 454, 472,  
     689
- D**, 216–218, 220, 284, 526  
     distinction with **diff**, 216, 217  
**DAG**, 154–157  
 damped oscillator, 529, 530  
**DASH**, line style, 413, 473  
**DASHDOT**, line style, 413, 473  
 data, exporting,  
     **ExportMatrix**, 115, 315  
     **writedata**, 115, 116, 125, 315  
 fitting,  
     **BsplineCurve**, 469  
     **fit**, 104, 468  
     **leastsquare**, 468  
     **LeastSquares**, 469  
 importing, 113–115, 125–127,  
     315, 466–468, 629  
 plotting, 411, 459–469  
 reading,  
     **importdata**, 114, 466, 468  
     **ImportMatrix**, 114, 115, 315,  
         466, 467, 629  
     **readdata**, 113, 114, 315  
     **readline**, 108, 112–114, 125–127  
 smoothing, 248, 249  
 statistical analysis of, 465–469  
 writing,  
     **ExportMatrix**, 115, 315  
     **writedata**, 115, 116, 125, 315
- data types, 83–86  
 data vectors (DAGs), 42, 43, 45, 47,  
     58, 73, 74, 77, 83, 154–157,  
     266, 290  
**datatype**, option of, **Matrix** and  
     **Vector**, 621,  
     **rtable**, 322,  
**dchange**, 549, 553, 586–590, 608
- Dchangevar**, 586, 588  
 decimal place, in formatted output,  
     120, 121  
**declare**, option of **C** and **Fortran**,  
     128, 129  
**declare** (**PDEtools** function), 524,  
     526, 541, 543, 544, 551, 566,  
     570, 580, 590, 601, 603, 606,  
     607, 609  
 defining, attributes, **setattribute**,  
     88, 89, 125, 331, 462  
 constants, 48, 49  
 coordinates 415, 416, 424, 453, 454,  
     586–589, 688–693  
 derivatives, 213–220  
 functions, 189–212  
 indexing functions, 311–313, 317  
     values, *see* assignment  
 definite integration, 234–239  
 definite matrix, 642  
**DefiniteSum**, 258  
**degree**, 30, 140, 142, 211  
 degree of, ODEs, 522  
     PDEs, 601  
     polynomials, **degree**, 30, 140, 142,  
         211  
**delay**, option of **simplify**, 388  
**DeleteColumn**, 641  
 deleting, elements from lists, 296, 297  
     files, **fremove**, 124  
     symbols, 67–69, 71, 72  
**del** operator, **Del**, **Gradient**, **Nabla**,  
     582, 687, 688, 690, 691, 693  
**DeleteRow**, 641  
 Delta function, **Dirac**, 242, 246, 250,  
     612  
 Denavit-Hartenberg, matrix, 663  
     parameters, 665, 695  
 denesting, of radicals, **radnormal**,  
     12, 54, 252  
     of square roots, 54  
**denom**, 147, 161, 177, 184, 254, 382,  
     677, 678  
 denominator, *see* **denom**  
 density plot, **densityplot**, 435, 436  
**dependent**, option of **allvalues**, 497,  
     637  
**DEplot**, 581–583  
**DEplot3d**, 583–585

- derivatives, at some point, 216  
 defining, 213–220  
 partial, 216  
 symbolic, 213–220
- DERIVE**, 3
- Descartes' rule of signs, 512
- DESol**, 326, 537–538
- Determinant**, 15, 20, 102–105, 559, 634, 638, 641, 668, 675, 676, 683
- determinant of a matrix, *see Determinant*
- determine**, 616
- determining system,  
 computing,  
**determine**, 616  
**eta\_k**, 541, 542, 555  
**odepde**, 542–545, 551, 560
- definition of, 542, 616
- simplification of, **autosimp**, 616  
 solving, **syngen**, 546, 549, 550, 552, 555, 558, 560  
 validating solutions of, **symtest**, 546, 552, 560
- DEtools** package, 560, 580–590  
**canoni**, 548, 553, 560  
**DEplot**, 581–583  
**DEplot3d**, 583–585  
**Dchangevar**, 586, 588  
**eta\_k**, 541, 555, 560  
**intfactor**, 547, 560  
**PDEchangecoords**, 586–589, 591  
**PDEplot**, 585, 601  
**odeadvisor**, 536, 545, 547, 553, 560  
**syngen**, 546, 549, 550, 552, 555, 558, 560  
 option, **HINT**, 550  
**way**, 550  
**symtest**, 546, 552, 560
- Dexp**, integration method, 240
- diagonal**, special indexing, 309, 310
- DiagonalMatrix**, 624, 625
- diagonal matrix, 624, 625
- diamond**, plot symbol, 468, 473
- dictionary order, *see* lexicographic ordering
- diff**, 6, 9, 198, 200, 213–218  
 distinction with **D**, 216, 217
- Diff**, 213, 214
- difference equations, *see* recurrence relations
- differential equations,  
 analytical solution of, 523–538, 601–610  
 change of variables in,  
**dchange**, **Dchangevar**,  
**PDEchangecoords**, 586–590
- checking solutions of,  
**odetest**, 524, 525, 537, 538  
**pdetest**, 603, 604, 606, 607, 609
- definition of, 522, 601
- degree of, 522, 601
- graphical solution of, 580–586
- in functional notation, 526
- Lie point symmetries for,  
 first-order ODEs, 544–551  
 second order ODEs, 551–559
- numerical solution of, 566–580, 611–615
- partial, 600–615
- partial solution of, 537, 538
- perturbation solution of, 590–600
- plotting solutions of, 567, 568, 573–575, 580–586
- power series solution of, 561–566
- solving by Lie point symmetries, 538–560, 615–618
- Taylor series solution of, 560, 561
- differential operator, **D**, 216–218, 220, 284, 526
- differentiation, automatic, 220–224  
 constants in, 217
- implicit**, **implicitdiff**, 215, 219, 220
- in functional style with **D**, 216–218, 526
- of formal power series, **powdiff**, 283
- of formulas and functions, **diff**, 6, 9, 198, 200, 213–218
- diffusion equation, 589, 601, 609–615
- digamma function, **polygamma** function, **Psi**, 50, 256, 354
- digits**, option of **evalf/int**, 240
- digits, number of, **Digits**, 32, 47, 48, 52, 79, 80, 96, 239, 449

- displayprecision**, 47, 135, 136, 276, 566, 570, 639, 685
  - in formatted output, 120, 121
- Digits**, 32, 47, 48, 52, 79, 80, 96, 239, 449
- dilog**, 37, 49, 50
- dilogarithm, **dilog**, 37, 49, 50, 268, 362
  - expansion of, 362
- dimension of algebraic set, **hilbertdim**, 721, 725
- dimension of matrix,
  - ColumnDimension**,
  - RowDimension**,
  - Dimension**, 641, 677, 693
- dimensional analysis, 512–514
- dimensionless variables, 514, 561
- Dirac**, 242, 246, 250, 612
- Dirac delta function, **Dirac**, 242, 246, 250, 612
- directed acyclic graph (DAG), 42, 43, 45, 47, 58, 73, 74, 77, 83, 154–157, 266, 290
- disassemble**, 174
- discarding elements of lists, 296, 297
- discont**, plot option, 196, 236, 408, 421, 474
  - discontinuities, handling of, *discont*, 196, 236, 408, 421, 474
- discrim**, 676
- discriminant of polynomial, **discrim**, 676
- disk**, 424
- dismantle**, 159, 174, 266
- display**, 402, 408–410, 412–417, 420, 422–427, 431, 435, 445, 446, 454, 456–458, 460–463, 465, 468–471, 477, 576, 580, 611, 615
  - display of, information, **infolevel**, 98, 99, 192, 193, 227, 233, 243, 251, 420, 486, 516, 525, 526, 532, 648, 656
  - printlevel**, 51, 79, 98–100, 197
  - results, 29, 30, 34, 99–101, 134, 483
  - several plots in one picture, 426, 427, 431, 435, 444, 445, 457, 458, 460, 461, 470, 471
- displayprecision**, interface variable, 47, 135, 136, 276, 566, 570, 639, 685
- distinction between, arrays and Arrays, 301–305
  - Arrays and Matrices, 306
  - D** and **diff**, 216, 217
  - sets and lists, 295
  - tables and rtables, 323
- distributed form of polynomials, 185, 186
- ditto operators, 46, 79, 83, 100, 101, 208, 209
- Divergence**, 687, 688
- divergence of a vector function, **Divergence**, 687, 688
- divergent sum, 258
- Divide**, 144
- division, by zero, 491
  - of integers, 44
  - of numbers, /, 34
  - of polynomials, 75, 142
- documentation, on-line, 36–41
- documents, worksheet as interactive, 29, 30, 106
- dodecahedron, plot of, 456
- dollar sign (\$), 214, 291
- domain, specification of,
  - of transformations, 333–335, 353, 356
  - of computations, **RealDomain**, 334, 354, 356, 372, 374, 386–388, 484
- Domains** package, 25
- done**, 35
- DOT**, line style, 413, 473
- dot product, **DotProduct**, 632, 645
- DotProduct**, 632, 645
- dots, multiple (..), 84, 292
  - single (.), 307, 315, 631–633
- dotted lines, in graphics,
  - see linestyle*
- double**, precision option, 16, 128
- double colon (::), 86, 87, 201
- double factorial (!!), 34
- double integral, 240
- double logarithmic plots, **loglogplot**, 429, 430
- double pendulum, 569–576

animation of, 576  
**double precision, double**, 16, 128  
 double precision floats, 16, 128  
**draw**, 437, 438, 727  
 drivers, graphics, 405, 406  
**dsolve** 522, 538, 566, 572  
**'dsolve/methods'**, 534  
**dsolve option, [Bessel]**, 534  
 can, 552, 554  
 can2, 552  
 class, 538  
 coeffs, 522, 564  
 dif, 552, 557  
 explicit, 538  
 gon, 552  
 gon2, 552  
 [**hypergeometric**], 535  
 implicit, 525, 538  
**Lie**, 538  
**method**,  
 bvp, 577  
 classical, 576  
 dverk78, 569, 572, 574, 576  
 fourier, 530, 538  
 fouriercos, 530, 531, 538  
 fouriersin, 530, 538  
 gear, 577  
 laplace, 529, 531, 538, 592–594  
 lsode, 576, 577  
 mgear, 577  
 rkf45, 577  
 rosenbrock, 577  
 taylorseries, 577, 578  
**output**,  
 basis, 536, 538  
 listprocedure, 578  
**type**,  
 exact, 523–538  
 formal\_series, 564  
 numeric, 566, 568, 572, 574, 577,  
     580  
 series, 560, 561  
**way**,  
 2, 3, ..., 7 (algorithm), 550  
 abaco1, abaco2, 550  
 exp\_sym, 550  
 formal, 550  
 mu, 550  
 patterns, 550  
**pdsolve**, 546, 555  
 Duffing's equation, 618  
 duplicates, elimination of, 293  
**dverk78**, option of **dsolve**, 569, 572,  
     574, 576  
 dynamic data vector, 42, 43, 45, 47,  
     58, 73, 74, 77, 83, 154, 156,  
     157, 266, 290  
 dynamic graphics, *see* animation  
 dynamic type checking, 201, 202  
  
*e*, base of natural logarithm, 49, 78  
 echelon form, row reduced,  
**HermiteForm**, 620, 626, 646, 653  
**ReducedRowEchelonForm**, 642,  
     646, 674  
 RREF method of  
**LUDecomposition**, 647, 654  
**echo**, interface variable, 107, 137  
**edges**, 727, 728  
 editing of input, 35, 96  
**Ei**, exponential integral, 229  
 eigenvalues, **Eigenvalues**, 635–638,  
     641, 683, 685  
     in radical notation, 637  
     in **RootOf** notation, 637, 683  
     numerical, 685  
**Eigenvalues**, 635–638, 641, 683, 685  
 implicit, option, 637, 683  
 output=list, option, 638, 685  
 eigenvectors, **Eigenvectors**, 635–641,  
     683–685  
     in **RootOf** notation, 637–639, 683  
     numerical, 685  
**Eigenvectors**, 636–641, 683–685  
 implicit, option, 637–639, 683  
 output=list, option, 636, 639, 685  
 elapsed time, *see* computing time  
 electric field, 434, 435, 582  
 electrical engineering, 62, 430, 492  
 electronic charge distributions, 680,  
     686, 687  
 electronic circuit, 430, 492  
 electrophylic substitution, 687  
 electrostatic potential, 336–338,  
     434–436, 583  
 elementary functions, 229, 230, 262  
 elementary symmetric polynomials,  
     743

**eliminate**, 500, 501  
**elimination**, of variables, **eliminate**,  
     **resultant**, 500, 501  
     fraction-free Gaussian,  
         **FractionFree** method of  
         **LUDecomposition**, 650, 654,  
         674, 676, 677  
     **GaussianElimination**, 646,  
         653, 654, 674  
     vs. Gröbner basis method, 503  
**elimination ordering**, **lexdeg**, 704,  
     706, 709  
**ellipse**, 424  
**ellipsis operator** (...), 84, 292  
**ellipsoidal coordinates**, 689  
**elliptic coordinates**, 415, 688  
**elliptic functions**, 238, 239  
     **EllipticCE**, 238  
     **EllipticCPi**, 238  
     **EllipticE**, 238  
     **EllipticF**, 238  
     **EllipticK**, 238  
     **EllipticPi**, 238  
**elliptic integrals**, 238, 239  
     associated complete,  
         of the 1st kind, **EllipticK**,  
             **LegendreKc1**, 238  
         of the 2nd kind, **EllipticE**,  
             **LegendreEc1**, 238  
         of the 3rd kind, **EllipticPi**,  
             **LegendrePic1**, 238  
     complete,  
         of the 1st kind, **EllipticK**,  
             **LegendreKc**, 238  
         of the 2nd kind, **EllipticE**,  
             **LegendreEc**, 238  
         of the 3rd kind, **EllipticPi**,  
             **LegendrePic**, 238  
     incomplete,  
         of the 1st kind, **EllipticF**,  
             **LegendreF**, 238  
         of the 2nd kind, **EllipticE**,  
             **LegendreE**, 238  
         of the 3rd kind, **EllipticPi**,  
             **LegendrePi**, 238  
**emacs** (external editor), 35  
**empty**, list, [], 295  
     name, ‘’, 80, 82  
     remember table, **NULL**, 206, 208  
     sequence, 100, 290  
     set, { }, 292  
     statement, **NULL**, 100, 154, 297, 422  
     string, "", 114  
     table, **table()**, 318  
     title in a plot, **NULL**, 474  
**Encapsulated PostScript**, 405  
**end of file**, **feof**, 124, 125, 127  
**ending**, a session, **quit**, 35  
     an input statement, 6, 34, 96  
**endless evaluation**, 68, 75  
     loops, 68, 75  
     recursion, 68, 75  
**ends**, 728  
**Enneper's minimal surface**, 479, 741  
**Enter key**, 34  
**entries**, 316 462  
**entries of tables**, **entries**, 316, 462  
**environment variables**, 60, 79, 80,  
     209, 267, 275  
     **Envadditionally**, 339, 340  
     **EnvAllSolutions**, 488, 489  
     **Envdiffopdomain**, 558  
     **EnvExplicit**, 80, 105, 484, 499,  
         502, 504, 509  
     **EnvFormal**, 260  
     **EnvGroebnerPairSelectionStrategy**,  
         717  
     **Envsignum0**, 60, 61, 80  
     **EnvTry**, 341  
**Digits**, 32, 47, 48, 52, 79, 80, 96,  
     239, 449  
**ditto operators**, 46, 79, 83, 100,  
     101, 208, 209  
**Normalizer**, 275, 286  
**Order**, 267  
**printlevel**, 51, 79, 98–100, 109,  
     197  
**Testzero**, 275, 286  
**UseHardwareFloats**, 79, 648, 657  
**enzyme-catalyzed reaction**, 494  
**epsilon**, option of **evalf/int**, 240  
**epsilon-gcd**, **EpsilonGCD**, 145  
**EpsilonGCD**, 145  
**Equal**, 642, 643, 647, 648, 651–653,  
     656, 694  
**equal sign**, 84, 87  
**equality testing**,

- for matrices, **Equal**, 642, 643, 647, 648, 651–653, 656, 694  
**general**, **testeq**, 44, 142, 245, 441, 497, 498, 524  
**literal**,  $=$ , 84, 87  
**equations**, characteristic, 635  
 cubic, 481, 482  
 differential, *see* differential equations  
 difficulties with solving, 485–492  
 eliminating variables in,  
   **eliminate**, **resultant**, 500, 501  
 in one unknown, 481, 482  
 integer solutions of (systems of), 512  
 left-hand sides of, *see* **lhs**  
 linear, 492–495, 625, 626, 650, 658  
 modulo integers, 514  
 nonlinear, 494–501  
 parameterized solutions of, 491  
 polynomial, 501–508, 698–703  
 preprocessing of, 490  
 recurrence, 514, 562, 563  
 right-hand sides of, *see* **rhs**  
 simultaneous, 491–501  
 solving,  
   analytically, 481–509, 702, 703  
   modulo integers, 514  
   numerically, 510–512  
   over integers, 512–514  
 splitting systems of, 504, 505  
 trigonometric, 489, 490  
**erf**, 49, 50, 195, 229, 237, 253, 316  
**error**, 130, 204, 431  
 error curves, in approximation theory, 277–281  
 error function, **erf**, 49, 50, 195, 229, 237, 253, 316  
 error messages, 35, 68, 69, 97, 186  
**errorbreak**, interface variable, 136  
 escape character, 121  
 escape codes, 121, 122  
**eta\_k**, 541, 555, 560  
 Euclid's gcd algorithm, 23, 24  
 Euler's beta function, **Beta**, 203  
 Euler's method, for solving ODEs, 576  
 Euler's Theorem, 480  
 Euler-Lagrange, equation, 570  
 formalism, 569–572  
 function, 569  
 Euler-Mascheroni's constant, **gamma**, 16, 48  
**eval**, 9, 10, 16, 19, 73–75, 78, 79, 90–93, 163, 191, 207, 208, 244, 269, 283, 293, 304, 317, 318, 320, 321, 355, 361, 364, 385, 388, 482, 486, 494, 496, 497, 524, 526–528, 535, 536, 543, 545–548, 552–557, 559–561, 567, 573, 574, 578, 602–605, 609, 635, 673  
**Eval**, 191, 192, 216  
**evala**, 56, 177, 178, 180, 185, 230, 232, 233, 484  
**evalc**, 59–62, 208, 326, 327, 376, 430, 431, 447  
**evalf**, 7, 8, 46–48, 50–52, 57, 77, 78, 80, 96, 100, 106, 135, 144, 145, 223, 224, 236, 239–241, 257–260, 327–328, 346, 415, 431, 435, 450, 483–488, 538, 583, 614, 639, 640, 685  
**evalf/int**, 32, 39, 240  
**evalhf**, 51, 52, 449, 659  
**evalm**, 304, 315  
**evaln** (keyword in procedure definition), 202  
**evaln**, 69, 71, 72, 75, 76, 116, 318, 667  
**evalp**, 88  
 evaluation, 34, 66  
   chain, 73, 74, 320, 321  
   exceptions of full, 71, 304, 319–321  
   full, 67, 69, 71, 73–75, 123, 304, 306, 317, 319, 320, 322, 323  
   in assignments, 68, 71, 72, 90, 493  
   in function definitions, 208, 209  
   last name, 319–321  
   in algebraic number field or function field, *see* **evala**  
   in complex arithmetic, *see* **evalc**  
   in floating-point arithmetic, *see* **evalf**, **evalhf**  
   in matrix context, **evalm**, 304, 315  
   in plotting, 439, 440  
   preventing, 75, 76, 87, 439

of arrays, table, and procedures, 319–321  
**of RootOfs**, 55  
 to a name, *see evaln*  
 suppressing, 75, 76, 87, 439  
 even numbers, 89, 342, 343  
**example**, 39  
**exact**, option of **algsubs**, 167  
 exact vs. approximate computation, 8, 46, 51  
 executing external commands,  
**ssystem**, 108, 466  
 executing time, *see* computing time  
**Exit** (menu item), 35  
**exp**, simplification option, 366, 367, 375, 376, 389, 589  
**exp**, 50  
**exp(1)**, base of natural logarithm, 49, 78  
**expand**, 8, 11, 28, 30, 50, 53, 56, 140–144, 154, 157, 175–177, 182, 184, 201, 244, 270, 356–364, 372–374, 378–381, 389–397, 491, 482, 516, 537, 548, 570, 589, 605, 651, 653, 677, 678  
**Expand**, 143, 144, 177, 178, 653  
**expand/cos**, 357, 358  
**expand/tanh**, 391–397  
**expanded**, keyword of **normal**, 183, 184, 226–228, 246, 389, 515  
 expanded form of polynomials, 139–144, 154, 157, 176, 182, 185, 186  
**expandoff**, 97, 363  
**expandon**, 97, 363  
 expansion (of functions), 356–364  
 controlling, 97, 359–363, 390  
 defining your own procedures for, 391–397  
 of Bessel functions, 244, 362  
 of binomials, 362  
 of dilogarithms, 362  
 of exponential mappings, 359, 372  
 of factorials, 362, 378  
 of hyperbolic functions, 357, 359, 391–397, 605  
 of hyperbolic tangents, 391–397  
 of logarithms, 359, 360, 372, 389  
 of polynomials, 7, 139–144, 175–178, 651, 653, 677, 678  
 of powers, 177, 201, 360, 361, 373, 374  
 of radicals, 53, 56, 360, 361, 481, 482  
 of rational functions, 184  
 of Riemann  $\zeta$ -function, 50, 362  
 of trigonometric functions, 28, 356–359, 363, 379–381, 389, 390, 537  
 partial, 357, 363  
 preventing, 363  
 expansion (of polynomials), 139–144, 175–178  
 suppressing full, 176  
 vs. factorization, 7  
 expansion point of series 265, 266, 274  
**explicit**, option of **dsolve**, 538  
 explicit return from procedure,  
**return**, 78, 92, 93, 130, 131, 202, 312, 392, 394  
 explicit roots of polynomials, 499, 502, 504, 509  
 explicit solution of an ODE, 524, 525, 538  
**expln**, option of **simplify**, 376  
**EXPLODE**, 531  
 exponent, of power, 47  
 exponential forms, conversion into,  
 hyperbolic functions, 376  
 trigonometric functions, 376  
 exponential function, **exp**, 50  
 exponential integral, **Ei**, 229  
 exponential mappings, **exp**, 50  
 approximating, 276–282  
 combining, 366, 367  
 expanding, 359  
 properties of, 345, 346  
 simplifying, 372  
 exponentiation, of formal power series, **powexp**, 283  
 of numbers,  $\wedge$ , 34  
 exporting, data,  
**ExportMatrix**, 115, 315  
**writedata**, 115, 116, 125, 315  
 graphics, **save**, 421  
 worksheets, in format,

