Overview of Important Areas

Magma Computational Algebra System

Computational Algebra Group
University of Sydney

John Cannon, Allan Steel, and Geoff Bailey

with prior contributions by Steve Donnelly, Mike Harrison, Bill Unger, and Mark Watkins.

June 2022

Complete edition: the overview together with all 45 linked topic pages

Typeset from `magma.maths.usyd.edu.au/magma/info/overview`

Contents

1 Introduction **1**
Contents by area . . . 1

2 Algebraic Geometry **4**
2.1 Varieties and Schemes . . . 4
2.1.1 Introduction . . . 4
2.1.2 Ambient Spaces . . . 4
2.1.3 Affine Patches . . . 5
2.1.4 Rational Maps . . . 6
2.1.5 Properties of a Scheme . . . 6
2.1.6 Constructions Associated with a Scheme . . . 7
2.1.7 Zero-dimensional Schemes . . . 7
2.1.8 Isolated Singularities . . . 7
2.1.9 Singularities of Schemes in Weighted Projective Space . . . 8
2.2 Coherent Sheaves and Divisors . . . 9
2.2.1 Introduction . . . 9
2.2.2 Invertible Sheaves . . . 9
2.2.3 Cohomology . . . 10
2.2.4 Divisors . . . 11
2.3 Algebraic Curves . . . 12
2.3.1 Introduction . . . 12
2.3.2 The Algebraic Function Field Associated with a Curve . . . 12
2.3.3 Places and Divisors . . . 12
2.3.4 Automorphism Group . . . 13
2.3.5 Plane Curves with Ordinary Singularities . . . 13
2.4 Algebraic Surfaces . . . 14
2.4.1 Introduction . . . 14
2.4.2 Surface Desingularisation using the Hirzebruch–Jung Method . . . 15
2.4.3 Surface Desingularisation using Blow-Up . . . 15
2.4.4 Hypersurface Singularities . . . 16
2.4.5 Surface Invariants . . . 17
2.4.6 Minimal Models . . . 18
2.4.7 Parametrisation of Rational Surfaces . . . 18
2.4.8 Del Pezzo Surfaces . . . 19
2.5 Toric Geometry . . . 20
2.5.1 Introduction . . . 20
2.5.2 Major Functionality . . . 20
2.5.3 Databases . . . 20

3 Associative Algebras **22**

3.1 Associative Algebras Given by Structure Constants . . . 22
3.1.1 Introduction . . . 22
3.1.2 Algebras Given by Structure Constants . . . 22
3.1.3 Construction of Algebras . . . 22
3.1.4 Decomposition . . . 23
3.1.5 Ideals and Orders . . . 23
3.2 Finitely-Presented Algebras . . . 24
3.2.1 Gröbner Bases . . . 24
3.2.2 Ideals . . . 24
3.2.3 Exterior Algebras . . . 25
3.3 Matrix Algebras . . . 25
3.3.1 Introduction . . . 25
3.3.2 Construction of Algebras . . . 25
3.3.3 Structure of the Algebra . . . 26
3.3.4 Presentations and the Wedderburn Decomposition . . . 26
3.4 Quaternion Algebras . . . 27
3.4.1 Introduction . . . 27
3.4.2 Local Properties . . . 27
3.4.3 Orders . . . 27
3.4.4 Enumeration of Ideal Classes . . . 27

4 Combinatorial Theory **29**
4.1 Introduction to Combinatorial Theory . . . 29
4.2 Partitions, Young Tableaux, and Symmetric Functions . . . 30
4.2.1 Partitions . . . 30
4.2.2 Young Tableaux . . . 30
4.2.3 Symmetric Function Algebras . . . 30
4.3 Graph Theory . . . 31
4.3.1 Introduction . . . 31
4.3.2 Constructions for Graphs . . . 31
4.3.3 Cliques and Independent Sets . . . 31
4.3.4 Equivalence and Automorphisms . . . 32
4.3.5 Networks . . . 32
4.3.6 Applications . . . 33
4.4 Design Theory . . . 33
4.4.1 Introduction . . . 33
4.4.2 Construction of Incidence Structures . . . 33
4.4.3 Difference Sets . . . 34
4.4.4 Resolutions and Parallelisms . . . 34
4.4.5 Automorphism Group and Isomorphism . . . 34
4.4.6 Applications . . . 35
4.5 Finite Planes . . . 35
4.5.1 Introduction . . . 35
4.5.2 Construction of Finite Planes . . . 35
4.5.3 Arcs, Conics, Parallelisms, and Unitals . . . 36
4.5.4 The Collineation Group . . . 36
4.5.5 Translation Planes . . . 36
4.5.6 Near-Field Planes . . . 36
4.5.7 Applications . . . 37
4.6 Hadamard Matrices . . . 37
4.6.1 Introduction . . . 37
4.6.2 Equivalence and Automorphism Groups . . . 37

4.6.3 Databases . . . 38
4.6.4 Applications . . . 38

5 Commutative Algebra 39
5.1 Gröbner bases . . . 39
5.1.1 Introduction . . . 39
5.1.2 The Faugere algorithm . . . 40
5.1.3 Gröbner Bases over Finite Fields and Algebraic Attacks . . . 40
5.1.4 Gröbner Bases over Characteristic Zero Fields . . . 41
5.2 Multivariate Polynomial Rings and Their Ideals . . . 41
5.2.1 Polynomial Ideals and Varieties . . . 41
5.2.2 Prime Ideals and Primary Decomposition . . . 42
5.2.3 Homogeneity and Graded Modules . . . 42
5.2.4 Normalisation, Regular Sequences, and Rees Algebras . . . 43
5.3 Invariant Theory . . . 44
5.3.1 Invariants of Finite Groups . . . 44
5.3.2 The Molien Series . . . 44
5.3.3 Invariants of Linear Reductive Groups . . . 44
5.4 Modules over Polynomial Rings . . . 45
5.4.1 Syzygies and Submodules . . . 45
5.4.2 Module Homomorphisms . . . 46
5.4.3 Free Resolutions . . . 46

6 Commutative Rings 48
6.1 Introduction to Commutative Rings . . . 48
6.2 Integer Rings . . . 49
6.2.1 Introduction . . . 49
6.2.2 Primality . . . 49
6.2.3 Factorisation . . . 49
6.2.4 The Ring . . . 50
6.2.5 Solving Conics . . . 50
6.3 Finite Fields . . . 50
6.3.1 Introduction . . . 50
6.3.2 Representation and Basic Arithmetic . . . 50
6.3.3 Compatibility and Subfield Embeddings . . . 51
6.3.4 Discrete Logarithms . . . 52
6.4 Real and Complex Fields . . . 52
6.4.1 Introduction . . . 52
6.4.2 Complex Arithmetic and Special Functions . . . 52
6.4.3 Numerical Linear Algebra . . . 53
6.4.4 Real and Complex Roots of Polynomials . . . 53
6.4.5 Linear Relations and Algebraic Dependencies . . . 53
6.4.6 Differentation and Integration . . . 53
6.5 Polynomial Rings . . . 54
6.5.1 Introduction . . . 54
6.5.2 Multiplication . . . 54
6.5.3 Greatest Common Divisor . . . 55
6.5.4 Factorisation of Univariate Polynomials . . . 55
6.5.5 Factorisation of Multivariate Polynomials . . . 56
6.6 General Commutative Rings in Magma . . . 56
6.6.1 Introduction . . . 56
6.6.2 Exact and Non-exact Rings . . . 57

6.6.3 Standard Operations for Commutative Rings . . . 57
6.6.4 Exact Fields . . . 57
6.6.5 Exact Euclidean Domains . . . 57
6.6.6 Exact Integral Domains . . . 58
6.6.7 Exact Principal Rings having an Euclidean Function . . . 58
6.6.8 Other Exact Commutative Rings . . . 58
6.6.9 Non-exact Fields . . . 58
6.6.10 Non-exact Commutative Rings . . . 59

7 Error-Correcting Codes 60
7.1 Linear Codes over Finite Fields . . . 60
7.1.1 Introduction . . . 60
7.1.2 Constructions for Codes . . . 60
7.1.3 Minimum Weight and Weight Distribution . . . 61
7.1.4 Equivalence and Automorphisms . . . 61
7.1.5 Decoding . . . 62
7.1.6 Low Density Parity Check Codes . . . 62
7.1.7 The McEliece Cryptosystem . . . 62
7.1.8 Use and Applications . . . 63
7.2 Linear Codes over Finite Rings . . . 63
7.2.1 Codes Over Galois Rings . . . 63
7.2.2 Constructions for Codes . . . 63
7.2.3 Minimum Weight and Weight Distribution . . . 64
7.3 Additive Codes . . . 64
7.3.1 Introduction . . . 64
7.3.2 Constructions for Additive Codes . . . 65
7.3.3 Minimum Weight and Weight Distribution . . . 65
7.3.4 Automorphism Group . . . 65
7.4 Quantum Error-correcting Codes . . . 65
7.4.1 Quantum Error-correction . . . 65
7.4.2 Stabiliser Codes and the Error Group . . . 66
7.4.3 Constructions for Quantum Codes . . . 67
7.4.4 Minimum Weight and Weight Distribution . . . 67
7.4.5 Automorphism Group . . . 68
7.4.6 Simulation on Hilbert Spaces . . . 68
7.4.7 Applications . . . 68

8 Group Theory 69
8.1 Finitely Presented Groups . . . 69
8.1.1 Introduction . . . 69
8.1.2 Coset Enumeration: Todd–Coxeter Procedure . . . 69
8.1.3 Low Index Subgroups . . . 70
8.1.4 Presentations for Subgroups . . . 70
8.1.5 Simplification of Presentations . . . 70
8.1.6 Quotients of a Finitely Presented Group . . . 71
8.1.7 Finite or Infinite? . . . 71
8.1.8 Classes of FP-Groups with Solvable Word Problem . . . 72
8.2 Permutation Groups . . . 72
8.2.1 Constructions for Permutation Groups . . . 72
8.2.2 Base and Strong Generating Set . . . 73
8.2.3 Natural Actions of a Permutation Group . . . 73
8.2.4 Conjugacy Classes . . . 74

8.2.5 Normal Structure . . . 74
8.2.6 Socle and Composition Series . . . 74
8.2.7 Backtrack Search Algorithms . . . 74
8.2.8 Subgroups . . . 75
8.2.9 Automorphism Group and Isomorphism . . . 75
8.3 Matrix Groups . . . 75
8.3.1 Introduction . . . 75
8.3.2 Construction of Matrix Groups . . . 76
8.3.3 Natural Actions of a Matrix Group . . . 76
8.3.4 BSGS Model . . . 76
8.3.5 Composition Tree Model . . . 77
8.4 Abelian Groups . . . 77
8.4.1 Finitely-Presented Abelian Groups . . . 77
8.4.2 Generic Abelian Groups . . . 78

9 Modules and Representation Theory 79
9.1 Module Theory . . . 79
9.1.1 Vector Spaces . . . 79
9.1.2 Modules with Scalar Action . . . 80
9.1.3 Modules with Matrix Action . . . 81
9.2 Representation Theory . . . 83
9.2.1 Character Theory . . . 83
9.2.2 Modular Representations . . . 83
9.2.3 Group Representations in Characteristic Zero . . . 84

10 Number Theory 86
10.1 Elementary Number Theory . . . 86
10.1.1 Introduction . . . 86
10.1.2 Factorisation . . . 86
10.1.3 Primality . . . 86
10.1.4 The Ring . . . 87
10.1.5 Solving Conics . . . 87
10.2 Algebraic Number Theory . . . 87
10.2.1 Introduction . . . 87
10.2.2 Basic Arithmetic . . . 87
10.2.3 Local Rings and Fields . . . 87
10.2.4 Subfields . . . 88
10.2.5 Maximal Order . . . 88
10.2.6 Ideal Class Group . . . 88
10.2.7 The Galois Group . . . 88
10.2.8 Class Field Theory . . . 88
10.3 Local Rings and Fields . . . 89
10.3.1 Introduction . . . 89
10.3.2 Implementation . . . 89
10.3.3 Exact -adic Rings and Fields . . . 89
10.3.4 Applications . . . 89
10.4 Lattices . . . 90
10.4.1 Introduction . . . 90
10.4.2 Reduction of Lattices . . . 90
10.4.3 Enumeration of Short Vectors . . . 91
10.4.4 Theta Series as Modular Forms . . . 91
10.4.5 Automorphism Group . . . 91

10.5 Elliptic Curves over and Number Fields . . . 92
10.5.1 Introduction . . . 92
10.5.2 Curves over the Rationals . . . 92
10.5.3 Curves over a Number Field . . . 93
10.5.4 Elliptic Curves of Given Conductor . . . 93
10.5.5 Mordell–Weil group . . . 93
10.5.6 Points . . . 94
10.5.7 Heights of Points . . . 94
10.5.8 Morphisms . . . 94
10.6 Hyperelliptic Curves over and Number Fields . . . 94
10.6.1 Introduction . . . 94
10.6.2 Points . . . 95
10.6.3 Jacobians over Number Fields or . . . 95
10.6.4 Kummer Surfaces . . . 95
10.6.5 Mordell–Weil Group . . . 95
10.6.6 Chabauty's Method . . . 96
10.6.7 Analytic Jacobians . . . 96
10.6.8 Invariants . . . 97
10.6.9 Automorphism Group and Isomorphism Testing . . . 97
10.6.10 Superelliptic Curves . . . 97
10.7 Curves Over Finite Fields . . . 98
10.7.1 Introduction . . . 98
10.7.2 Point Counting on Elliptic Curves . . . 98
10.7.3 Point Counting on Hperelliptic Jacobians . . . 98
10.7.4 Pairings . . . 98
10.7.5 Discrete Logarithms . . . 99
10.8 Artin Representations . . . 99
10.9 L-Functions . . . 99
10.9.1 Introduction . . . 99
10.9.2 Sources of L-Functions . . . 100
10.9.3 Artin Representations . . . 100
10.9.4 Elliptic Curves over and Number Fields . . . 100
10.9.5 Hyperelliptic Curves over and Number Fields . . . 101
10.9.6 Hecke Grossencharacters . . . 101
10.9.7 Hilbert Modular Forms . . . 101
10.9.8 Hypergeometric Motives . . . 101
10.9.9 Jacobi Motives . . . 102
10.9.10 Tensor Products . . . 102
10.9.11 Applications . . . 102
10.10 Modular Forms and Modular Curves . . . 102
10.10.1 Classical Modular Forms . . . 102
10.10.2 Representations from Modular Forms . . . 103
10.10.3 Modular Abelian Varieties . . . 103
10.10.4 Modular Forms over Number Fields . . . 103
10.10.5 Modular Curves . . . 104

Chapter 1

Introduction

We list here some of the more significant areas of mathematics covered by Magma. Each item links to a page giving some general information about the topic as it is viewed in Magma, followed by a short mention of some of the more important functionality that is available for that area. Where practical, some indication of the algorithms used and their performance may be given.

Here are some anticipated uses of this material:

- The general reader can learn much about the state of contemporary computational tools that are currently available in many areas of algebra.
- A person looking to use computational algebra can quickly get general information about what is available in Magma and hence decide if Magma is likely to be the right tool to meet their needs.
- For a person already using Magma, this information can be used as a guide showing what is available in the system, somewhat more easily than working through the approximately 6,500 pages of the Magma Handbook.

A reader desiring more detail beyond what is covered in this Overview should consult the relevant chapters of the Handbook.

The version of the Overview as it stands on its release on June 9, 2022 is an interim version as the construction of this site is still underway. This version covers about 50 areas whereas the completed version will cover approximately 100.

Contents by area

Algebraic Geometry

- Varieties and Schemes
- Coherent Sheaves and Divisors
- Algebraic Curves
- Algebraic Surfaces
- Toric Geometry

Associative Algebras

- Associative Algebras Given by Structure Constants
- Finitely Presented Algebras
- Matrix Algebras
- Quaternion Algebras

Combinatorial Theory

- Introduction to Combinatorial Theory
- Partitions, Young Tableaux, and Symmetric Functions
- Graph Theory
- Design Theory
- Finite Planes
- Hadamard Matrices

Commutative Algebra

- Groebner bases
- Multivariate Polynomial Rings and Their Ideals
- Invariant Theory
- Modules over Polynomial Rings

Commutative Rings

- Introduction to Commutative Rings
- Integer Rings
- Finite Fields
- Real and Complex Fields
- Polynomial Rings
- General Commutative Rings

Error-Correcting Codes

- Linear Codes over Finite Fields
- Linear Codes over Finite Rings
- Additive Codes
- Quantum Error-correcting Codes

Group Theory

- Finitely Presented Groups
- Permutation Groups
- Matrix Groups
- Abelian Groups

Modules and Representation Theory

- Module Theory
- Representation Theory

Number Theory

- Elementary Number Theory
- Algebraic Number Theory
- Local Rings and Fields
- Lattices
- Elliptic Curves over Q and Number Fields
- Hyperelliptic Curves over Q and Number Fields
- Curves over Finite Fields
- Artin Representations

- L-Functions
- Modular Forms

Chapter 2

Algebraic Geometry

2.1 Varieties and Schemes

2.1.1 Introduction

Magma has a wide range of functionality for performing hard arithmetic and geometric computations with algebraic varieties or more general schemes of finite-type over a base ring. A scheme is defined by a finite set of multivariate polynomials on an "ambient" space or the corresponding polynomial ideal generated by these. It can be thought of as the space of simultaneous solutions of this system of polynomials; however, schemes may be non-reduced. The reader should recall that the scheme defined by an ideal is not considered the same as the scheme defined by the radical of that ideal. Amongst other cases, this is important for questions involving multiplicity of intersections or map fibres.

A "subscheme" Y of a scheme X is one in the same ambient space whose defining ideal $I(Y)$ contains the defining ideal $I(X)$ of X, up to a redundant ideal factor. In the literature this is often referred to as a "closed" subscheme, but we use the more concise terminology here. A "variety" is a scheme that is reduced and irreducible whose base ring is a field. Thus, a variety cannot be decomposed into a finite union of subschemes none of which contain the other. The dimension of a scheme X can be defined as being one more than the length of a maximal totally ordered (by inclusion) chain of subvarieties of X. It is equal to the maximum of the dimensions of the irreducible components of X. The dimension is the most important property of a scheme. Curves are one-dimensional schemes and surfaces are two-dimensional.

2.1.2 Ambient Spaces

The ambient spaces A are generally defined by a multivariate polynomial ring R over the base ring k with a single or multiple $\mathbb{Z}$-grading of the n variables. If there is no grading, this is taken to mean n-dimensional affine space k^n. The ring R, referred to as the coordinate ring of A, gives the allowable global algebraic functions on A in the scheme category; that is, polynomials in the variables. Quotients of polynomials give "rational" functions on A. With a single grading where all variables have weight 1, A is an ordinary projective space over k, $\mathbb{P}^{n-1}(k)$.

The defining data for a scheme X lying in ambient A are polynomials or ideals of R that are jointly homogeneous with respect to each grading (there is no extra condition in affine spaces). The points of X over a field (or ring) L are the points of A on which the defining polynomials all vanish.

As well as the importance of schemes/varieties for a large number of algebraic and arithmetic problems in pure mathematics and its applications, many of the real or complex manifolds arising

in analysis are spaces of real or complex points of varieties defined over a number-field. In the case of compact complex manifolds coming from a projective scheme, rational functions and maps on the manifolds correspond precisely to the corresponding algebraic objects on the schemes by the GAGA principle. Other objects like vector bundles, coherent sheaves and their cohomology groups, and global divisors also correspond in a one-to-one fashion to those on the algebraic side. Thus, problems like the computation of cohomological invariants, intersection numbers, global embeddings into projective space, or isomorphisms can be tackled with algebraic geometry packages such as Magma's.