- HTML (with MathML), 30  
**LATEX**, 30  
RTF, 30  
XML, 30
- ExportMatrix**, 115, 315  
  **delimiter** option, 115  
  **target** option, 115  
  **transpose** option, 115
- expression sequences, 289–292
- expressions, attributes of, 88–89  
  collecting, *see collect*  
  combining, *see combine*  
  defining attributes of,  
    **setattribute**, 88, 89, 125, 331,  
    462  
  display of large, using labels, 135,  
    482  
  using **LargeExpressions**  
    package, 187  
  expanding, *see expand*  
  freezing of, **freeze**, 172  
  generalized rational, 159–161  
  hiding of, **Veil**, 187  
  indeterminates, **indets**, 483, 486,  
    489, 491  
  internal representation of, 153–161  
  listing attributes of, **attributes**, 88  
  normalization of, *see normal*  
  reading from files, *see read*  
  replacing parts in, *see subsop*  
  removing parts in, *see remove*  
  revealing of, **Unveil**, 187  
  saving to files, *see save*  
  selecting parts in, based on,  
    criteria, *see select*  
    representation, *see op*  
  size of, **length**, 42, 70, 82, 126  
  storage of, 42, 43, 45, 47, 58, 73, 74,  
    77, 83, 154, 156, 157, 266, 290  
  substitution in, *see algsubs, subs,*  
    **subsop**, and **trigsubs**  
  surgery of, **dismantle**, 159, 174  
  testing properties of, *see is*  
  trigonometric, *see trigonometric*  
    functions  
  thawing of, **thaw**, 172  
  unhiding of, **Unveil**, 187  
  writing to streams, 106, 107,  
    117–119
- zero equivalence of, 181
- exprofile**, 205, 206
- exprseq** (data type), 84, 289
- extended greatest common divisor of  
  integers, **igcdex**, 44  
  polynomials **gcdex**, 161
- extending Maple's facilities, 326,  
  327, 392–397, 431, 432, 463,  
  691–692
- extension, of fields, 144, 178, 651–656
- external behavior, 670
- extrema, of functions, **extrema**, 9  
**extrema**, 9
- eye position, **orientation**, 443–445,  
  476
- f* and *g* series, 284
- factor**, 13, 18, 19, 21, 27, 56, 110,  
  136, 143, 144, 146, 170, 171,  
  178, 179, 184, 216, 256, 428,  
  500, 502, 505, 515, 517, 563,  
  587, 603, 608, 632, 638, 675,  
  676, 682, 683, 692
- Factor**, 143, 144, 146, 179, 180
- factored normal form, of rational  
  functions, 183
- factorial**, !, 34, 362, 378
- factorials, conversion of, into gamma  
  functions, 378  
  expansion of, 362
- factorization, absolute, **AFactor**, 180
- algorithms, 178  
  complete, **Split**, 179  
  of integers, **ifactor**, 37–39, 43, 44,  
    362  
  of polynomials, *see factor*  
  over a finite field, 178, 179  
  over algebraic function fields, 178,  
    179
- square free, **Sqrfree**, 180  
  square free, **sqrfree**, 180  
  vs. expansion, 7
- FAIL**, boolean value, 48, 340
- false**, boolean value, 48, 340
- Fast Fourier Transform, **FFT**,  
  247–249
- fclose**, 115, 117, 118, 124, 125, 127
- feof**, 124, 125, 127

- Fermat surface in the projective space, 520
- Feuerbach's nine-point circle, 480
- fflush**, 124
- FFT**, 247–249
- Fibonacci polynomials, 514
- field, electric, 434, 435, 582
- field, finite, 22, 143, 144, 178, 179, 646
- field, plotting a vector, **fieldplot**, 433
- field extensions, polynomial calculus over 143, 144, 178, 179
- linear algebra over, 651–656
- field lines, 583
- fieldplot**, 433
- File** (menu item), 35
- file, end of, **feof**, 124, 125, 127
- filepos**, 124–126
- files, appending to, **appendto**, 107, 112, 132
- automatic loading of library, 101, 109
- batch, 109–112, 393–396
- binary, 117–119
- closing of, **fclose**, 115, 117, 118, 124, 125, 127
- displaying output only, 106, 107
- end of, **feof**, 124, 125, 127
- in internal format, 106–109
- in user-readable format, 106–108
- initialization, **.mapleinit**, 109  
  **maple.ini**, 34, 109
- opening of, **fopen**, 117–119, 124–126
- reading from, **read**, 22, 96, 107, 109, 110, 112, 348, 349, 396, 422  
  **readdata**, 113, 114, 315  
  **readline**, 108, 112–114, 125–127
- removing, **fremove**, 124
- saving to, **save**, 107, 108, 110, 112, 348, 349, 421
- types of, 106
- worksheet, 29, 30, 106
- writing to, **writeto**, 106, 107, 112, 132, 134, 205, 206  
  **writedata**, 115, 116, 125, 315
- fill**, option of **Array**, 312, 313, 324  
**Matrix**, 307–309, 313, 621
- rtable**, 323, 324
- Vector**, 314, 621, 622, 624
- filled**, plot option, 426, 435, 474, 475
- filled plots, 426, 435, 474, 475
- fine-tuning Maple, 92, 93, 133–136, 275, 276, 286, 287, 317, 358
- finite fields, 22, 143, 144, 178, 179, 646
- finite Fourier series, 365, 570, 592
- finite sums, **add**, 255
- fisheye**, 3D-graphics projection, 443, 476
- fit** (statistical subpackage), 104, 468
- fitting, b-splines, **BsplineCurve**, 469
- least-squares, **leastsquare**, 468  
  **LeastSquares**, 469
- Flatten**, 299
- Float**, 47
- floating-point arithmetic, 7, 8, 46, 48, 53, 656–658
- at user-defined precision, 8, 47, 48
- precision of, 8, 47, 51
- speed of, 52, 419, 449, 656
- vs. exact arithmetic, 8, 46, 51
- vs. hardware precision, 25, 52, 449, 648, 649, 656, 657
- with hardware precision, 52, 419, 449, 648, 649
- floating-point normalization,  
  **fnormal**, 145, 248, 332, 649, 686
- floating-point numbers, 46–52
- formatted output of, 120, 121, 123, 124, 128, 135
- hardware, 51, 449, 450, 648, 649, 657
- internal representation of, 47
- fluid dynamics, 514
- fluid variable, *see* environment variable
- fnormal**, 145, 248, 332, 649, 686
- Fokker-Planck equation, 588
- foldl**, 210, 211
- foldr**, 210, 211
- Folium, 478
- font**, plot option, 410, 424, 435, 437, 438, 445, 462, 463, 473

font,  
 family,  
   **COURIER**, 473  
   **HELVETICA**, 473  
   **SYMBOL**, 473  
   **TIMES**, 473  
 style,  
   **BOLD**, 473  
   **BOLDITALIC**, 473  
   **BOLDOblique**, 473  
   **DEFAULT**, 473  
   **ITALIC**, 473  
   **OBLIQUE**, 473  
   **ROMAN**, 473  
**FONT**, graphics primitive, 422, 423, 452  
 fonts, of text in a graph, 409, 410  
**fopen**, 117–119, 124–126  
 forbidden configurations, 668  
 forcing evaluation, *see eval*  
**forget**, 97, 98, 207, 208, 276, 287, 341, 358, 364, 486, 489, 499  
   **reinitialize**, option, 358  
**FORM**, 4, 26  
 formal power series, 281–284  
 format specification,  
   of **printf**, 120, 121  
   of **sscanf**, 122, 123  
 formatted I/O, 119–127  
 formfeed, 122, 403  
 FORTRAN code generation,  
   **Fortran**, 127, 128, 131  
 FORTRAN language, 16, 25, 65, 81, 87, 127, 128  
**Fortran**, 16, 127, 128, 131  
**Fortran** option, **coercetypes**, 16  
   **declare**, 128  
   **functionprecision**, 16, 128  
   **optimize**, 128  
   **output**, 129  
   **precision**, 128, 131  
 forward automatic differentiation, 222  
**fourier**, method of **dsolve**, 530, 538  
**fourier**, 241, 246, 247, 612  
 Fourier transform, **fourier**, 241, 246, 247, 612  
 Fourier-Bessel transform, **hankel**, 241, 249  
 Fourier-Cosine transform,  
   **fouriercos**, 241, 249  
 Fourier-Sine transform, **fouriersin**, 241, 249  
**fouriercos**, method of **dsolve**, 530, 531, 538  
**fouriercos**, 241, 249  
**fouriersin**, method of **dsolve**, 530, 538  
**fouriersin**, 241  
**fprintf**, 119, 124  
**FPseries**, 562, 563  
**FPSstruct**, 562, 563  
**FTseries**, 562, 563  
**fraction**, property, 343, 350  
 fraction, *see rational number*  
 fraction-free Gaussian elimination,  
   **FractionFree** method of  
     **LUdecomposition**, 650, 654, 674, 676, 677  
**GaussianElimination**, 646, 653, 654, 674  
**frame**, **axesstyle**, 416, 426, 429, 432, 433, 443, 445–447, 456, 472, 584, 610, 611, 614  
**Framemaker**, 405  
**frames**, animation option, 477  
**framescaling**, animation option, 477  
 free variables, 66  
**freeze**, 172  
 freezing of expressions, **freeze**, 172  
**fremove**, 124  
 frequency of garbage collection,  
   **gcfreq**, kernel option, 134  
**Frobenius**, 646, 652  
**FrobeniusForm**, 646, 652  
**FromInert**, 209  
**frontend**, 161, 363, 390  
**fscanf**, 114, 119, 123, 125, 127  
**fsolve**, 32, 510, 511, 578, 600  
   option, **avoid**, 510, 511  
   **complex**, 510,  
   **fulldigits**, 510  
   **maxsols**, 510  
   options of, 510  
 full evaluation, 67, 69, 71, 73–75, 123, 304, 306, 317, 319, 320, 322, 323  
 enforcing, **eval**, 320, 321

- exceptions of, 71, 304, 319–321
  - suppressing, 75, 76, 439
- full partial fraction decomposition, **fullparfrac**, 150
- function calls, 326–328
  - macros for arguments of, 327
  - pretty printing of, 326
- functional assignments, 207
- functional relationship,
  - as a formula, 191
  - as a function, 192
  - as a procedure, 192
- functions, addition of, 210
  - additive, 391
  - algebraic, 80, 177–179, 229, 230, 233, 503
  - alphabetical listing of all built-in, **?index,function**, 40
  - anonymous, 210, 211
  - antiderivatives of, 225–234
  - approximation of, **numapprox**
    - package, 10, 149, 265, 267, 273, 276–281, 610
  - arrow notation of, 12, 193–195, 204, 208, 210
  - Bessel, 50, 237, 242–246, 362, 409, 410, 534–535
  - branches of multiple-valued, 53, 58, 190, 355, 484, 489
  - complex, 58
  - composition of, @, 210, 217
  - defining, 189–212
  - defining numerical evaluation of, 51
  - derivatives of, 213–220
  - differentiation of, **diff**, **D**, 213–220
  - distinction with formulas, 216, 217
  - drawing, *see plot*
  - elementary, 229, 230, 262
  - elliptic, 238–239
  - forgetting previous results of, *see forget*
  - generating, 12–13, 514
  - gradient of, **Gradient**, **Del**, **Nabla**, 582, 687, 688, 690, 691, 693
  - hyperbolic, 50, 356–359, 365, 366, 370, 371, 376, 377, 391–396
  - implicitly defined, 215, 217
  - index, 308, 315, 621, 624
    - indexed, 203
    - indexing, 309–311, 313, 315, 318, 324, 621
    - initially known, **?inifcns**, 49, 189
    - integration of, 225–234
    - inverse hyperbolic, 50, 376, 388, 527
    - inverse trigonometric, 50, 190, 369, 371
    - Kummer, 535
    - Lambert's *W*, 189–191, 372, 487, 489, 509, 526, 554, 557, 558
    - mapping onto lists, arrays, etc., *see map*
    - minima of, 6
    - multiple-valued, 58, 59, 190, 355
    - multiplication of, 210
    - operations on, 210
    - optional arguments of, **hasoption**, 431, 432
    - piecewise defined, **piecewise**, 195–202, 326, 439, 440
    - plotting, *see plot*
    - rational, 147, 148, 159–161, 183, 184, 187, 226–228, 273
    - recursive, 204–208
    - remember tables of, 193, 206–208, 363
    - remembering results of, 193, 206–208
    - stationary points of, 6, 9
    - subscripted, 203
    - trigonometric, 50
    - using the assume facility, 340
    - with variable number of arguments, 290
    - Whittaker, 535
    - fundamental theorem of analysis, 235
    - Galois field, 144, 180
    - Galois theory, 485
    - gamma**,  $\gamma$ , as Euler-Mascheroni's constant, 48
      - as Greek character, 16
    - GAMMA**, 49, 50, 58, 59, 266, 354, 375, 378, 456
    - gamma function, **GAMMA**, 49, 50, 58, 59, 266, 354, 375, 378, 456
      - definition of, 49

conversion into factorial, 378  
 plot of, 58, 59, 455, 456  
 simplification of, 375  
**GAP**, 3  
 garbage collection, 134  
**Gausselim**, 646, 655  
 Gaussian elimination,  
     **GaussianElimination**,  
     **LUDecomposition**, 31, 646,  
     649, 650, 653, 674  
     fraction-free, **FractionFree** method  
         of **LUDecomposition**, 650,  
         654, 674, 676, 677  
**GaussianElimination**, 646, 653,  
     654, 674  
 Gaussian integers, 143, 464  
**GaussianInteger**, property, 344, 346  
**GaussInt** package, 464  
 Gauss-Jordan form,  
     **ReducedRowEchelonForm**,  
     642, 646, 674  
**Gaussjord**, 646, 655  
**RREF** method of  
     **LUDecomposition**, 647, 654  
**Gaussjord**, 646, 655  
**gbasis**, 383, 501, 505, 507  
**gc**, 134  
**gcd**, 23, 143  
 gcd, *see* greatest common divisor  
**Gcd**, 144  
**gcdex**, 161  
**gcfreq**, kernel option, 134  
**GEAR**, 577  
 Gear extrapolation method, 576  
 general purpose simplification,  
     **simplify**, 370–375  
 general relativity, 3  
 generalized inverse of a matrix,  
     **MatrixInverse**, 693, 694  
 generalized rational expressions,  
     159–161  
 generalized series expansions, **series**,  
     268  
 generating functions, 12, 13, 514  
 generic problems vs. specialized  
     problems, 173, 174, 335, 361  
**genfunc** package, 13, 517  
     **rgf\_encode**, 13  
**GenerateEquations**, 658  
**GenerateMatrix**, 625, 626  
 geocentric Cartesian coordinates, 506  
 geocentric reference ellipsoid, 506  
 geodesy, 506–508, 700  
 geodetic coordinates, 506  
 geometrical drawings, 436–438  
**geometry** package, 436–438  
     **AreCollinear**, 438  
     **circle**, 436  
     **circumcircle**, 437  
     **draw**, 437, 438  
     **IsOnCircle**, 436, 437  
     **line**, 436, 438  
     **midpoint**, 436  
     **point**, 436, 437  
     **projection**, 436, 437  
     **triangle**, 436, 437  
**GetCoordinateParameters**, 690  
**getenv**, 137  
**gif**, graphics output, 406  
**GivensRotationMatrix**, 624  
**global**, declaration, 201, 202  
 global vs. local variables, 194, 195,  
     201–203  
 golden ratio, 194  
 good extension of term ordering, 722  
 Gosper’s summation method, 256,  
     257  
 graded lexicographic ordering, 704,  
     707, 708  
 graded reverse lexicographic ordering,  
     *see* total degree ordering  
**Gradient**, 582, 687, 688, 690, 691,  
     693  
 gradient of function, **Gradient**, 582,  
     687, 688, 690, 691, 693  
 Gradshteyn–Ryzhik, 17–19, 263  
**GramSchmidt**, 645, 686  
 GramSchmidt orthogonalization,  
     **GramSchmidt**, 645, 686  
**graph**, 726  
 graph coloring, 726–729  
 graphics, 2D, 403–440  
     3D, 441–458  
     2D-plot specialties, 426–436  
     2D-plot style, 410–412, 417, 475  
     3D-plot specialties, 452–459  
     3D-plot style, 442, 476, 477  
     adding a title in, 409, 410, 474

- adding text in, 410
- altering, 424–426
- angle of view in 3D, 443–445, 476
- animated, **animate**, **animate3d**, **animatecurve**, 469–471
- array, 425, 427, 445
- axes in, **axes**, 416, 417, 443, 472
- camera position in 3D, 443–445, 476
- changing colors of, **color**, 412–415, 472
- colors in, 412–415, 472
- combining into, *see also display*
  - a graphics array, **display**, 427, 445, 470, 471, 477
  - a single picture, 426, 427, 431, 435, 445, 454, 461, 462, 470
  - an animation, 471
- constrained**, scaling option, 407, 473
- contour, **contourplot**, 434, 435
- contour lines in 3D, **contours**, 447, 475
- coordinate systems, in 2D, 415, 416, 472
  - in 3D, 453, 454, 472
- creating a legend in, 435
  - through **legend** option, 417, 418, 461, 474
- data structures of, 418, 419, 422, 423, 448, 449, 451
- density, **densityplot**, **densityplot3d**, 435, 436
- display devices, **plotsetup**, 403, 405, 406
- drawing formulas in, 403
- drawing functions, 404, 405
- drawing geometrical, **draw**, 437, 438
- evaluation in, 439–441
- eye position in 3D, **orientation**, 443–445, 476
- filling of, **filled**, 426, 435, 474, 475
- grid in 3D, **grid**, 445, 475
- hardware arithmetic in, 51, 52, 450
- headings in, **title**, 409, 410, 474
- horizontal range of, **view**, 408
  - in (semi-, double-) logarithmic scaling, **semilogplot**, **loglogplot**, **logplot**, 429–432
- in plane geometry, **draw**, 437, 438
- labels in, 409, 410, 473
- legend in, 417, 418, 435, 461, 474
- lighting in 3D, **light**, 446, 447, 476
- line styles in, **linestyle**, 412, 413, 417, 473
- line thickness in, **thickness**, 412, 417, 474
- low-level 2D-objects, 418, 419, 422, 423
- low-level 2D-plot primitives, 422, 424
- low-level 3D-objects, 448, 449
- low-level 3D-plot primitives, 451
- manipulating objects in, 422, 424, 425
- of an electric field, 434, 435, 582
- orientation in 3D, **orientation**, 443–445, 476
- perspective in 3D, **projection**, 443, 476
- plane algebraic curves, 406, 427, 428, 470, 471
- plot aliasing in, 438, 439, 445
- PostScript output of, 405
- printing of, 403, 405
- range of, 405, 407, 408
- reading from file, **read**, 422
- redrawing in, 419, 420
- removing discontinuities in, **discont**, 408, 474
- resetting plot device, **plotsetup**, 403, 405, 406
- resolution of, **resolution**, 419, 474
- rotating, **rotate**, 424, 425
- saving, **save**, 421
- scaling of, **scaling**, 407, 473
- screen dump of worksheet with, 6, 404
- setting default options of, **setoptions**, **setoptions3d**, 416, 447
- shading of 3D, **shading**, 442, 443, 476
- structure of, 418–422, 448–452

- style of lines in, `linestyle`, 412,  
     413, 417, 473  
 survey of, 401, 402  
 thickness of lines in, `thickness`,  
     412, 417, 474  
 tick marks in, `tickmarks`, 409, 474  
 title in, `title`, 409, 410, 474  
 transforming, `transform`, 424, 425  
 translating, `translate`, 424, 425  
 tubes in 3D, `tubeplot`, 454  
**unconstrained**, scaling option,  
     407, 473  
 vertical range of, 407  
 view frame in 3D, `view`, 408, 446,  
     474  
 view of, `view`, 408, 446, 474  
 without border lines, 405  
 zooming in, 420  
**grave accents**, 80, 81, 97  
**gravitational term**, 571  
**greater-than symbol**, as prompt, 6,  
     33, 107  
     as relational operator, 84  
**greatest common divisor**, of integers,  
     `igcd`, `igcdex`, 21, 44, 308  
     of polynomials, `gcd`, `gcdex`, 23,  
         143, 161  
**Greek characters**, 16, 77  
**grid**, plot option, 445, 475  
**GRID**, graphics primitive, 449, 451  
 grid points, 442, 449–451  
**GRIDSTYLE**, graphics primitive, 452  
**Groebner** package, 168, 383, 384,  
     501–508, 719–745  
**fglm**, 731, 732  
**gbasis**, 383, 501, 505, 507  
**gsolve**, 504–506, 736, 737  
**hilbertdim**, 721, 725  
**hilbertpoly**, 722–724  
**hilbertseries**, 721, 724, 725  
**is\_finite**, 506, 729  
**is\_solvable**, 725, 726, 728  
**leadcoeff**, 710  
**leadmon**, 710, 730–732  
**leadterm**, 710  
**normalf**, 384, 710–713, 715, 731,  
     743  
**spoly**, 713, 715  
**termorder**, 706–709, 731, 744,  
     745  
**testorder**, 706–709  
**univpoly**, 508, 735  
 Gröbner basis, 168, 383, 384, 501–508,  
     697–745  
**algorithm**  
     Buchberger's method, 714–716  
     improved Buchberger's method,  
         716–719  
     Faugère-Gianni-Lazard-Mora  
         method, 717, 731, 732  
**application** of, 719–745  
     counting of finite solutions,  
         730–732  
     dimension, 721  
     equivalence of systems of  
         polynomial equations, 720, 721  
     finding a univariate polynomial,  
         508, 734, 735  
     finite solvability, 729  
     Hilbert polynomial, Hilbert  
         series, 721–725  
     solvability, 725, 726  
     simplification of expressions, 742,  
         743  
     basics, 703–719  
     characterization, 712, 713  
     computing of, `gbasis`, 716  
     minimal, 715  
     reduced, 715  
**group theory**, 3  
**gsolve**, 504–506, 736, 737  
  