Magma schemes are all globally represented and the ones that can be defined are quasi-projective as abstract schemes — mainly affine or projective. However, as described in the introduction, there is a wide range of projective space ambients in which they may lie and any closed subscheme of one of these ambients is representable. Most of the deeper functionality is only available for schemes whose base ring k is an (exact) field. The ring k may be an arbitrary Magma ring, however, and it can be useful to define schemes over the integer ring $\mathbb{Z}$, for example, and maps over $\mathbb{Z}$ in order to conveniently specialise to multiple prime fields by reduction mod p. Gröbner basis computations over $\mathbb{Z}$ can be convenient to determine a finite set of primes outside of which there is good reduction. The geometry machinery is designed for exact algebraic computation and very little beyond the basic constructors will work for inexact fields like $\mathbb{R}$ and $\mathbb{C}$.

As an example, consider the homogeneous polynomial

$$f = x^4 - y^4 + z^4 + xz(x^2 + z^2) + 3xy^2z - 2x^2z^2$$

defining a projective plane curve C over $\mathbb{Q}$. This is a canonical model for the genus 3 modular curve $X_0(34)$. We would like to find all of the primes at which f reduces to a singular equation. Using the Jacobi criterion, we take the ideal in $\mathbb{Z}[x, y, z]$ generated by f and its three partial derivatives and compute a Gröbner basis with respect to the *grevlex* ordering. We find the polynomials $136z^7$, $4y^3 - 6xyz$, and a monic polynomial in x: $x^3 + ..$, where the remaining terms are all divisible by y or z. This means that for $p \neq 2$ or 17 (the prime divisors of $4 * 136$) a singular point on C mod p would have to have homogeneous coordinates all 0, which is not a valid projective point. Thus, f reduces to the equation of a non-singular projective curve mod p for all p not equal to 2 or 17. This is as good as we could hope for since $X_0(34)$ has bad reduction modulo those two primes, so "no" model for it with integer coefficients can reduce to a non-singular curve.

In projective ambients, a one-to-one correspondence between ideals and schemes does not exist. Two ideals define the same scheme if and only if their primary components differ by a redundant ideal that depends on the gradings of the ambient. There is a maximal (saturated) ideal that defines a given scheme and it can be necessary to compute this to make correct computations of dimension, images of maps, tests for being reduced or irreducible, and a number of other properties.

2.1.3 Affine Patches

Projective schemes are covered by standard "affine patches" which are open subsets for the Zariski toplogy. These are needed internally for certain computations where it is necessary to reduce to commutative algebra operations in an affine setting and they can be useful to the user for the same reasons. It should be noted, however, that most of the algorithms work with homogeneous coordinate rings and ideals, which is generally more efficient.

Currently in Magma not all of the standard affine patches are constructible in some weighted projective spaces. However, all of the patches are available in the commonest cases of ordinary and product projective spaces. One place where affine patches are essential is in working with

individual points in pointsets on schemes. A point is definable only if it lies in one of the constructible affine patches although it is represented by its homogeneous coordinates in the projective space. In bad cases, it is always possible to consider the point as a zero-dimensional subscheme of the base scheme X and work with that.

There are functions to return the affine patches of a projective scheme and to restrict various structures like maps to patches. There is also a projective closure map to compute the projective closure of an affine scheme. If the affine scheme is the patch of an existing projective one, that is returned (the information linking patches and projective schemes is stored internally). Otherwise, the usual projective closure in ordinary projective space is returned.

2.1.4 Rational Maps

There is a lot of functionality for working with rational maps between schemes. Two very useful functions for rational maps are the following: An extension function is provided which attempts to find alternative equations for the map so as to define it on a larger open subset of the domain. If the codomain is affine, ordinary projective, or product projective, then this will extend the map to its maximal domain of definition. There is also a function to test whether a map is invertible (as a rational map) and compute and add inverse defining equations to explicitly present it as a birational map.

Magma can compute the image scheme $f(Z)$ for a subscheme Z of X and a rational map $f : X \to Y$. It can also compute inverse images (pullbacks) of subschemes of Y. The same is true for points on X or Y. The image computations will return an error if Z or the point lies in the complement of the open set on which f is defined by its current set of defining equations or alternative defining equations. This is a situation in which the extension function is often very useful because more complicated rational maps are usually not maximally defined by a single set of mapping equations.

A variant on standard maps are *graph maps*, which can only be defined for maps between ordinary projective schemes. Here the map is defined by the scheme-theoretic closure of its graph in $X \times Y$. The advantage of graph maps is that they are automatically defined on their maximal domain of definition and testing for invertibility as well as computing the inverse (which will also be maximally defined) is much quicker and more efficient.

Of the particular rational maps for which special construction functions are available in Magma, *blow-ups* should be mentioned. A function is provided to compute the (global) blow-up scheme Z of scheme X by an arbitrary subscheme Y of X together with the blow-up map ϕ of Z to X. The map ϕ gives an isomorphism of the Zariski-open subsets $Z/\phi^{-1}(Y)$ and X/Y and $\phi^{-1}(Y)$ is of codimension 1 in Z whatever the codimension of Y in X. If Y is a non-singular subvariety of a non-singular variety X, the points of Z above a point P in Y correspond to the tangent directions in X that are normal to Y, which explains the name 'blow-up'. Blow-ups of points (as reduced subschemes) and their inverses, *blow-downs*, are extremely important for surfaces in particular. Any birational map between two non-singular projective surfaces can be factored into a sequence of blow-ups and blow-downs.

2.1.5 Properties of a Scheme

Properties and objects that can be computed for a scheme X are listed below:

- Algebraic dimension. For projective schemes: arithmetic genus, the Hilbert polynomial/function, and degree. More complex cohomological invariants are described in the section on Coherent Sheaves and Divisors.
- Determine if a scheme is irreducible and whether or not it is reduced. The irreducible components in reduced or non-reduced form.

- Singular points. Determine the singularity type of the scheme at a particular point: non-singular, Gorenstein, Cohen–Macaulay, normal (the last only for surfaces).
- Determine whether an ordinary projective scheme is arithmetically Gorenstein or Cohen–Macaulay.
- Tangent spaces, tangent cones, and multiplicities of a given point.
- Tangent and secant varieties of a variety X in affine or ordinary projective space.
- Classification and computation of normal forms of isolated singular points. See the subsection on *Isolated Singularities* below.
- Unions and intersections of schemes. Test equality and inclusion of pairs of schemes.
- The linear subsystems of the full linear system of degree d hypersurfaces on an ordinary projective space that have specified basepoints.
- Over a number field K, test X for local solubility over the completion of K at a prime ideal. Find (approximately) all points over K up to a specified naive height.
- For ordinary projective schemes over a finite field, find all points over the base field.

2.1.6 Constructions Associated with a Scheme

The following constructions are available:

- Basic functions are provided to construct scheme-theoretic unions and intersections of a finite sets of schemes as well as ones which test whether two schemes in the same ambient space are equal or if one is a subscheme of the other.
- The linear subsystems of the full linear system of degree d hypersurfaces on an ordinary projective space that have specified basepoints may be constructed.
- Over a number field K there are functions to test for local solubility of schemes over the completion of K at a prime ideal (i.e., whether the scheme has a point defined over the completed p-adic field) and to lift solutions at given p-adic precision to higher precision. There is also a function to search for approximately all points over K up to a specified naive height on an affine or ordinary projective scheme.
- For ordinary projective schemes over a finite field, the set of all points over the base field may be enumerated using a fibration technique.

More sophisticated functionality for schemes of arbitrary dimension is available in the sheaves and divisors package.

2.1.7 Zero-dimensional Schemes

Zero-dimensional schemes are discrete schemes that only contain a finite number of points over any extension of the base field. These are the solutions of algebraic systems of equations with only finitely many solutions. Some special functionality is provided for these schemes. The total number of solutions with multiplicity may be computed with the degree function (which does not require the scheme to be ordinary projective) and the total number of distinct solutions is given by the degree of the reduced subscheme. There are functions that list all points (solutions) over a given extension of the base field or list all of them over an algebraic closure of the base field.

2.1.8 Isolated Singularities

Code is being developed in Magma to classify isolated hypersurface singular points according to the classification scheme of V. I. Arnold and to compute their normal form. The analytic transformation to the normal form can also be computed to arbitrary precision.

To extend the range of applicability of Magma's singularity analysis, a function is provided that transforms a singular point P on a scheme X to an analytically equivalent hypersurface singularity if this is possible. When it is, the singularity is referred to as an *analytic hypersurface*

singularity or AHS for short. When X is reduced and all irreducible components of X through P have the same dimension d, the necessary and sufficient condition for P to be singular and an AHS is that the tangent space of X at P has dimension $d+1$.

Let P on X be an isolated singular point (that is, P does not lie on a positive dimensional component of the singular subscheme of X) and assume that X is a hypersurface or that P is an AHS on X. The classification function determines whether P is a singularity that lies in one of the families of Arnold's classification. If so, the function returns the name of the family and also computes Arnold's polynomial normal form of the singularity. Optionally, the user may also ask for an analytic automorphism $R \to R$, computed to desired precision, that maps the analytic hypersurface equation F to the normal form.

An example of a singularity that is not on a hypersurface but is analytically isomorphic to a hyperplane singularity is the origin O of the surface S in 4-space defined by the two polynomials $x^2 + yt + t^2$ and $z^3 + xt + y^2 - t$. The above functions first determine that O is an AHS and then that it is an E_7 singularity with normal form (in three variables) $x^2 + y^3 + yz^3$. In this case, projection onto the 3-dimensional tangent space at O does not give an isomorphism to a hypersurface singularity (this global projection technique works in some examples). Instead, it produces a non-isomorphic singularity that is no longer isolated.

2.1.9 Singularities of Schemes in Weighted Projective Space

Full testing for singularity/non-singularity of schemes lying in ambients other than affine, ordinary projective, or product projective space has been a major omission of the geometry package since the introduction of schemes into Magma. The basic singularity test formally applies the Jacobian criterion to the affine or projective coordinate ring of scheme X. In the projective cases given above, the coordinate rings are those of faithfully-flat affine cones over X for which the fibres are non-singular. By standard results on faithfully-flat scheme coverings, applying the Jacobian criterion to the projective coordinate ring gives correct results. However, for projective ambients with different weightings, the result will generally be incorrect. Essentially, the criterion does not apply where X meets the singular locus of the ambient, where it will usually give the conclusion of singularity, even when X is non-singular there.

The most useful ambients apart from those mentioned above are the weighted projective spaces (WPSs) with positive integral weights. The singularities of the ambient occur in the affine patches that are undefined internally in Magma; that is, the patches corresponding to the localisation of a variable whose weight does not divide all other weights (i.e., of weight > 1 in the usual case where the set of weights has been normalised to have no common divisor). These affine patches exist but are not isomorphic to affine space with one fewer variable than the WPS.

A facility for constructing "user-level" affine patches for WPS at indices of variable weight greater than 1. While not yet integrated into the low-level scheme structures, they can be obtained by a function call which also returns a projective closure map to give the birational restriction from the original ambient to the patch and the inclusion map of the patch into the original ambient. Another difference is that the inclusion map is also a birational map that is not everywhere-defined. Mathematically, it is everywhere defined of course, but it cannot be everywhere defined using ordinary Magma scheme maps.

The affine patches are computed in a natural way using Magma's invariant theory functionality applied to a finite cyclic group action on a full polynomial ring. This gives the patch as an affine variety with relations. For weights of even a medium size, the number of variables of the patch is often much larger than the original number of projective variables, but it is usually still much smaller than the number of variables need to re-embed the WPS into ordinary projective space by d-uple embedding, say.

Routines have also been implemented to embed product projective schemes into ordinary projective space using *Segre embeddings*. This increases ambient dimension but can still be a useful tool in order to apply sheaf or divisor functionality that is only available at present for ordinary projective schemes. The functions work for an arbitrary number of components of the product projective space and also return isomorphisms from the original to the re-embedded scheme that have alternative inverse defining equations giving the inverse everywhere.

2.2 Coherent Sheaves and Divisors

2.2.1 Introduction

Magma has functionality for defining and working with coherent sheaves on ordinary projective schemes. On normal ordinary projective varieties there is also a package for divisors and divisor classes which is linked to the invertible sheaf machinery. The functions allow the computation of cohomology groups, divisor maps, and Riemann–Roch spaces, amongst other things.

Coherent sheaves on a scheme X correspond to finitely-generated graded modules over the (graded) coordinate ring R of the ambient of X which are annihilated by the defining ideal of X. If X is ordinary projective then the module corresponding to a sheaf is not quite unique. With some minor restrictions on the sheaf, there is a maximal, separated, finitely-generated module that represents it. Separated here means that the module has no m-torsion, where m is the maximal homogeneous ideal of R. This maximal module is extremely important. Any module representing the sheaf differs from the maximal module in only a finite number of the graded summands but it is usually the smaller graded parts that are incomplete and these are the most important. For example, the 0th graded summand of the maximal module is the finite-dimensional k-vector space that gives the global sections of the sheaf.

As well as having a suite of functions to perform the standard Gröbner-based commutative algebra operations on (graded) modules over polynomial rings, a good method for computing the maximal module from a given module representation is a key ingredient in general computational work with sheaves. Magma has an efficient algorithm for this computation based on a finite projection of the exact support of the sheaf module to a linear subspace, computed via Noether normalisation combined with a double dual computation on the module considered as a module over the coordinate ring of this subspace. This avoids computing complicated dualising modules and requires only very weak assumptions on the support of the sheaf; namely, that it is equidimensional of positive dimension.

There are functions to construct general sheaves by giving a representing module as well as functions for special important sheaves like the basic structure sheaf as well as the tangent sheaf and the canonical sheaf. There is also a function to give the Serre twists of sheaves which are represented by the same modules but with a shift in the grading. Functions are also provided for the construction of sheaves from other sheaves using operations such as direct sums, duals, and tensor products.

2.2.2 Invertible Sheaves

Some of the most important types of sheaf are the locally-free sheaves and there is a test for local-freeness which also computes the rank. The basic version of this test is a standard algorithm based on Fitting ideals but a more complex algorithm can be optionally used which is much faster for sheaves on low-dimensional varieties in high codimension spaces. The latter test applied to the tangent sheaf or sheaf of differentials gives a much faster non-singularity (actually, smoothness) test than the default Magma algorithm in these high-codimensional cases.

Particularly important are rank 1 locally-free (*invertible*) sheaves. Isomorphism classes of these

correspond to rational equivalence classes of Cartier divisors and there is a correspondence between these and rational maps, up to a linear change of variables, of the base scheme X into ordinary projective spaces. Embeddings into projective space come from what are known as *very ample* invertible sheaves/divisor classes and ample sheaves give embeddings into weighted projective spaces with a smaller number of variables. The map coming from a Cartier divisor class or equivalent invertible sheaf is referred to as the *divisor map*.

There are functions to compute the divisor maps of invertible sheaves as scheme graph maps or ordinary scheme maps. For example, along with the canonical and anticanonical maps which are the divisor maps associated with the canonical sheaf and its dual, the important *adjunction* mapping can easily be computed. In the case of surfaces, this is just the divisor map of the first Serre twist of the canonical sheaf.

There is a function to compute the invertible sheaf corresponding to an effective Cartier divisor D, given as a subscheme of X, and to also compute a basis of the *Riemann–Roch* space of D along with the divisor map. The Riemann–Roch space is the finite-dimensional space of rational functions on X whose divisor of poles is $\leq D$. Although this space is closely related to the divisor map, the two are computed in a slightly different manner here.

There are times when it is desirable to anti-project a variety from a lower-dimensional projective space to a higher-dimensional one. This may occur when the higher-dimensional embedding has a relatively simple natural form and the lower-dimensional one is a projection down that has fewer variables but unwieldy equations and bad expressions for automorphisms. A case in which this arose naturally is in 8-descent on elliptic curves where a model in projective 3-space for a homogeneous space E is constructed which is a projection from 7-space of a projective normal embedding. To recover the full embedding with sheaves is simply a matter of computing the divisor map of the structure sheaf and taking its image! Computing the maximal module for the structure sheaf is equivalent to constructing the coordinate ring of E in its 7-space embedding as a module and taking the image under the divisor map essentially recovers this coordinate ring as an algebra.

There is some functionality for working with specific global sections of a sheaf or its twists, represented by homogeneous elements of the original or maximal module. In particular, there is a function to compute the subscheme of zeroes of a global section of a locally-free sheaf. Applied to an invertible sheaf this gives a corresponding effective Cartier divisor. Two sheaves may be tested for isomorphism and an explicit isomorphism is returned if they are found to be isomorphic. One use of this is to test whether two Cartier divisors are rationally equivalent. This will be the case precisely when their corresponding invertible sheaves are isomorphic.

2.2.3 Cohomology

Most of the important discrete invariants of projective algebraic varieties are expressible as the k-dimension of a sheaf cohomology group: the geometric genus and higher plurigenera, the irregularity of a surface, and the Hodge diamond numbers. These cohomology dimensions also occur in deformation theory. For the tangent sheaf, the H^0 dimension gives the dimension of the connected component of the automorphism group, the H^1 dimension gives the dimension of the tangent space of the local deformation ring, and if the H^2 cohomology has dimension zero then there are no obstructions to deformation and the local deformation ring is smooth. There is a function to compute dimension of the cohomology group $H^n(X, S)$ for any sheaf S and $n \geq 0$. The algorithm used is the one based on the Bernstein–Gelfand–Gelfand correspondence. As an alternative, the H^0 dimension can also be computed more directly with a function that gives the dimension of global sections of a sheaf.

2.2.4 Divisors

Divisors are finite $\mathbb{Z}$-linear combinations of reduced, irreducible subschemes of codimension one of X, $\sum_1^n n_i D_i$. The D_i are referred to as prime divisors. If the n_i are all ≥ 0 then the divisor is said to be *effective*. On a normal variety, effective divisors are in one-to-one correspondence with certain subschemes (technically, ones whose primary components are all of codimension one) and we sometimes identify the divisor with the corresponding subscheme. Prime divisors correspond to subschemes whose defining ideal is prime. A non-zero rational function f determines a divisor $div(f)$, called a *principal* divisor, equal to the effective divisor of its zeros minus the effective divisor of its poles.

Abstractly, the multiplicity n_i to which any prime divisor D_i occurs in the decomposition of $div(f)$ is equal to the valuation of f (considered as an element of the field of all rational functions) with respect to the local ring of the scheme-theoretic generic point of D_i, which is a discrete valuation ring.

Two divisors are *rationally equivalent* if their difference is a principal divisor. A divisor is called *Cartier* if it is locally a principal divisor in the neighbourhood of any point of X. On a non-singular variety all divisors are Cartier, but divisors may fail to be Cartier in the neighbourhood of singular points. There is a one-to-one correspondence between rational equivalence classes of Cartier divisors and invertible sheaves (see below) and this allows us to use sheaf machinery for the more complex divisor computations.

Divisors are represented as a $\mathbb{Z}$-linear combination of effective divisors given by their defining ideals. These effective divisors do not have to be reduced or irreducible. There are functions to refine the representing decomposition into the full decomposition of a $\mathbb{Z}$-linear combination of prime divisors or just into a relatively prime decomposition where any two of the effective divisors occurring have no common prime divisor factor. Some operations for divisors are summarised below.

- Basic arithmetic operations such as addition, subtraction, and scalar multiplication can be performed on divisors and there is a function for testing whether a divisor is Cartier.
- There are functions to compute the multiplicity of a divisor with respect to a given prime divisor without computing the full prime decomposition.
- There are functions to compute whether a divisor D is principal (and also return a rational function f such that $D = div(f)$ (this uses the Riemann–Roch space)), whether a divisor is equivalent to an effective divisor (with such an effective divisor also returned), or whether two divisors are rationally equivalent.
- For divisors or invertible sheaves on surfaces, intersection numbers can be computed and there are tests for properties such as the Nefness of effective divisors.
- There is a function that computes the associated invertible sheaf for a Cartier divisor D which generalises the earlier function for effective divisors.
- The Riemann–Roch space of a divisor may be computed. This uses the same algorithm as for sheaves in the effective case and a simple variant in the general case.

An example of divisor use is to blow down disjoint lines on a Del Pezzo surface. If the divisor corresponding to the sum of the lines and a hyperplane section is computed then the associated divisor map blows down the lines and gives the anticanonical embedding of the resulting higher-degree Del Pezzo surface. With sheaves, this can be performed by taking the divisor map of the first Serre twist of the invertible sheaf corresponding to the sum of the lines.

2.3 Algebraic Curves

2.3.1 Introduction

An algebraic curve C in Magma is defined as a one-dimensional scheme. Practically all of the non-trivial curve functionality applies only to curves defined over a field and so we will assume that k is a field in this section. The easiest examples of curves are plane curves given by a single defining equation in two-dimensional affine or ordinary projective space. These two dimensional ambients are referred to as *planes*. The Fermat curves are a family of plane curves being defined by the polynomials $x^n + y^n - z^n$, $n \geq 2$, in the projective plane, with x, y, z as the coordinate variables.

There are a number of specialised types of curves in Magma for which there is specific extra functionality. The most general subtype is that of plane curves. There are also subtypes for conics in the ordinary projective plane (curves defined by irreducible homogeneous polynomials of degree 2 in 3 variables), elliptic curves, and hyperelliptic curves. Elliptic curves and hyperelliptic curves in Magma are defined by a non-singular two variable Weierstrass equation.

2.3.2 The Algebraic Function Field Associated with a Curve

Most of the specific machinery that is available only for curves (or comes in a different flavour to the corresponding functionality for general varieties) makes use of Magma's *algebraic function field* machinery. These are the intrinsics concerned with divisors, places, differentials, and related objects. This functionality only applies to curves C that are varieties (reduced and irreducible — integral for short) and where function fields arise, this assumption about C holds. It is possible to use the sheaf machinery in some cases to work with reducible curves. However, the function field methods for curves have a big advantage in that they do not assume the non-singularity of C. Any integral curve is birationally equivalent over k to a non-singular projective curve that is unique up to isomorphism. This curve will be referred to as the *projective normalisation* of C and denoted by $\tilde{C}$.

Let $k(C)$ denote the function field of C, the field of rational functions (defined over k) on C. The projective normalisation $\tilde{C}$ is completely determined by $k(C)$. The function field $k(C)$ is of transcendence degree 1 over the base field k. Such fields have many properties in common with number fields (finite extensions of $\mathbb{Q}$). In particular, they have *places* which are the analogues of places in a number field, and which in turn correspond to prime ideals in the ring of integers. Magma is able to compute the integral closures in $K = k(C)$ given K as a finite field extension of a $k(t)$ using an algorithm that is a variant of the round 4 algorithm used to construct the ring of integers of a number field.

An algorithm is provided for constructing an abstract algebraic function field K isomorphic to $k(C)$ along with explicit translation maps back and forth between the elements of K and rational functions on C given as quotients of polynomials of the ambient coordinate ring. Thus K can be used to make explicit computations of Riemann–Roch spaces, divisor maps, etc., with the results being converted back to the language of points, subschemes, rational functions, and scheme maps.

2.3.3 Places and Divisors

In this subsection, the term divisor will be taken to refer to a divisor on $\tilde{C}$ which is represented internally by a divisor object on the function field K. The divisor can be decomposed into a $\mathbb{Z}$-linear combination of places in the usual way. The list below indicates some some of the vast range of functionality available for curves, ignoring functions that apply to general schemes or varieties.

- Function field of a curve together with the translation map between elements of the function field and rational functions is also returned.
- Geometric genus of $\tilde{C}$.
- Divisor of $\tilde{C}$ corresponding to the pullback of a finite subscheme of C.
- Canonical divisor.
- All places lying over a point P in pointset $C(L)$ where L is a finite field extension of k.
- Decomposition of a divisor into a sum of multiples of places.
- For a rational function on C, find its associated principal divisor or its divisor of zeros or poles.
- Riemann–Roch space and Riemann–Roch bases as elements of the function field for a divisor.
- Meromorphic differentials on a curve and a basis for the holomorphic differentials and the divisor of a differential.
- Weierstrass places of $\tilde{C}$. Weierstrass gap numbers of a given place.
- Basis of the space of all linear relations over the base field k between a given sequence of function field elements. (These functions are very efficient and are much faster than Gröbner-based methods for computing polynomial relations of given degree.)

There are special functions available for non-constant maps between curves that make use of the function field K. These allow the computation of the degree of the map, the ramification divisor, and pullbacks and pushforwards of functions, divisors, and differentials. For a map f from a curve C to any affine or ordinary projective scheme and a point P in $C(L)$, the image $f(P)$ is computed using place computations if P does not lie in the current domain of definition of f. Effectively, the function field is used to compute images of f on its maximal domain of definition without having to use the expensive map extension function.

2.3.4 Automorphism Group

There is a package of functionality for automorphism groups of curves of genus ≥ 2. The full group of (birational) automorphisms over the base field can be computed, again using a function field algorithm based on Weierstrass points and unique affine models for the full curve minus a point. Note that the birational automorphisms of C are in one-to-one correspondence with the actual automorphisms (birational maps which are everywhere defined with everywhere defined inverses) of $\tilde{C}$. Subgroups generated by a given set of automorphisms can be computed and also returned as a two-way isomorphic map of the automorphism group to a finite permutation group. This can be used for group-theoretic computations of normalisers, centralisers, or Sylow subgroups, for example, and is used internally for fast multiplication and powering of automorphisms. There is also a function to return the matrix representation of an automorphism group acting on the space of holomorphic differentials via pullback.

Complementing the automorphism group code, there is a function to compute the quotient of a curve by a finite group of automorphisms. This also returns a quotient map from C to the quotient curve.

Functions are provided to parametrise a rational (genus 0) curve and to test whether a curve of genus ≥ 2 is hyperelliptic and, if so, to return a Weierstrass model for it along with an isomorphic scheme map to the Weierstrass model. The extensive machinery available to compute arithmetic properties of elliptic and hyperelliptic curves is summarised in the respective sections on elliptic curves and hyperelliptic curves.

2.3.5 Plane Curves with Ordinary Singularities

As a partial alternative to function field methods, there is some specialised functionality for plane curves with only ordinary singularities. Along with tests for ordinariness, there are functions for quickly computing the canonical map and other *adjoint* maps. There are also alternative methods for parametrising ordinary plane rational curves and rational normal curves in higher dimensional

projective ambients. These methods often lead to a parametrisation map with smaller-degree defining equations than the function field method. There are also functions to produce a random plane curve of specified degree and with a specified number of ordinary singularities of specified multiplicities.

The ordinary singularity code relies on the fact that ordinary singularities are resolved by a single blow-up of the plane at the point. This means that the conditions that the singularities place on polynomials to belong to the canonical or adjoint linear systems can be directly computed from the singular subscheme of C rather than having to do local computations at each singular point. There is also a module of functions to resolve general singularities of plane curves via repeated blow-ups of the ambient plane. This involves local work at each singular point and is computationally heavier than the ordinary case code, but it gives information about the resolution graph over each singular point and local blow-up equations that together allow the computation of the canonical or adjoint systems for general plane curves.

2.4 Algebraic Surfaces

2.4.1 Introduction

With the projected specialised functionality for surfaces in mind, in contrast to curves, the dimension-2 schemes covered by the Magma surface subtype are assumed to be a variety (reduced and irreducible) defined over a field.

In the classification of algebraic curves, the geometric genus of the non-singular projective model is the unique discrete invariant. For surfaces, there are a number of independent discrete invariants besides the geometric genus and Magma has functionality to compute these which will be described below. If X is a non-singular projective surface then K_X denotes the canonical sheaf or a canonical divisor. The notation nK_X, $n \geq 1$, denotes the *n-canonical* sheaf or divisor and the associated linear system is also termed n-canonical. The adjective *pluricanonical* is used to refer generally to one of these sheaves, divisors, or linear systems. The nth *plurigenus*, P_n, is the dimension of the space of global sections of the n-canonical sheaf. The first plurigenus is just the geometric genus. The higher plurigenera are also important surface invariants that cannot be automatically deduced from the genus as in the curve case.

An important classifier of a projective non-singular surface is its *Kodaira–Enriques type*. This refines the most basic classification by *Kodaira dimension* d, where $-1 \leq d \leq 2$ can be thought of as the maximum dimension of the image of the surface under birational divisor maps coming from positive multiples of the canonical divisor/sheaf (multi-canonical maps). Kodaira dimension d is -1 if and only there are no effective multi-canonical divisors of any degree. The notation -1 will be used in this section rather than the conventional -∞.

Another big difference between curves and surfaces is non-existence of a unique (up to isomorphism) non-singular projective surface in a birational equivalence class of surfaces. However, apart from surfaces of Kodaira dimension -1, there is a unique *minimal model* up to isomorphism in the equivalence class. The minimal models are characterised by having no *exceptional curves* (or *(-1)-curves*), which are non-singular rational curves on the surface with self-intersection number -1.

If Y is a minimal model of Kodaira dimension ≥ 0 then any birational map from a non-singular projective X to Y is a morphism (everywhere defined) and is purely a sequence of blow-downs. Thus, Y can be thought of as the unique surface at the base of an infinite blow-up graph representing the surfaces in a birational equivalence class. The minimal model is therefore an important surface that provides a canonical birational representative of a given class. Minimal models for surfaces can be computed with some restrictions on the surface presentation as

described below.

2.4.2 Surface Desingularisation using the Hirzebruch–Jung Method

As with curves, the ideal situation is to be able to get information about and perform operations on the non-singular projective surfaces that are birational desingularisations of an input surface X. This is a more difficult problem for surfaces. Function field methods based on normalising Dedekind domains are not available for varieties of dimension greater than 1. However, Magma does have code for computing the *formal desingularisation* of a surface in $Proj^3$ using the Hirzebruch–Jung resolution procedure. The information returned by the algorithm does not give the full resolution graph of curves over a singular point or singular curve. It gives the reverse inclusion map between the formal power series rings which are the completions of the local rings at the generic points of a curve lying over the singular locus and its image in X, a point or curve.

In a similar way, a function is provided to compute the more general *adjoint* linear systems, which correspond to the sheaves $mK_X + nH$ where H is a hyperplane divisor on X or its pullback to the desingularisation. This allows the computation of the plurigenera as well as important canonical, anticanonical, pluricanonical, or adjoint maps on the desingularisation as explicit birational maps on X and their images such as pluricanonical models, etc.

This provides enough information to explicitly compute the m-canonical linear systems of the desingularisation as a sublinear system of the full linear system of homogeneous forms of degree $m(d-4)$ on X where d is the degree of X. In a similar way, there is a function to compute the more general *adjoint* linear systems, which correspond to the sheaves $mK_X + nH$ where H is a hyperplane divisor on X (or its pullback to the desingularisation), as sublinear systems of the degree $n + m(d-4)$ forms. This allows the computation of the plurigenera as well as important canonical, anticanonical, pluricanonical, or adjoint maps on the desingularisation as explicit birational maps on X and their images (pluricanonical models, etc.). The computed (formal) desingularisation is a non-uniquely defined member of a birational equivalence class which is often not minimal. However, all of the things listed above are birational invariants that are independent of whichever desingularisation is constructed.

The formal desingularisation package can be used for projective surfaces in higher-dimensional ambients by first projecting birationally down to $Proj^3$. There are some problems with this approach, however. An alternative approach taken in Magma is to provide functions that can efficiently compute properties of interest for surfaces lying in higher-dimensional ambients with only singularities of restricted type. A suite of general functions based on the sheaf machinery is provided; these functions work on (mainly ordinary projective) surfaces that only have *simple* singularities.

2.4.3 Surface Desingularisation using Blow-Up

An algorithm for desingularising surfaces by local blow-up was developed and implemented by M. Harrison (CAG). This complements the existing machinery for formally desingularising projective hypersurfaces (surfaces in $Proj^3$) using the Hirzebruch–Jung method. The issue with the older method is that it is much too slow when applied to many practical examples, often hanging.

The local blow-up procedure, applied to a scheme-theoretic singular point p/k (p corresponding to a Galois-conjugate set of singular points on S) returns:

1. A set of affine surfaces with explicit birational maps down to an affine patch of S containing p;
2. A finite set of k-irreducible (affine) curves in each of the above surfaces which are affine patches of complete IEDs over p of the blow-up resolution;
3. Data listing which of the curves in 2) over different patches form part of the same complete IED and (for convenience) which of these curves meet the singular locus of the affine surface

they lie in.

The blow-up desingularisation data includes the same functionality as formal desingularisation and also extra functionality which is impossible with the formal version. This comes in two flavours: functions which relate to the canonical divisor class on the desingularisation and others which are unrelated. The former have only been implemented for surfaces in $Proj^3$ because it is relatively easy to handle canonical sections in the hypersurface case. A natural basic pseudo-differential is available on which the computation of sections of pluri-canonical, pluri-anticanonical and, more generally, multi-adjoint sheaves on the desingularisation can be based.

The non-canonical functionality includes the computation of the intersection matrix of the IEDs above each singular point, the multiplicities w.r.t. IEDs of the pullbacks of divisors on S to the desingularisation X and the intersections with IEDs of the strict transforms of S-divisors to X (the pullbacks with all IED components removed). A function is also available to compute the restriction of a linear system on S by imposing vanishing (with multiplicity) conditions at the IEDs. This is extremely important for computing Riemann–Roch spaces and divisor maps of the strict transforms of divisors on S. For example, it is used to compute multi-adjoint maps and can also be used to compute the blow-down of exceptional lines on degenerate Del Pezzo surfaces.

One of the examples that was analysed rigorously using the new machinery was a degree 9 hypersurface S over $\mathbb{Q}$ that was sent in by a Magma user. S has only point singularities but these are quite nasty. The formal desingularisation method hung at an early stage when applied to S. In contrast, the blow-up method is able to resolve all of the singularities in about 2 seconds. The singularities of S consist of 2 conjugate A_3 singularities, 2 pairs of conjugate A_1 singularities, and two non-simple singularities s_1 and s_2 defined over $\mathbb{Q}$. The minimal resolution of s_1 has 11 rational IEDs above it, 10 (-2)-curves and a (-3)-curve. The minimal resolution of s_2 has 38 rational IEDs above it, 33 (-2)-curves, 4 (-3)-curves and a (-4)-curve.

We were able to work out for the user that the surface was (birationally) an elliptic fibration of general type (Kodaira dimension 1) with geometric genus 3 and that the canonical map gave an explicit elliptic fibration map. These further computations took about 1 second in total. The intersection matrices — giving the graphs of IEDs over the singular points — also took less than a second except for the singularity s_2, where the computation of the self-intersections of the 38 IEDs took about 5 minutes. However, the non-self-intersections for s_2 were all computed in a fraction of a second.

We used the full suite of functions to go further and locate 4 (-1)-curves and another 23 curves on the desingularisation that needed to be contracted to give a minimal model. We were also able to work out the entire structure of the set of bad fibres of the minimal elliptic fibration. Apart from the fibres containing IEDs over the simple singularities, the IEDs over s_1 and s_2 along with a number of linking curves in S gave 4 distinct singular fibres in the minimal model each of Kodaira/Neron type I_8.

2.4.4 Hypersurface Singularities

As described elsewhere, structures have been created for testing whether an isolated singular point on a scheme (over a field) is analytically isomorphic to a (isolated) analytic hypersurface singularity and, if so, to compute an equivalent analytic equation and transformation to this up to an arbitrary precision. This precision can be increased at a later stage if required.

Isolated hypersurface singularities up to analytic equivalence can be transformed into a sum of a diagonal quadric part $x_1^2 + \ldots + x_r^2$ in the first r variables and a residual part $F(x_{r+1}, \ldots, x_n)$ in the remaining variables. The residual part is a power series with all terms of total degree ≥ 3. Two singularities are equivalent if and only if their residual parts are equivalent as (isolated) hypersurface singularities in the ambient space given by the final variables. This is all true

formally over any (algebraically closed) field of characteristic not equal to 2. The *corank* of the singularity is $m = n - r$, the number of variables of the residual part. Classification up to analytic equivalence comes down to the classification of the residual parts in all possible coranks and transformation to normal form consists of first transformation to diagonal plus residual form and then transformation of the residual form to an appropriate normal form.

Functionality has been implemented which determines whether an isolated analytical hypersurface singularity lies in one of Arnold's families and, if so, computes the exact family, normal form and the analytic transformation to that normal form to any required degree of precision. The explicit transformation allows the user to transfer any geometric problem that can be defined in terms of local functions at the singularity over to the corresponding one on the normal hypersurface form, which is usually of a much simpler shape. As opposed to other computer algebra packages that deal with Arnold normal forms, the original singularity is not required to be on an actual hypersurface. It only has to be analytically equivalent to a hypersurface singularity. Another useful feature is that this machinery for singularities applies over fields of characteristic $p > 0$. Arnold's classification and analysis applies to characteristic 0. However, all of his families make sense in characteristic $p > p_0$, where p_0 is a small prime depending on the family (and its indices).

The base field over which the input singularity is defined will usually be a finite field or number field k. To effect the full transformation to normal form, which is only possible in general over an algebraically-closed field, requires an extension of k. To avoid unnecessarily large field extensions, certain trivial final steps are ignored in the normalisation and also slightly different but easily equivalent normal forms are used in places. The major changes have been to avoid normalising coefficients of terms of the form ax^n to 1 by taking an nth root of a, and to remove binomial terms of the form ax^2y^n rather than axy^{2n} via $x \to x + \alpha y^n$ transformations. The latter avoids making unnecessary cubic field extensions. The differences between the normal forms used in this approach and Arnold's are all documented, and the user can easily make their own final transformation to Arnold normal form.

Other features of the implementation include a one or two-stage reduction to normal form and a uniformised computation of the transformation that works through one of only 3 auxiliary functions which deal with the semiquasihomogeneous cases, the non-sqh cases where Arnold's condition A is satisfied, and the non-sqh cases where the full 'spectral sequence' processing is required.

2.4.5 Surface Invariants

The functions that are described below apply to either the case of surfaces in $Proj^3$, where formal desingularisation is used, or to surfaces in ambients of arbitrary dimension when the simple singularity assumption is required.

- Simple singularities if any exist, and the A-D-E types of the singular points.
- Intersection number of two divisors on a surface (at least one of which should be Cartier) or two invertible sheaves (giving the intersection number of the two Cartier divisor classes that they represent).
- Geometric genus p_g, the *irregularity* q, and the two *Chern numbers*, c_1^2 and c_2, which are related by $c_1^2 + c_2 = 12(1 - q + p_g)$.
- The values of p_g and q for the desingularisation. Also the Chern numbers for a surface X having only simple singularities. Additionally, there is functionality to compute the *minimal* Chern numbers, which are the Chern numbers of a minimal model of X.
- The higher plurigenera P_n.
- The Hodge diamond numbers $h^{p,q}$ of a minimal desingularisation of a projective surface with only simple singularities.
- The Kodaira–Enriques type of a desingularisation of a projective surface X.

Again, once the Kodaira–Enriques class has been determined (see below), these can all be computed from the basic invariants and class or with the computation of only a few of the smaller plurigenera. For example, for surfaces of general type, only the basic invariants and second plurigenus P_2 are needed to compute all P_n by formula, and only the basic invariants are required if the surface is known to be minimal.