**Hamiltonian**, 579  
     matrix, 681  
**handling**, discontinuities, `discont`,  
     196, 236, 408, 421, 474  
     errors, 130, 204, 431  
**hankel**, 241, 249  
 Hankel matrix, `HankelMatrix`, 624  
 Hankel transform, `hankel` 241, 249  
**HankelMatrix**, 624  
 hardware floating-point numbers, 51,  
     449, 450, 648, 649, 657  
 harmonic oscillator, 564–566  
     energy levels of, 565  
**hasassumptions**, 89, 337–339, 349  
 hash sign `#`, 9, 97

- hash tables, 316
- hasoption**, 431, 432
- hastype**, 85, 86
- headings, in graphics, 409, 410, 474
- heat equation, 601, 609–615
- Heaviside**, 199, 242, 246, 249, 421, 422, 478, 529
- Heaviside step function, *see* **Heaviside**
- helicoid, 453
- Helmert’s projection, 506, 700
- help**, 37
- help system, 36–41
  - help**, 37
  - info**, 37, 38
  - synopsisusage**, 38, 39
  - text search, 40
  - topic search, 39
  - related topics, **related**, 39
- help windows, samples of, 36, 37
- Hensel lifting, 178
- Hermite**, 646, 653
- Hermite polynomial, **HermiteH**, 181, 566
- Hermite reduction, 231
- HermiteForm**, 620, 626, 646, 653
- HermiteH**, 181, 566
- hermitian**, special indexing, 310
- hermitian product of vectors, 314, 315
- Hermitian transpose of matrix, **HermiteanTranspose**, 641
- HermiteanTranspose**, 641
- Hessenberg**, special indexing, 310
- HessenbergForm**, 646
- Hessian**, 688
- Hessian matrix, **Hessian**, 688
- Heun’s method, 576
- hidden**, option of **plot3d**, 442, 477
- hidden surface removal, **hidden**, 442, 477
- hierarchy of,
  - functional properties, 351
  - miscellaneous properties, 352
  - numerical properties, 350
  - properties on matrices, 351
- high-energy physics, 3
- hilbert**, 243, 250
- Hilbert function, affine, 721, 723
  - projective, 723
- Hilbert matrix, **HilbertMatrix**, 309, 624
- Hilbert polynomial, affine, 722
  - projective, 723
- Hilbert transform, **hilbert** 243, 250
- Hilbert series, **hilbertseries**, 721, 724
- Hilbert-Serre theorem, 723
- hilbertdim**, 721, 725
- HilbertMatrix**, 309, 624
- hilbertpoly**, 722–724
- hilbertseries**, 721, 724, 725
- HINT**, option of, **pdsolve**, 602, 606–609
  - symgen**, 550
- histogram**, 462, 463, 465
- Hoeij’s knapsack algorithm, 178
- home directory, 108
- homogenization, 722, 723
- horizontal tabulates, \t, 122
- horner**, option of **convert**, 148, 182
- Horner form, of polynomials, 148, 149, 182, 183, 276
- hornerform**, 149, 276, 279
- Horowitz-Ostrogradsky reduction, 228
- HouseholderMatrix**, 624
- HSV**, color space, 413, 414, 472
- HUE**, color chart, 414, 415
  - color space, 414, 472, 475
- Hückel matrix, 681
  - theory, 680–687
- hybrid algorithmic structure, 31, 32, 353
- hyperbolic coordinates, 688
- hyperbolic functions, 50, 356–359, 365, 366, 370, 371, 376, 377, 391–396
  - combining, 365, 366
  - conversion into exponential forms, 376, 377
  - conversion into hyperbolic sines and cosines, 377
  - expansion of, 357, 359, 391–397, 605
  - simplification of, 370, 371
- hypercylindrical coordinates, 689
- hypergeom**, 50, 375, 378

**hypergeometric**, option of **convert**, 535  
 option of **dsolve**, 535  
**Hypergeometric**, 258  
 hypergeometric function,  
   **hypergeom**, 50, 375, 378  
 conversion into standard function, 378, 536  
 simplification of, 375  
 hypergeometric identities, 257, 258  
 hyperspherical coordinates, 13, 14

**I**,  $\sqrt{-1}$ , 58, 62, 78  
 $i$ , complex root,  $\sqrt{-1}$ , 58, 62, 78  
 I/O, at user level, 106–116  
   exporting data, **ExportMatrix**, 115, 315  
   formatted, 119–127  
   importing data,  
     **importdata**, 114, 466, 468  
     **ImportMatrix**, 114, 115, 315, 466, 467, 629  
   low-level, 116–127  
   of binary data, **readbytes**,  
     **writebytes**, 118  
   reading data, **readdata**, 113, 114, 315  
   streams, 106, 107, 117–119  
   writing data, **writedata**, 115, 116, 125, 315

ideal, generated by polynomials, 703  
 identifiability,  
   Markov parameter  
     matrix approach to, 672  
   similarity-transformation  
     approach to, 673, 678–680, 701  
   transfer function approach to, 672, 677, 678, 701

identifiable, structurally globally, 672, 701, 733, 734  
   structurally locally, 671  
 identifiers, 77, 80  
**identity**, special indexing, 310, 324  
 identity matrix, 310, 324, 624, 633, 642, 674, 681  
 identity test of,  
   expressions **testeq**, 44, 142, 245, 497, 498, 524

matrices, **Equal**, 642, 643, 647, 648, 651–653, 656, 694  
**IdentityMatrix**, 624, 631, 642  
**ifactor**, 37–39, 43, 44, 362  
**iFFT**, 247, 249  
**igcd**, 21, 44, 308  
**igcdex**, 44  
**Im**, 58, 336, 356, 484  
 imaginary part, of complex numbers, **Im**, 58, 336, 356, 484  
 imaginary unit, **I**, 58, 62, 78  
 $J$  as, 62, 492, 493  
**imaginearunit**, interface variable, 62, 430, 432, 492  
**implicit**, option of **dsolve**, 525, 538  
   option of **Eigenvalues** and  
     **EigenVectors**, 637–639, 683  
 implicit differentiation,  
**implicitdiff**, 215–220  
 implicit functions, plotting of  
**implicitplot**,  
**implicitplot3d**, 427, 428, 455  
**implicitdiff**, 215, 219, 220  
 implicitization of parametric objects, 740–742  
**implicitplot**, 428, 429  
**implicitplot3d**, 455  
**importdata**, 114, 466, 468  
 importing data, 113–115, 125–127, 315, 466–468, 629  
**ImportMatrix**, 114, 115, 315, 466, 467, 629  
   **delimiter** option, 467  
   **transpose** option of, 467  
   **source** option, 115, 467  
 impulse response function, 670  
 incomplete beta function, 203  
 indefinite integration, **integrate**, **int**, 225–234  
**independent**, option of **allvalues**, 497  
 indeterminates, in expressions, **indets**, 483, 486, 489, 491  
**indets**, 483, 486, 489, 491  
**index**, option of **RootOf**, 57  
 index function, 308, 315, 621, 624  
 index of regularity, 722  
**index/newtable**, 317  
 indexed functions, 203

- indexed names, 68, 69, 79, 84, 85, 203, 316, 341, 673
  - assignment of, 316
- indexing function, 309–311, 313, 315, 318, 324, 621
  - antihermitean**, 310
  - antisymmetric**, 310, 311, 313, 622
  - band**, 310
  - constant**, 310
  - creation of, 311–313, 317
  - diagonal**, 309, 310
  - hermitian**, 310
  - Hessenberg**, 310
  - identity**, 310, 324
  - scalar**, 310
  - symmetric**, 14, 20, 309–311, 318, 629, 634, 637
  - triangular**, 309, 310, 324, 629, 655
  - unit**, 310
  - using more than one, 309–310, 313
  - zero**, 310
- indices**, 316, 318, 462, 463, 534
- indices, applying functions to parts of expressions with specific, **applyop**, 170, 171, 211
  - of tables, **indices**, 316, 318, 462, 463, 534
- selecting parts of expressions with specific, *see op*
- substituting parts of expressions with specific, **subsop**, 169–171, 208, 297, 317, 327, 486, 517
- inequalities, linear 508, 509,
- inert forms, 209, 213, 226, 234, 241, 285
- infinite evaluation, 68, 75
  - loops, 68, 75
  - recursion, 68, 75
- infinitesimal, *see Lie point symmetry*
- infinity**, 48, 355
- infnorm**, 277–279, 610
- info**, 37
- infolevel**, 98
  - all**, 98
  - dsolve**, 525, 526, 532, 536
  - integrate**, 99, 227, 233, 234, 251
  - invlaplace**, 243
  - LinearAlgebra**, 648, 656
- plot**, 420
- rsolve**, 516
- solve**, 192, 193, 486
- init**, initialization file, 109
- initial value problem, 527–529, 560, 566, 568, 577
  - derivatives in, 529
  - problems with aliases in, 528
- initialization files, 34, 35, 62, 109, 163, 416
- initially known functions, listing of, **?inifcns**, 49, 189
- inner product, **DotProduct**, 632, 645
- inplace**, option in matrix arithmetic, 659, 660
- input, editing of, 35, 96
  - files containing, 106–110
  - termination of, 6, 34, 96
- insequence**, option of **display**, 470, 471, 477
- int**, **Int**, *see integrate, Integrate*
- integers, 41–45
  - arithmetic of, 41, 42
  - base of, 42
  - big, 42, 43
  - complex, 143, 464
  - division with remainder of, **iquo**, 44
  - even, 89, 342, 343
  - factorization of, **ifactor**, 37–39, 43, 44, 362
  - Gaussian, 143, 464
  - greatest common divisor of, **igcd**, **igcdex**, 21, 44, 308
  - internal representation of, 42, 43
  - multiple-precision, 42, 43
  - odd, 89, 342, 343
  - operations on, 43
  - primality test of, **isprime**, 43, 294
  - remainder of division of, **irem**, 44
  - single precision, 41
  - size limitation of, 42, 43
- integral, Chebyshev, 232
  - exponential, **Ei**, 229
  - Sine, **Si**, 478
- integral equations, 245
- integral tables, 17, 263
- integral transforms, 241–251, 529, 538

adding, 246  
**Fourier, Fourier**, 241, 246, 247, 612  
**Fourier-Bessel, hankel**, 241, 250  
**Fourier-Cosine, fouriercos**, 241, 249  
**Fourier-Sine, fouriersin**, 241  
**Hankel, hankel**, 241, 249  
**Hilbert, hilbert** 241, 250  
**Laplace, laplace**, 241–246  
**Mellin, mellin**, 241, 250  
**integrals, elliptic**, 238, 239  
**integrate, Integrate**, 9, 17, 19, 26–28, 62, 72, 99, 100, 133, 149, 172, 173, 200, 226–228, 230–241, 243–245, 250–255, 610  
 continuous option, 235  
 discont option, 236  
 integrating factor, **intfactor**, 547, 560  
 integration, 19  
 algorithms,  
     Lazard/Rioboo/Trager, 228  
     heuristic, 227  
     Horowitz-Ostrogradsky  
         reduction, 228  
     Risch, 225, 228, 229  
     Risch-Norman, 230  
     Rothstein/Trager, 228  
     Trager, 230  
 by parts, **Rule[parts, ...]**, 253  
 change of variables in,  
     **Rule[change, ...]**, 252  
 constant of, 226  
 contour, 253  
 definite, 234–239  
 definite vs. indefinite plus  
     limits, 235  
 elliptic, 238, 239  
 indefinite, 225–234  
 Monte Carlo method, **\_MonteCarlo**, 240  
 multidimensional, 240  
 numerical, 239–241  
     adaptive Gaussian quadrature  
         method, **\_Gquad**, 32, 329  
     adaptive sinc method, **\_Sinc**, 32, 329  
     Clenshaw-Curtis quadrature  
         method, **\_CCquad**, 32, 239  
         double exponential method,  
             **\_Dexp**, 32, 239, 240  
     Monte Carlo method,  
         **\_MonteCarlo**, 240  
     NAG routine, **\_d01akc**, 240  
     Newton-Cotes method, **\_NCrule**, 32, 239  
     of formal power series, **powint**, 283  
     of rational functions, 226–229  
     of algebraic functions, 230, 232  
**interactive, Interactive**, 402, 403  
**Interactive**, 469  
**interface, displayprecision**, 47, 135, 136, 276, 566, 570, 639, 685  
 echo, 107, 137  
 errorbreak, 136, 138  
 imaginaryunit, 62, 430, 432, 492  
 labeling, 135  
 prettyprint, interface variable, 101, 135  
 prompt, 135  
 quiet, 132, 134  
 rtablesize, 323  
 screenwidth, 132  
 showassumed, 89, 91, 92, 136, 242, 333, 436, 687  
 verboseproc, 135, 136, 206, 345, 357, 391  
 warnlevel, 583  
 intermediate expression swell, 24  
 internal address ordering, 292  
 internal Maple format, 106, 107, 109  
 internal representation of,  
     floating-point numbers, 47  
 graphics, 418, 419, 422, 423, 448, 449, 451  
 integers, 42, 43  
 lists, 295  
 polynomials, 153–159  
 rational functions, 159–161  
 rational numbers, 45  
 sequences, 290  
 series, 266  
 sets, 292  
 tables, 319  
 interrupt key, 35

- interrupting calculations, 35
- intersect**, 293
- intersection of sets, **intersect**, 293
- IntersectionBasis**, 645, 646
- intfactor**, 547, 560
- inttrans** package, *see* integral transforms
- invariance, of ODEs, 542, of PDEs, 616
- inverse**, 283
- inverse, Fast Fourier Transform, **iFFT**, 247, 249
- Fourier transform, **invfourier**, 246, 247, 613
- Laplace transform, **invlaplace**, 242–245
- Mellin transform, **invmellin**, 250
- of *z*-transforms, **invztrans**, 517
- inverse, of matrix, **MatrixInverse**, 15, 633, 641, 643, 657, 693, 694
- of polynomial mapping, 22, 742
- of power series, **inverse**, 282
- inverse hyperbolic functions, 50, 376, 388, 527
- conversion into logarithmic expressions, 376
- simplification of, 371
- inverse kinematics, of ROMIN robot, 699, 700, 739, 740
- inverse trigonometric functions, 50, 190, 369, 371
- conversion into logarithmic forms, 376
- simplification of, 371
- invfourier**, 246, 247, 613
- invlaplace**, 242–245
- invoking Maple, 33
- invztrans**, 517
- iostatus**, 119
- iquo**, 44
- IRb-6 robot, 695
- irem**, 44
- irrational numbers, 46–57
- is**, 197, 198, 398, 340, 341, 345, 347, 484
- is\_finite**, 506, 729
- is\_solvable**, 725, 726, 728
- IsDefinite**, 642, 648
- isolate**, 245, 525, 549, 563
- isolve**, 512, 513, 561
- isOnCircle**, 436, 437
- IsOrthogonal**, 642, 644
- isprime**, 43, 152, 294
- isqrt**, 43
- IsSimilar**, 642, 644
- IsUnitary**, 642
- IsZero**, 642
- iterated cosine, 411, 412
- i<sup>th</sup>* prime number, **ithprime**, 75, 76, 291, 315, 411
- Jacobi's notation of elliptic integrals, 238
- Jacobian conjecture, 742
- Jacobian, matrix, **Jacobian**, 666, 688, 691, 692
- JACOBIAN**, 131
- Jacobian**, 666, 688, 691, 692
- JACOBIAN**, code generation, 131
- Java, code generation, **Java**, 129–131
- Java**, 129–131
- Java** option, **defaulttype**, 130
  - optimize**, 129, 130
  - returnvariablename**, 130
- joining points, in graphics, 459, 460
- Jordan block matrix,
  - JordanBlockMatrix**, 624, 642
- Jordan form, **JordanForm**, 173, 174, 643, 646
- Josephson's junction circuit, 288
- jpeg**, graphics output, 405
- Julia sets, 450, 451
- jumps, in antiderivatives, 236
- Kalman's criterion for
  - controllability, 671
  - observability, 671
- KANT, 3
- Karatsuba multiplication,
  - complexity of, 515
- Kepler's equation, 272, 288
- kernel, for integral transforms, 241
  - of Maple, 29, 30
  - of matrix, **NullSpace**, 635, 641, 645
- kernel options, **kernelopts**, 133, 134, 205, 223, 224

- kernelopts**, `bytesalloc`, 134  
   `bytesused`, 134, 223, 224  
   `cputime`, 134  
   `gcfreq`, 134  
   `maxdigits`, 42  
   `printbytes`, 133, 134  
   `profile`, 205, 206  
   `wordsize`, 134
- KernelOpts**, procedure from `Maplets` package, 134
- key**, control, 35  
   interrupt, 35
- keyword search**, 37–39
- kidney**, 669, 670
- kink angle**, in 2D plots, 419, 449
- Kirchhoff's laws**, 492
- Klein's bottle**, 479
- knot**, plot of, 455
- Knuth**, 516
- Korteweg-de Vries equation**, 601, 615–617
- Kovacic's algorithm**, 531, 533, 534
- Kronecker-Trager factorization algorithm**, 178
- label**, option of `RootOf`, 57
- labeldirections**, plot option, 461, 473
- labelfont**, plot option, 473, 612
- labels**, plot option, 413, 429–431, 451, 460–463, 473, 612
- labels**, in formulas, 135, 482  
   in graphics, 409, 410, 473  
   selecting, 482
- Lagrange multipliers**, 746
- Lagrangian function**, 569, 571
- LaguerreL**, 375
- Laguerre polynomial**, `LaguerreL`, 375
- Lambert's W function**, *see* `LambertW`  
   branches of, 190, 489  
   simplification of, 372
- LambertW**, 189–191, 372, 487, 489, 509, 526, 554, 557, 558
- laplace**, option of `dsolve`, 529, 531, 538, 592–594
- laplace**, integral transform, 241–246
- Laplace equation**, 224, 587, 588, 601, 607, 608
- Laplace transform**, `laplace`, 241–246
- Laplace-Beltrami operator**, 13, 219, 687, 688, 693
- Laplacian**, 687, 688, 693
- Laplacian**, `Laplacian`, 687, 688, 693
- LargeExpressions** package, 187  
   `Unveal`, 187  
   `Veal`, 187
- last name evaluation**, 319–321
- last result**, %, 17, 46, 79, 83, 101, 208
- LATEX**, 15, 30, 77, 78, 91, 132, 133
- latex**, 15, 77, 78, 132
- latitude**, 506, 507
- launching Maple**, 6, 33
- laurent**, 267
- Laurent series**, `series`, 266  
   `laurent`, 267
- layout**, of large results,  
   using labels, 135, 482  
   using the `LargeExpressions` package, 187
- of plots**, `plotsetup`, 403, 405, 406
- Lazard/Rioboo/Trager improvement**, 228
- lcoeff**, 140–142
- leadterm**, 710
- leading coefficient**, of polynomial, `lcoeff`, 140–142  
   `leadcoeff`, 710
- leading monomial**, `leadmon`, 710, 730–732
- leading term**, of polynomial, `leadterm`, 710  
   of series, `leadterm`, 271, 286
- leadmon**, 710, 730–732
- leadterm**, 271, 286, 710
- leaf**, Maple, 406, 458
- least-squares fit**, `leastsquare`, 468  
   `LeastSquares`, 469
- leaving Maple**, `quit`, 35
- left**, option of `limit`, 285
- left quotes**, 80, 81, 97
- left-adjusted format**, 101
- left-hand sides**, of equations, *see* `lhs`
- legend**, plot option 417, 418, 461, 474

- Legendre polynomial, **LegendreP**, 212  
 Legendre's form, of elliptic integrals, 238  
**LegendreCK**, 238  
**LegendreE**, 238  
**LegendreEc**, 238  
**LegendreEc1**, 238  
**LegendreF**, 238  
**LegendreKc**, 238  
**LegendreKc1**, 238  
**LegendrePi**, 238  
**LegendrePic**, 238  
**LegendrePic1**, 238  
 Lemniscate of Bernoulli, 478  
**length**, 42, 70, 82, 126  
 length limitations, 42, 43, 77, 186  
 Lenstra, factorization algorithm, 178  
 Levin's *u*-transform, 258  
**lexdeg**, 704, 706, 709  
 lexicographic ordering, **plex**, 145, 383, 501, 507, 704, 706, 711–713, 715, 716, 720, 729–733, 735, 736, 738, 740–745  
**lhs**, left-hand sides of equations, 233, 252–254, 378, 383, 491, 499, 501, 504, 517, 543, 553, 557, 572, 587–589, 591–594, 596–598, 603  
**libname**, 70, 82, 111, 112, 395, 396  
 library, creating your own, 111, 112, 391–397  
 packages, 101  
 private, 101, 111, 112, 391–397  
 share, 29, 101, 105  
 standard, 29, 30, 101  
 update, 101  
 LibraryTools package, 111, 396  
**SaveToLibrary**, 396  
**LiE**, 3  
 Lie point symmetry, 544–551  
 commutator, **mult**, 559  
 computation of determining system  
**determine**, 616  
**eta\_k**, 541, 542, 555  
**odepde**, 542–545, 551, 560  
 contact transformation, 542, 551  
 definition of, 539, 615, 616  
 dynamical, 542  
 prolongation, 542, 551  
 Lie symmetry method, 544–551, 615–618  
 canonical variables, **canoni**, 548, 553, 560  
 contact transformation, 542, 555  
 determining system,  
 computing,  
**determine**, 616  
**eta\_k**, 541, 542, 555  
**odepde**, 542–545, 551, 560  
 definition of, 542, 616  
 simplification of, **autosimp**, 616  
 solving, **syngen**, 546, 549, 550, 552, 555, 558, 560  
 validating solutions of, **symtest**, 546, 552, 560  
 dynamical symmetry, 542  
 integrating factor, **intfactor**, 547, 560  
 list of methods for computing symmetries, '**syngen/methods**', 550  
 prolongation, 541, 542  
 computing, **eta\_k**, 541, 555, 560  
 Lie symmetry package, **liesymm**, 615–618  
 Lie theory, 3, 12, 16  
**liesymm** package 615–618  
**autosimp**, 616  
**determine**, 616  
**light**, plot option, 446, 447, 476  
 lighting, background, **ambientlight**, 446, 475  
 models, **lightmodel**, 476  
 sources, **light**, 446, 447, 476  
**lightmodel**, plot option, 476  
**limit**, 10, 50, 152, 285–287, 336–338, 354, 491  
 option, **complex**, 285  
**left**, 285  
**real**, 285  
**right**, 285  
**Limit**, 285  
 limit cycle, 581, 590, 595  
 limitations, of computer algebra, 23–29  
 of computing time, **timelimit**, 136