2.4.6 Minimal Models

For surfaces of Kodaira dimension ≥ 0 there are functions to compute a minimal model and the birational morphism from the original surface to it. For Kodaira dimensions 0 and 1 these functions can currently only be applied to non-singular (ordinary projective) surfaces. They use repeated *adjunction* maps, iterating until the termination condition $c_1^2 = 0$ applies to the image. If H is a hyperplane divisor of X in its projective embedding, the adjunction map is the divisor map of the $H + K_X$ divisor. Adjunction theory says that this map is birational, everywhere defined (a morphism), and that its image is again a non-singular surface if X is not rational or ruled. Furthermore, it blows down all exceptional curves of degree 1 in the projective embedding and it reduces the degree of higher degree exceptional curves or maps them to non-exceptional curves. It is therefore clear that iteration will eventually blow down all exceptional curves and give a minimal model. For Kodaira dimension 1 the function also returns the fibration map to a curve that presents the minimal model as an elliptic fibration.

- For surfaces X of general type, the minimal model functions differ slightly. There is an option to produce a non-singular minimal model using repeated adjunction mappings as above. The termination value of c_1^2 is determined from the second plurigenus of X.
- For surfaces X of general type, again, there is a more significant model than the minimal model or the individual n-canonical models. This is the model in weighted projective space associated to the almost-ample canonical divisor K_X. As an abstract scheme it is isomorphic to the minimal model with (-2)-curves contracted. It is usually referred to simply as the *canonical model* and there is a function to compute it.
- For rational and ruled surfaces, the adjunction process works as above except that it eventually reaches a *terminal* surface on which the adjunction map is not everywhere defined or its image is a curve or it is not birational. The terminal surfaces form a small number of families of specially-embedded surfaces. In the non-rational cases, they are line and conic bundles over the base curve. In the rational case, we have line and conic bundles along with anticanonically-embedded Del Pezzo surfaces and one or two other possibilities.
- For surfaces having Kodaira dimension -1 the minimal model function returns one of these above-mentioned terminal surfaces.

2.4.7 Parametrisation of Rational Surfaces

There are *parametrisation* functions for rational surfaces. These construct a birational map from X to $Proj^2$ by first mapping to a terminal surface X_1 and then calling special parametrisation routines for Del Pezzo surfaces and line and conic bundles which will be described below. For rational surfaces with only simple singularities, the map to the terminal surface is constructed as described above. For rational surfaces in $Proj^3$ with general singularities, a special routine is used that finds an appropriate multi-adjunction map, associated to a $nH + mK_X$ divisor, which maps X to a terminal surface in one step. This first map can be constructed over an arbitrary base field k. Parametrisation of the terminal surfaces is an arithmetic question, though, and the existence of such a parametrisation depends on k. The routines for parametrising the terminal surfaces assume that k is $\mathbb{Q}$ or a number field.

There are a number of functions to construct random surfaces from interesting families of non-singular projective surfaces in $Proj^4$. These families cover a number of different Kodaira–Enriques

types. The constructions use 'codimension 2' commutative algebra techniques and are based on functions in the Macaulay Computer Algebra System for the same families.

An example of a rational surface in $Proj^3$ parametrisable over $\mathbb{Q}$ is the non-normal surface with defining polynomial $w^3y^2z + (xz + w^2)^3$. The classification routine maps it to a degenerate degree-6 Del Pezzo surface in $Proj^6$ with an A2 singularity. Then the parametrisation routines for Del Pezzo surfaces construct a parametrisation map $Proj^2 \to X$ given by degree-6 polynomials with rational coefficients.

2.4.8 Del Pezzo Surfaces

A (strong) Del Pezzo surface X is a rational surface for which the anticanonical divisor $-K_X$ is ample. This means that the divisor map of a multiple of the divisor gives an isomorphic embedding into ordinary projective space. The degree d of X is equal to its first Chern number $K_X.K_X$ and takes values between 1 and 9 inclusive. For $d \geq 3$, $-K_X$ gives an embedding of X as a non-singular surface of degree d in $Proj^d$. For degrees 1 and 2 it is necessary to take the multiple $(4 - d)(-K_X)$ to get an ordinary projective embedding. In these last two cases there is an anticanonical embedding as a hypersurface in an appropriately-weighted projective space of dimension 3. A *weak* Del Pezzo surface is one where $-K_X$ is not necessarily ample but is close to ampleness; technically, it is big and *nef*. These differ only slightly from the (strong) Del Pezzo surfaces.

Del Pezzo surfaces are represented in Magma by their anticanonical models. Both strong and weak Del Pezzo surfaces are allowed but some of the current functionality only applies to strong Del Pezzo surfaces. A Del Pezzo surface of degree d can also be defined as a surface which is isomorphic over the projective closure of its base field k to the projective plane $Proj^2$ blown up at $9 - d$ points. The points have to be distinct and satisfy certain conditions for a strong Del Pezzo surface. For weak Del Pezzo surfaces, the blow-up points can be infinitely-near (i.e., lying on the exceptional curve of a point that has already been blown up) and the set of points has to satisfy weaker conditions. There are functions to check the set of conditions on a given set of plane points and construct the corresponding Del Pezzo surface. There is also a function to test whether a given non-singular projective surface X is abstractly a strong Del Pezzo surface and return the anticanonical model and isomorphism from X to it.

There is a suite of functions that constructs a parametrisation of a Del Pezzo surface over $\mathbb{Q}$ and number fields. This covers all degrees and also the weak case. For high degree Del Pezzo surfaces the functions use a method based on explicitly computing the Lie algebra of the automorphism group of the surface. Lower degree cases are reduced to the higher degree case by blowing down Galois-stable sets of disjoint exceptional curves. For the lower degree degenerate cases, special algorithms based on projecting from singular points are used. The functions return whether the input Del Pezzo surface X is parametrisable over the base field k and also returns an explicit parametrisation (birational map $Proj^2 \to X$) over k in the affirmative case.

Other functionality includes functions for computing and working with the lines on a cubic (degree 3 Del Pezzo) surface in $Proj^3$. There are also functions to compute the classical invariants and covariants of a cubic surface and to transform its equation to pentahedral form (that is, the sum of 5 cubes of linear forms) if possible.

On the arithmetic side, minimisation and reduction routines are provided for degree 3 and degree 4 Del Pezzo surfaces over $\mathbb{Q}$ which find isomorphisms over $\mathbb{Q}$ to models with simpler defining equations containing smaller integer coefficients.

2.5 Toric Geometry

2.5.1 Introduction

A torus is a connected, affine algebraic group that decomposes into a direct product of the multiplicative group over the algebraic closure of the base field. Toric varieties are special varieties with the algebraic action of a torus that have a cellular decomposition into orbits under this action. They can be represented combinatorially as fans, which are unions of rational polyhedral cones in a finite-dimensional rational vector space, or dually, as polytopes in the dual space. Toric-invariant divisors and other interesting geometric structures attached to the variety can be similarly represented.

Toric varieties are an important area of current research. Many interesting rational varieties are toric, including higher-degree Del Pezzo surfaces and many Fano varieties, which are higher-dimensional analogues of Del Pezzo surfaces. The one-to-one correspondence with fans allows the construction of toric varieties and the investigation of their geometric properties through combinatorial geometric methods.

Magma contains a large body of code for toric geometry and working more generally with polytopes and polyhedra. Related to this are a number of important databases of geometric families that can be described in toric terms. In addition to the standard features of a computational toric package, there is a new algorithm to deal with more general maps than the usual toric type, code for Cox rings, and links to Magma schemes.

2.5.2 Major Functionality

Along with many other routines for cones and polyhedra, important functions for toric geometry include the following:

- Creation of fans to represent toric varieties.
- Basic properties such as non-singularity, completeness, and being Gorenstein or $\mathbb{Q}$-Gorenstein (a multiple of the canonical sheaf is invertible).
- Blowing up toric subvarieties.
- Traditional toric maps defined as combinatorial maps between fans.
- New algorithm for maps using radical rational functions. This allows the definition of maps which cannot be defined combinatorially as above.
- Wide range of functionality for toric divisors, represented by polytopes, and the divisor class group. This includes tests for linear equivalence, principality, being Cartier or $\mathbb{Q}$-Cartier (a multiple is Cartier), ampleness, very ampleness, nefness, and being big.
- Riemann–Roch spaces for toric divisors, returned as a polytope or a set of functions giving a basis. Computation of the full graded ring of multiple sections of an ample divisor.
- Cox rings of toric varieties.
- Resolution of singularities.
- Mori theory: computation of the Mori and nef cones; contraction of extremal rays; flips; minimal models.

2.5.3 Databases

The oldest database listed here is not a 'toric' database but is similar in style. It is the Magma database of families of K3 surfaces embedded in small-dimensional weighted projective spaces. The computation of these is part of a project that has been running since the mid-1970s, and now contains 24099 entries. The entries are candidates for the Hilbert series of the graded coordinate ring of a K3 surface in weighted projective space.

The toric databases contain data for polytopes that represent toric Fano varieties of certain types.

Fano varieties are higher-dimensional analogues of Del Pezzo surfaces: complete varieties whose anticanonical divisor is ample. The three big toric databases contain:

- Terminal Fano polytopes in dimension 3.
- Canonical Fano polytopes in dimension 3.
- Smooth (i.e., non-singular) Fano polytopes up to dimension 8.

Chapter 3

Associative Algebras

3.1 Associative Algebras Given by Structure Constants

3.1.1 Introduction

Finite dimensional algebras supported in Magma are organised as follows. Any algebra may be defined as a structure constant algebra; that is, defined by specifying a multiplication table for the basis elements. This is how most algebras are represented in Magma; the main exceptions, which have their own special representations, are group algebras, matrix algebras, and commutative algebras (which are best represented using polynomial rings and modules).

Apart from the special types just mentioned, and apart from Lie algebras, most algebras supported fall under the banner of associative algebras. At this level of generality there is considerable functionality: support for orders and ideals, computation of radical, and idempotent decomposition. Depending on the base field, more nontrivial routines are provided. For an algebra over $\mathbb{Q}$, an algorithm to compute a maximal order in the algebra is implemented (de Graaf).

An extensive package of algorithms is provided for the special case of quaternion algebras. In the future, it is likely that a package will be developed for central simple algebras.

3.1.2 Algebras Given by Structure Constants

In the rest of this page, tools for finite-dimensional associative algebras defined by structure constants (ASC-algebras) will be reviewed. This is a very general way of presenting a finite-dimensional algebra, though as the dimension increases the memory needed for the structure constant matrices can become substantial and so it is limited to dimensions of at most a few thousand. Certain other types of associative algebras such as quaternion and group algebras have different representations that make use of special features of the particular type of algebra.

A general finite-dimensional associative algebra can also be represented either as a matrix algebra or as a finitely-presented algebra. One can move between these representations. Thus, given a finite-dimensional finitely-presented algebra (fp-algebra) one can often construct its regular representation using a non-commutative Gröbner Basis algorithm.

3.1.3 Construction of Algebras

Specifying a structure constant algebra A is straightforward. A sparse representation is available if it is believed that the structure constant vectors are sparse. There are a number of functions that create ASC-algebras directly from other algebraic structures. If F is a number field or finite field with subfield E, a function is provided which constructs the ASC-algebra of dimension

$[F : E]$ over E which is isomorphic to F. Given a simple algebra A with centre K, another function returns a K-algebra B which is K-isomorphic to A. Finally, a group algebra may be converted to an ASC-algebra.

Constructors are available to create subalgebras and quotient algebras of an ASC-algebra A. However, quotient algebras are only available for the case in which A is defined over a field. If A is an R-algebra and S is a ring such that there exists a monomorphism $f : R \to S$ then a function allows the user to construct the S-algebra B obtained by applying f to map the coefficients of elements of A into S.

Every algebra over a field comes with a basis and the usual operations are available: extract basis, extend basis, test independence of specified algebra elements, etc. Functions are provided to create various subalgebras: centre, centraliser, idealiser, left and right annihilators, commutator of two subalgebras (as a module or ideal), etc.

3.1.4 Decomposition

The functions described in this paragraph assume that A is defined over a finite field. An algebra A can be regarded as a (left- or right-) module for itself so that if A is defined over a finite field, the machinery for A-modules can be used to decompose modules arising from the regular module for A and so compute structural information for A. The information that can be computed includes: a composition series, the composition factors, the Jacobson radical (the largest nilpotent ideal), and the decomposution of A into a direct sum of indecomposable ideals. Tests for an algebra being simple or semisimple are available. In addition the minimal two-sided (left, right) ideals and maximal two-sided (left, right) ideals can be computed. The machinery is very efficient so that composition series (and factors) have been computed for algebras having dimension up to 10,000.

The functions described in this paragraph assume that A is defined over a number field. The Jacobson radical $J(A)$ (the largest nilpotent ideal) of A can be computed using an algorithm due to A. Cohen, G. Ivanyos, and D. Wales. A function is provided which computes the decomposition of A into a direct sum of ideals of A such that each is indecomposable. A closely related function returns a sequence of primitive orthogonal idempotents in the centre Z of A such that their images in $A/J(Z)$ span $J(Z)$. This machinery will soon be replaced by a more powerful approach based on the use of a Meataxe that works for modules defined over number fields.

3.1.5 Ideals and Orders

Let K be a number field and let R be the ring of integers of K. For an associative algebra A over a field K, an order O is a subring of A such that O contains a K-basis for A and is also a finitely generated R-module. Orders in an algebra are represented by a pseudobasis, similar to the way in which modules over Dedekind domains are represented.

Arbitrary orders of an algebra can be created in Magma. A function is also provided to construct a maximal order in an algebra. The standard operations for orders and their elements are provided. Left, right and two-sided ideals of orders are supported. Apart from the obvious arithmetic operations, sums, products, and intersections of ideals are supported as is the computation of colon ideals.

3.2 Finitely-Presented Algebras

3.2.1 Gröbner Bases

A finitely-presented algebra (FPA) is a quotient of a free associative $K<x_1, \ldots, x_n>$ algebra by an ideal of relations, where K is a field. To compute with such ideals and quotients, one constructs *noncommutative* Gröbner bases (GBs), which have many parallels with the standard commutative GBs. At the heart of the theory is a noncommutative version of the *Buchberger algorithm* which computes a GB of an ideal of an algebra starting from an arbitrary generating set of the ideal. One significant difference with the commutative case is that a noncommutative GB may not be finite for a finitely-generated ideal, but in such cases it may still be useful to construct a truncated GB up to a certain degree d.

Magma contains a powerful suite of algorithms for computing with noncommutative GBs and FPAs. Ideals can be defined to be left-, right-, or two-sided and a GB can be computed for each type of ideal. Besides the standard Buchberger algorithm, Magma includes a noncommutative generalisation by A. Steel (2005) of the Faugere F_4 algorithm for GBs, which is based on sparse linear algebra techniques. As for the commutative case, this uses optimal packed representations for small finite fields and modular algorithms for fields of characteristic zero and so this algorithm typically runs dramatically faster than the Buchberger algorithm.

For example, here are some standard benchmarks taken from a paper [V. Levandovskyy, G. Studzinski, B. Schnitzler, *Enhanced computations of Groebner bases in free algebras as a new application of the letterplace paradigm*, Proc. ISSAC 2013] in which the authors compare their algorithm to compute noncommutative GBs in Singular with Magma. Let F be the free algebra $\mathbb{Q}<x, y, z>$ and let I be the two-sided ideal generated by the following *braid relations*:

$$yxy - zyz, \quad xyx - zxy, \quad zxz - yzx, \quad x^3 + y^3 + z^3 + xyz.$$

Magma computes the noncommutative GB of I up to degree 11 in 1.5 seconds; the GB has 726 elements having up to 316 terms each. The largest step in the F_4 algorithm involves reducing a 16256×17895 sparse matrix (density 0.1%) over $\mathbb{Q}$.

As another example, let F be the free algebra $\mathbb{Q}<f_1, f_2, f_3, f_4>$ and let I be the two-sided ideal generated by the following *Serre relations* (related to Cartan matrices of Lie algebras):

$$f_1f_2 - f_2f_1, \quad f_1f_4 - f_4f_1, \quad f_2f_3 - f_3f_2, \quad f_1f_3^2 - 2f_3f_1f_3 + f_3^2f_1,$$

$$f_2f_4^2 - 2f_4f_2f_4 + f_4^2f_2, \quad f_1^2f_3 - 2f_1f_3f_1 + f_3f_1^2, \quad f_3f_4^2 - 2f_4f_3f_4 + f_4^2f_3,$$

$$f_2^2f_4 - 2f_2f_4f_2 + f_4f_2^2, \quad f_3^3f_4 - 3f_3^2f_4f_3 + 3f_3f_4f_3^2 - f_4f_3^3.$$

Magma computes the noncommutative GB of I up to degree 15 in 13.5 seconds; the GB has 43 elements having up to 1338 terms each. The largest step in the F_4 algorithm involves reducing a 1898503×1916549 very sparse matrix (density 0.0001%) over $\mathbb{Q}$. In comparison, Singular takes 147.3 seconds for the same computation.

3.2.2 Ideals

For ideals of free algebras and their quotients, Magma supports analogues of all the basic operations for the commutative case when applicable. Let A be the FPA $K<x_1, \ldots, x_n> / I$, where I may be a left-, right-, or two-sided ideal. If the GB of I is finite and has been computed, then it is easy to determine whether A has finite dimension or not. If the dimension d is finite, then Magma can easily compute d and a representation of A as a matrix algebra or a structure constant algebra, together with the isomorphism, assuming d is not too large (since this involves constructing $d \times d$ matrices over K). Using this matrix representation, one can also compute

properties of elements of A such as minimal polynomial and also structural properties of A such as the centre or Jacobson radical, with the results involving elements of A explicitly (i.e., as words in the free generators, reduced modulo I). The advantage of the FPA representation of an algebra is that is often *sparse*: one can construct algebras with dimensions in the tens of thousands or more which have compact presentations as FPAs but would require matrices which are too large in practice with the matrix or structure constant algebra types.

3.2.3 Exterior Algebras

Magma also has special support for *exterior algebras*. Such an algebra is skew-commutative and is a quotient of the free algebra $K<x_1, \ldots, x_n>$ by the relations $x_i^2 = 0$ and $x_i x_j = -x_j x_i$ for $1 \leq i, j \leq n$, $i \neq j$. Because of these relations, elements of the algebra can be written in terms of commutative monomials in the variables (via a collection algorithm), and the associated algorithms are much more efficient than for the general noncommutative case. Also, a Gröbner basis of an ideal of an exterior algebra is always finite and another special extension of the Faugere F_4 algorithm enables Magma to compute this GB of an ideal very rapidly. One can also construct modules over exterior algebras and construct free resolutions and perform operations similar to those for modules over commutative polynomial rings. As an example of an application in algebraic geometry, free resolutions over exterior algebras are heavily used in the Decker–Eisenbud–Fløystad–Schreyer algorithm to compute the cohomology of coherent sheaves.

3.3 Matrix Algebras

3.3.1 Introduction

Matrix algebras are one of the associative algebra types in Magma. They arise in many areas of which an important one is the representation theory of groups and algebras. Much functionality is provided for computing with individual elements. However, at present some important structural functionality has yet to be implemented.

A general finite-dimensional associative algebra can also be represented either as a matrix algebra or as a finitely-presented algebra. One can move between these representations. Thus, given a finite-dimensional finitely-presented algebra (fp-algebra) one can often construct its regular representation using a non-commutative Gröbner Basis algorithm.

3.3.2 Construction of Algebras

A matrix algebra A is normally specified by giving a set of generating matrices. It can be defined over any exact ring supported by Magma. Given a pair of matrix algebras, functions are provided to construct their direct sum and tensor product algebras.

Constructors are available to create subalgebras and quotient algebras of an ASC-algebra A. However, quotient algebras are only available for the case in which A is defined over a field. If A is an R-algebra and S is a ring such that there exists a monomorphism $f : R \to S$ then a function allows the user to construct the S-algebra B obtained by applying f to map the coefficients of elements of A into S. Functions are provided to create the centre of A and the centraliser of a subalgebra of A.

A large number of operations are implemented for the matrix elements. These include properties such as being a unit, nilpotent, or unipotent; invariants such as determinant, trace, order, projective order (when over a finite field); and canonical forms such as Jordan form, rational form, invariant factors (when over a field) and echelon form, Hermite form, and Smith form (when over an Euclidean domain). Other operations include adjoint, characteristic polynomial, minimal

polynomial, Hessenberg form, and eigenvalues. The algorithms used for these problems are often very sophisticated so as to reduce the runtime as much as possible.

3.3.3 Structure of the Algebra

In order to perform various structural operations, a matrix algebra A over a field K is treated as a vector space M by regarding each $n \times n$ matrix as a vector of length n^2. Using elementary linear algebra, it is straightforward to construct a basis for A thereby determining its dimension and making it possible to test membership of A. Left-, right- and two-sided ideals can be created. Over a field, a basis is found by a form of spinning algorithm. This allows the user to determine dimension, basis and to test membership of the ideal.

Noting that A has a natural action on M, it can be regarded as an A-module. In the case in which K is a finite field the Meataxe can be applied to M to determine such things as a composition series and the composition factors for A. The minimal left-, right-, or two-sided ideals can also be found. At present no other ideal operations are available.

At present algorithms of Brooksbank and O'Brien are used to compute the Jacobson radical and unit group of A though these will be shortly be replaced by algorithms based on data described in the *Section on presentations* below. If A is defined over a field that is not a finite field then an algorithm suggested by Willem de Graaf is used.

3.3.4 Presentations and the Wedderburn Decomposition

The functions described in this subsection assume that A is defined over a finite field. A series of algorithms developed by Jon Carlson and Graham Matthews enable the user to construct a presentation for A. In the course of doing this, much important structural information about A is derived:

- A list of primitive idempotents for A, one idempotent for each irreducible module.
- The simple quotient algebras of the matrix algebra A.
- The algebra eAe, where e is a sum of primitive idempotents, one for each simple A-module.
- The Jacobson radical of A.
- The Wedderburn decomposition of A into a direct sum of indecomposable ideals.
- The Cartan matrix for A.
- A presentation in generators and relations of A.

Once a presentation is available, a non-commutative Gröbner basis algorithm is used to construct a basis for the finitely-presented algebra. In particular, this yields the dimension of the algebra. The presentation machinery also gives a test for membership in a matrix algebra. If A is a subalgebra of the full matrix ring $Mat(n, \mathbb{F}_q)$, Magma can decide whether a matrix of $Mat(n, \mathbb{F}_q)$ is an element of A. If the matrix is in A then it can be written as a polynomial in the associate algebra defined by the presentation. The construction of a presentation is usually quite fast and algebras of dimension 10,000 or more can be handled.

Let G be the wreath product of the cyclic group of order 2 with the alternating group A_7, a group of order 322,560. The matrix algebra A over $\mathbb{F}_2$ generated by matrices giving the action of G on the coset space of its Sylow 3-subgroup has degree 1120. Using the Carlson–Matthews algorithms, it takes 68.27 seconds to construct a presentation for A and a further 2.80 seconds to determine that the dimension of A is 77,045. If the same matrix algebra is taken over $\mathbb{F}_5$, the runtime is 2363 seconds and the dimension of the algebra is 112,002.

3.4 Quaternion Algebras

3.4.1 Introduction

There are only two quaternion algebras over the reals: 2 by 2 matrices, and the Hamiltonians. Over other fields such as $\mathbb{Q}$, there are infinitely many isomorphism classes. The Magma package mainly deals with arithmetic questions about quaternion algebras over global fields; that is, $\mathbb{Q}$, number fields, and functions fields $k(t)$ over a finite field k. Unless explicitly stated, these methods were developed by Kohel ($\mathbb{Q}$), Donnelly, Kirschmer and Voight (number fields), and Kirschmer (function fields).

3.4.2 Local Properties

The isomorphism type of an algebra is determined by local data; i.e., by properties of the algebra tensored with completions of the base field (these are simpler objects to study). A consequence is that existence and construction of isomorphisms are completely different computational problems, with the latter being much harder. The local calculations needed to decide the isomorphism class amount to computing Hilbert symbols. Functions are also provided for computing Hilbert symbols $(a, b)_P$ for field elements a, b without reference to quaternion algebras. Hilbert symbols are also used in the algorithms to compute maximal orders in quaternion algebras. Special maximal order algorithms are implemented for quaternion algebras over each type of field. (The general maximal order algorithm for associative algebras could be used, of course; however it is advantageous to make use of the very special structure of quaternions.)

A different kind of local calculation is: Given an algebra A whose completion $A \otimes F_v$ is trivial (as shown by computing a Hilbert symbol), explicitly construct an isomorphism over F_v to the trivial algebra, namely the algebra of 2 by 2 matrices. This is not a computationally hard problem, but requires the algorithm to be implemented very carefully. Magma contains a robust implementation over each kind of field.

3.4.3 Orders

The preceding paragraph discussed local properties of algebras; we now turn to orders in algebras. For an algebra A over a field F, an order may be defined as a subring of A which is a module of finite rank over some order of F. Note that while all modules are free over rings such as $\mathbb{Z}$ or the polynomial ring $k[t]$, this is not true for orders in number fields. Therefore, we have the computational issue of how to represent and handle orders and ideals. Magma has a well-developed infrastructure for dealing with non-free modules, originally developed for handling orders in relative number fields, and which has also been made available to the user as a package for modules over Dedekind domains in general. In particular, this furnishes a canonical representation (echelon form) of a module.

When working in quaternion algebras this machinery is mainly kept in the background, while the user interracts with the system via more user-friendly commands. Orders and ideals can be defined by various kinds of generating data, and many arithmetic operations on them are implemented. As well as the most obvious ones, these include constructions like intersections, normalisers, and colon ideals.

In the theory of quaternion algebras, special kinds of orders play important roles. Magma contains algorithms for testing whether a given order is maximal, Eichler, hereditary, Gorenstein, or Bass.

3.4.4 Enumeration of Ideal Classes

For a given (maximal or Eichler) order in a quaternion algebra, two key computational problems stand out: to describe the group of units in the order, and to describe the set of (right or left)

equivalence classes of ideals. Here we see a very close analogy with number fields, for which the key problems are to compute the maximal order, the unit group, and the ideal class group. The biggest difference in the non-commutative setting is that the set of ideal classes of a quaternion order does not have a natural group structure. On the other hand, the classes of two-sided ideals *do* form a group. Computing this is another important problem, in its own right, and also as a subproblem in a variant of the algorithm for finding all ideal classes. Magma contains algorithms for each of these problems (note: the units and ideal classes are not always finite, but they admit finite descriptions). Of most interest is the case of *definite* quaternion algebras: an algebra over $\mathbb{Q}$ is definite when all its subfields are imaginary quadratic fields, and the notion extends to algebras over number fields and function fields. The algorithms in Magma for this important case are especially refined (Donnelly and Kirschmer); in particular, a number of strategies and tricks have been developed to quickly test ideal isomorphism during the enumeration of ideal classes.

There is an especially significant application which relies on this ideal class problem in definite algebras, together with a closely related problem: finding all elements of given norm in an ideal. The Magma algorithm for this problem uses many of the same techniques used in the enumeration of ideal classes. Together these algorithms enable us to efficiently do "Brandt module" calculations over number fields; Magma is the only software with this ability over totally real number fields in general. Brandt module calculations form the core of the main algorithm in Magma for Hilbert modular forms (which are discussed in the section on arithmetic geometry).

Chapter 4

Combinatorial Theory

4.1 Introduction to Combinatorial Theory

A substantial number of algorithms have been implemented for graph theory. Both simple graphs and simple digraphs are supported as well as multigraphs and multidigraphs. In each case vertices and/or edges may be labelled or unlabelled. Graphs may be represented using the adjacency matrix (dense representation), or using an adjacency list (sparse representation). Thus, graphs with millions of vertices may be handled if the sparse representation is used,. The algorithms implemented for graphs are mainly concerned with determining various properties of the graph, such as planarity, and locating important substructures such as cliques. A linear time algorithm is used to test whether a graph is planar. The `nauty` program developed by B. Mckay is used to compute the automorphism group of a graph and to test whether two graphs are isomorphic.

Besides graphs, flow networks have been implemented. A *flow network* (or *transportation network*) is a directed graph where each edge has a *capacity* associated with it. An edge represents a flow of material between two vertices and the capacity represents the maximal amount of material that can be transported along the edge in some fixed time interval. The fundamental network flow problem is to determine a maximum flow at minimum cost from a specified source to a specified sink. Two sophisticated algorithms for computing the maximum flow are provided: the *Dinic algorithm* and the *push-relabel algorithm*; these were improved and implemented by P. Lieby.

Incidence structures arise in various areas of discrete mathematics including in the study of finite groups. A range of related incidence structures including general (finite) incidence structures, near-linear and linear spaces, and t-designs have been implemented in Magma. A particularly useful algorithm provided for incidence structures is a test for t-balance. Other functionality is concerned with searching for resolutions, parallelisms, and parallel classes. An efficient backtrack algorithm developed by W. Unger is provided for computing the automorphism group and for testing pairs of designs for isomorphism.

A structure closely related to block designs is the *finite plane*. This is separate type in Magma and some geometric tools are provided. Both the affine and projective versions of a plane are available. Finite planes may be constructed over either finite fields or finite nearfields. Some special constructions for finite planes such as Hughes' planes are also provided. Tools for finding substructures such as the Baer subplane and k-arcs are available but much is missing. Again finding the collineation group of a plane and testing planes for isomorphism is performed using the backtrack algorithm mentioned above.

Another structure closely related to t-designs is the *Hadamard matrix*. The main algorithms implemented are concerned with testing Hadamard matrices for equivalence and for computing their automorphism group. These functions are based on Mckay's `nauty` program. Associated with

equivalence testing is the provision of a function to compute a canonical form for an isomorphism class of Hadamard matrices. A large database of Hadamard matrices is provided.

4.2 Partitions, Young Tableaux, and Symmetric Functions

4.2.1 Partitions

One significant combinatorial building block is that of partitions. A *partition* of a (positive) integer n is a way of writing n as a sum of positive integers (and is represented within Magma by a sequence of those integers, arranged in decreasing order). A *restricted partition* is a partition where the summands are constrained to come from some particular subset of the integers.

Magma provides the standard counting functions for partitions and restricted partitions, as well as functions for enumerating partitions. When enumerating it is possible to restrict to partitions of a specific size. Such enumerative functions can greatly simplify enumeration through an ad hoc combinatorial structure of the user's creation.

4.2.2 Young Tableaux

Magma builds on the partition functionality to provide tools for working with Young tableaux and associated monoids. The monoids in question are ordered monoids, which will usually be the positive integers, although Magma supports more general constructions. Related to such an ordered monoid is the associated plactoid monoid, which is the quotient of the monoid with respect to Knuth equivalence. Magma provides constructions for ordered monoids and the matching plactic monoid, and testing of words for Knuth equivalence.

A Young diagram (also called a Ferrers diagram) is a collection of boxes arranged in rows, where each row is aligned at the left and has at least as many boxes in it as the row below it. A Young tableau (or just tableau) is obtained from a Young diagram by placing an element from an ordered monoid in each box in such a way that the entries are increasing when read down columns and non-decreasing when read across rows. Young tableaux are thus created from the words of an ordered monoid. It turns out that each word is Knuth equivalent to that of some (unique) tableau, and thus that the elements of the associated plactic monoid may be put into correspondence with (distinct) tableaux.

Magma provides facilities for enumerating all tableaux with specified properties or for generating random tableaux. It also provides skew tableaux (the results of deleting a smaller tableau from the top left of some other tableau). Other functionality includes conjugate tableaux, hook length computations, *jeu de taquin*, and a realisation of the Robinson–Schensted–Knuth correspondence.

The irreducible complex representations of $\mathrm{Sym}(n)$ are in one-to-one correspondence with a set of Young diagrams. Thus Young tableaux are an important tool in the study of representations of the symmetric group and the general linear group. This machinery is used within Magma to construct representations of $\mathrm{Sym}(n)$.

4.2.3 Symmetric Function Algebras

A further extension of partitions and Young tableaux lies in the realm of symmetric function algebras. Here the object of interest lies in a basis for the algebra and the main operation of interest is changing between a pair of the five bases of interest. The five main bases of a symmetric function algebra are best understood in terms of operations on a set of associated tableaux. Symmetric function algebras allow the manipulation of symmetric functions over an infinite number of indeterminates, whereas the more usual polynomial rings are restricted to a finite number.

Symmetric function algebras in Magma may be created over any of the five standard basis types: Schur, monomial, elementary, homogeneous, or power sum functions. Arithmetic and composition operations are provided and are able to transition smoothly between the different basis types as required. Conversions are also possible to and from symmetric polynomials.

The Schur functions are the generating basis for the standard tableaux, and it is possible to get the tableau corresponding to such a function. There is also a correspondence between Schur functions and characters of the symmetric group — the irreducible characters are precisely those arising from Schur functions indexed by a single partition — and Magma provides an explicit construction of this correspondence.

4.3 Graph Theory

4.3.1 Introduction

A *graph* G consists of a set of *vertices* and a set of *edges*, where each edge denotes a connection between two of the vertices. A graph is *simple* if it has no loops (a *loop* is an edge connecting a vertex to itself) and has at most one edge between any given pair of vertices. A *directed graph* (shortened to *digraph*) is one in which the connections are "one-way": An edge from u to v is not equivalent to an edge from v to u. Consequently, a simple digraph may have an edge from v to u as well as one from u to v.

Non-simple graphs (and digraphs) are called *multigraphs* (respectively, *multidigraphs* or directed multigraphs). Magma supports all four of these graph types. In the rest of this section, the term "graph" will be used as an inclusive term for all of these graph types.

Vertices and edges of a graph may have additional information associated to them. In the case of vertices it is traditional to think of this as assigning a colour to each vertex. Edges are very commonly assigned numeric values, which are usually referred to as "weights".

4.3.2 Constructions for Graphs

Graphs arise naturally in a good many contexts, and this is reflected in the rich variety of construction tools that Magma provides for graphs. To start with, there is a completely general means of specifying a graph via either an explicit listing of the edge relations or an adjacency matrix. Graphs may also be constructed from groups (Cayley graph, Schreier graph, orbital graph, closure graph, Paley graph, Paley tournament) and designs (incidence graph, point graph, block graph, line graph, Hadamard graph). Then there are iterative techniques for constructing graphs from other graphs; examples include addition or removal of vertices and/or edges, union, product, complement, contraction, and switching.

Magma also provides easy ways to create graphs of various well-known families, such as bipartite graphs, multipartite graphs, complete graphs, k-cube graphs, polygon graphs, path graphs, and others. It is also possible to create random graphs and trees. Additionally, several optional databases of graphs are made available for use with Magma. These include a database of strongly regular graphs, and several databases of small graphs, including Eulerian graphs, planar graphs, and self-complementary graphs.

4.3.3 Cliques and Independent Sets

A *clique* of a graph is a set of vertices such that any two are connected to each other. An *independent set* is a set of vertices such that no two of them are connected to each other. An independent set thus corresponds to a clique in the complement graph.

Two algorithms are provided for finding cliques in a graph, depending on whether the aim is to find cliques of a given size, or to find the largest possible cliques. To find the largest clique size the branch and bound algorithm of Bron and Kerbosch is preferred. When the size is specified in advance then the algorithm used is a dynamic one due to Walker, Myrvold, and Prsa.

Note that the Magma implementation of the branch and bound algorithm has been adapted into a variant that can also search for non-maximal cliques of a given size. Also, the dynamic algorithm can be iterated in order to determine a maximum clique size. Consequently, each approach can be used to solve each problem, and which is the better option will depend on the structure of the graph. In general, though, the above algorithm choices are often best.

4.3.4 Equivalence and Automorphisms

An important (and difficult) problem in the theory of graphs is to determine whether two graphs are isomorphic. A related problem is to determine a canonical representation of a given graph; if this can be done then it leads to a simple test for isomorphism: Two graphs are isomorphic exactly when their canonical representations are the same.

Magma takes this approach to the graph isomorphism problem, using B. McKay's excellent `nauty` software, widely regarded as state of the art in this area and included in Magma with his permission. Additionally, `nauty` is able to produce the full group of automorphisms of a graph, solving another problem of key importance.

In each case the morphisms will take into consideration any vertex-colouring of the graphs, ensuring that consistent colouring properties are maintained. As an example of this in action, Hadamard matrices can be represented by an equivalent graph with a suitable vertex colouring; thus the graph algorithms above make it possible to find canonical forms of Hadamard matrices. This is important for efficient use of the Hadamard matrix database.

4.3.5 Networks

Multigraphs further extend to networks, which in this context are directed multigraphs where each edge has associated to it a *cost* and a *capacity*. This enables modelling of problems involving *flow*, where items or information must travel along the network, subject to constraints of cost. The two key problems in this area are *maximum flow* and *minimum cut*. In maximum flow, the goal is to determine just how much can travel from one point in the network to another; this includes producing a valid routing of that amount while minimising the associated cost. For minimum cut, on the other hand, the aim is to compute sets of key network edges which must be preserved in order to maintain that rate of flow.

Magma includes two algorithms for solving these problems: The Dinic algorithm and a general push-relabel algorithm. Both have the same theoretical complexity in the general case, but several heuristics (such as those described by Cherkassky and Golberg) are applicable to the general push-relabel algorithm that make it faster in most cases. The exception is the case of very sparse networks in which all edges have the same capacity; in such cases the Dinic algorithm performs better.

These tools find natural application in real world situations wherever networks arise. Examples include freight transportation, traffic planning, communication networks, supply chain management, timetabling, and more. Generally speaking, applying maximum flow enables efficient use of the network, and minimum cut identifies critical components. Both combine to guide expansion of the network in order to obtain larger flows while increasing reliability (such as by adding redundancy for critical paths).

4.3.6 Applications

The graph theory machinery is used very widely. Perhaps one of the largest areas of application is to the study of questions in algebraic graph theory. This work is mainly carried out by three groups: The group headed by M. Conder at the University of Auckland, a group based in the Former Republic of Yugoslavia including A. Malnic, D. Marusic and P. Potocnik, and finally the group of C. Praeger at the University of Western Australia.

Other significant applications have been to the factorisation of certain types of graph, and graph theoretic constructions for codes, designs and planes. An application by G. Brown uses a Magma graph as the basis of a database of geometric objects. The Magma profiler also uses the graph theory machinery to manage trees of function calls. Approximately 200 papers have been located that cite Magma for computations in graph theory. Some earlier papers are listed at:

- `http://magma.maths.usyd.edu.au/magma/citations/topic/Graphs/`

4.4 Design Theory

4.4.1 Introduction

An *incidence structure* is a triple $D = (P, B, I)$, where:

- P is a set, the elements of which are called *points*;
- B is a set, the elements of which are called *blocks*; and
- I is an incidence relation between P and B, so that $I \subset P \times B$. The elements of I are called *flags*.

A *near-linear space* is an incidence structure in which every block contains at least two points and any two points lie in at most one block. A *linear space* is a near-linear space in which any two points lie in *exactly* one block. It is usual, when discussing near-linear spaces, to use the term *line* in place of the term *block*.

Let v, k, t and λ be integers with $v \geq k \geq t \geq 0$ and $\lambda \geq 1$. A *t-design* with v points and *blocksize* k is an incidence structure $D = (P, B, I)$ where:

- The cardinality of P is v;
- D is uniform with blocksize k;
- D is no repeated blocks; and
- For each t-subset T of P there are exactly λ blocks of B incident with all the points of T.