- of size of expressions, 42, 43, 77, 186
- limits, **limit**, 10, 50, 152, 285–287, 336–338, 354, 491
  - assisting the computation of, 286, 287, 336–338,
  - bidirectional, 285
  - directional, 285
- line**, 424, 436, 438
- line**, style option, 410, 442, 475, 477
- line continuation, 35
- line feed, 34
- linear damped oscillator, 529, 530
  - general solution of, 529
  - with underdamping, 529
  - with various damping factors, 530
- linear equations, 492–495, 625, 626, 650, 658
- linear inequalities, 508, 509
- linear output, *see lprint*
- linear systems, controllability of, 671, 675, 676
  - external behavior of, 670
  - impulse response function of, 670
  - observability of, 671, 676
  - structural identifiability of, 671–680
- LinearAlgebra** package, 619–662
  - Add**, 630
  - Adjoint**, 641
  - BandMatrix**, 624
  - Basis**, 645
  - BezoutMatrix**, 624
  - BidiagonalForm**, 646
  - CharacteristicMatrix**, 641, 682
  - CharacteristicPolynomial**, 635, 638, 641, 682
  - Column**, 641
  - ColumnDimension**, 641, 693
  - ColumnOperation**, 641
  - ColumnSpace**, 635, 645
  - CompanionMatrix**, 624
  - ConditionNumber**, 641
  - CrossProduct**, 645
  - DeleteColumn**, 641
  - DeleteRow**, 641
  - Determinant**, 15, 20, 102–105, 559, 634, 638, 641, 668, 675, 676, 683
  - DiagonalMatrix**, 624, 625
  - Dimension**, 641
  - DotProduct**, 632, 645
  - Eigenvalues**, 635–638, 641, 683, 685
    - implicit**, option, 637, 683
    - output=list**, option, 638, 685
  - Eigenvectors**, 635–641, 683–685
    - implicit**, option, 637–639, 683
    - output=list**, option, 636, 639, 685
  - Equal**, 642, 643, 647, 648, 651–653, 656, 694
  - Frobenius**, 646, 652
  - FrobeniusForm**, 646, 652
  - GaussianElimination**, 646, 653, 654, 674
  - GenerateEquations**, 658
  - GenerateMatrix**, 625, 626
  - GivensRotationMatrix**, 624
  - GramSchmidt**, 645, 686
  - HankelMatrix**, 624
  - Hermite**, 646, 653
  - HermiteForm**, 620, 626, 646, 653
  - HermitianTranspose**, 641
  - HessenbergForm**, 646
  - HilbertMatrix**, 309, 624
  - HouseholderMatrix**, 624
  - IdentityMatrix**, 624, 631, 642
  - IntersectionBasis**, 645, 646
  - IsDefinite**, 642, 648
  - IsOrthogonal**, 642, 644
  - IsSimilar**, 642, 644
  - IsUnitary**, 642
  - IsZero**, 642
  - JordanBlockMatrix**, 624, 642
  - JordanForm**, 173, 174, 643, 646
  - LinearSolve**, 626, 650, 658
  - LUDecomposition**, 646–650, 654, 655, 657, 658, 674, 676, 677
    - method, **Cholesky**, 648
    - FractionFree**, 650, 654, 674, 676, 677
    - NAG**, 650, 655, 658, 659
    - RREF**, 647, 654
  - Map**, 633, 639, 653
  - MatrixAdd**, 630, 631
  - MatrixInverse**, 15, 633, 641, 643, 657, 693, 694
  - MatrixMatrixMultiply**, 631

- MatrixNorm**, 103, 104, 645  
**MatrixOptions**, 622, 659  
**MinimalPolynomial**, 635, 641  
**Minor**, 32, 641  
**Modular** subpackage, 104, 655  
**Multiply**, 631, 660  
**Norm**, 41  
**Normalize**, 645, 686  
**NullSpace**, 635, 641, 645  
**Permanent**, 641  
**PopovForm**, 646  
**QRDecomposition**, 646  
**RandomMatrix**, 626, 628, 629, 693, 694  
**RandomVector**, 628  
**Rank**, 103, 634, 641, 674  
**ReducedRowEchelonForm**, 642, 646, 674  
**Row**, 641  
**RowDimension**, 641, 677  
**RowOperation**, 641, 668  
**RowSpace**, 635, 645  
**ScalarMatrix**, 677, 682, 693  
**ScalarMultiply**, 14, 630  
**SchurForm**, 646  
**SingularValues**, 641, 685  
**SmithForm**, 646, 652  
**SubMatrix**, 16, 626, 641, 643, 665, 677  
**SubVector**, 626, 641, 665  
**SumBasis**, 645, 646  
**SylvesterMatrix**, 624  
**ToeplitzMatrix**, 624, 629, 634, 636  
**Trace**, 14, 634, 641  
**Transpose**, 14, 632, 641, 644, 648, 649, 667, 676, 682, 686, 691–693  
**TridiagonalForm**, 646, 648, 649  
**UnitVector**, 624  
**VandermondeMatrix**, 624  
**VectorAngle**, 645  
**VectorNorm**, 645  
**ZeroMatrix**, 645  
**ZeroVector**, 645  
**LinearSolve**, 626, 650, 658  
lines, commands across several, 34, 35  
lines in diagrams, color of, *see* **color**  
dashing of, *see* **linestyle**  
in contour plots, 435  
plotting 3D, **spacecurve**, 452, 574, 575  
points to sample on, *see* **numpoints**  
styles of, *see* **linestyle**  
thickness of, *see* **thickness**  
**linestyle**, plot option, 10, 412, 413, 417, 424, 438, 447, 461, 473, 578  
linkage system  $M_6$ , 736–739  
Liouville’s principle, 229  
Liouvillian functions, 229  
Liouvillian solutions, of 2nd order ODEs, 531  
Lissajous figures, 478  
LISP, 3  
list plots, 459–465  
**histogram**, 462, 465, 466  
**listcontplot**, 459  
**listcontplot3d**, 459  
**listdensityplot**, 459, 464, 465  
**listplot**, 459–461, 615  
**listplot3d**, 459  
**matrixplot**, 462, 463  
**pointplot**, 459–461  
**pointplot3d**, 459  
pie charts, 463  
**listcontplot**, 459  
**listcontplot3d**, 459  
**listdensityplot**, 459, 464, 465  
**listplot**, 459–461, 615  
**listplot3d**, 459  
**listprocedure**, option of **dsolve**, 573, 574, 578  
lists, 294–300  
adding elements to, 297  
appending elements to, 297  
applying functions to, **map**, **map2**, 210, 211  
assigning elements of, 297, 299, 300  
catenating, 297  
changing parts of, 297–299  
concatenating, 297  
conversion to sets, 295  
deleting elements of, 296, 297  
discarding elements of, 296, 297  
empty, [], 295  
extracting elements from, 296, 297  
extracting sublists from, 296, 297

- flattening, **Flatten**, 299  
 inserting elements in, 297  
 internal representation of, 295  
 length of, 295  
 mapping functions onto, **map**,  
     **map2**, 210, 211  
 membership test of, 295  
**op** vs. brackets [], 296  
 plots of, 459–465  
 prepending elements to, 297  
 rearranging, **sort**, 295, 298  
 replacing elements of, 297, 298  
 reversal of, 299  
 rotating, 298, 299  
 searching in, 295  
 selecting elements of, 296, 297  
 sorting, 298  
 term ordering in, 295  
 vs. sets, 295  
**ListTools** package, 299  
**Flatten**, 299  
 Livermore Stiff ODE solver, **lsode**,  
     576, 577  
**ln**, option of, **combine**, 242, 367,  
     368, 513  
**collect**, 545  
**convert**, 376  
**simplify**, 274, 367, 372, 389  
**ln**, 50  
 loading of package functions, **with**,  
     103, 104  
**local**, declaration, 203  
 local vs. global variables, 194, 195,  
     201–203  
**log**, 50  
**log10**, 50  
**log[b]**, 203  
 logarithm, natural, **log**, **ln**, 50  
     of base 10, **log10**, **log[10]**, 50, 203  
     of formal power series, **powlog**,  
         283  
     with arbitrary base, 203  
 logarithmic plots, **loglogplot**,  
     **logplot**, **semilogplot**,  
         429–432,  
 logarithms, combining, 242, 367, 368,  
     513  
     expansion of, 359, 360  
     simplification of, 372, 373  
 logical constants, **FAIL**, 48, 340  
     **false**, 48, 340  
     **true**, 48, 340  
 logical expressions, 84  
 logical operators, **and**, 84  
     **not**, 84  
     **or**, 84  
**loglogplot**, 429  
**logplot**, 429  
**LONGDASH**, line style, 473  
longitude, 506  
look-up tables, 97, 236, 246, 247  
loops, endless, 68, 75  
    infinite, 68, 75  
lower-case vs. upper-case characters,  
 77  
**lprint**, 99–101, 120, 121, 304, 327,  
     418, 419, 448, 449  
**LREtools** package, 516  
**lsode**, option of **dsolve**, 576, 577  
LU decomposition, of matrix,  
 see **LUDecomposition**  
Lucas numbers, 204–208  
**LUDecomposition**, 646–650, 654,  
     655, 657, 658, 674, 676, 677  
 method, **Cholesky**, 648  
**FractionFree**, 650, 654, 674, 676,  
     677  
**NAG**, 650, 655, 658, 659  
**RREF**, 647, 654  
**.m**, internal format file extension,  
 107–110  
Macaulay, 3  
Mach number, 514  
Maclaurin series, **series**, 265–267  
**macro**, 49, 102, 103, 163, 210, 327,  
     494, 673, 733  
MACSYMA, 3, 219, 590  
Magma, 4  
magnitude, of complex numbers, **abs**,  
 50, 58, 59, 334, 336  
**makehelp**, 396, 397  
mantissa, 47  
**map**, 14, 15, 56, 106, 127, 151, 171,  
     210, 211, 248, 249, 272, 276,  
     284, 320, 321, 371, 381, 383,  
     411, 425, 428, 457, 462–464,  
     484, 486, 489, 493, 496, 497,

- 501, 504, 507, 549, 552, 570,  
 572, 582, 592, 593, 608,  
 626, 632–634, 638, 639, 644,  
 649, 651–653, 655, 656, 667,  
 675–679, 685, 686, 691–693  
 difference with **Map**, 633
- Map**, 633, 639, 653
- map2**, 210, 211
- Maple, adapting, 92, 93, 133–136,  
 275, 276, 286, 287, 317, 358,  
 391–396  
 as a programming language, 11, 25,  
 26, 31  
 assisting, 242, 243, 251–255, 275,  
 276, 285–287, 536, 602  
 availability of, 29  
 built-in functions,  
**?index,function**, 40  
 design of, 29–32  
 fine-tuning, 92, 93, 133–136, 275,  
 276, 286, 287, 317, 358  
 help facilities of, 36–41  
 hybrid algorithmic structure of, 31,  
 32, 353  
 initialization files of, 34, 35, 62,  
 109, 163, 416  
 invoking, 6, 33  
 Iris, 29, 30  
 kernel of, 29, 30  
 launching, 6, 33  
 leaving, **quit**, 35  
 library of, 29, 30, 101–106  
 on-line documentation, 36–41  
 origin of, 29  
 parallel version of,  
**||MAPLE||**, 5,  
 parts of, 29, 30  
 prompt, 6, 34, 106, 135, 136  
 quitting, **quit**, 35  
 share library of, 29, 101, 105  
 source code of, 92, 135, 136, 357,  
 391  
 starting, 6, 33  
 surprising results of, 26  
 user interface of, 6, 29, 30  
 worksheet, 29, 30, 106  
**||MAPLE||**, 5  
 Maple Application Center, 105  
 Maple procedures, list of built-in,  
**?index,functions**, 40  
**maple.ini**, initialization file, 34, 109  
**.mapleinit**, initialization file, 109  
**maple.lib**, 395  
**Maplets** package, 29, 134  
**march**, 111, 112, 395, 396  
 Markov parameter matrix, 671, 672,  
 675  
**match**, 385, 518  
*Mathematica*, 4  
 mathematical constants, defining, 48,  
 49  
 Catalan's number, **Catalan**, 48  
 Euler-Mascheroni number,  
**gamma**, 16, 48  
**infinity**,  $\infty$ , 48  
**Pi**,  $\pi$ , 48, 77, 155  
 mathematical correctness, 354, 355,  
 364, 388  
 mathematical functions, list of  
 built-in, **?inifcns**, 49  
 matrices, addition of, **MatrixAdd**,  
**Add**, 629  
 addition of scalars, 630, 631  
 adjoint, **Adjoint**, 641  
 antisymmetric, **antisymmetric**,  
 310, 313, 622  
 arithmetic of, 629–634  
 augmentation of, 623, 624, 642,  
 666, 668, 674, 675  
 banded, **band**, 310  
**BandMatrix**, 624  
**Bezout**, **bezout**, 624  
 block, 635  
 characteristic,  
**CharacteristicMatrix**, 641,  
 682  
 characteristic polynomials of,  
**CharacteristicPolynomial**,  
 635, 636, 641, 682  
 Cholesky decomposition of, 648  
 column dimension,  
**ColumnDimension**, 641, 693  
 column operations in,  
**ColumnOperation**, 641  
 column rank of, **Rank**, 634  
 column space of, **ColumnSpace**,  
 635, 645

combining columns of,  
**ColumnOperation**, 641  
 combining rows of,  
**RowOperation**, 641  
 concatenation of, 623, 624, 642, 666, 668, 674, 675  
 condition numbers of,  
**ConditionNumber**, 641  
 construction of (general), 621–629  
     block, 635  
     special, 624  
 creating, **Array**, **Matrix**, 300–315, 621–629  
 definite, **IsDefinite**, 642, 648  
 deleting columns of,  
**DeleteColumn**, 641  
 deleting rows of, **DeleteRow**, 641  
 Denavit-Hartenberg, 663  
 determinant of, **Determinant**, 15, 20, 102–105, 559, 634, 638, 641, 668, 675, 676, 683  
 diagonal, **DiagonalMatrix**, 624, 625  
 eigenvalues of, **Eigenvalues**, 635–638, 641, 683, 685  
     in radical notation, 637  
     in **RootOf** notation, 637, 683  
     numerical, 685  
 eigenvectors of, **Eigenvectors**, 635–641, 683–685  
     in radical notation, 637  
     in **RootOf** notation, 637–639, 683  
     numerical, 685  
 equality of, **Equal**, 642, 643, 647, 648, 651–653, 656, 694  
 evaluating entries of **array**-based, 320  
 evaluating **array**-based, 319–321  
 exporting, **ExportMatrix**, 115, 315  
 extension of, 623, 624, 642, 666, 668, 674, 675  
 floating-point arithmetic of, 656–660  
 fraction-free Gaussian elimination of, **FractionFree** method of  
**LUDecomposition**, 650, 654, 674, 676, 677  
**GaussianElimination**, 646, 653, 654, 674  
 Frobenius form of, **Frobenius**, 646, 652  
**FrobeniusForm**, 646, 652  
 Gauss-Jordan form of,  
**ReducedRowEchelonForm**, 642, 646, 674  
**Gaussjord**, 646, 655  
 RREF method of  
**LUDecomposition**, 647, 654  
 generalized inverse of,  
**MatrixInverse**, 693, 694  
 generating systems of linear equations associated with,  
**GenerateEquations**, 658  
 Givens rotation,  
**GivensRotationMatrix**, 624  
 Hückel, 681  
 Hamiltonian, 681  
 Hankel, **HankelMatrix**, 624  
 Hermite normal form of, **Hermite**, 646, 653  
**HermiteForm**, 620, 626, 646, 653  
 Hermitian transpose of,  
**HermitianTranspose**, 641  
 Hessenberg form of,  
**HessenbergForm**, 646  
 Hessian, **Hessian**, 688  
 Hilbert, **HilbertMatrix**, 309, 624  
 identity, 310, 324, 624, 633, 642, 674, 681  
     via **IdentityMatrix**, 624, 631, 642  
 importing, **Importmatrix**, 114, 115, 315, 466, 467, 629  
 invariant factors of, 651  
 inverse of, **MatrixInverse**, 15, 633, 641, 643, 657, 693  
 Jacobian, **Jacobian**, 666, 688, 691, 692  
 Jordan block,  
**JordanBlockMatrix**, 624, 642  
 Jordan form of, **JordanForm**, 173, 174, 643, 646

kernels of, **NullSpace**, 635, 641, 645  
 last name evaluation of, 319–321  
 LU decomposition of, *see*  
**LUDecomposition**  
 manipulating all entries of, **Map**, **map**, 320, 633, 639, 653  
 Markov parameter, 671, 672, 675  
 minimum polynomials of,  
**MinimalPolynomial**, 635, 641  
 minors of, **Minor**, 31, 641  
 Moore-Penrose inverse of,  
**MatrixInverse**, 693, 694  
 multiplication of, . operator, 307, 315, 631–634  
**Multiply**, 631, 660  
 nonsingular, 633  
 norm of, **MatrixNorm**, 103, 104, 645  
 number of, columns of,  
**ColumnDimension**, 641, 693  
 rows of, **RowDimension**, 641  
 numerical eigenvalues of,  
**Eigenvalues**, 685  
 numerical eigenvectors of,  
**Eigenvectors**, 685  
 operating on all elements of,  
**Map**, **map**, 320, 633, 639, 653  
 options for representing,  
**MatrixOptions**, 622, 659  
 orthogonal, **IsOrthogonal**, 642, 644  
 over finite fields, 648–660  
 overlap, 681  
 permanent of, **Permanent**, 641  
 Popov form of, **PopovForm**, 646  
 powers of, 632, 633  
 properties on, 351  
 pseudo-inverses of, 693, 694  
 QR decomposition of,  
**QRDecomposition**, 646  
 random, **RandomMatrix**, 626, 628, 629, 693, 694  
 rank of, **Rank**, 634, 641  
 rational canonical form of,  
**FrobeniusForm**, 646, 652  
 row rank of, **Rank**, 103, 634, 641, 674  
 row reduced echelon form of,  
**HermiteForm**, 620, 626, 646, 653  
**ReducedRowEchelonForm**, 642, 646, 674  
 RREF method of  
**LUDecomposition**, 647, 654  
 row space of, **RowSpace**, 635, 645  
 scalar multiplication of,  
**ScalarMultiply**, 14, 630  
 Schur form of, **SchurForm**, 646  
 selecting columns in, **Column**, 641  
 selecting rows in, **Row**, 641  
 singular values of,  
**SingularValues**, 641, 685  
 Smith normal form of,  
**SmithForm**, 646, 652  
 sparse, 315, 622  
 special, 309–315  
 stacking, 623, 624, 642, 666, 668, 674, 675  
 standard forms of, 646–656  
 structural operations on, 641–644  
 submatrices of, **SubMatrix**, 16, 626, 641, 643, 665, 677  
 swapping columns of,  
**ColumnOperation**, 641  
 rows of, **RowOperation**, 641, 668  
 Sylvester, **SylvesterMatrix**, 624  
 symmetric, **symmetric**, 14, 20, 309–311, 629, 634, 637  
 test for, definite, **IsDefinite**, 642, 648  
 equal, **Equal**, 642, 643, 647, 648, 651–653, 656, 694  
 orthogonal, **IsOrthogonal**, 642, 644  
 similar, **IsSimilar**, 642, 644  
 zero, **IsZero**, 642  
 Toeplitz, **ToeplitzMatrix**, 624, 629, 634, 636  
 topological, 681, 682  
 trace of, **Trace**, 14, 634, 641  
 transpose of, **Transpose**, 14, 632, 641, 644, 648, 649, 667, 676, 682, 686, 691–693  
 triangular, **triangular**, 309, 310, 324, 629, 655

tridiagonal, 311  
 tridiagonal form of,  
**TridiagonalForm**, 646, 648, 649  
 Vandermonde,  
**VandermondeMatrix**, 624  
 zero, **ZeroMatrix**, 645  
**matrix**, 124, 304, 311, 330, 331, 445, 624  
     vs. **Matrix**, 304  
**Matrix**, 14, 20, 173, 174, 306–310, 313, 622, 624, 629, 643, 644, 655, 664, 666, 679, 682, 708, 737  
     option, **attributes**, 621  
         **datatype**, 621  
         **fill**, 307–309, 313, 621  
         **readonly**, 310, 621  
         **shape**, 14, 20, 309, 310, 313, 621, 622, 655  
         **storage**, 621, 622  
 matrix browser, 308  
 matrix operations, elementary, 641, 668  
**MatrixAdd**, 630, 631  
**MatrixInverse**, 15, 633, 641, 643, 657, 693, 694  
**MatrixOptions**, 622, 659  
**matrixplot**, 462, 463  
**MatrixPolynomialAlgebra** package, 620  
**maximize**, 277  
**\_MaxSols**, 489  
**maxsols**, option of **solve**, 510  
**mean**, 105  
 Mellin transform, **mellin**, 241, 250  
**member**, 101, 102, 293, 295, 686  
**MEMBER** (user-defined function), 202  
 membership tests, **member**, 101, 102, 203, 293, 295, 686  
 memory, allocation of, **bytesalloc**, 133, 134  
     clearing of, **gc**, 134  
     used, **bytesused**, 133, 134  
 merge sort, complexity of, 515  
**MESH**, graphics primitive, 451  
 messages, bytes-used, 133  
     controlling frequency of bytes-used, 134  
         **gcfreq**, 134  
         **printbytes**, 133  
     run-time, 107  
     warning, 192, 486, 620  
 methyl group, 687  
 metric tensor, 13–16, 691, 692  
     inverse of, 15  
 Michaelis-Menten theory, 494  
**midpoint**, 436  
**minimax**, 279–281  
 minimal polynomial, of matrix, **MinimalPolynomial**, 635, 641  
**MinimalPolynomial**, 635, 641  
 minimax approximation, **minimax**, 279–281  
**minimize**, 277  
**Minor**, 32, 641  
 minor of a matrix, **Minor**, 32, 641  
**minus**, 70, 78, 110, 293, 483, 678, 679  
 mistakes, correcting, 35  
     in function definitions, 193–195, 197  
     in plotting, 439, 445  
**mkdir**, 111  
**mod**, 44, 45, 144, 651–653, 655, 656  
**Mod**, 104  
**mods**, 45  
**Modular**, subpackage of  
     **LinearAlgebra**, 104, 619, 620, 655  
 modular arithmetic, 44, 45, 144, 514, 648–660  
 module, 86, 325, 611, 620  
     selection operator of member of, **::**, 87, 102, 104, 325, 388, 611, 612, 620  
     using a, **use**, 103, 326, 334, 354, 356, 372, 374, 387, 388, 484  
 Möbius function, 22  
 Möbius strip, plot of, 479  
 molecular-orbital Hückel theory, 680–687  
 monomial, 704  
 Monte Carlo integration, **MonteCarlo**, 240  
 Moore-Penrose inverse, of matrix, 693, 694

movies, *see* animation  
**msolve**, 514, 744  
**mtaylor**, 111, 270  
**mult**, commutator of infinitesimals, 559  
 multi-degree, of a monomial, 708  
 multidimensional arrays, 300, 301  
 multiple curves, plotted in distinct styles, 10, 405, 412, 462  
 multiple input lines, 35  
 multiple libraries, 82, 111, 112, 395, 396  
 multiple-precision integers, 42, 43  
 multiple-valued functions, 58, 59, 190, 355  
 multiplication, of functions, \*, 210  
     of matrices and vectors, ., 307, 315, 631–634  
     of numbers, \*, 34  
**multiply**, 282  
**Multiply**, 631, 660  
 multivariate polynomials, 145, 146  
     distributed form of, 185, 186  
     expanded form of, 185, 186  
 multivariate Taylor series expansions, **mtaylor**, 111, 270  
**MuMATH**, 3  
**MuPAD**, 5  
.**mws**, Maple worksheet file extension, 106  
  