Such a design is usually referred to as a t-(v, k, λ) design. If b denotes the cardinality of B, a t-design with $v = b$ and $t \geq 2$ is called a *symmetric design*.

In the following text, operations described for incidence structures also apply to near-linear spaces, linear spaces, and block designs unless otherwise noted.

4.4.2 Construction of Incidence Structures

An incidence structure is usually constructed by giving the parameters together with a list of blocks; an alternative is to give the incidence matrix. Functions are provided to construct incidence structures from graphs and codes. Given an incidence structure, other incidence structures may be derived using a number of standard operations: complement, dual, contract at a point of block, residual with respect to a point or block, and restriction to a specified set of points. Given two or more incidence structures their sum or union may be constructed.

Various tests are available for determining basic properties of an incidence structure D. These include simple, self-dual, uniform, balanced, complete, symmetric, point regular, block regular,

near-linear space, linear space, design, and Steiner. Testing t-balance can be expensive and so, as well as a brute force algorithm, two alternatives are provided. One alternative algorithm first computes the automorphism group A of D and then uses the A-orbits of t-sets of elements chosen from P to reduce the amount of testing required. Another alternative is a brute force algorithm optimised for $t \geq 4$.

4.4.3 Difference Sets

Let G be a group of order v and let k and λ be positive integers such that $1 < k < v$. A (v, k, λ) *difference set* for G is a set D of k group elements such that the set

$$\{\ gh^{-1} : g, h \in D \mid g \neq h\ \}$$

contains every non-identity element of G exactly λ times.

A function is provided to construct difference sets belonging to the families `"Q"`, `"H6"`, `"T"`, `"B"`, `"B0"`, `"O"`, `"O0"`, or `"W4"` as listed in Marshall Hall's book, *Combinatorial Theory*. Another function is provided for constructing the Singer difference set corresponding to a hyperplane of $\mathrm{PG}(n, q)$.

Given a difference set whose elements belong to an integer residue class ring, a finite abelian group, or a finite group, a function is provided that constructs the corresponding symmetric design.

4.4.4 Resolutions and Parallelisms

Let D be an incidence structure with v points. A *resolution* of D is a partition of the blocks of D into classes C_i such that each class is a 1-design with v points and index λ. The positive integer λ is called the *index* of the resolution. A resolution with $\lambda = 1$ is called a *parallelism*. In this case the classes C_i are called *parallel classes*.

An incidence structure can be tested to determine whether of not it has a resolution of any index or a resolution having a particular index. It is also possible to obtain all resolutions. Given a uniform incidence structure a function is provided to search for a parallelism. A variant will return all parallelisms. It is also possible to determine the parallel classes. The basic algorithm employs a backtrack search which is efficient when the design has a parallelism. If it is suspected that there are no parallelisms an alternative algorithm that works by searching for cliques in a graph will be much faster. For example, in the case of a non-parallelisable design with 272 blocks, the clique method was an order of magnitude faster than the backtrack method (0.02 seconds as compared to 0.2 seconds).

4.4.5 Automorphism Group and Isomorphism

A particularly important function for incidence structures is the construction of its automorphism group. Magma uses a very efficient backtrack search algorithm developed by J. Leon which employs his partition refinement technique. A variation of this algorithm is used to test pairs of incidence structures for isomorphism.

The automorphism group of an incidence structure D is returned as a permutation group G acting on a set which is determined by properties of D. The Magma G-set machinery may be used to obtain the action of G on different sets associated with D. The two most important actions are those acting on the point set and block set. As the group G does not directly act on D, elements of G that are required to act on D are first cast as mappings of D into itself.

4.4.6 Applications

The concept of a block design arises in many different areas of discrete mathematics and coding theory as well as in other areas such as group theory. Consequently, the Magma machinery for incidence structures has been heavily used across a wide range of areas. Frequently, in a project that involves calculations with designs, the design aspect forms a small part of a project based in a different area altogether. However, there are many projects known to us that employed the Magma machinery to gain insight into problems in design theory.

An example of a project that has used the design machinery extensively over the 20 year life of Magma is the long-running project of E. Assmus and J. Key in which they study the application of algebraic coding theory to the analysis and construction of designs. Details may be found in the book *Designs and their Codes* by Assmus and Key.

4.5 Finite Planes

4.5.1 Introduction

Let $D = (P, L)$ be a pair of sets such that

- P is a finite set, the elements of which are called *points*; and
- B is a set of subsets of P, the elements of which are called *lines*.

A *finite projective plane* is a point-line pair D such that

- There is a unique line containing any two distinct points;
- Any two distinct lines intersect in a distinct point; and
- There exist four points, no three of which are collinear.

A finite projective plane is a symmetric 2–$(n^2 + n + 1,\ n + 1,\ 1)$ design where the lines of the plane become the blocks of the design. A *finite affine plane* is a point-line pair D such that

- There is a unique line containing any two distinct points;
- A point p not on a line l is on precisely one line that does not intersect l; and
- There exists a set of three non-collinear points. A finite affine plane can also be regarded as a 2–$(n^2,\ n,\ 1)$ design.

4.5.2 Construction of Finite Planes

The basic method of defining a finite plane is to specify its point set and the line set. A classical finite projective plane can be defined by specifying a 3-dimensional vector space V over a finite field. The points are the 1-dimensional subspaces while the lines are the 2-dimensional subspaces. A classical finite affine plane is defined by specifying a 2-dimensional vector space V over a finite field. The points are the 1-dimensional subspaces while the lines are the cosets of the 1-dimensional subspaces.

A constructor is provided to create subplanes of projective or affine planes that contain a specified set of points. In the case of classical planes, subfield subplanes may be created. Given a specified line l in a projective plane π, the affine plane obtained from π by removing l can be constructed. Conversely, given an affine plane π, the projective completion of π can be constructed.

A test for a plane being Desarguesian is provided. In addition, in the case of projective planes, a function for testing whether the plane is self-dual is available.

4.5.3 Arcs, Conics, Parallelisms, and Unitals

A *k-arc* in a projective or affine plane P is a set of k points of P, no three of which are collinear. A k-arc is *complete* if it cannot be extended to a $(k+1)$-arc by the addition of another point. A *tangent* to an arc A is a line which meets A exactly once; a *secant* is a line which meets A exactly twice; and a *passant*, or *external line*, is a line which does not meet A at all. A *unital* in the classical projective plane $PG_2(q^2)$ is a set of q^3+1 points such that every line meeting two of these points meets exactly $q+1$ of them. Functions are provided to find arcs, conics, parallelisms, and unitals.

- Determine parallel classes.
- Construct a k-arc, or complete k-arc.
- The set of tangent (secant, passant) lines for a given arc.
- Given a set S of five points in general position belonging to a classical projective plane P of order $n > 3$, construct the unique conic that passes through them.
- Find the knot of a conic.
- Find the interior (exterior) points of a conic.
- Test whether a set of points in a projective plane $PG_2(q^2)$ is a unital.
- Find the set of tangents to the points of a unital.
- Find the set of intersections of the unital U with the tangents to U which pass through the point p (feet of the unital).

4.5.4 The Collineation Group

A particularly important calculation for finite planes is the determination of its automorphism group. Magma uses a very efficient backtrack search algorithm developed by J. Leon which employs his partition refinement technique. A variation of this algorithm is used to test pairs of planes for isomorphism.

The automorphism group of an incidence structure D is returned as a permutation group G acting on a set which is determined by properties of D. The Magma G-set machinery may be used to obtain the action of G on different sets associated with D. The two most important actions are those acting on the point set and line set. As the group G does not directly act on D, elements of G that are required to act on D are first cast as mappings of D into itself.

Let p be a point and l a line of a projective plane P. A *(p,l)-central collineation* is a collineation α of P which fixes l pointwise and p linewise. The line l is called the *axis* of α and the point p is called the *centre* of α. A function is provided to find the group G of (p,l)-central collineations of a projective plane. In addition, functions are provided to find the central collineation group with specified axis or centre.

4.5.5 Translation Planes

A small group of functions is provided for constructing translation planes by derivation. Let q be a power of a prime. Given the projective plane $PG_2(q^2)$, a function is provided for finding a Baer subplane of order q. An affine plane may be constructed from $PG_2(q^2)$ by derivation with respect to a Baer subplane. Finally, given the projective plane $PG_2(q)$, where q is a power of 2, a function constructs a translation plane by derivation with respect to an oval.

4.5.6 Near-Field Planes

Given a finite near-field N, there is an *affine* plane A with point set $N \times N$ and lines given by the equations $y = xm + b$, $x = c$. Let P be the corresponding projective plane, obtained from A by adjoining a line L_∞ called the *line at infinity*. By carefully labelling the points of A and the

line L_∞ with triples of elements of N, every collineation of A extends to a collineation of P. A function is provided to construct this plane.

In 1957, Hughes discovered a class of finite projective planes constructed from the Dickson near-fields which have rank 2 over their kernel. Neither these planes nor their duals are translation planes and therefore they cannot be obtained by the coordinatisation method of the previous paragraph. Hughes' methods required the kernel to be central but in 1960 the construction was generalised by Rosati to include the Zassenhaus near-fields. For simplicity, the term 'Hughes plane' will include both Hughes planes and generalised Hughes planes. A function is provided which constructs the Hughes plane given a near-field having rank 2 over its kernel.

As an example, we can construct four distinct projective planes of order 49. These are the Desarguesian plane $PG_2(49)$, the non-Desarguesian near-field plane over the Dickson near-field of order 49, the non-Desarguesian near-field plane over the Zassenhaus near-field of order 49, and finally the Hughes plane over the Dickson near-field of order 49. The orders of their respective collineation groups are 66437613849600, 22127616, 33191424, and 11261376. Thus, the four planes are distinct from one another. The computation took 5 seconds.

4.5.7 Applications

Finite geometry is an active area of research. Many of the major centres for finite geometry use Magma. R. Baker and G. Ebert have studied questions arising in $PG_n(q)$ for $n \geq 3$ as well as in planes. E. Assmus and J. Key did many computations with finite planes in the course of their work applying coding theory to the analysis and construction of designs. Many workers have used Magma in their study of blocking sets and spreads in finite planes.

4.6 Hadamard Matrices

4.6.1 Introduction

A Hadamard matrix is a square matrix, all of whose entries are ± 1, such that every pair of rows differ in exactly half the number of places (and similarly, every pair of columns differ in exactly half the number of places). Given a Hadamard matrix it is always possible to find an equivalent matrix (a *normalised matrix*) in which the first row and column consist entirely of $+1$'s. Given a normalised Hadamard matrix, a symmetric $(4t-1, 2t-1, t-1)$ design may be constructed, and conversely.

4.6.2 Equivalence and Automorphism Groups

Two Hadamard matrices are said to be equivalent if one can be transformed into the other by performing a sequence of row swaps, column swaps, row negations, or column negations. Determining whether two Hadamard matrices are equivalent is a hard problem, due to the regularity of the structure. Magma determines this equivalence by first transforming the matrices to equivalent graph as described by Brouwer *et al.*, and then applying the `nauty` tools to determine if the resulting graphs are isomorphic. A variant of this approach can be used to get a canonical form for a Hadamard matrix. A further modification of the algorithm is used to compute the automorphism group.

Magma also implements the 4-profile, an invariant based on considering sets of four rows at a time of the matrix. Since equivalent matrices have equal 4-profiles, this can sometimes be used as a fast test for inequivalence before commencing the more expensive full tests. More recently, Magma has installed the alternative graph isomorphism code of `Traces` for the Hadamard computations. `Traces` functions much like `nauty` but with a different choice of strategy that is sometimes much

more efficient. Testing has shown that for larger Hadamard matrices the time savings of using `Traces` can be considerable.

4.6.3 Databases

Magma includes a large database of Hadamard matrices, with at least one example in all possible dimensions up to 256. It contains complete enumerations up to dimension 28, and a decent number of representatives up to dimension 72 (note that the dimension of a Hadamard matrix must be 1, 2, or a multiple of 4). A recent project has produced 29,613 Hadamard matrices of order 256 from difference sets constructed in groups of order 256. These will be added to the database in the near future.

4.6.4 Applications

The Hadamard matrix machinery has been used in a number of applications. The main application known to us is in equivalence testing by groups constructing new Hadamard matrices. Another application has been to the construction of codes and other combinatorial objects from Hadamard matrices.

Chapter 5

Commutative Algebra

5.1 Gröbner bases

5.1.1 Introduction

A Gröbner basis of an ideal $I = \langle f_1, \ldots, f_k \rangle$ of the multivariate polynomial ring $R = K[x_1, \ldots, x_n]$ (where K is a field or Euclidean ring) is a canonical generating set of I which yields information from which many important properties of I can be determined. For example, if G is a Gröbner basis of I, then one can easily test whether an arbitrary polynomial $f \in R$ is in I, or one can instantly determine from G whether the underlying system of polynomial equations has a solution (and also the number of solutions over an algebraic closure when the number is finite); if G is also written with respect to a lexicographical order (see below), then one can quickly read off all solutions to the system of equations. The Gröbner basis machinery in Magma was designed and implemented by A. Steel.

The worst-case theoretical complexity for computing a Gröbner basis is doubly exponential in the maximum degree of the input polynomials; this holds for the space usage as well as the time taken. Thus, it may not be feasible to construct a Gröbner basis within the memory of any computer, even ignoring the time taken. However, there is a vast range of ideals for which computing a Gröbner basis is quite feasible. Much of Algebraic and Arithmetic Geometry depends heavily on Gröbner bases for non-trivial operations on ideals and modules of multivariate polynomial rings, but major applications occur in a huge number of disciplines.

Magma is widely recognised by workers in the field as having the broadest and fastest Gröbner Basis machinery in general for any CAS. Magma's package for Gröbner bases is doubly generic in the following sense:

- All the Gröbner basis machinery works generically for ideals defined over an arbitrary field or Euclidean ring (as is the case with Magma's linear algebra). Apart from the generic versions, highly optimised versions have been developed for working with ideals defined over specific types of field which arise frequently in practice such as small finite fields and $\mathbb{Q}$.
- All of the Gröbner basis machinery works generically for a wide range of machine representations of monomials. These includes not only several forms of packed representations for the standard commutative monomials (including sparse and dense variants, selected according to the number of variables), but special representations such as bit-vector representations for system solving over GF(2) and for non-commutative settings, such as exterior algebras, solvable algebras, and general non-commutative algebras.

5.1.2 The Faugere F_4 algorithm

The traditional algorithm for computing a Gröbner basis is *Buchberger's algorithm*, published by Buchberger in 1965. In 1999, J.-C. Faugère introduced his so-called F_4 algorithm which is generally more efficient than the Buchberger algorithm. The F_4 algorithm basically consists of a series of steps in each of which a hash table of monomials is built up and a related matrix is constructed in parallel (with columns labelled by the hash table entries) and then the matrix is reduced by Gaussian elimination; this avoids expensive repeated operations on common monomials which dominate the computation when the Buchberger algorithm is used.

The matrices constructed in the F_4 algorithm are typically globally sparse but may be extremely large and have very dense patches (e.g., for the 80-variable HFE challenge below, the largest matrix is $293,287 \times 1,666,981$ and many rows have hundreds of thousands of non-zero entries). Traditional linear algebra algorithms are designed for either sparse matrices or for dense matrices and are not applicable to this situation, so the reduction of the matrix requires highly specialist code which works row by row with the mixture of sparse and dense rows. Thus Magma's implementation has a special representation for these kinds of matrices (only used in the F_4 algorithm) and based on this, the algorithm is highly optimised for several different types of finite fields and also includes modular algorithms for the rational field and number fields.

A major new variant of the F_4 algorithm was developed by A. Steel in 2013 and is applicable to polynomial systems with rather dense inputs; this is a common case for attacking certain types of cryptosystem and includes the HFE and MQQ cryptosystems and also Minrank systems. This variant makes Magma even faster than before for several types of polynomial systems.

All the timings given below are for runs on a 64-bit Dell workstation with a 4-core Intel Core i7-7700 CPU (3.60 GHz); timings are given for runs in both the default single core case and the parallel 4-core case.

5.1.3 Gröbner Bases over Finite Fields and Algebraic Attacks

The case in which the coefficient ring is a finite field is of considerable interest. For computing such Gröbner bases, Magma has a variety of optimal representations for the linear algebra phase of the F_4 algorithm. The critical case of GF(p), where p is a moderately small prime, is handled by exact floating point operations so that the primitive vector operations are highly optimised. This approach is also the basis of the characteristic zero algorithms below. For example, Magma V2.27 computes:

- The Gröbner basis of the Cyclic-9 roots ideal over GF(32003) in 10.2 secs (4 cores) or in 29.3 secs (1 core);
- The Gröbner basis of the Cyclic-10 roots ideal over GF(32003) in 601.2 secs (4 cores) or in 1957.1 secs (1 core).

Algebraic attacks on cryptosystems have become of increasing importance and basically involve solving large multivariate polynomial systems over a small finite field, particularly GF(2). Magma contains special extensions of the F_4 algorithm for solving large multivariate polynomial systems over GF(2). This includes a special bit vector representation for monomials ($x^2 = x$ for each variable in this context) together with an optimal numbering scheme for monomials. (A hash table takes too much memory when there is a huge number of variables.)

For example, Magma apparently remains the only publically-available software which is able to solve Patarin's first Hidden Field Equations (HFE) Crypto Challenge. The input for this system consists of 80 quadratic polynomials in 80 variables over GF(2) and Magma V2.27 solves the challenge in 28.2 secs (4 cores) or in 85.1 secs (1 core) with the dense variant.

5.1.4 Gröbner Bases over Characteristic Zero Fields

Since knowledge of a Gröbner basis is a popular approach to solving systems of polynomial equations, being able to compute Gröbner bases in characteristic zero is required by a great number of applications in areas of mathematics such as commutative algebra, algebraic geometry, and arithmetic geometry. For ideals over $\mathbb{Q}$, Magma contains a highly optimised modular algorithm for the linear algebra phase of the F_4 algorithm which can compute large Gröbner bases with large rational coefficients in reasonable time. For example, Magma V2.27 computes:

- The Gröbner basis of the Cyclic-9 roots ideal over $\mathbb{Q}$ in 144.3 secs (4 cores) or in 286.1 secs (1 core), using 750 MB for either computation;
- The Gröbner basis of the Cyclic-10 roots ideal over $\mathbb{Q}$ in 5.2 hours (4 cores) or in 16.1 hours (1 core), using 46 GB for either computation.

For computing Gröbner bases of ideals over a number field $K = Q(\alpha)$, Magma also contains a 'double' modular algorithm for the linear algebra phase of the F_4 algorithm, which works by mapping K into several finite fields and using both interpolation and Chinese Remaindering. This modular algorithm is often dramatically faster than the standard (non-modular) algorithm since it avoids the intermediate coefficient blowup which quickly arises in any non-trivial computation with number fields.

5.2 Multivariate Polynomial Rings and Their Ideals

5.2.1 Polynomial Ideals and Varieties

In this section, functions for computing with the ideals I of a multivariate polynomial ring $R = K[x_1, \ldots, x_n]$ will be discussed. Since it is assumed that the base ring K is a field or Euclidean domain, all polynomial rings R are *Noetherian*, meaning that any ideal I is finitely generated. The notation $<a_1, \ldots, a_m>$ is used to denote the ideal I generated by this collection of elements.