**Nabla**, 582, 687, 688, 690, 691, 693  
 nabla operator, **Nabla**, **Del**, **Gradient**, 582, 687, 688, 690, 691, 693  
**NAG**, integration method, **d01akc**, 240  
     method of **LUDecomposition**, 650, 655, 658, 659  
 name protection, **protect**, 48, 49, 79, 87  
 names, abbreviated, **macro**, 49, 102, 103, 163, 210, 327, 494, 673  
     aliasing of, **alias**, 55, 62, 91, 102, 103, 494, 525  
     assigned, **anames**, 69, 70, 72  
     assignments of, 66, 71, 87, 91, 316, 493  
     beginning with underscore, 54, 77, 78, 80, 562  
     case sensitivity, 77  
     characters allowed in, 77  
     concatenation of, **cat**, ||, 82, 84, 292  
     conventional, 7  
     function without, 70, 210, 211  
     indexed, 68, 69, 79, 84, 85, 203, 316, 341, 673  
     introducing sensible, 327  
     length limitation of, 77  
     of initially known of packages, ?**index,packages**, 40, 104  
     of variables, 77–83  
     philosophy of Maple, 7  
     protected, 48, 49, 78, 87  
     protecting, **protect**, 48, 49, 78, 87  
     recursive definition of, 68, 75  
     special characters in, 80, 81  
     unassigned, **unames**, 70  
     underscores in, 54, 77, 78, 80, 562  
     vs. conventional strings, 81  
**nargs**, number of arguments, 290  
 natural logarithm, **log**, **ln**, 50  
     base of, 49, 78  
**negation**, **Non**, 343, 344, 350  
     **not**, 84  
 nested data types, 86  
 nested square roots, 54  
**networks** package, 727, 728  
     **chromopoly**, 728  
     **draw**, 727  
     **edges**, 727, 728  
     **ends**, 728  
     **petersen**, 727  
     **vertices**, 727  
 neural network, 503, 698, 725, 729, 736  
 newline character, 34, 35, 81, 97, 115, 121, 122  
 Newton method, of **fsolve**, 32, 511  
 Newton-Cotes method, for numerical integration, 32, 239  
 Newton-Euler formalism, 569  
**nextprime**, 43  
 non-commutative algebra, 25, 26, 631  
**none**, plot option, 443, 472, 476  
**nonnegint**, data type, 11, 26, 294  
**nonuniform**, animation option, 477  
**nops**, 158, 160, 293, 295–299

**Norm**, 41, 645  
**normal**, axesstyle, 443, 444, 472  
**normal**, 7, 9, 14, 15, 19, 147, 148, 160, 161, 164, 171, 172, 183–186, 213, 215, 216, 218–220, 226–228, 243, 269, 270, 275, 362, 370, 371, 380, 389, 391, 495, 515, 545–547, 553, 556, 557, 564, 572, 582, 590, 633, 634, 652  
**Normal**, 184, 185, 232, 233, 652, 655, 656  
normal distribution, **normald**, 248  
normal form, of polynomials, 181, 182, 710–712  
of rational expressions, 183–186  
normal form simplification, 182, 710–712  
normal simplifier, 182, 710–712  
**normalf**, 384, 710–713, 715, 731, 743  
normalization, 183–185  
of small numerical values, **fnormal**, 145, 248, 332, 649, 686  
of vectors, **Normalize**, 645, 686  
**Normalize**, 645, 686  
**Normalizer**, 275, 286  
**not**, 84  
not equal ( $< >$ ), 84  
notation, mimicing of mathematical, 6, notched, format of **boxplot**, 465 nucleophytic substitution, 687 **NULL**, 100, 154, 206, 208, 290, 297, 422, 474  
**NullSpace**, 635, 641, 645  
**numapprox** package, 10, 149, 265, 267, 273, 276–281, 610  
**chebdeg**, 278  
**chebmult**, 278  
**chebpade**, 10, 278, 279  
**chebsort**, 278  
**chebyshev**, 273, 278  
**confracform**, 149, 273, 277  
**hornerform**, 149, 276, 279  
number of arguments, **nargs**, 290  
number of digits of an integer, **length**, 42  
number theory package, **numtheory**, 63, 87, 88  
numbers, algebraic, 53–57, 80, 177, 178, 487, 636  
approximate, 46–52  
arbitrary precision, 46–52  
as continued fractions, **cfrac**, **confrac**, 87  
Bernoulli, 256  
complex, 58–62  
exact vs. approximate, 51  
floating-point, 46–52  
hardware floating-point, 51, 449, 450, 648, 649, 657  
irrational, 46–57  
Lucas, 204–208  
nonnegative integral, **nonnegint** (data type), 11, 26, 294  
normalization of floating-point, **fnormal**, 145, 248, 649, 686  
p-adic, 88  
radical, 46, 53–56  
rational, 45  
truncation of small, **fnormal**, 145, 248, 332, 649, 686  
**numer**, 147, 161, 184, 381, 553, 557, 588, 677, 678  
numerator, of rational expression, *see numer*  
Numerical Algorithms Group (NAG), 32, 239, 240, 305, 621, 648–650, 656–660  
numerical approximations, **evalf**, 32, 46–52  
numerical constants, defining, 48, 49  
numerical data, I/O of, 112–117, 124–127, 466, 629  
numerical evaluation, *see evalf and evalhf*  
numerical integration, 31, 239–241 methods, adaptive Gaussian quadrature method, **Gquad**, 32, 329  
adaptive sinc method, **Sinc**, 32, 329  
Clenshaw-Curtis quadrature method, **CCquad**, 32, 239

- double exponential method,  
  `Dexp`, 32, 239, 240
- Monte Carlo method,  
  `MonteCarlo`, 240
- NAG routine, `_d01akc`, 240
- Newton-Cotes method, `NCrule`,  
  32, 239
- numerical properties, hierarchy of,  
  350
- numerical root-finding, `fsolve`, 510,  
  511
- numerical solving, of differential  
  equations, 566–580, 611–615
- of eigenvalue problem, 685
- of polynomial equation,  
  `realroot`, `sturm`, 512
- of (systems of) equations, `fsolve`,  
  32, 510, 511, 578, 600
- numerical Taylor series method for  
  solving ODEs, 577, 578
- `numpoints`, plot option, 59, 194, 406,  
  410, 416, 417, 421, 429, 432,  
  434, 435, 439, 452, 454, 458,  
  469, 473, 574, 575, 579
- `numtheory` package, 63, 87, 88
- `0`, order symbol, 266, 267, 269
- `object too large`, 43, 186
- oblate spheroidal coordinates, 689
- observability, of linear systems, 671,  
  675, 676
- observability matrix, 671
- `octahedron`, 446, 471
- `odd`, 89, 342, 343
- odd numbers, assuming, 89,  
  computing with properties of, 342,  
  343
- `odeadvisor`, 536, 545, 547, 553, 560
- ODEPACK, 577
- `odeplot`, 567, 568, 574, 578, 580  
  `refine` option, 568
- ODEs, analytical solution of,  
  `dsolve`, 523–538
- change of coordinates in,  
  `dchange`, `Dchangevar`,  
  586–590
- checking solutions of, `odetest`, 524,  
  525, 537, 538
- definition of, 522
- degree of, 522
- explicit solutions of, 525, 526
- formal power series solution of,  
  `powsolve`, 561–566  
  `formal_series`, 564  
  `FPseries`, 562  
  `FTseries`, 643
- graphical solution of, 580–586
- implicit solutions of, `implicit`, 525
- in functional notation, 526
- Lie point symmetries for, 538–560
- linear, 522
- nonlinear, 522
- numerical solutions of, 566–580
- order of, 522
- partial solution of, 537, 538
- perturbation solution of, 590–600
- plotting numerical solutions of,  
  `odeplot`, 567, 568, 574, 578,  
  580
- series solution of, `series` option,  
  560–566
- systems of, 531
- Taylor series method for, 560, 561,  
  577, 578
- tracing the finding of solutions of,  
  526
- `odetest`, 524, 525, 537, 538
- on-line documentation, 36–41
- one-dimensional output, `Iprint`,  
  99–101, 120, 121, 304, 327,  
  418, 419, 448, 449
- `op`, 57, 110, 158, 159, 165, 169–171,  
  176, 203, 207, 208, 211, 233,  
  269, 275, 289, 293, 296–299,  
  304, 311, 312, 317, 318,  
  327–329, 331, 358, 382, 391,  
  394, 428, 429, 431, 445, 457,  
  462–464, 504–506, 511, 513,  
  528, 561, 563, 599, 604, 605,  
  616, 658, 676, 679, 682
- `open`, 117, 119
- `Open`, 20, 62, 90, 91, 93,  
  339–341, 343–349, 509
- open streams, status of, `iostatus`,  
  119
- opening, of streams, `fopen`, `open`,  
  `popen`, 117–119
- operands, of expressions, *see op*

applying functions to all, **map**, 171  
 applying functions to specific,  
**applyop**, 170, 171, 211  
 substitution of specific, **subsop**,  
 169–171, 208, 297, 317, 327,  
 486, 517  
 operations, arithmetic, 34  
     functional, 210  
     logical, 84  
     precedence of, 34, 633, 634  
     on series, 270  
 operator, arrow,  $\rightarrow$ , 12, 193–195,  
 204, 208, 210  
 assignment operator ( $:=$ ), 66, 71,  
 87  
 association ( $::$ ), 86, 87, 201  
 composition,  $(@)$ , 217  
 commutative multiplication,  $*$ , 34  
 concatenation ( $\parallel$ ), 82, 84, 292  
 del operator, **Del**, **Gradient**,  
**Nabla**, 582, 687, 688, 690,  
 691, 693  
 differential, **D**, 216–218, 220, 284,  
 526  
 ditto, 46, 79, 83, 100, 101, 208, 209  
 ellipsis operator,  $(..)$ , 84, 292  
 logical, **and**, **not**, **or**, 84  
 matrix multiplication,  $\&*$ , 304, 315  
     . (dot), 307, 315, 631–634  
 nabla, **Nabla**, **Del**, **Gradient**,  
 582, 687, 688, 690, 691, 693  
 relational,  $(>, \leq, <>, \text{etc.})$ , 84  
 selection,  $[]$ , 291, 293, 296, 316  
 sequence,  $\$$ , 214, 291  
**optimize**, option of code generation,  
 128, 129  
**optimize**, 128, 129  
     **tryhard** option, 129  
 optimizing expressions, 128, 129  
**option remember**, 193, 205–208, 222,  
 267, 341, 358, 486  
 optional arguments in plotting  
     routines, **hasoption**, 432  
 options, setting default graphics,  
     **setoptions**, 416  
     **setoptions3d**, 447  
**or**, 84  
**order**, option of **rtable**, 323  
**Order**, environment variable, 267  
 order, of ODEs, 522  
     of PDEs, 601  
     of series, 266, 267, 269  
**order**, 267  
 order symbol,  $0$ , 266, 267, 269  
 ordering of terms in polynomials, *see*  
     term ordering  
 ordinary differential equations, *see*  
     ODEs  
 ordinates, **AXES**, **axes**, 443, 452, 472  
**Ore\_algebra** package, **poly\_algebra**,  
 706, 731, 744, 745  
**orientation**, option of **Vector**, 621,  
 622  
     plot option, 443–445, 476  
**ORIENTATION**, graphics primitive, 448,  
 452  
 orientation in 3D graphics,  
     **orientation**, 443–445, 476  
**OrProp**, 343, 344  
**orthogonal**, perspective in 3D  
     graphics 443, 476  
 orthogonal curvilinear coordinates,  
 687–689  
 orthogonal polynomials, 278  
     Chebyshev, **ChebyshevT**, 274,  
 278, 279, 288  
     Hermite, **HermiteH**, 181, 566  
     Laguerre, **LaguerreL**, 375  
     Legendre, **LegendreP**, 212  
 orthogonality testing of matrix,  
     **IsOrthogonal**, 642, 644  
 oscillator, *see* pendulum  
 outer product of vectors,  
     **CrossProduct**, 645  
 output, columnated, **writedata**, 115,  
 116, 125, 315  
     defining pretty printing of, 326  
     file for graphics, **plotoutput**, 403,  
 405  
     files, 106  
     for C language, **C**, 129  
     for Fortran language, **Fortran**,  
 127, 128, 131  
     for Java language, **Java**, 130, 131  
     for L<sup>A</sup>T<sub>E</sub>X, **latex**, 15, 77, 132, 133  
     formatted 119–127,  
     linear, *see* **lprint**  
     low-level, 116–127

- number of digits in formatted I/O, 120, 121  
 of binary data, **writebytes**, 118  
 PostScript, 405  
 streams, 117  
 overlap integrals, 691  
 overriding Maple functions, 92, 93, 133–136, 275, 276, 286, 287, 317, 358, 391–396
- p-adic numbers, 87  
 package for,  
   algebraic curves, **algcurves**, 429  
   approximation of functions,  
     **numapprox**, 10, 149, 265, 267, 273, 276–281, 610  
   combinatorics, **combinat**, 312, 293, 394  
   curve fitting, **CurveFitting**, 469  
   calculus over domains, **Domains**, 25  
   code generation, **codegen**, 128, 148, 149, 182, 183  
     **CodeGeneration**, 16, 127–133  
   elementary calculus, **Student**, 252, 253  
   Euclidean geometry, **geometry**, 436–439, 480  
   formal power series, **Slode**, 561–564  
     **powseries**, 281–284, 561, 562, 565, 566  
   Gaussian integers, **GaussInt**, 464  
   generating functions, **genfunc**, 13, 517  
   graph theory, **networks**, 727, 728  
   Gröbner bases, **Groebner**, 168, 383, 384, 501–508  
   integral transforms, **inttrans**, 241–250, 612, 613  
   large expressions,  
     **LargeExpressions**, 187  
   Lie symmetries, **liesymm**, 615–618  
   linear algebra, **linalg** (deprecated), 173, 304, 619  
     **LinearAlgebra**, 102–104, 619–662  
   linear recurrence equations,  
     **LREtools**, 516  
   list manipulation, **ListTools**, 299  
   maplets, **Maplets**, 134  
   matrix polynomials,  
     **MatrixPolynomialAlgebra**, 620  
   multi-processing, **process**, 118, 119  
   number theory, **numtheory**, 63, 87, 88  
   ordinary differential equations,  
     **DDEtools**, 521, 536, 541–548, 551, 559, 560, 580–591  
   Ore algebras, **Ore\_algebra**, 706  
   p-adic numbers, **padic**, 87  
   partial differential equations,  
     **PDEtools**, 524, 526, 541–549, 553, 566, 570, 580, 585, 586, 601–612  
   plotting tools, **plottools**, 414, 415, 422–426, 435, 446, 452, 456, 458, 461, 463, 471, 48  
   polynomial calculus,  
     **PolynomialTools**, 142, 179  
   real calculus, **RealDomain**, 334, 354, 356, 372, 374, 386–388, 484  
   special plotting, **plots**, 402, 409–416, 420, 423, 424, 426–436, 447, 452, 459–464, 469, 471, 567, 574, 575, 689  
   scientific constants,  
     **ScientificConstants**, 48  
   statistics, **stats**, 104, 105, 114, 124, 125, 248, 429, 459, 465, 466, 468  
   string manipulation, **StringTools**, 82  
   summation, **sumtools**, **SumTools**, 256, 258  
   symbolic-numeric algorithms for polynomials, **SNAP**, 144  
   units, **Units**, 318  
   vector calculus, **VectorCalculus**, 582, 666, 687–693  
 packages, listing of,  
   **?index,packages**, 40, 104  
   loading of, **with**, 103, 104  
   subpackage of, 104  
 pad character, 120  
 Padé approximation, **pade**, 10, 273, 277, 278  
   of rational functions, 273

**pade**, 10, 273, 277, 278  
**padic package**, **evalp**, 88  
 parabolic coordinates, 689  
 paraboloidal coordinates, 689  
 parallel version of Maple,  
     ||MAPLE||, 5,  
 parameter identification, 669–680  
 parameters, in arrow operators, 193,  
     194  
 parametric plots, in 2D, 406  
     in 3D, 452, 453  
**parametrization**, 429  
 parametrization of solutions, 491  
 parentheses, (), 34  
**pareto**, 464  
 Pareto plots, **pareto**, 464  
**parfrac**, 149–151  
 PARI, 3  
**parse**, 124  
 parsing of strings, **parse**, 124  
 partial derivatives, 216  
 partial differential equations, *see*  
     PDEs  
 partial fraction decomposition,  
     full, **fullparfrac**, 150  
     of rational functions, **parfrac**,  
         149–151  
 parts of expressions, applying  
     functions to all, **map**, 171,  
         210, 211, 320, 633, 639, 653  
     applying functions to specific,  
         **applyop**, 170, 171, 211  
     removing specific, **remove**, 210,  
         211, 294, 599  
     replacing specific, **subsop**,  
         169–171, 208, 297, 317, 327,  
         486, 517  
     selecting specific, **select**, 70, 210,  
         211, 294, 484, 498, 587, 599,  
         603, 616  
 parts of Maple, 29–31  
 Pascal’s theorem, 48  
**patch**, plotstyle option, 410, 414,  
     442–446, 453, 454, 456, 457,  
     462, 463, 475, 477  
**patchcontour**, plotstyle option, 58,  
     194, 442, 447, 454–456, 475,  
     477, 610, 611  
**patchnogrid**, 52, 422, 436, 442, 446,  
     450, 451, 457, 458, 465, 475,  
     477  
 pattern matching, **match**, 385, 518  
**pclose**, 117  
**PDEchangecoords**, 586–589, 591  
**PDEplot**, 585, 601  
 PDEs, analytical solution of,  
     **pdsolve**, 521, 545, 546, 548,  
         550, 552, 553, 555, 601–610  
     change of variables in,  
         **dchange**, **Dchangevar**,  
         **PDEchangecoords**, 587–589,  
         591  
     checking solutions of, **pdetest**,  
         603, 604, 606, 607, 609  
     definition of, 600  
     degree of, 601  
     graphical solution of, **PDEplot**,  
         585, 601  
     Lie point symmetries, **liesymm**  
         package, 615–618  
     linear, 601  
     nonlinear, 601  
     numerical solution of, 610–615  
     order of, 601  
     plotting solutions of, 585, 601, 611,  
         612, 614  
     quasi-linear, 601  
**pdetest**, 603, 604, 606, 607, 609  
**PDEtools** package, 524, 526, 541–549,  
     553, 566, 570, 580, 585, 586,  
     601–612  
**casesplit**, 603–605  
**dchange**, 549, 553, 586–590, 608  
**declare** 524, 526, 541, 543, 544,  
     551, 566, 570, 580, 590, 601,  
     603, 606, 607, 609  
**PDEplot**, 585, 601  
**separability**, 603, 604  
**pdintegrate**, 618  
**pdsolve**, 521, 545, 546, 548, 550, 552,  
     553, 555, 601, 602, 606–612  
     **HINT** option, 602, 606–609  
 Pell’s equation, 514  
 pendulum, 269, 528, 529, 560, 561,  
     590  
 double, 569–576

- percentage symbol (%), 17, 46, 79, 83, 101, 208  
in labels, 135, 482
- performance analysis, 24, 52, 204–206, 296
- Permanent**, 641
- permanent of matrix, **Permanent**, 641
- perspective in 3D graphics, projection, 443, 476
- Petersen graph, 726–729
- petersen**, 727
- phase, of a complex number, argument, 58
- picking elements, based on criteria, select, 70, 210, 211, 294, 484, 498, 587, 599, 603, 616  
based on representation, *see op*  
from lists, 296, 297  
from sets, 293
- phase portrait, 578, 579, 581, 582  
of anharmonic oscillator, 578, 579, 582  
of van der Pol equation, 581
- $\pi$ , **Pi**, 48, 77, 155
- $\pi$ -bonds, 681
- $\pi$ -electron density, 686
- $\pi$ -electron energy, 680, 686
- pie charts, 463, 464
- piecewise**, 196, 199, 200, 440
- PIECEWISE**, 326
- piecewise defined functions, 195–201, 439–440  
as case statements, 195  
as procedures, **piecewise**, 196, 199, 200, 440  
complications with, 197–199
- pieslice**, 415, 463
- pipe**, 117, 118
- placeholder, **DESol**, 538
- Diff**, 213, 214
- Eval**, 191, 192, 216
- Expand**, 143, 144, 177, 178, 653
- Factor**, 143, 144, 146, 179, 180
- FPSstruct**, 562, 563
- Frobenius**, 646, 652
- function call as, 326
- Gausselim**, 646, 655
- Gaussjord**, 646, 655
- Gcd**, 144
- Hermite**, 646, 653
- Integrate**, 17, 19, 26, 27, 226–228, 230–241, 250–254, 610
- Limit**, 285
- Normal**, 184, 185, 232, 233, 652, 655, 656
- Primfield**, 56, 64, 230
- PIECEWISE**, 326
- Quo**, 144
- RootOf**, 54
- Simplify**, 232, 484
- Smith**, 646, 652
- plane algebraic curves, plots of, 406, 427, 428, 470, 471
- plex**, 145, 383, 501, 507, 704, 706, 711–713, 715, 716, 720, 729–733, 735, 736, 738, 740–745
- plot**, 10, 59, 190, 191, 196, 197, 199, 200, 221, 236, 248, 249, 277–289, 402–413, 415–421, 423, 426, 427, 429, 439, 440, 458, 468, 469, 488, 508, 567, 573, 574, 576–580, 594, 600, 611
- plot aliasing, 439, 445
- plot arrays, 427, 435, 445, 470, 471
- PLOT objects, 418, 419, 423
- plot options, common, 472–474
- axes,
- box, 412–414, 433–436, 442, 443, 446, 448–455, 462, 463, 465, 472, 530, 574, 585
  - frame, 416, 426, 429, 432, 433, 443, 445–447, 456, 472, 584, 610, 611, 614
  - none, 406, 415, 418, 423, 424, 436–438, 443, 458, 463, 471, 472
  - normal, 416, 443, 472
- axesfont, 408, 410, 427, 472
- color, 10, 406, 412–415, 421–424, 426–428, 434, 435, 438, 445, 448, 449, 452, 458, 460, 461, 463, 468, 469, 471, 472, 475, 574, 575, 578, 580, 585, 611, 612
- coords, 415, 472

**bipolar**, 472, 688  
**cardiod**, 472  
**cartesian**, 472, 688, 689  
**cylindrical**, 453, 472, 689  
**ellcylindrical**, 689  
**ellipsoidal**, 689  
**elliptic**, 415, 688  
**hyperbolic**, 688  
**hypercylindrical**, 689  
**oblatespheroidal**, 689  
**parabolic**, 688  
**paraboloidal**, 689  
**polar**, 415, 688  
**prolatespheroidal**, 689, 690  
**spherical**, 453, 455, 472, 689  
**font**, 410, 424, 435, 437, 438, 445, 462, 463, 473  
**labelfont**, 473, 612  
**labels**, 413, 429–431, 451, 460–463, 473, 612  
**linestyle**, 10, 412, 413, 417, 424, 438, 447, 461, 473, 578  
**numpoints**, 59, 194, 406, 410, 416, 417, 421, 429, 432, 434, 435, 439, 452, 454, 458, 469, 473, 574, 575, 579  
**scaling**, 407, 473  
    **constrained**, 407, 473  
    **unconstrained**, 407, 473  
**symbol**,  
    **box**, 411, 473  
    **circle**, 410, 429, 460, 461, 473, 568, 580  
    **cross**, 473  
    **diamond**, 468, 473  
    **point**, 473  
**symbolsize**, 410, 460, 461, 473  
**thickness**, 406, 412, 417, 421, 422, 424, 426, 432, 437, 438, 447, 452, 455–457, 471, 474  
**tickmarks**, 412–414, 425, 427, 429, 444, 462, 463, 466, 470, 471, 474  
**title**, 10, 196, 409, 412, 413, 419, 420, 449, 461, 463, 474, 567, 568, 573–575, 584, 585, 611  
**titlefont**, 409, 410, 474  
**view**, 58, 408, 417, 420, 428, 433, 446, 456, 458, 460, 461, 474  
**plot options**, related to animations, 477  
    **frames**, 477  
    **framescaling**,  
        **nonuniform**, 477  
        **uniform**, 477  
    **insequence**, 470, 471, 477  
**plot options**, specific, 474, 475  
    **adaptive**, 420, 474  
    **discont**, 196, 236, 408, 421, 474  
    **filled**, 426, 435, 474  
    **legend**, 417, 418, 461, 474  
    **resolution**, 419, 474  
    **sample**, 410, 420, 474  
    **style**,  
        **line**, 410, 475  
        **patch**, 410, 475  
        **point**, 410, 411, 429, 473, 475, 568, 580  
    **xstickmarks**, 408, 409, 465, 475  
    **ystickmarks**, 409, 465, 475  
**plot window**, 404  
**plotdevice**, **bmp**, 406  
    **gif**, 406  
    **jpeg**, 405  
    **wmf**, 406  
**2D plots**, general format of, 404  
    options, 407–418, 472–475  
    range of, 405, 407, 408  
    scaling of, 407, 473  
    vertical range of, 407  
    view of, **view**, 407, 408, 474  
**3D plots**, general format of, 441  
    options, 442–447, 472–477  
**plots package**, 402, 409–416, 420, 423, 424, 426–436, 447, 452, 459–464, 469, 471, 567, 574, 575, 689  
    **animate**, 469, 612  
    **animate3d**, 469, 471  
    **animatecurve**, 470  
    **changecoords**, 415, 416, 424  
    **complexplot**, 432  
    **complexplot3d**, 455  
    **conformal**, 432, 433  
    **contourplot**, 435  
    **contourplot3d**, 459  
    **coordplot3d**, 689  
    **densityplot**, 435, 436

- display**, 402, 408–410, 412–417, 420, 422–427, 431, 435, 445, 446, 454, 456–458, 460–463, 465, 468–471, 477, 576, 580, 611, 615  
**fieldplot**, 433  
**implicitplot**, 428, 429  
**implicitplot3d**, 455  
**interactive**, 402, 403  
**Interactive**, 469  
**listcontplot**, 459  
**listcontplot3d**, 459  
**listdensityplot**, 459, 464, 465  
**listplot**, 459–461, 615  
**listplot3d**, 459  
**loglogplot**, 429  
**logplot**, 429  
**matrixplot**, 462, 463  
**odeplot**, 567, 568, 574, 578, 580  
**pareto**, 464  
**polygonplot**, 413  
**polygonplot3d**, 448  
**polyhedraplot**, 456  
**semilogplot**, 429–431  
**setoptions**, 416  
**setoptions3d**, 447  
**spacecurve**, 452, 574, 575  
**surfdata**, 614  
**textplot**, 409, 410, 424, 435, 461, 463  
**textplot3d**, 445  
**tubeplot**, 454  
**plotsetup**, 403, 405, 406  
plotting, 2D-fields, **fieldplot**, 433, 434  
 2D-plot options, 407–418  
 2D-plot specialties, 426–436  
 2D-plot styles, 410–412  
 2D-plot window, 404  
 3D-plot options, 442–447  
 3D-plot specialties, 452–458  
 3D-plot styles, 442  
 3D-space curves, 452, 574, 575  
 a Bode diagram, **Bodeplot**, 430–432  
 a cob-web model, 411, 412  
 a common mistake in, 439–441  
 adaptive, 410, 419–421, 474  
 adding a title, 409, 410, 474  
 adding text, 410  
 altering, 424–426  
 angle of view in 3D, 443–445, 476  
 axes in, **axes**, 416, 417, 443, 472  
 camera position in 3D, 443–445, 476  
 colors in, **color**, 412–415, 472  
 combining plots, *see also display*  
     a graphics array, **display**, 427, 445, 470, 471, 477  
     a single picture, 426, 427, 431, 435, 445, 454, 461, 462, 470  
     an animation, 471  
 complex curves, **complexplot**, 432  
 complex functions,  
     **complexplot3d**, 455  
 conformal mappings, **conformal**, 432, 433  
**constrained**, scaling option in, 407, 473  
 contour diagrams, **contourplot**, 435, 435  
 contour lines in 3D, **contours**, 447, 475  
 coordinate systems, in 2D, 415, 416, 472  
     in 3D, 453, 454, 472  
 data, 411, 459–469  
 density, **densityplot**,  
     **densityplot3d**, 435, 436  
 evaluation in, 439–441  
 geometrical figures, **draw**, 436–439  
 grid in 3D, **grid**, 445, 475  
 hardware arithmetic in, 51, 52, 450  
 headings in, **title**, 409, 410, 474  
 histograms, **histogram**, 462, 463, 465  
 implicitly defined functions,  
     **implicitplot**,  
     **implicitplot3d**, 427, 428, 455  
 in (semi-, double-) logarithmic scales, **semilogplot**,  
     **loglogplot**, **logplot**, 429–432  
 in cylindrical coordinates, 453  
 in functional notation, 440  
 in spherical coordinates, 453  
 infinite range, 405  
 infinity plot, 405  
**jpeg**, graphics driver, 403

- kink angle in, 419, 449  
 labels in, 409, 410, 473  
 legends in, 435  
     through **legend** option, 417, 418,  
         461, 474  
 lighting in 3D, **light**, 446, 447, 476  
 line styles in, **linestyle**, 412, 413,  
     417, 473  
 line thickness in, **thickness**, 412,  
     417, 474  
 lists of points, **listplot**, **listplot3d**,  
     459–461, 615  
 low-level 2D-objects in, 418, 419,  
     422, 423  
 low-level 3D-objects in, 448, 449  
 manipulating objects in, 422, 424,  
     425  
 method, in 2D, 418–421  
     in 3D, 448–452  
**noborder**, plot option, 405  
 options, 472–476  
 orientation in 3D, **orientation**,  
     443–445, 476  
 parametric curves and surfaces,  
     406, 452, 453  
 perspective in 3D, **projection**,  
     443, 476  
 piecewise defined functions,  
     439–441  
 pie charts, 463  
 plane algebraic curves, 406, 427,  
     428, 470, 471  
**plotdevice**, 406  
 polyhedra, 456, 457  
 PostScript output, 405  
 printing, 405  
 reading, **read**, 422  
 removing discontinuities, **discont**,  
     408, 474  
 resetting plot device,  
     **plotsetup**, 403, 405, 406  
 resolution, **resolution**, 419, 474  
 scales in, **scaling**, 407, 473  
 sample points, 410, 419, 420, 474  
 saving, **save**, 421  
 setting default options,  
     **setoptions**,  
         **setoptions3d**, 416, 447  
 setting **plotoptions** in, 405  
 several graphs in one picture, 426,  
     427, 431, 435, 445, 454, 461,  
         462, 470  
 shading in 3D, **shading**, 442, 443,  
     476  
 statistical, 465–467  
 tick marks, **tickmarks**, 409, 474  
 title in, **title**, 409, 474  
**unconstrained**, scaling option in,  
     407, 473  
 view frame in, **view**, 408, 446, 474  
     without border lines, 405  
**plottools** package, 414, 415,  
     422–426, 435, 446, 452, 456,  
         458, 461, 463, 471, 480  
**arc**, 424  
**arrow**, 424  
**cuboid**, 414  
**curve**, 424, 425  
**disk**, 424  
**ellipse**, 424  
**line**, 424, 436, 438  
**octahedron**, 446, 471  
**pieslice**, 415, 463  
**point**, 461  
**rectangle**, 423, 435, 458  
**rotate**, 424, 425  
**scale**, 424, 425, 457  
**stellate**, 424  
**tetrahedron**, 456  
**transform**, 424, 425, 458  
**translate**, 424, 425, 457  
**plot3d**, 52, 58, 59, 194, 402, 414,  
     441–447, 449–451, 453, 530,  
     610, 611  
**PLOT3D** objects, 448, 449, 451  
**plot3d** options, specific, 475–477  
     **ambientlight**, 446, 475  
     **color**, 10, 406, 412–415, 421–424,  
         426–428, 434, 435, 438, 445,  
         448, 449, 452, 458, 460, 461,  
         463, 468, 469, 471, 472, 475,  
         574, 575, 578, 580, 585, 611,  
         612  
     **contours** 434, 435, 447, 451, 455,  
         475  
     **filled**, 475  
     **grid**, 445, 476  
     **gridstyle**, 476

- light**, 446, 447, 476
- lightmodel**, 476
- orientation**, 443–445, 476
- projection**,
  - fisheye**, 443, 476
  - normal**, 443, 476
  - orthogonal**, 443, 476
- shading**, 442, 443, 476
  - none**, 443, 476
  - xy**, 442, 476
  - xyz**, 442, 443, 476
  - z**, 442, 476
  - zgrayscale**, 442, 443, 476
  - hue**, 443, 476
- style**, 442, 476
  - contour**, 442, 447, 475, 477
  - hidden**, 442, 477
  - line**, 410, 442, 476
  - patchcontour**, 58, 194, 442, 447, 454–456, 475, 477, 610, 611
  - patchnogrid**, 442, 476
  - patch**, 410, 442, 476
  - point**, 410, 442, 475, 477
  - wireframe**, 442, 477
- PLOT3Doptions**, option of **convert**, 451
- Plücker coordinates, 520
- Poincaré-Lindstedt method, 590–594
- point**, style option, 410, 442, 475, 477
  - symbol, 473
- point**, procedure from, **geometry** package, 436, 437
  - plottools** package, 461
- POINT**, graphics primitive, 422, 451
- point plots, 410, 411
  - pointplot**, 459–461
- pointto**, 174
- polar coordinates, 415, 688
- polar form of a complex number, 326, 328
  - conversion to Cartesian form, 326, 328
- polarplot**, 428
- poly\_algebra**, 706, 731, 744, 745
- polygamma function, **Psi**, 50, 256, 354
- polygonplot**, 413
- polygonplot3d**, 448
- POLYGONS**, graphics primitive, 422, 423, 448, 451, 458
- polyhedra, 456, 457
- polyhedraplot**, 456
- polylogarithm, 369
- polynomial mapping, 22, 742
  - inverse of, 22, 742
- polynomial reduction, 710, 711
- polynomial remainder sequence, 23, 24
- polynomial simplification, 25, 139–144, 154, 157, 176, 181–183, 185–187, 384, 385, 710–712
- polynomials, absolute factorization of, **AFactor**, 180
  - Bernoulli, 256
  - canonical forms of, 139, 384
  - characteristic, **CharacteristicPolynomial**, 635, 638, 641, 682
  - Chebyshev, **ChebyshevT**, 274, 278, 279, 288
  - coefficients in, **coeffs**, 141, 142, 146, 513, 678
  - collected form of, **collect**, 139–142, 182, 186
  - complex roots of, 487, 510
  - composition of, **compoly**, 25, 385
  - cyclotomic, 152
  - degree of, **degree**, 30, 140, 142, 211
  - discriminant of, **discrim**, 676
  - division of, **quo**, 75, 142
  - expanded canonical form of, 139–144, 154, 157, 176, 182, 185, 186
  - expansion of, **expand**, 7, 139–144, 175–178, 651, 653, 677, 678
  - factorization of, **factor**, 13, 18, 19, 21, 27, 56, 110, 136, 143, 144, 146, 170, 171, 178, 179, 184, 216, 256, 428, 500, 502, 505, 515, 517, 563, 587, 603, 608, 632, 638, 675, 676, 682, 683, 692
  - Fibonacci, 514
  - full expansion of, 176
  - gcd of, **gcd**, **gcdex**, 23, 143, 161

- greatest common divisor of, 23, 143, 161
- Hermite, **HermiteH**, 181, 566
- Horner form of, **hornerform**, 148, 149, 182, 183, 276
- internal representation of, 153–159
- Laguerre, **LaguerreL**, 375
- leading coefficient of, **lcoeff**, 140–142
- Legendre, **LegendreP**, 212
- minimal, **MinimalPolynomial**, 635, 641
- multivariate, 145, 146
- normal forms of, **normalf**, 181, 182, 384, 710–712, 743
- numerical complex roots of, 510
- ordering of terms in, 141, 145, 298, 383, 501, 507, 508
- orthogonal, 278
- Chebyshev, **ChebyshevT**, 274, 278, 279, 288
  - Hermite, **HermiteH**, 181, 566
  - Laguerre, **LaguerreL**, 375
  - Legendre, **LegendreP**, 212
- partial expansion of, 176
- real roots of univariate, **realroot**, **sturm**, 512
- resultant of, **resultant**, 56, 500
- roots with multiplicity of, **roots**, 66
- sorting terms in, **sort**, 13, 23, 24, 111, 140, 141, 143, 145, 146, 175, 184, 187, 188, 385, 507
- symmetric, 743
- Taylor, 265, 276, 277
- univariate, 139–145
- PolynomialTools** package, 142, 179
- CoefficientList**, 142
  - Split**, 179
- popen**, 117, 119
- posint**, data type, 204, 206, 249, 250, 311, 343
- PostScript, 405
- powadd**, 282
- powcreate**, 282, 284
- powdiff**, 283
- power**, option of simplification procedure, 368, 373, 374, 389
- power series, 281–284
- adding, **powadd**, 282
- composing, **compose**, 282
- creating, **powcreate**, 282
- derivative of, **powdiff**, 283
- exponentiation of, **powexp**, 283
- integrating, **powint**, 283
- inverting, **inverse**, 282
- logarithm of, **powlog**, 283
- multiplying, **multiply**, 282
- package, **powseries**, 281–284, 562, 565
- reversion of, **reversion**, 283
- showing first terms of, **tpsform**, 282–284, 562, 565, 566
- solving ODEs by, **dsolve**, 564
- FPseries**, 562
  - FTseries**, 643
  - powsolve**, 562
- powers, combining, 368, 369
- expansion of, 360, 361
- simplification of, 373, 374
- powseries** package, 281–284, 562
- compose**, 283
  - inverse**, 283
  - multiply**, 283
  - powadd**, 282
  - powcreate**, 282, 284
  - powdiff**, 283
  - powexp**, 283
  - powint**, 283
  - powlog**, 283
  - powsolve**, 562–564
  - reversion**, 283
  - tpsform**, 282–284, 562, 565, 566
- powerset**, 293
- powexp**, 283
- powint**, 283
- powlog**, 283
- powsolve**, 562–564
- precedence rules, of arithmetic operations, 34, 633, 634
- prepending elements to lists, 297
- presence in, testing of, **member**, 101, 102, 203, 293, 295, 686
- prettyprint**, 101, 135
- prettyprinting, 29, 101, 135, 326
- previous results, 46

forgetting, **forget**, 98, 207, 208, 276, 287, 341, 358, 364, 486, 489, 499  
 referring to, 17, 46, 79, 83, 100, 101, 208, 209  
 remembering, 98, 193, 206–208, 267, 358  
**primality testing**, **isprime**, 43, 152, 294  
 prime number,  $i^{\text{th}}$ , **ithprime**, 75, 76, 291, 315, 411  
 next, **nextprime**, 43  
 testing for, **isprime**, 43, 152, 294  
**Primfield**, 56, 64, 230  
 primitive part of a polynomial, **primpart**, 716  
**primpart**, 716  
 principal branch of multiple-valued function, 53, 58, 190, 355, 484, 489  
**print**, 21, 24, 92, 99, 100, 112, 116, 135, 136, 193, 206, 290, 316, 318, 345, 357, 391, 472, 550, 620  
**printf**, 119–121  
 conversion codes of, 120, 121  
 flags of, 120  
 printing routine, defining your own, 326  
**printlevel**, 51, 79, 98–100, 109, 197  
 private libraries, 101, 111, 112, 391–397  
**profile**, kernel option, 205, 206  
 problem of specialization, 173, 174, 361  
 procedures, defining, 201–204, 206  
 recursive, 204–208  
**process** package, 118  
**pipe**, 118  
**popen**, 119  
**procname**, 193, 197–199, 203, 327, 328, 345, 441  
**projection**, plot option, 443, 476  
**projection**, **geometry** procedure, 436, 437  
 prolate spheroidal coordinates, 689–693  
**prompt**, interface variable, 135  
 prompt, 6, 34, 106, 135, 136  
 change of, 135  
 properties, 89–93, 333–352  
 adding, **addproperty**, 338, 344, 345  
**attributes**, 88, 89  
 algebra of, 342, 343  
**AndProp**, 343, 344  
**BottomProp**, 343, 350, 352  
**composite**, 343, 350  
 constructors of, **and**, **not**, **or**, 343  
 defining, 89, 338, 344, 345  
**fraction**, 343, 350  
 functional, 351  
**GaussianInteger**, 344, 346  
**hasassumptions**, 89, 337–339, 349  
 hierarchy of, 343, 350–352  
**imaginary**, 344  
**InfinitelyDifferentiable**, 343  
**integer**, 343, 350  
 investigating, **is**, 197, 198, 338, 340, 341, 345, 347, 484  
**irrational**, 343, 350  
 lattice of, 343  
 linear, 344  
**LinearProp**, 344  
 modes of queries on, 341  
**\_EnvTry**, 341  
**monotonic**, 343, 351  
**MutuallyExclusive**, 344  
**natural**, 343  
**Non(0)**, 343, 344, 350  
**nonneg**, 344  
**nonnegint**, 11, 26, 294  
**Non(singular)**, 344, 351  
 numerical, 350  
**odd**, 343  
 on matrices, 351  
 parametric, 344  
**OddMap**, 344, 351  
**OrProp**, 344  
**posint**, 204, 206, 249, 250, 311, 343  
**PositiveDefinite**, 344, 351  
**prime**, 343, 344, 350  
**PropRange**, 344  
 queries on, 340, 347  
**realcons**, 70, 344  
**RealRange**, 20, 62, 90–93, 339–341, 343–349, 371, 387, 488, 509  
 removing, 90, 91, 335, 337