The ideal I can be thought of as coming from the polynomial system $a_1, \ldots, a_m$ and, conversely, any polynomial system generates an ideal I. The significance of the ideal is roughly as follows. The *variety* $V(a_1, \ldots, a_m)$ of a polynomial system is the set of n-tuples $(u_1, \ldots, u_n)$ over the algebraic closure of K at which all a_i simultaneously vanish. The polynomials that vanish on V are precisely those in the *radical* of I, $rad(I)$. The radical is the ideal formed by all polynomials f such that a power $f^k \in I$ for some $k \geq 1$. Thus, there is a 1–1 correspondence between *algebraic sets* (sets of simultaneous solutions/zeroes of sets of polynomials), and radical ideals in the polynomial ring. The variety $V(I) = V(a_1, \ldots, a_m)$ is a (reduced) affine algebraic *scheme* in the terminology of algebraic geometry.

There will be many possible distinct finite bases for an ideal I. Computationally, the most important bases are the Gröbner bases. There is a unique Gröbner basis for each total ordering satisfying a number of properties of the monomials of R. An appropriate Gröbner basis is used in the algorithms for the implementation of all of the non-trivial functionality on polynomial ideals.

An essential basic function is the one that computes the *dimension* of an ideal (strictly speaking, the Krull dimension of the quotient ring R/I) which is the Zariski dimension of the associated algebraic set $V(I)$. The ideal I has dimension 0 precisely when $V(I)$ consists of a finite set of points. If I can be generated by r elements, its dimension is at least $n - r$ but equality holds only for the special case of complete intersections. For example, when $n = 3$, Magma computes that the dimension of $\langle x_1^2 + x_2^2, x_2^3 + x_3^3 \rangle$ is 1 but that of $\langle x_2^2 - x_1 x_3, x_1 - x_1^3 - x_2 x_3^2, x_2 - x_1^2 x_2 - x_3^3 \rangle$ is also 1 but the second ideal cannot be generated by two elements.

Two of the most important operations are the computation of the *colon ideal* $(I : J)$ of two ideals

and the *saturation* $(I : J^\infty)$. These perform a type of ideal division. The polynomial $f \in (I : J)$ if and only if $f * J \subseteq I$ and $f \in (I : J^\infty)$ if and only if there is a power of f, f^k with $k \geq 1$ such that $f^k * J \subseteq I$. Geometrically, these correspond to difference operations on varieties. The variety $V((I : J^\infty))$, for instance, is the *Zariski closure* of the difference set $V(I)/V(J)$. As a simple example with $n = 2$ consider $I = <x_1^3 - x_1 - x_2^2, x_1^2 + x_1x_2 - 1>$ for which $V(I)$ is the finite set of points on the intersection of an elliptic curve and a conic in the affine plane. We compute the saturation of I by $<x_2>$ to get $<x_1x_2 + x_1 + x_2^2, x_1^2 + x_2, x_2 - x_1x_2 + 1>$. These new equations define the set of points in $V(I)$ with $x_2 \neq 0$ (two points have been removed).

Elimination ideals may also be computed. These are the ideals formed by intersecting I with the polynomial subring of R generated by a subset of the variables. This gives the polynomials in I that are polynomials in the chosen subset of the variables and can be thought of as resulting from the elimination of a number of variables from the original system of polynomial equations. Resultants are also available, which can be used to get a partial result when eliminating one variable.

Amongst many other things, elimination ideals are used to compute images of varieties under polynomial maps. For example, to find the image of the affine line under the map $t \to (t+t^2, t^3 - t, t^3 - t^2 + 1)$ into 3-space, we can eliminate t from the ideal $<x - t - t^2, y - t^3 + t, z - t^3 + t^2 - 1>$ to get the ideal of defining equations of the image:

$$\langle xz - x - 2y^2 + 3yz - 3y - z^2 + 1, xy - 3y^2 + 5yz - 5y - 2z^2 + 2, x^2 - 2x - y^2 + 2yz - 4y - z^2 + 1\rangle$$

5.2.2 Prime Ideals and Primary Decomposition

From this point of the section onwards, most of the functions only apply when K is a field and this is now assumed to be the case.

The radical $rad(I)$ of an ideal I gives the full set of polynomials vanishing on $V(I)$. It can be essential to compute this in order to determine $\#V(I)$ or the degree of $V(I)$ in the positive dimensional case, and also to compare whether 2 algebraic sets are equal.

More generally, there are a number of important decompositions of an ideal as an intersection of ideals of a particular type. Every ideal has a *primary* decomposition into primary ideals, which are generalisations of prime-power ideals. *Radical* decomposition decomposes the radical of the ideal into an intersection of prime ideals. These define the irreducible components of $V(I)$. *Equidimensional* decomposition is an easier and faster decomposition into an intersection of ideals each of which is the intersection of all primary components of I of a given dimension. This can be computed without computing the individual primary factors and is useful in a number of contexts.

Magma can compute all of these decompositions. They are all heavy computations but we have sped things up significantly in recent years by using homogenisation techniques.

In the particular case of dimension zero, the following may also be computed:

- All of the (finitely-many) points in $V(I)$ defined over a given extension L of the base field K or all of the points over an algebraically-closed field.
- *Triangular* decomposition of I. This is generally faster than radical/primary decomposition but the triangular factors allow the points of $V(I)$ to be quickly recovered.

5.2.3 Homogeneity and Graded Modules

In geometry and other mathematical applications of commutative algebra, homogeneity often plays a key role. A polynomial f is homogeneous with respect to a (positive) integer weighting $(w_1, \ldots, w_n)$ of the variables if each monomial that occurs in f has the same weight. The standard

case is where each variable has weight 1, which is traditionally what is referred to as homogeneity. The term *weighted-homogeneous* is often used used when referring to properties and functions that apply to cases of more general weightings. An ideal is (weighted)-homogeneous if it can be generated by homogeneous polynomials. Modules with the homogeneity property are referred to as graded and functionality for them will be described in their section.

There is a filtration and finiteness associated to homogeneous ideals and graded modules that is theoretically and computationally very important. The ideal or module breaks up into a direct sum of pieces of given weight (or grading) and each piece is finite-dimensional as a vector space over the base field K. Magma has an option for Hilbert-driven Gröbner basis computation. This allows some the computation to proceed by graded piece with more efficient criteria for termination.

The most fundamental objects associated to a graded module are the *Hilbert series* and *Hilbert polynomial* (the latter only for the standard weighting). These are computed in Magma using a fast Gröbner-based algorithm. They are used to determine important geometric invariants of the algebraic scheme in (weighted) projective space defined by I, such as the degree and arithmetic genus. For instance, the Hilbert polynomial of the ideal defining a Veronese surface X in 5-dimensional projective space is computed to be $P(t) = 2t^2 + 3t + 1$. This shows that the degree is $4 = 2 \times 2$ (twice the leading coefficient) and the arithmetic genus is $0 = P(0) - 1$.

For many operations on ideals it is important to try to minimise the number of generators that define the ideal I. In general, it is a very hard problem to determine a genuinely minimal set of generators of I. However, it is possible in the weighted-homogeneous case and in that case a *minimal basis* can be computed in Magma.

5.2.4 Normalisation, Regular Sequences, and Rees Algebras

The following important computational tools for a polynomial ideal I of dimension d or its quotient algebra R/I are also available in Magma:

- *Noether normalisation of I.* If d is the dimension of I, this is a set of d algebraically-independent polynomials $y_1, \ldots, y_d$ that are independent for I — if $S = K[y_1, \ldots, y_d]$ then $S \to R/I$ is injective — and such that variables x_i mod I are *integral* over S. A *linear* Noether normalisation is computed except in some exceptional cases over small finite fields. This is where each y_i is a K-linear sum of the x_j. They have many computational applications and are often used to reduce affine algebra problems to ones over a plain polynomial ring.
- *Normalisation of a reduced affine algebra R/I.* The normalisation is the integral closure of R/I in its total ring of fractions. It is a finitely-generated algebra and is returned in the form S/J where S is a polynomial ring over K with additional variables and J an ideal of S. It is based on the standard algorithm of de Jong but also uses Magma's algebraic function field machinery for a faster, more efficient algorithm in the case that I has dimension 1.
- *Regular sequences in I.* These are sequences of d polynomials $f_1, \ldots, f_d$ in I that generate a *complete intersection* subideal of dimension d. They are important for depth-related problems. The algorithm used is that of Eisenbud and Sturmfels.
- *Rees algebras.* These important algebras occur frequently in desingularisation routines and other places in birational geometry. They represent the *blow-up* of an ideal I/J in an affine algebra R/J where $J \subseteq I$. They are computed explicitly as finitely-generated algebras.

A nice example of normalisation is for the principal ideal I in $R = \mathbb{Q}[x, y]$ generated by the polynomial $(x - y)x(x^2 + y)^3 - y^3(x^3 + xy - y^2)$. This is a singular, affine model of the the modular curve $X_1(11)$ that arises naturally. Basic normalisation of I with the function field method is very fast (about 0.03s on our current machines) and returns a reasonable presentation of the normalisation of R/I as a 5-variable algebra which can be tidied up to give 3-variables by

some straightforward substitution. The normalisation using the function field method with the option of simplification by Riemann–Roch filtration takes 0.11s but returns the result in the form R/J (so just using the original 2 variables) where J is generated by a single degree 3 polynomial which is an affine form of an elliptic curve equation for $X_1(11)$. This is much faster than de Jong's method that can take several hours.

5.3 Invariant Theory

5.3.1 Invariants of Finite Groups

Invariant theory here refers to the situation where there is a finite group G acting on a polynomial ring $R = k[x_1, \ldots, x_n]$ through a linear action on the variables and we wish to explicitly compute (homogeneous) generators for the finitely-generated subring of invariants R^G. From a group-theoretic point of view, the action of G on the space of linear forms in the x_i is a representation V and R with its G-action represents the direct sum of all of the symmetric powers of this representation. From a geometric point of view, G represents a finite group of automorphisms on a scheme X and computation of the ring of invariants gives the quotient scheme X/G. In fact, the invariant theory package is used for certain special cases by Magma's curve quotient functionality.

The computation of the ring of invariants is in two phases. Firstly, a basis for the *primary* invariants is computed, using an algorithm of Kemper. The primary invariants are a set of n algebraically-independent (homogeneous) polynomials in R^G, which therefore generate a subring R_1 over which R^G is a finite ring extension. Next, *secondary* invariants are computed which give a set of (homogeneous) generators for R^G as an algebra over R_1. Different algorithms are used for this, depending on whether the order of G is coprime to the characteristic of k. The ring R_G is defined as a (graded) k-algebra by the full set of primary and secondary invariants and the ideal of algebraic relations between these invariants. There is a function to compute and return this ideal of relations.

5.3.2 The Molien Series

The Hilbert series of R^G can be found through the computation of the *Molien series* of G. The latter is computable from representation theory (modular in the characteristic p case, when G should be given as a permutation group) before performing the algebraic computation of polynomial invariants. It is also used in the actual algorithm for computing invariants to check that a full set of generators have been found by comparison with the Hilbert series.

5.3.3 Invariants of Linear Reductive Groups

Using quite different algorithms, invariants may be computed for the algebraic linear action of an algebraic subgroup G of the algebraic group of n-by-n matrices GL_n. This algebraic linear representation may be defined by giving explicit matrices for the action with entries as rational functions on the dimension n^2-affine space in which GL_n naturally lies. The algorithm used is Derksen's algorithm, which will not necessarily terminate if G is not a linearly reductive group (when R^G may not even be a finitely-generated k-algebra). This is a computationally-heavy algorithm that may take some time to complete, but there are also functions to compute a k-basis for the finite-dimensional space of invariants of any given degree. This function is generally quite fast when computing the invariants of a specific degree and it may be useful for producing the full ring of invariants in cases where the Hilbert series of R^G can be determined theoretically.

An interesting special case is for SL_2 acting on the polynomial ring with $n = 2g + 3$ for $g \geq 2$, where the linear action is given by the natural action on the space of homogeneous polynomials

of degree $2g + 2$ in two variables. The projective variety defined by the homogeneous ring of invariants gives the (affine) moduli space of algebraic hyperelliptic curves of genus g on removing the hyperplane corresponding to $D = 0$, where D is the discriminant function.

5.4 Modules over Polynomial Rings

5.4.1 Syzygies and Submodules

In this section a module refers to a finitely-generated module. A module can be represented in presentational form as the quotient of a free module $F = R^m$ by a submodule S of F (referred to as the relation module), which is a subgroup of F invariant under the R-action. Modules occur naturally in problems involving multivariate polynomials. An ideal in R can itself be considered as a finitely-generated module over R. In geometry, the *sheaves* of differentials or tangent vectors on an affine scheme are represented by R-modules and those on ordinary or weighted-projective schemes are represented by graded R-modules.

In order to compute effectively with such modules, operations such as element membership, intersection, and computing the presentational form of a submodule are required. A general submodule of F/S is equal to S_1/S, where S_1 is a submodule of F containing S, and it is necessary to be able to find generators and relations for such an object. In particular, it is necessary to compute a basis for the module of R-relations between the generators of any submodule S of F. These are referred to as *syzygies* of S (or rather of its given basis).

The definition of the Gröbner basis for an ideal of R can be adapted so as to allow the computation of Gröbner bases for submodules S of a free module F. Then the Gröbner-based methods used for ideals extend to submodules S. Thus a syzygy can be computed using a standard Gröbner method. Using this approach, functions are provided for computing the intersection of a finite number of submodules of a module, the colon ideal of two submodules (with the natural definition), Hilbert series, Hilbert polynomials, and minimal bases of graded modules. Magma supports a variety of monomial orders for modules, including 'position-over-term' and 'term-over-position' orderings. For computing Gröbner bases of modules, Magma uses an extension of the Faugere F_4 algorithm to handle modules (developed in 2004 by A. Steel); this apparently remains the only implementation in any software of such an extension. A variant of this is also present for computing syzygies directly using the F_4 method.

Basic operations on modules include:

- Annihilator of a module M. This is the ideal I of all elements of R that act as the zero map on M and is computed by a slight variant of the syzygy method.
- Fitting ideals of M. These are important invariants of M that can be defined for a given presentational form of M and then shown to be independent of the presentation chosen.
- Construction of submodules generated by a finite number of elements and quotient modules. Direct sums of modules, R-tensor products and tensor powers.
- Serre twists of a graded module. This is a formal twisting operation that leaves the underlying module alone but shifts the grading.

As an example, we can take the graded module M that represents the locally-free sheaf of differentials on the projective cubic surface X defined by the vanishing of the polynomial $f = x^3 + y^3 + z^3 + t^3$ in $\mathbb{Q}[x, y, z, t]$. The module M has a minimal presentation as a quotient of a free module of rank 6, R^6. We check that the annihilator of M is precisely the ideal $<f>$, as it should be for a locally-free sheaf. We compute the first Fitting ideal of M and check that it lies in $<f>$. Then we compute the second Fitting ideal and its ideal saturation with respect to the maximal ideal of R generated by the 4 variables. The result is the full ideal R. These facts together demonstrate that M defines a locally-free sheaf of rank 2 on the surface X.

5.4.2 Module Homomorphisms

R-linear homomorphisms can be defined between modules in a natural way, and are represented internally by matrices giving an R-linear map between the free modules occurring in the reduced forms of the modules. If modules M and N are graded modules then a homomorphism from M to N can be decomposed uniquely into a sum of homogeneous homomorphisms. A homomorphism is *homogeneous of weight* d if it maps the rth graded piece of M into the $(r+d)$th graded piece for every r. All of the usual homomorphism operations are available. Kernels, cokernels and images can be computed as sub- or quotient modules of the domain or codomain. There are tests for injectivity or surjectivity.

The set of homomorphisms from M to N, $\mathrm{Hom}_R(M,N)$, has a natural structure as an R-module and Magma can compute it as a module along with a map that takes an element of the module to the actual homomorphism that it represents. If M and N are graded, $\mathrm{Hom}_R(M,N)$ is computed as a graded module where the weight of a homogeneous element corresponds to its weight as a homogeneous homomorphism as defined above. Magma's algorithm for computing Hom-modules is again based on an efficient adaptation of the Faugere F_4 algorithm to compute syzygies.

5.4.3 Free Resolutions

One of the most important objects attached to a module over polynomial ring R is a *free resolution* of R. Hilbert proved that if we are careful about the bases we choose and we take the syzygy module of a module and the syzygy module of that and so on then, after at most n steps, the syzygy module will be trivial. Here n is the number of variables of R. In the language of homological algebra, this says that M has a free resolution of length n: There is a long exact sequence of R-modules

$$0 \to F_n \to F_{n-1} \to \ldots \to F_0 \to M \to 0$$

where the F_i are free modules (some of which may be 0). Computation of free resolutions can be performed by iteratively computing syzygy modules but there is a more efficient way using what is known as the La Scala algorithm. Magma computes free resolutions using an extremely efficient implementation of this.

Free resolutions are particularly significant for graded modules, where they capture a number of invariants of the module. More precisely, in the graded case there are *minimal* free resolutions where the matrices representing the maps between the F_i have entries that are all homogeneous of positive weight. In that case, the resolution is essentially unique and the F_i can be given gradings by assigning column weights so that the maps in the resolution are all homogeneous of weight 0. The column weights of the F_i in the graded case form an array traditionally referred to as the *Betti table* of the module. The entries of this array — the Betti numbers — contain much useful information about M.

Magma contains an optimised algorithm to compute a minimal free resolution in the graded case and to compute and return the Betti numbers of the module. This algorithm combines the La Scala resolution algorithm with the Faugere F_4 algorithm and was developed by A. Steel in 2005; it actually involves running k separate instances of the F_4 algorithm at the same time, where k is the length of the resolution. As an example of its efficiency, in February 2014 Richard Stanley asked for the Betti numbers of the 4x4 magic squares (the semigroup algebra of all 4x4 matrices of nonnegative integers with all row and column sums equal; the algebra has 24 generators of degree one and Krull dimension 10). Magma's F_4-based algorithm computes the Betti numbers in only 1700 seconds; this involves reducing 811,620 polynomials of degree up to 20 in 24 variables

at 16 different levels to collect all the syzygies and the resulting Betti numbers are:

1 178 1837 7416 16440 25144 35562 42204 35562 25144 16440 7416 1837 178 1.

Another benchmark involves computing the full free resolution of a tangent developable of a rational normal curve of degree 13; here Magma computes the resolution in 412 seconds, while Macaulay 2 (the fastest competitive software) computes the resolution in about 10 hours.

A related quantity is the *Castelnuevo–Mumford* regularity of a module that Magma also has a function for (the CM regularity is actually a simple function of the Betti numbers). This invariant is important, for example, in the computation of sheaf cohomology and it is used by the cohomology package in Magma.

Free resolutions and functionality for Hom modules allow the computation of the homological bi-functors $Ext^i_R(M,N)$ and $Tor^i_R(M,N)$ for pairs of modules, M and N and any $i \geq 0$ (they are all zero for $i > n$). There are user functions to compute these for any (i, M, N).

Chapter 6

Commutative Rings

6.1 Introduction to Commutative Rings

Commutative rings are one of the most fundamental structures in algebra. In particular, the integers, the rational numbers, the real field and the complex field are all commutative rings. Many other objects such as polynomial rings, matrix rings, vector spaces, and modules are constructed from these.

Here only the principal fields supported will be noted. The field of rational numbers $\mathbb{Q}$ is the starting point for constructing a large number of commutative rings, the most important of which is the ring of integers $\mathbb{Z}$. Another basic field is a finite field $\mathbb{F}_q$, where $q = p^k$ for some prime p.

Global fields comprise two families of fields:

- A *number field* is a finite extension K of $\mathbb{Q}$. The ring of integers $\mathcal{O}_K$ of a number field K is a Dedekind domain and is the main focus in the study of number fields.
- A *global function field* is a finite extension of the field $\mathbb{F}_q(t)$; that is, the field of rational functions over the finite field $\mathbb{F}_q$.

A great deal of machinery has been constructed for working with global fields. This has been developed jointly by the KANT group (TU, Berlin) and the Magma group. Magma is the only widely used system that provides machinery for global function fields, which are of great importance in the study of the arithmetic properties of curves. In fact the machinery for function fields also allows function fields that are extensions of the rational function field $\mathbb{Q}(t)$.