- saving variables with, 348, 349  
**SquareMatrix**, 344, 351  
**TopProp**, 343, 344, 350, 352  
 unary, 344, 351  
 property, functions, 343, 345  
   names, 344  
   ranges, 344  
**property/ChildTable**, 344, 345  
**property/object**, 347, 348  
**property/ParentTable**, 344, 345  
**PropRange**, 344  
**protect**, 48, 49, 79, 87  
**protected**, attribute, 88, 155  
 protected words, 78, 155  
 pseudo steady state, 494, 495  
**Psi**, 50, 256, 354  
 Puiseux series, 267  
 PUMA robot, 694  
 punctuation marks, 34, 83, 86, 87,  
   122, 201  
 pure lexicographic ordering, **plex**,  
   145, 383, 501, 507, 704,  
   706, 711–713, 715, 716,  
   720, 729–733, 735, 736, 738,  
   740–745
- quantile**, format of **scatterplot**, 465  
 quantum chemistry, 680  
 quantum mechanics, 564  
 quartic equation, 508  
 question mark (?), 36, 37, 39, 77, 122,  
   301  
**quiet**, interface variable, 132, 134  
   option of **declare**, 524, 526, 541,  
   543, 544, 551, 566, 570, 580,  
   601, 603, 606, 607, 609  
**quit**, 35  
 quitting Maple, **quit**, **done**, **stop**, 35  
**Quo**, 144  
**quo**, 75, 142  
 quotation marks, *see* accents  
 quotes, back, 80, 81, 97  
   double, 81–83  
   left, 80, 81, 97  
   right, 67, 68, 71, 75, 76, 83, 87,  
   143, 202, 213, 291, 318, 440  
**Quotient**, 144  
 quotient, of integer division, **iquo**, 44  
   of polynomial division, **quo**, 75,  
   142  
 QR decomposition of matrix,  
**QRDecomposition**, 646  
**radical**, option of **convert**, 8, 50, 55,  
   57, 179, 180, 185, 378, 486, 498  
 radical numbers, 46, 53–56  
 radical simplification, 374  
 radicals, 52, 232, 487, 636  
   arithmetic of, 46, 53  
   combining, 369  
   conversion to **RootOfs**, 55, 185,  
   232  
   denesting of, **radnormal**, 12, 54,  
   252  
   expansion of, 53, 56, 360, 361, 481,  
   482  
   simplification of, **radsimp** 373, 374  
**radnormal**, 12, 54, 252  
**radsimp**, 374  
 raising to a power, *see* exponentiation  
**rand**, 212, 294, 355, 628  
**RandomMatrix**, 626, 628, 629, 693,  
   694  
   option, **density**, 629  
   **entries**, 628  
   **generator**, 628, 629  
   **outputoptions**, 628, 629  
**random** subpackage of **stats**, 104,  
   124, 125, 248, 429  
 noraml distribution, **normald**, 248  
 uniform distribution, **uniform**,  
   124, 125, 429  
 random matrix, **RandomMatrix**,  
   626, 628, 693, 694  
 numbers, **rand**, 212, 294, 355, 628  
 polynomials, **Randpoly**,  
   **randpoly**, 212, 629, 694  
   vectors, **RandomVector**, 628  
**randpoly**, 212  
**Randpoly**, 694  
 range, data type, 292  
 rank, of matrix, **Rank**, 634, 641, 674  
**Rank**, 103, 634, 641, 674  
 rational canonical form of a matrix,  
**FrobeniusForm**, 646, 652  
 rational expressions, combining of,  
**combine**, 370

normal form of, **normal**, 183–186  
 rational functions, 147, 148, 159–161,  
     183, 184, 187, 226–228, 273  
 continued fraction form of,  
     **confrac**, **confracform**, 148,  
     149, 273, 278  
 expansion of, 184  
 factored normal form of, 183  
 full partial fraction decomposition  
     of, **fullparfrac**, 150  
 integration of, 226–229  
 internal representation of, 159–161  
 Padé approximation of, **pade**, 273  
 partial fraction decomposition of,  
     **parfrac**, 149–151  
 rational numbers, 41, 45, 46  
     arithmetic of, 41  
     canonical form of, 45  
     continued fractions of, **confrac**,  
         **cfrac**, 87  
     internal representation of, 46  
**rationalize**, 53, 482  
**RATLODE**, 531  
**ratrecon**, 273  
 raw characters, 122  
**Re**, 58, 249, 336, 340  
 real part of a complex number, **Re**,  
     58, 249, 336, 340  
**READ**, 117, 125, 126  
**read**, 22, 96, 107, 109, 110, 112,  
     348, 349, 396, 422  
**readbytes**, 118  
**readdata**, 113, 114, 315  
 reading, data,  
     **importdata**, 114, 466, 468  
     **ImportMatrix**, 114, 115, 315,  
         466, 467, 629  
     in binary format, **readbytes**,  
         118  
     **readdata**, 113, 114, 315  
     **readline**, 108, 112–114, 125–127  
 expressions from files, *see* **read**  
 plots, 422  
**readline**, 108, 112–114, 125–127  
**readonly**, option of, **Matrix** and  
     **Vector**, 310, 621  
**rtable**, 323  
**real**, option of, **convert**, 149  
**limit**, 285  
 real numbers, *see* floating-point  
     numbers  
 real-valued root, 53, 374, 387, 388  
**RealDomain** package, 334, 354, 356,  
     372, 374, 386–388, 484  
**RealRange**, 20, 62, 90–93, 339–341,  
     343–349, 371, 387, 488, 509  
**realroot**, 512  
 rearranging lists, **sort**, 295, 298,  
     706–709  
 polynomials, **sort**, 13, 23, 24, 111,  
     140, 141, 143, 145, 146, 175,  
     184, 187, 188, 385, 507  
**Record**, 325, 326  
 record data structure, 325, 326  
**rectangle**, 423, 435, 458  
**rectangular**, coordinates, 688, 689  
     grid style, 476  
 rectangular coordinates, 688, 689  
 recurrence equations, 514–517, 562,  
     563  
     solving, **rsolve**, 514–517  
 recurrence relations, *see* recurrence  
     equations  
 recursion, infinite, 68, 75  
 recursive procedures, 204–208  
 redrawing a plot, 419, 420  
**reduce**, 711  
**REDUCE**, 3, 232  
**ReducedRowEchelonForm**, 642,  
     646, 674  
 reduction, of polynomials, 710, 711  
 regression analysis, **leastsquare**, 468  
     **LeastSquares**, 469  
**related**, 39  
 relational operators, ( $>$ ,  $\leq$ ,  $<$ , etc.),  
     84  
**reinitialize**, option of **forget**, 358  
**rem**, 75, 142  
**remainder**, **algsubs** option, 167  
 remainder, of integer division, **irem**,  
     44  
     of polynomial division, **rem**, 23,  
         75, 142  
 remember option, 193, 205–208, 222,  
     267, 341, 358, 486  
 remember tables, 193, 206, 363  
     clearing, *see* **forget**  
     empty, **NULL**, 206, 208

- of functions, 193, 206–208, 363
- remembering, previous results, 98, 193, 206–208, 267, 358
- results of functions, 193, 206–208
- Remez algorithm, 279, 280
- remove**, 210, 211, 294, 599
- removing, discontinuities in plots, **discont**, 408, 474
- elements, based on criteria, **remove**, 210, 211, 294, 599
- elements from tables, 207, 318
- files, **fremove**, 124
- properties, 90, 91, 335, 337
- repeated evaluation, endless, 68, 75
- replace mode of Maple, 96
- replacing, elements of lists, 297, 298
  - parts in expressions, **subs**, **subsop**, 169–171, 208, 297, 317, 327, 486, 517
- representation tree, 157
- reserved words, list of, **?reserved**, 78
- resetting plot device, **plotsetup**, 403, 405, 406
- residue**, 101, 102, 254
- resolution**, plot option, 419, 474
- resonant terms, 592, 597
- restart**, 35, 72
- restarting Maple, **restart**, 35, 72
- resultant**, 56, 500, 501
- results, display of, 29, 30, 34, 99–101, 134, 483
  - forgetting previous, *see* **forget**
  - layout of large, 135, 482
  - referring to previous, 17, 46, 79, 83, 100, 101, 208, 209
  - remembering, 193, 206–208
- return**, 78, 92, 93, 130, 131, 202, 312, 392, 394
- Return key, 6, 34
- returnvariablename**, 130
- reverse automatic differentiation, 222
- reversed apostrophes, 80, 81, 97
- reversion**, 283
- reversion, of a power series, **reversion**, 283
- Reynold's number, 514
- RGB**, color space, 413, 472, 475
- RGB color chart, 413, 414
- rgf\_encode**, 13
- rhs**, right-hand sides of equations, 191, 226–228, 230, 233, 235, 236, 241, 245, 252–254, 372, 373, 378, 383, 468, 499, 501, 504, 517, 530, 535–538, 543, 547, 553, 557, 561, 587, 589, 592–594, 603–606, 613
- Riccati equation, 532, 536, 550
- Rich Text Format (RTF), 30
- Riemann hypothesis, 59
- Riemann  $\zeta$ -function, **Zeta**, 49, 50, 59, 362
  - critical line of, 59
  - expansion of, 50, 362
- right**, option of **limit**, 285
- right-hand sides, of equations, *see* **rhs**
- right quotes, 67, 68, 71, 75, 76, 83, 87, 143, 202, 213, 291, 318, 440
- Risch algorithm, for integration, 8, 20, 225, 228, 229
- Risch-Norman front end, for integration, 229
- rkf45**, Runge-Kutta method, 566, 577
- robot arm, tip of, 665
- ROMIN robot, 699, 700, 739, 740
- root-finding, **roots**, 56, 66
  - numerical, **realroot**, **sturm**, 512
- Root0f**, argument of **convert**, 55, 185, 232, 638
- RootOf**, 54–57, 78, 144, 177–180, 230, 232, 233, 272, 326, 483–487, 497–499, 538, 637, 638
  - avoiding indexing of, 56, 486, 487
  - extra arguments in, 57, 498
    - index**, 57
    - label**, 57
  - simplification of, 54, 55
- roots**, 66
- roots of negative numbers, **surd**, 53
- rotate**, 424, 425
- rotational velocity, 665–667
- rotation of graphics objects, 424, 425
- Rothstein/Trager integration method, 228
- Row**, 641
- row rank of matrix, **Rank**, 634, 641, 674

row reduced echelon form,  
**HermiteForm**, 620, 626, 646, 653  
**ReducedRowEchelonForm**, 642,  
  646, 674  
  RREF method of  
    **LUDecomposition**, 647, 654  
row space of matrix, **RowSpace**,  
  635, 645  
**RowDimension**, 641, 677, 693  
**RowSpace**, 635, 645  
RREF, option in **LUDecomposition**,  
  647, 654  
**rsolve**, 514–516  
**rtable**, 301–303, 322, 324  
  options  
    **datatype**, 322  
    **fill**, 323, 324  
    **order**, 323  
    **readonly**, 323  
    **storage**, 324  
    **subtype**, 324  
  special, 324  
**rtable\_dims**, 325  
**rtable\_num\_dims**, 325  
**rtable\_num\_elems**, 325  
**rtablesizer**, interface variable, 323  
RTF, Rich Text Format, 30  
**Rule[change, ...]**, change of  
  variables in integration, 252  
**Rule[parts, ...]**, integration by  
  parts, 253  
run-time messages, 107  
Runge-Kutta methods, 566, 569, 572,  
  574, 576  
  
S-polynomial, 712, 713, 717, 718  
**sample**, plot option, 410, 420, 474  
sampling, 410, 419–421, 420, 474  
saturation, of colors, 414  
**save**, 107, 108, 110, 112, 348, 349,  
  421  
**savelib**, 112, 395, 396  
**savelibname**, 112, 395, 396  
**SaveToLibrary**, 396  
saving, expressions to files, 107, 108,  
  110, 112, 348, 349, 421  
plots, 421  
variables with associated  
  properties, 348  
  
**scalar**, special indexing, 310  
scalar multiplication of matrices and  
  vectors, **ScalarMultiply**, 14,  
  630  
**ScalarMultiply**, 14, 630  
scalar product, **DotProduct**, 632,  
  645  
**scale**, 424, 425, 457  
**scaling**, plot option, 407, 473  
**SCALING**, graphics primitive, 418, 423,  
  448, 452  
**scanf**, 119  
scatter plots, **scatterplot**, 465, 468  
  **format**, **quantile**, 468  
  **symmetry**, 468  
Scherk's minimal surface, 479  
SCHOONSCHIP, 3  
Schrödinger equation, 564, 586  
Schur form of a matrix, **SchurForm**,  
  646  
**SchurForm**, 646  
scientific constants,  
  **ScientificConstants**,  
  48  
scientific notation, 120  
**ScientificConstants** package, 48  
Scratchpad, 5  
screen dump of, help browser, 36  
  help page, 36, 37  
Maple session, 6  
text search, 40  
topic search, 39  
**screenwidth**, interface variable, 132  
searching, by topic, 39  
  full-text, 40  
  in multiple libraries, 109, 112, 396  
**SearchText**, 70  
**sec**, 50  
secant method, 32, 511  
**sech**, 50  
secular terms, 592  
**select**, 70, 210, 211, 294, 484, 498,  
  587, 599, 603, 616  
selection, **select**, by criteria, 70, 210,  
  211, 294, 587, 599, 603, 616  
  in sequences, set, and lists, 300  
  of a root, 57  
  of solutions, 484, 498  
  of subexpressions, 70

- selection operator, `[]`, 291, 293, 316  
 selection strategy, in reduction process, 717  
**selectremove**, 294  
 semicolon, 6, 34, 86, 87, 96  
**seq**, 105, 125, 126, 187, 248, 290–292, 298, 300, 301, 394, 395, 411, 413–415, 425, 429, 435, 445, 457, 463, 465, 466, 471, 490, 493, 494, 561, 576, 583, 591, 593–595, 614, 615, 664–667, 673, 677, 686, 691, 692  
 vs. `$`, 291  
 sequence operator, `$`, 214, 291  
 sequences, 289–292  
   catenating, 290  
   creating, `seq`, 290–292  
   empty, `NULL`, 290  
   internal representation of, 290  
   selecting elements of, 291  
 sequential substitution with **subs**, 163, 164  
**series**, 10, 201, 266–272, 274–276, 285–287, 535, 538, 561, 613, 614  
 series, Chebyshev, **chebyshev**, 273, 278, 279  
   converting into polynomials, 270, 273, 274  
   expansion point of, 265, 266, 274  
   fine-tuning of, 275, 276  
   generalized, **series**, 268  
   integrating and differentiating, 271  
   internal representation of, 266  
   Laurent, **laurent**, **series**, 266, 267  
   leading term of, **leadterm**, 271, 286  
   of multivariate functions, **mtaylor**, 111, 270  
   Puiseux, 267  
   reversion of, 271, 272, 283  
   Taylor, **taylor**, 265, 276, 277  
   truncation order of, **order**, 266, 267  
 sessions, starting Maple, 6, 33  
   ending Maple, 35  
**setAttribute**, 88, 89, 125, 331, 462  
**SetCoordinateParameters**, 690  
**SetCoordinates**, 687, 690, 691  
**setoptions**, 416  
**setoptions3d**, 447  
 sets, 292–294  
   creating all subsets of, **powerset**, 293  
   difference of, **minus**, 293  
   empty, `{ }`, 292  
   internal representation of, 292  
   intersection of, **intersect**, 293  
   membership test of, **member**, 101, 102, 293, 686  
   number of elements in, **nops**, 293  
   selecting elements of, 293  
   term ordering in, 292  
   union of, **union**, 293  
   vs. lists, 295  
 setting default options of plotting,  
   **plotsetup**, 403, 405, 406  
   **setoptions**, 416  
   **setoptions3d**, 447  
**sgn**, 202  
**shading**, plot option, 442, 443, 476  
**SHADING**, graphics primitive, 451, 452  
 Shape lemma, 732  
 share library, 29, 101, 105  
 sharp symbol, `#`, 9, 97  
 SHEEP, 3  
**showassumed**, 89, 91, 92, 136, 242, 333, 436, 687  
 Sierpinsky's tetrahedron, 456, 457  
 $\sigma$ -bonds, 680, 681  
**signum**, option of **convert**, 334  
**signum**, 19, 60, 61, 80, 251, 334, 336, 340  
 silent operation, 134  
 SIMATH, 3  
 similarity of matrices, **IsSimilar**, 642, 644  
 similarity transformation approach to identifiability, 673, 678–680  
 simplification, automatic, 7, 8, 62, 354–356  
   automatic vs. user-control, 7, 8, 147  
   canonical forms in, 181–183  
   controling, 354, 357, 358, 363, 366, 367, 370, 371, 386–391  
   correctness of, 8, 61, 354, 355  
   difficulties with automatic, 355  
   general purpose, **simplify**, 370–375

- normal form, *see normal*  
 of algebraic numbers, 53–55  
 of complex expressions, 58, 60, 61  
 of exponential mapping, 372  
 of gamma function, 375  
 of hyperbolic functions, 370, 371  
 of hypergeometric function, 375  
 of inverse hyperbolic functions, 371  
 of inverse trigonometric functions, 371  
 of logarithm, 372, 373  
 of powers, 373, 374  
 of radicals, **radsimp**, 373, 374  
 of **RootOfs**, 54, 55  
 of trigonometric functions, 370, 371, 379–382  
 polynomial, 25, 139–144, 154, 157, 176, 181–183, 185–187, 384, 385, 710–712, 742, 743  
 radical, 373, 374  
 restricting, 389–391  
 symbolic (*symbolic* option of **combine** and **simplify**), 32, 359–361, 364–369, 371–373, 388, 549, 597, 605, 630  
 table, 141  
 trigonometric, 370, 371, 379–382  
 validity of, 354, 355  
 with assumptions, **assuming**,  
   **assume** option of **simplify**, 15, 18, 19, 61, 78, 89, 198, 249–252, 254, 286, 334–336, 338, 339, 354, 356, 359–361, 364, 367, 369, 371–374, 380, 386–388, 524, 525, 527, 529, 561, 587, 610  
 with respect to side relations, 169, 17, 370, 382–386, 493  
 without validity checks, 355, 359–361, 364–369, 371–373, 388, 549, 597, 605, 630  
 simplification routines, defining your own, 391–397  
**simplify**, 31, 46, 54, 89, 169, 327, 333, 370, 371, 375  
**arctrig**, 371  
**delay**, 388  
**exp**, 375, 389  
**ln**, 274, 367, 372, 389  
**power**, 374  
**radical**, 374, 375  
**size**, 391  
**symbolic**, 32, 359–361, 371–373, 388, 549, 597, 605, 630  
**trig**, 371, 375  
**Simplify**, 232  
 Simson’s Theorem, 437  
 simultaneous substitution, 164  
**sin**, 50  
 sine integral, **Si**, 478  
 single-precision integers, 41  
 SINGULAR, 3  
 singular state, of robot arm, 668, 669  
 singular values of matrix, **SingularValues**, 641, 685  
 singularity, 235, 239, 240, 428  
**SingularValues**, 641, 685  
**sinh**, 50  
 size, of integers, 42  
   of expressions, 42, 70, 82, 126, 187  
   of strings, 70, 82, 126  
 slash, /, division, 34  
   in names, 80–82  
 Slater-Zener type atomic orbitals, 691  
**Slode**, package, 562, 563  
**FPseries**, 562, 563  
**FTseries**, 562, 563  
**Smith**, 646, 652  
 Smith form of a matrix, **SmithForm**, 646, 652  
**SmithForm**, 646, 652  
 smoothing, of data by convolution, 248, 249  
   of plots, *see numpoints*  
**SNAP** package, 144  
**Quotient**, 144  
**SOLID**, line style, 473  
 solids, 3D, 456, 457  
 solution bases for ODEs,  
   **output=basis** option, 536, 538  
 solutions, assigning, **assign**, 493  
   checking of, for determining  
     system, **symtest**, 546, 552, 560  
     equations, 481, 482, 496, 497, 502, 503  
     ordinary differential equations,  
       **odetest**, 524, 525, 537, 538

- partial differential equations,
  - pdetest**, 603, 604, 606, 607, 609
  - parametrized, 491, 492
  - selecting, 498
  - substituting, 482, 496, 497
- SolutionsMayBeLost**, 485
- solvable types, of 1st order ODEs, 532
  - of 2nd order ODEs, 533
- solve**, 9, 65, 71, 127, 272, 481–487, 492, 493, 503, 509
  - abbreviations in, 483, 484
  - arguments of, 483
  - difficulties with, 485–492
  - no solutions found by, 485, 486
- solvefor**, 492
- solve/identity**, 518, 519
- solving,
  - analytically, **solve**, 481–509
  - inequalities, **solve**, 508, 509
  - numerically, **fsolve**, 510–512
  - over integers, **isolve**, 512
  - recurrence equations, **rsolve**, 514–517
  - systems of equations, 492–508
  - systems of linear equations, 492–495
  - systems of nonlinear equations, 495–508
  - systems of polynomial equations, 495–508, 698–702
- solving ODEs, analytically, **dsolve**, 523–538
  - by integral transforms, 530, 531, 538
  - in explicit form, 525, 538
  - in implicit form, 525, 538
- by perturbation methods, 590–600
- by power series, 561–566
- by Taylor series, 560, 561
- numerically, 566–580
  - by Euler method, 576
  - by Gear extrapolation methods, 576
  - by Gear multistep extrapolation methods, 576
  - by Heun's method, 576
  - by Livermore Stiff ODE solver, 576, 577,
- by Runge-Kutta methods, 576, 577
- by Taylor series method, 577, 578
- solving PDEs, analytically, **pdsolve**, 521, 545, 546, 548, 550, 552, 553, 555, 601, 601–610
  - by integral transforms, 602
  - by Lie point symmetries, 615–618
  - numerically, **PDEplot**, 585, 601
- sort**, 13, 23, 24, 111, 140, 141, 143, 145, 146, 175, 184, 187, 188, 295, 298, 385, 507
- sorting, lists, 298, 706–709
  - polynomials, 13, 23, 24, 111, 140, 141, 143, 145, 146, 175, 184, 187, 188, 385, 507
- source code, viewing, 92, 135, 136, 357, 391
- sources of light, **light**, 446, 447, 476
- spacecurve**, 452, 574
- SPACEDASH**, 473
- SPACEDOT**, 473
- spaces, 9, 35, 80
- sparse matrix, 315, 622
- special, characters, 80
  - indexing, 311–313, 317
  - matrices, 309–315
  - tables, 318
- specialization, problem of, 173, 174, 335, 361
- specialized problems vs. generic problems, 173, 174, 335, 361
- sphere, plot of, 453
- spherical coordinates, 453, 689
- Split**, 179
- splitting field, **Split**, 179
- spoly**, 713, 715
- sprintf**, 119
- sqrfree**, 180
- Sqrfree**, 180
- sqrt**, 50
- square brackets, 294, 296, 300, 328
- square root function, **sqrt**, 50
- square root of -1, **I**, 58, 62, 78
- square roots,
  - combining, 364, 369
  - denesting, 54
  - simplifying, 333, 334

- square-free factorization, **Sqrfree**,  
     **sqrfree**, 180  
 square-free form, 180  
**sscanf**, 119, 122, 123  
     conversion codes of, 123  
**ssystem**, 108, 466  
 stacking matrices, 623, 624, 642, 666,  
     668, 674, 675  
 Stanford Manipulator, 663–669  
 starting Maple, 6, 33  
 start-up files, *see* initialization files  
**statement**, option of **parse**, 124  
 stationary point, 5, 9  
 statistical plots, 465–469  
 Statistics Netherlands, 126, 459  
**stats** package, 104, 105, 114, 124,  
     248, 429, 459, 465  
     **anova**, subpackage of, 104  
     **describe**, subpackage of, 104  
     **fit**, subpackage of, 104, 468  
     **histogram**, 462, 463, 465  
     **importdata**, 114, 466, 468  
     **mean**, 105  
     **random**, subpackage of, 104  
         **normaldf**, 248  
         **uniform**, 124, 125, 429  
     **statevalf**, subpackage of, 104  
     **statplots**, subpackage of, 104,  
         465, 466, 468  
     **boxplot**, 465  
     **changecolor**, 466  
     **histogram**, 462, 463, 465, 466  
     notched, format of **boxplot**, 465  
     quantile format of **scatterplot**,  
         465  
     **scatterplot**, 465  
     **symmetry**, format of **scatterplot**,  
         465  
     **xshift**, 466  
     **xychange**, 466  
     **transform**, subpackage of, 104  
 steady state, 503  
**stellate**, 424  
 STENSOR, 3  
 step function, **Heaviside**, 199, 242,  
     246, 249, 421, 422, 478, 529  
 stereospecificity, of electrophyllic  
     substitution, 687  
 Stirling's formula, 274  
 stoichiometry, of chemical reaction,  
     519  
**stop**, 35  
 stopping Maple, 35  
**storage**, option of, **Matrix** 621, 622  
     **rtable**, 324  
 storage management, 30, 134  
 storage of expressions, 42, 43, 45, 47,  
     58, 73, 74, 77, 83, 154, 156,  
     157, 266, 290  
**Strand**, 5  
 streams, 117  
     closing of, **close**, **fclose**, **pclose**,  
         117  
     listing status of, **iostatus**, 119  
     opening of, **fopen**, **open**,  
         **popen**, 117–119  
     status of, **iostatus**, 119  
     types of, 117  
     writing to, 106, 107, 117–119  
 strings, beginning with underscore,  
     54, 77, 78, 80, 562  
     case sensitivity, 77  
     characters allowed in, 77  
     compared to names, 81  
     concatenation of, **cat**, **||**, 82, 84,  
         292  
     length limitation of, 77  
     parsing of, **parse**, 124  
     size of, **length**, 70, 82, 126  
     size limitation of, 77  
     special characters in, 77  
**StringTools** package, 82  
 structural identifiability, definition of,  
     671, 672  
     of systems, 671–680  
 structural operations on matrices,  
     641–644  
**Student** package, 252  
     change of variables in integration,  
         **Rule[change, ...]**, 25  
     hints, **Hint**, 252, 253  
     integration by parts,  
         **Rule[parts, ...]**, 253  
**sturm**, 512  
 Sturm's theorem, 512  
**style**, plot option, 410–412, 417, 442,  
     475, 477  
 subexpressions, freezing of, 172

- sharing of common, 45, 155
- recognizing, 155, 165–167, 169
- selection of, 70
- SubMatrix**, 16, 626, 641, 643, 665, 677
- subroutines, exiting of, **error**, 130, 204, 431
- subs**, 14, 15, 18, 56, 57, 66, 79, 107, 110, 111, 127, 155, 162–166, 169, 171–174, 187, 191, 209, 210, 215, 218, 235, 252, 269, 270 274, 278, 279, 298, 337, 381, 382, 385, 386, 389, 422, 425, 428, 445, 454, 458, 481, 491, 497, 498, 502, 507, 513–517, 525, 528 543, 561, 564, 565, 571, 595–597, 605, 607, 608, 610, 612, 616, 635, 668, 678, 679, 682, 690, 691
  - vs. **algsubs**, 166, 168
  - vs. **subsop**, 169
- subscripted functions, 203
- subscripted names, 68, 69, 79, 84, 85, 203, 316, 341, 673
  - assignment of, 316
- subsop**, 169–171, 208, 297, 317, 327, 486, 517
  - vs. **applyop**, 171
  - vs. **subs**, 169
- substitution, 66, 155, 162–173
  - algebraic, **algsubs**, 166–168, 381
  - multiple, 164, 165
  - of operands, **subsop**, 169–171, 208, 297, 317, 327, 486, 517
  - of solutions into equations, 496, 497
  - sequential, **subs**, 163, 164
  - simultaneous, **subs**, 164
  - trigonometric, **trigsubs**, 379–382
  - vs. assignment, 66, 67
- subtype**, option of **rtable**, 324
- SubVector**, 626, 641, 665
- sugar strategy, 717
- sum**, 75, 76, 256, 258
- Sum**, 255–260
- SumBasis**, 645, 646
- summation, 255–260
  - Cesaro, 260
  - finite, **add**, 255
  - numerical, 258–260
- symbolic, 256–258
- summation method, Abramov's, 256, 257
  - (extended) Gosper, 257
  - based on Bernoulli polynomials, 256
  - based on hypergeometric identities, 257, 258
  - Wilf-Zeilberger, 256
- sums, divergent, 260
- sumtools**, **SumTools**, 256, 258
- suppressing, evaluation, 75, 76, 439
  - full trigonometric expansion, 357, 358
- surd**, 53
- surface types, list of, 84
- surfdata**, 614
- surgery of expressions, **dismantle**, 159, 174
- Sylvester matrix, **SylvesterMatrix**, 624
- symbol**, plot option,
  - box**, 411, 473
  - circle**, 410, 429, 460, 461, 473, 568, 580
  - cross**, 473
  - diamond**, 468, 473
  - point**, 473
- symbolic**, simplification option of,
  - combine**, 364–369
  - simplify**, 32, 359–361, 371–373, 388, 549, 597, 605, 630
  - sqrt**, 355
- symbolic-numeric algorithms for
  - polynomials, **SNAP**, 144
- symbolsize**, plot option, 410, 417, 429, 460, 461, 468, 473, 568
- symgen**, 546, 549, 550, 552, 555, 558, 560
  - method, **way**, 550
  - option, **HINT**, 550
- '**symgen/methods**', 550
- symmetric**, special indexing, 14, 20, 309–311, 318, 629, 634, 637
- symmetric, matrix, 310
  - table, 318
- symmetry**, format of **scatterplot**, 465

- symmetry, Lie, *see* Lie point symmetry  
 symmetry plots of data,  
     **symmetry** option of  
     **scatterplot**, 465, 580  
**syntest**, 546, 552, 560  
 synonyms, 102  
 synopsis, part of help text, 38  
 syntax errors, 35, 96, 97  
 system calls to external, **ssystem**,  
     108, 466  
 system theory, 669–680  
 systems,  
     as programming languages, 11  
     facilities of, 11  
     historical survey of, 2–5  
     of general purpose, 3–5  
     of special purpose, 3  
     properties of, 5–11
- table**, 125–127, 316–318, 614, 615  
 tables, 316–318  
     changing elements of, 316, 321  
     copying, **copy**, 321  
     creating, **table**, 316–318  
     empty, **table()**, 318  
     entries of, **entries**, 316  
     evaluation of, 319–321  
     indices of, **indices**, 316, 318, 462,  
         463, 534  
     internal representation of, 319  
     special, 318  
     symmetric, 318  
     unassigning entries of, 318  
 tabulates, 122  
 tagging of expressions, 88, 89  
**tan**, 50  
 tangent surface to twisted cubic, 746  
**tanh**, 50  
**taylor**, 265, 276, 278  
 Taylor polynomials, 265, 276, 278  
 Taylor remainder theorem, 277  
 Taylor series expansion, **mtaylor**,  
     111, 270  
     extracting coefficients of,  
         **coeftayl**, 271, 540, 541  
     **taylor**, 265  
 Taylor series method, for solving  
     ODEs, 560, 561, 576, 578
- tcoeff**, 141, 142  
**tdeg**, 145, 508, 704–706, 721, 722,  
     724, 725, 730, 731  
 term, of a polynomial, 704  
     leading, **leadterm**, 709, 710  
 term ordering, admissible, 709, 717  
     defining you own, 707  
     elimination, **lexdeg**, 704, 706, 709  
     graded lexicographic, 704, 707, 708  
     graded reverse lexicographic, *see*  
         total degree  
     in sets, 292
- matrix**, **matrix**, 707, 708  
     pure lexicographic, **plex**, 145, 383,  
         501, 507, 704, 706, 711–713,  
         715, 716, 720, 729–733, 735,  
         736, 738, 740–745  
     total degree, **tdeg**, 145, 508,  
         704–706, 721, 722, 724, 725,  
         730, 731  
     weighted reverse lexicographic,  
         **wdeg**, 704, 705, 707, 708
- termorder**, 706–709, 731, 744, 745  
**testeq**, 44, 142, 245, 441, 497, 498,  
     524  
**testorder**, 706–709  
**Testzero**, 275, 286  
**TEXT**, 117–119, I/O option  
     graphics primitive, 422, 451  
**textplot**, 409, 410, 424, 435, 461, 463  
**thaw**, 172  
**thickness**, plot option, 406, 412,  
     417, 421, 422, 424, 426, 432,  
     437, 438, 447, 452, 455–457,  
     471, 474  
**THICKNESS**, graphics primitive, 422,  
     423  
**tickmarks**, plot option, 409, 474  
**TI-92**, 3  
**TI Interactive**, 4  
 tilde, as abbreviation for home  
     directory, 108  
     as indication of variable with  
         assumptions, 20, 61, 89–92,  
         333, 334, 339, 349  
     avoiding, **showassumed**, 89, 91, 92,  
         136, 242, 333, 436, 687  
     removal of, 92, 93  
**time**, 43, 44

**timelimit**, 136  
**title**, plot option, 10, 196, 409, 412, 413, 419, 420, 449, 461, 463, 474, 567, 568, 573–575, 584, 585, 611  
**titlefont**, plot option, 409, 410, 474  
Toeplitz matrix, **toeplitz**, 624, 629, 634, 636  
**ToeplitzMatrix**, 624, 629, 634, 636  
**ToInert**, 209  
top of rational expression, numerator, **numer**, 147, 161, 184, 381, 553, 557, 588, 677, 678  
topological matrix, 681, 682  
**TopProp**, 343, 344, 350, 352  
torusknot, plot of, 454  
total degree ordering, **tdeg**, 145, 508, 704–706, 721, 722, 724, 725, 730, 731  
**tpsform**, 282–284, 562, 565, 566  
    **Trace**, 14, 620, 634, 641  
trace, of matrix, **Trace**, 14, 634, 641  
Tractrix, 527, 528  
Trager’s integration method, 230  
trailing coefficient, of polynomials, **tcoeff**, 141, 142  
transfer function approach, of linear systems, 672, 677  
**transform**, 424, 425  
transforms, Fourier, **fourier**, 241, 246, 247, 612  
    Fourier-Bessel, **hankel**, 241, 249  
    Fourier-Cosine, **fouriercos**, 241, 249  
    Fourier-Sine, **fouriersin**, 241  
    Hankel, **hankel**, 241, 249  
    Hilbert, **hilbert**, 241  
    inverse Fourier, **invfourier**, 246, 247, 613  
    inverse Laplace, **invlaplace**, 242–245  
    inverse  $z$ -, **invztrans**, 517  
    Laplace, **laplace**, 241–246  
    Mellin, **mellin**, 241, 250  
     $z$ -, **ztrans**, 516  
**translate**, 424, 425  
translational velocity, 665  
transpose, of matrix, **Transpose**, 14, 632, 641, 644, 648, 649, 667, 676, 682, 686, 691–693  
**Transpose**, 14, 632, 641, 644, 648, 649, 667, 676, 682, 686, 691–693  
**transpose**, I/O option, 115, 467  
**triangle**, 436, 437  
**triangular**, grid style, 476  
    special indexing, 309, 310, 324, 629, 655  
**tridiagonal**, user-defined matrix option, 311  
**TridiagonalForm**, 646, 648, 649  
**trig**, option of, **combine**, 275, 365, 366, 380, 529, 592, 598  
    **convert**, 28, 376, 605, 608  
    **simplify**, 371, 375  
trigonometric equations, 489, 490  
trigonometric functions, 50  
    combining, 275, 365, 366, 380, 529, 592, 598  
    conversion into exponential forms, 375–377  
    conversion into sines and cosines, 376–378  
    conversions into tangents, 376  
    expansion of, 28, 356–359, 363, 379–381, 389, 390, 537  
    inverse, 50, 190, 369, 371  
    partial expansion of, 357  
    simplification of, 370, 371, 379–382  
trigonometric simplification, 370, 371, 379–382  
**trigssubs**, 379–382  
**true**, 48, 340  
truncation of powers, 167, 168  
truncation order, **Order**, 266, 267, 269  
**tryhard**, option of **optimize**, 129  
**tubeplot**, 454  
twisted cubic, tangent surface to, 746  
two-link robot arm, 569–576  
    animation of, 575, 576  
**type**, **dsolve** option,  
    **exact**, 523–538  
    **formal\_series**, 564  
    **numeric**, 566, 568, 572, 574, 577, 580  
    **series**, 560, 561

**type**, 85–87  
 \* , 392, 394  
 + , 85, 358, 391, 392, 394  
 And, 302, 303  
 array, 302, 319  
 Array, 302, 303  
 function, 511  
 indexed, 85, 92, 203  
 integer, 312, 317, 358, 392, 394  
 list, 302  
 matrix, 302  
 Matrix, 302, 303, 306  
 nonreal, 392  
 numeric, 358, 441  
 polar, 327, 328  
 Polar (user-defined procedure), 325  
 polynom, 86, 140  
 posint, 204  
 protected, 87  
 quadratic, 86  
 ratpoly, 159–161, 431  
 rtable, 302, 303, 306  
 set, 86, 317  
 specfunc, 392  
 vector, 302, 314  
 Vector, 302, 314, 324  
**typematch**, 85, 86, 518  
 types, = , 84, 358  
 <= , 84  
 <> , 84  
 \* , 84, 392, 394  
 + , 84, 358, 391, 392, 394  
 .. , 84  
 . , 84  
 ^ , 84  
 and, 84  
 And, 302, 303  
 anything, 56, 70, 202, 367, 368,  
 513, 518  
 array, 302, 319  
 Array, 302, 303  
 basic, 83–87  
 complex, 84  
 conversion between, 47, 87  
 exprseq, 84  
 float, 84  
 fraction, 84  
 function, 84  
 indexed, 84, 92, 203  
 integer, 84, 312, 317, 358, 392, 394  
 list, 84, 302  
 matrix, 302  
 Matrix, 302, 303, 306  
 nonnegint, 294  
 nonreal, 392  
 not, 84  
 numeric, 358, 441  
 of streams, 117  
 or, 84  
 polar, 327, 328  
 Polar (user-defined procedure), 325  
 polynom, 86, 140  
 posint, 204  
 power, 158  
 procedure, 84  
 product, 158  
 protected, 87  
 quadratic, 86  
 ratpoly, 159–161, 431  
 rtable, 302, 303, 306  
 series, 84, 270  
 set, 84, 317  
 specfunc, 392  
 string, 84  
 structured, 85  
 sum, 158  
 surface, 83, 8  
 symbol, 367  
 table, 84, 301  
 testing of, 85, 86  
 uneval, 84  
 vector, 302, 314  
 Vector, 302, 314, 324  
**unames**, 69, 70, 79  
**unapply**, 208, 209, 216, 218, 223,  
 224, 245, 276, 278, 279, 428,  
 592–594, 600, 610  
**unassign**, 71, 72  
 unassignment, of variables, 67–69, 71,  
 72, 75, 94, 217, 318  
 evaln, 69, 71, 72, 75, 318  
**unassign**, 71, 72  
 of table entries, 318  
 unbound variables, 66, 70  
 unconstrained, scaling option , 407,  
 473

undefined answer, 200, 260, 285, 355, 365, 374, 388  
 underscore, 54, 77, 78, 80, 562  
**uneval**, data type, 84  
 unevaluated expressions, 84  
**uniform**, animation option, 477  
**uniform**, statistics function, 124, 125, 429  
**union**, 293  
**unit**, special indexing, 310  
**Units** package, 318  
 univariate polynomials, 139–145  
   real roots of,  
     **realroot**, **sturm**, 512  
**univpoly**, 508, 735  
 Unix pipe, 117, 118  
 unknowns, in expressions, **indets**, 483, 486, 489, 491  
**unprotect**, 48, 49, 79  
**Unveil**, 187  
**?updates**, 39  
 upper-case vs. lower-case characters, 77  
**usage**, 38, 39  
**use**, 103, 326, 334, 354, 356, 372, 374, 387, 388, 484  
**UseHardwareFloats**, 79, 648, 657  
 user interface, worksheet, 6, 29, 30  
   text-based, 35  
**userinfo**, 98  
 user's home directory, ~ (tilde), 108  
  
 validity, of combine, 364  
   of simplifications, 354, 355  
**value**, 18, 19, 27, 192, 213, 214, 216, 226–228, 230–240, 250–254, 256–260, 285, 364, 365, 381, 517, 564, 610, 612  
 van der Pol equation, 533, 566–568, 581, 590, 591, 595, 596, 600, 700, 702, 734, 735  
 Vandermonde matrix,  
   **VandermondeMatrix**, 624  
**VandermondeMatrix**, 624  
 variables, assignment of, 66, 71, 72, 91, 297, 299, 300, 316, 493  
   assigned, **anames**, 69, 70, 72  
   bound, 69, 70, 72  
   dimensionless, 514, 561  
  
 elimination of, **eliminate**, 501, 502  
 environment, 60, 79, 80, 209, 267, 275  
 evaluation of, **eval**, 73–76  
 fluid, 79  
 free, 66, 70  
 global vs. local, 194, 195, 203  
 interface, **displayprecision**, 47, 135, 136, 276, 566, 570, 639, 685  
**echo**, 107, 137  
**errorbreak**, 136  
**imaginaryunit**, 62, 430, 432, 492  
**labeling**, 135  
**prettyprint**, 101, 135  
**prompt**, 135  
**quiet**, 132, 134  
**rtablesize**, 323  
**screenwidth**, 132  
**showassumed**, 89, 91, 92, 136, 242, 333, 436, 687  
**verboseproc**, 135, 136, 206, 345, 357, 391  
**warnlevel**, 583  
 local vs. global, 194, 195, 201–203  
 names of, 77–83  
 properties of, 88–93, 333–352  
 sorting of, **sort**, 298  
 types of, **type**, 83–86  
 unassignment, 72, 318  
 values of, 73, 74  
 variance, analysis of, **anova**, 104  
 variational method, 681  
**vector**, conversion into, 315, 330  
**vector**, 302, 311, 624  
**Vector**, 314, 622, 624  
 vector field, defining, **VectorField**, 687, 688  
   plotting of, **fieldplot**, 433  
 vector norm **VectorNorm**, 645  
 vector operations, 645  
 vector spaces, basis for, **Basis**, 645  
   basis for intersection of,  
     **IntersectionBasis**, 645, 646  
   basis for sum of, **SumBasis**, 645, 646  
**Vector [column]**, conversion into, 330  
**VectorAngle**, 645  
**VectorCalculus** package, 687–693

- curl, Curl**, 687, 688
- divergence, Divergence**, 687, 688
- Jacobian, Jacobian**, 666, 688, 691, 692
- Laplacian, Laplacian**, 687, 688, 693
- VectorField**, 687, 688
- VectorNorm**, 645
- vectors, 300
  - angle between, **VectorAngle**, 645
  - arithmetic of, 629–634
  - construction of, 621–624, 627, 628
  - creating, 621–624, 627, 628
  - cross product of, **CrossProduct**, 645
  - data structure, 42, 43, 45, 47, 58, 73, 74, 77, 83, 154, 156, 157, 266, 290
  - dimension of, 645
  - dot product, **DotProduct**, 632, 645
  - GramSchmidt orthogonalization of, **GramSchmidt**, 645, 686
  - in column notation, **column**, 621
  - inner product, **DotProduct**, 632, 645
  - normalization of, **Normalize**, 645, 686
  - outer product of, **CrossProduct**, 645
  - random, **RandVector**, 628
  - scalar multiplication of, **ScalarMultiply**, 14, 630
- Veil**, 187
- verboseproc**, interface variable, 135, 136, 206, 345, 357, 391
- vertical tabulate, 122
- vertices**, 727
- view**, plot option, 58, 408, 417, 420, 428, 433, 446, 456, 458, 460, 461, 474
- VIEW**, graphics primitive, 418, 419, 422
- viewing source code, 92, 135, 136, 357, 391
- viewpoint, in 3D graphics, **orientation**, 443–445, 476
- volume element, 15, 691, 692
- W*, function of Lambert, **LambertW**, 189–191, 372, 487, 489, 509, 526, 554, 557, 558
- warning messages, 192, 486, 620
- warnlevel**, interface variable, 583
- Waterloo Maple Inc., 29
- wave equation, 585, 586, 601
- wdeg**, 704, 705, 707 item weighted reverse lexicographic ordering, **wdeg**, 704, 705, 707
- whattype**, 79, 83, 86, 87, 114, 157–160, 174, 176, 266, 269, 290, 319, 322, 324, 327, , 622, 628, 633
- Whittaker functions, 535
- Wilf-Zeilberger summation method, 256
- wireframe**, plot option, 442, 477
- with**, 103, 104
- wmf**, graphics output, 406
- word size of computer, 134
- worksheet interface, 29, 30, 106
- WRITE**, 117, 119, 124
- writebytes**, 118
- writedata**, 115–116, 125, 315
- writeline**, 115, 118
- writeto**, 106, 107, 112, 132, 134, 205, 206
- writing, data
  - ExportMatrix**, 115, 315
  - writedata**, 115, 116, 125, 315
  - expressions to streams, 106–107, 117–118
  - to file or terminal, **writeto**, 106, 107, 112, 132, 134, 205, 206
  - your own conversion routines, 328
- Wronskian, 660
- xmaple**, 33
- XML, 30, 123
- xtickmarks**, plot option, 408, 409, 465, 475
- xy**, plot option, 442, 476
- xyz**, plot option, 443, 476
- ytickmarks**, 409, 465, 475
- z**, plot option, 442, 476
- z-transforms**, 516

inverse, **invztrans**, 517  
**zero**, speical indexing, 310  
zero equivalence, 181  
zero-finding, numerical 510, 511  
zero matrix, **ZeroMatrix**, 645  
zero, rounding to, **fnormal**, 145,  
    248, 332, 649, 686  
**Zeta**, 49, 50, 59, 362  
**zgrayscale**, plot option, 443, 476  
**zhue**, plot option, 443, 476  
**zip**, 210, 211, 247–249, 513  
zooming, in graphics, 420  
**ztrans**, 516