The second important class of fields are the *local fields* which are completions of the global fields:

- The field of real numbers $\mathbb{R}$ and the field of complex numbers $\mathbb{C}$.
- The field of p-adic numbers $\mathbb{Q}_p$, p a prime, and their finite extensions.
- The field of formal Laurent series $\mathbb{F}_q((t))$ over a finite field $\mathbb{F}_q$.

Many calculations are assumed to take place in the algebraic closure of a field. Mostly, it is possible to choose a suitable finite extension at the outset and so there is no need to define the algebraic closure. However, for cases where this is not possible, the formal algebraic closures of $\mathbb{Q}$ and $\mathbb{F}_q$ can be defined. Such fields are represented by polynomials defining the generating elements of the current approximation to the closure and most operations are performed using Gröbner basis techniques.

In this chapter only the integers, finite fields, the real and complex, fields and polynomial rings are considered. Information about global and local fields can be found under *Number Theory*. In the final section an attempt is made to summarise the commutative rings supported in Magma by classifying them according to the axioms they satisfy.

6.2 Integer Rings

6.2.1 Introduction

The Magma tools for working with the integers are based on very fast integer arithmetic. Basic arithmetic is performed mainly by Magma code developed by A. Steel in 1995, with GMP (The GNU Multiple Precision Arithmetic Library) linked in and used for some base cases of multiplication, division, and GCD. The algorithms used are all asympotically fast.

The other key tasks when computing with the integers are to be able to factor integers, including large numbers, and also to establish that an integer is a prime. The latter breaks up into two separate problems: fast probabilistic primality testing and primality proving (where a certificate is provided). All three problems arise frequently (often for reasonably large integers) in algorithms in all areas of the system.

6.2.2 Primality

Primality testing/proving is hard coded for reasonably small integers (it has been established by brute force computation that a certain procedure suffices). For other integers, the default probabilistic test consists of twenty iterations of the Miller–Rabin test. When a proof is desired, the probabilistic test is followed by invoking an implementation by Morain of the elliptic curve primality proving (ECPP) method. A primality certificate can be obtained from the same routine in terse form, and then another function can be used to translate this into a human-readable mathematical proof. An ingredient in the ECPP algorithm is a table of precomputed "Weber" polynomials related to modular curves. The Magma routine uses Morain's 1998 extended table.

6.2.3 Factorisation

Suppose n is a square-free integer whose factorisation is sought. The algorithms for integer factorisation in Magma fall into two families. The first family is characterised by the fact that the runtime (for extracting one factor) depends upon the size of the smallest prime factor so given a large n they are used to quickly locate small factors. The algorithms of this type available in Magma are trial division, table-driven factorisation of integers of the form $b^k \pm 1$, Pollard's rho and $p-1$ algorithms, Shanks' SQUFOF algorithm, and Lenstra's elliptic curve method (ECM). The second group comprises algorithms whose running time depends solely on the size of n. The Multiple Polynomial Quadratic Sieve (MPQS) and the Number Field Sieve (NFS) are the representatives of this family installed in Magma.

Magma's automated factorisation routine, by default, uses the following techniques controlled by parameters depending on the size of the input: trial division, checking for Cunningham factors, SQUFOF, Pollard methods, elliptic curve factorisation, and, lastly, the multiple polynomial quadratic sieve. Zimmerman's GMP-ECM code provides an implementation of the ECM and the Pollard methods. The MPQS code was designed and implemented by A. Lenstra. The number field sieve in Magma was implemented by S. Contini (Magma). Both of these sieves are able to exploit multiple cores.

Factorisation of integers of the form $n = b^k \pm 1$, where b and k are not too big, uses R. Brent's `factor` algorithm, which employs a combination of table lookups and attempts at "algebraic" factorisation (Aurifeuillian techniques). The tables contain 385,251 factors f of numbers $b^k \pm 1$, where $b < 10000$, $k < 10000$, and $f > 109$. For $b \leq 12$ these are mainly from the Cunningham tables (for k up to 1200) while for $13 \leq b \leq 99$ they are mainly from the Brent–Montgomery–te Riele extension of the Cunningham tables. For more information consult Richard Brent's website. The ability to factor integers of the form $b^k - 1$ is of considerable importance when working with the multiplicative group of a finite field. For instance, it allows much faster computation of the

order of a non-zero element of the field.

6.2.4 The Ring $\mathbb{Z}/N\mathbb{Z}$

The ring $\mathbb{Z}/N\mathbb{Z}$ (for composite N) is a fundamental structure which is the basis for some nontrivial (computational) problems. Many algorithms require the calculation of square roots modulo N or a primitive root mod N and Magma has very efficient algorithms for these tasks. Dirichlet characters modulo N are essentially characters on the multiplicative group of $\mathbb{Z}/N\mathbb{Z}$. They are basic objects used in constructions in all areas of number theory, from elementary problems to complicated objects such as L-functions and modular forms. Magma contains a well-developed package for working with Dirichlet characters.

6.2.5 Solving Conics

One of the simplest Diophantine equations encountered is Pell's equation, where it is desired to find integer solutions of the equation $x^2 - dy^2 = 1$. It is well known that Pell's equation can be solved by calculating the continued fraction of $\sqrt{d}$ and Magma can perform the necessary continued fraction calculations. Alternatively, it can be solved by calculating units in the ring $\mathbb{Z}[\sqrt{d}]$ as discussed in the section on Algebraic Number Theory. Another Diophantine problem is to find an integer solution to the conic equation $Ax^2 + By^2 + Cz^2 = 0$. Note that finding one solution yields a parametrisation of all solutions.

As it happens, conics are particularly important Diophantine equations from the point of computational algebra generally. For example, solving conics is a necessary step in some algorithms for decomposing algebras and constructing group representations. Simon's algorithm for solving conics with integer or rational coefficients has been implemented by M. Watkins (Magma), as has Simon's algorithm for the more general problem of finding an isotropic subspace for a quadratic form in more variables. Conics over number fields require a different algorithm, which is discussed in the section on Algebraic Number Theory.

6.3 Finite Fields

6.3.1 Introduction

A large amount of effort has been put into producing fast tools for computing with finite fields as well as with matrices and polynomials over finite fields. Thus, numerous techniques are employed to achieve fast arithmetic in very large finite fields, such as characteristic 2 fields of degree 10,000 upwards.

Linear algebra over finite fields is highly optimised, with a different scheme being used for packing the components of a $GF(q)$-vector used for each field of cardinality up to 8 so as to achieve fast parallel row reduction. For field cardinalities greater than 8, different vector packing schemes are employed for different ranges of possible cardinalities.

A range of methods are available for solving the discrete log problem. At present these concentrate on small characteristics but future work will provide faster methods for medium and large characteristics. A novel feature is the provision of databases containing the logarithms of factor bases for finite fields $GF(p^n)$, where p is small.

In this section, K will denote the finite field $GF(p^n)$, where p is a prime.

6.3.2 Representation and Basic Arithmetic

A variety of representations are used for finite fields in Magma, depending on the characteristic and degree. This ensures that operations are as fast as possible (not just for basic arithmetic but

also for in polynomial algebra and linear algebra over finite fields) while saving memory for small fields.

For a prime field $GF(p)$, a packed representation is used for small p and Magma's large integer package is used for large p. For an extension field $K = GF(q)$, where $q = p^d$, $d > 1$, a variety of representations are used:

- If $q \leq 2^{20}$ the Zech logarithm representation is used, where elements are stored as logarithms and addition/subtraction is done via lookup table taking $O(q)$ space. This is typically much faster than a polynomial representation since one does not need a separate memory block for each element and the lookup table is very fast to access.
- A polynomial representation, which uses the asymptotically-fast univariate polynomial operations over the prime field $GF(p)$; this is particularly efficient in very high degree.
- A 'hybrid' polynomial representation, which internally rewrites the field K over a subfield S with the Zech representation which is as large as possible. Then elements of K can be written as polynomials over S of degree lower than d so the polynomial arithmetic is faster. This representation is only applicable when d has suitable divisors and depends heavily on an automatic compatibility scheme (see below) to manage the mappings between the different representations in the background, but all of this is automated and hidden from the user. For example, the user may have $K = GF(2^3)$ and define L as a degree-8 extension of K with a certain polynomial; now L will be internally represented as a suitable degree-3 extension of $S = GF(2^8)$ where S has the Zech representation; this makes the arithmetic much faster, but L and its elements will still be explicitly presented to the user as an extension of K.
- A special packed representation is used in both characteristic 2 and 3 (when q is out of the Zech logarithm range). In characteristic 2, this also includes the use of special machine instructions for multiplication on recent Intel and AMD processors. For example, in $GF(2^{100043})$ one thousand element products can be computed in about 0.6 seconds.

Magma also has special code for fast evaluation of the Frobenius map based on linear algebra and this is exploited in several contexts. Apart from the basic arithmetic operations, there is a sophisticated algorithm to compute kth roots of elements of fields, where k may be an arbitrary positive integer.

6.3.3 Compatibility and Subfield Embeddings

The most critical feature which is unique to Magma's finite field machinery is automatic guarantee of *compatibility*; that is, one can construct an arbitrary lattice of finite fields of characteristic p by successively creating extension fields (via any irreducible polynomials) and subfields in any order and it is guaranteed that the diagram of subfields commutes (that is, when mapping an element through various relationships between subfields the result is the same no matter which path is taken).

The theoretical scheme which ensures this compatibility condition was developed by A. Steel in 1993 and strongly depends on linear algebra for managing the variety of background mappings. The scheme does not depend in any way on Conway polynomials (by which a lattice of subfields is defined only by direct extensions of the prime field using a fixed set of irreducible polynomials from a database). The Magma mechanism allows the user to create arbitrary chains of extensions (with arbitrary irreducible polynomials defining the extensions) and embed fields into each other when applicable; also, field elements can be embedded in any subfield and allow related operations, such as computing the minimal polynomial of an element with respect to any given subfield.

6.3.4 Discrete Logarithms

Magma provides machinery for computing discrete logarithms in any supported finite field no matter which field representation is used. Suppose w is a chosen primitive element of K. The *discrete logarithm problem* (DLP) for K is, given arbitrary non-zero element x of K, find an integer e such that $x = w^e$. Let $m = q - 1$ be the order of the multiplication group G of K. The Pohlig–Hellman algorithm reduces the problem to computing e modulo each prime power factor of m. Methods such as the Pollard rho algorithm with square root complexity can do this quite quickly provided that the cardinality l of the largest Sylow p-subgroup of G is less than 10^{15}.

If l is too large for the Pollard rho algorithm, then the *index calculus* method is used. This method has subexponential complexity. Magma has a number of implementations of the index calculus algorithm which are capable of solving the DLP in fields having small characteristics and cardinalities up to 10^{180}, at least. In 2012, a database was developed by A. Steel which contains the logarithms of a factor base for many finite fields having small characteristics thereby avoiding a precomputation step that potentially could take a very long time. The complete database currently includes logarithms at least for the fields $GF(p^d)$ for:

- $p = 2$: all $d \leq 450$ and all prime d up to 503;
- $p = 3$: all $d \leq 270$ (427 bits);
- $p = 5$: all $d \leq 219$ (509 bits);
- $p = 7$: all $d \leq 150$ (421 bits);
- $p = 11$: all $d \leq 85$ (294 bits);
- $p = 13$: all $d \leq 72$ (266 bits);
- $13 < p < 97$: several non-trivial fields.

6.4 Real and Complex Fields

6.4.1 Introduction

In this section, facilities for computing with real and complex-valued functions defined are summarised. Since a real or complex number is represented in the computer by a finite approximation, the functions are designed to produce results up to some specified precision. For example, if computations are to be performed over the complex field $\mathbb{C}$ with a precision of n digits, the user first creates an ambient complex field with n digits of precision. The functions and operations described here will then work to that precision given any elements belonging to that version of $\mathbb{C}$. The precision n can be set to a very large positive value and so calculations with millions of digits can be performed.

6.4.2 Complex Arithmetic and Special Functions

The packages MPFR and MPC are used to provide arbitrary precision real and complex arithmetic. These are principally developed by INRIA (particularly Paul Zimmermann originally) and Andreas Enge, with MPC being based on MPFR, and MPFR itself being based on GMP. These libraries allow guaranteed rounding according to ANSI/IEEE-754 standards in arbitrary precision upon zero-extending the input. The number of implemented functions is not overly large, but handles the basics (such as the trigonometric functions and their inverses).

Magma contains implementations of a number of other numerical functions, which are described in the Real and Complex Fields section of the Handbook. Examples are Bessel functions, polylogarithms, hypergeometric functions, Dawson's integral, and Weber's functions. The implementation of these largely follows PARI code from the mid 90s. Furthermore, some functions related to elliptic curves, such as the elliptic logarithm and exponential, have been re-implemented from scratch in a highly efficient manner. Another example is Θ-functions suitable

for working with Rosenhain invariants for analytic Jacobians of hyperelliptic curves (implemented by Paul van Wamelen).

6.4.3 Numerical Linear Algebra

Quite recently, the numerical facilities of Magma were extended to include some rudimentary methods for numerical linear algebra. These are based on the RQ decomposition of a matrix which is the reverse of the more traditional QR decomposition, as Magma is row-based in its view of matrices. The functionality currently includes:

- Rank
- Kernel
- Eigenvalues
- Pseudoinverse
- Solutions to linear systems of equations The user can specify a tolerance to be used, or this can be computed automatically via an analysis of a condition number.

Such numerical linear algebra methods are used internally in various places, such as when dealing with lattices over the real numbers, which occur in class and unit group calculations for number fields.

6.4.4 Real and Complex Roots of Polynomials

An implementation by Xavier Gourdon of Schönage's splitting circle method is used to find the complex roots of a polynomial to high accuracy. The Graeffe process is used to find suitable splitting circles, and given exact input, the method can then be iterated so as to determine the roots to within the precision of the ambient real field. Other algorithms include Laguerre's method and Newton–Raphson, which may have different time/accuracy trade-offs, and indeed are more useful for non-exact input.

The real roots of a polynomial are found using a version of the so-called "Uspensky's algorithm", based on Vincent's theorem, to isolate the roots. This uses translations of the original polynomial, as opposed to Sturm-based methods that use derivatives.

6.4.5 Linear Relations and Algebraic Dependencies

Given a complex number α and a target degree, Magma uses the LLL algorithm to determine a "best-fit" polynomial of the given degree for α. This now uses Damien Stehle's implementation of LLL, proceeding by finding a (near) linear relation between powers of α. More generally, a best-fit linear relation can be found for a sequence of complex numbers.

Other algorithms in Magma use real and/or complex approximations to determine an exact result. One example of this is the computation of Heegner points on rank 1 elliptic curves (over the rationals), where in conjunction with 12-descent, lattice-based methods can be used to recognise a rational point with 4600 digits using only 35 digits of precision.

6.4.6 Differentation and Integration

Some basic functionality is provided for integration, such as Simpson, trapezoidal, and Romberg quadrature. Interpolation is possible via the use of Neville's algorithm, and computing numerical derivatives is also available.

6.5 Polynomial Rings

6.5.1 Introduction

Since polynomials are an example of a particularly simple function, polynomial rings are of fundamental importance. In particular, they are the structures underpinning algebraic geometry. The data representation and algorithms for a *univariate polynomial* are quite different to those for *multivariate polynomial*. Let $R[x]$ be a ring of univariate polynomials over the ring R. Polynomials of $R[x]$ are generally represented by dense vectors of coefficients from R, or by bit packing when R is a very small finite field. The complexity of operations with such polynomials is generally dominated by the degree of the input polynomials.

Let $R[x_1, \ldots, x_n]$ be a ring of multivariate polynomials over R. Polynomials belonging to the ring $R[x_1, \ldots, x_n]$ are represented as a sparse list of coefficient-monomial pairs, where a monomial is represented by a list of exponents. The complexity of operations on such polynomials is generally dominated by the number of terms in the polynomials. Most multivariate polynomials which arise in Computer Algebra are *sparse*; that is, the number of terms in a polynomial is typically very much less than the product of the degrees in all variables. The arithmetic algorithms in the multivariate case are thus radically different to those in the univariate case.

This chapter is concerned with basic operations on polynomials such as multiplication, GCD, and factorisation. The tools for computing with ideals may be found in the section on Multivariate Polynomial Rings and Their Ideals.

6.5.2 Multiplication

Univariate polynomial multiplication based on the Fast Fourier Transform (FFT) is the critical component of all asymptotically-fast polynomial algebra. The basic idea behind the use of the FFT method for polynomial multiplication is to evaluate the degree-n input polynomials at $O(n)$ distinct roots of unity in a suitable ring, then multiply these evaluations pointwise and finally interpolate these to obtain the polynomial product. The evaluation and interpolation steps are done by the FFT operation in time $O(n \log(n))$ instead of the $O(n^2)$ time of the classical method.

The Schönhage–Strassen algorithm is a complete algorithm for multiplication based on the FFT method, using the ring of integers modulo $2^{2^k} + 1$ for a suitable integer k so that sufficient roots of unity are present. It is particularly efficient for multiplying very large integers or polynomials with large coefficients. For example, using Magma on a 3.2 GHz Xeon processor, one can multiply two million-decimal-digit integers in 0.025 seconds or two integer polynomials, each with degree 1000 and 1000-bit coefficients, in 0.026 seconds.

In characteristic 2, an algorithm of Cantor is used for polynomial multiplication, which follows the same basic approach as the FFT-based algorithms but which uses all elements of a suitable vector space as evaluation points instead of roots of unity. For example, on a 3.2 GHz Xeon processor, one can multiply two degree-million polynomials in $GF(2)[x]$ in 0.040 seconds.

All basic arithmetic operations in $R[x]$ (such as division, GCD, resultant) are reduced in Magma to multiplication using algorithms of Borodin and Moenck from the 1970s. These algorithms have complexity $O(n \log(n) \log(\log(n)))$ instead of the classical complexity $O(n^2)$, where n is the degree of the input, and so are asymptotically-fast. Furthermore, these algorithms work very well in practice even for moderately small degree (in the hundreds), so this is not just a theoretical result.

Multivariate arithmetic is based on a suite of algorithms which manage the sparse representation. Multiplication is performed by the Monagan–Pearce heap-based algorithm, saving much time and space over traditional methods.

6.5.3 Greatest Common Divisor

For the calculation of polynomial greatest common divisors (GCDs), Magma has implementations of the generic Euclidean algorithm for elements of $K[x]$ and the generic recursive subresultant GCD algorithm for elements of $K[x_1, \ldots, x_n]$. For several important field types, there are specific optimised algorithms for computing GCDs in $K[x]$:

- $K = GF(q)$: Moenck's asymptotically-fast algorithm (reducing to multiplication)
- $K = \mathbb{Q}$ or $\mathbb{Q}(\alpha_1, \ldots, \alpha_k)$: modular algorithm (mapping to the fast finite field case above) with Chinese remaindering
- $K = K_0(t_1, \ldots, t_k)$: fraction-free method mapping to an extended polynomial ring
- $K = (K_0(t_1, \ldots, t_k)[x_1, ..., x_n])/I$: asymptotically-fast evaluation/interpolation algorithm

For computing the GCD of multivariate polynomials over any of the above fields, a fast evaluation/interpolation algorithm is also provided (with both dense and sparse variants) which avoids intermediate coefficient blowup.

Similarly, Magma contains generic algorithms for computing the resultant of both univariate and multivariate polynomials over any field, but there are also special multiplication-based, modular, and interpolation algorithms for the important types of computable fields.

6.5.4 Factorisation of Univariate Polynomials

For factorisation in $K[x]$, where K is the finite field $GF(q)$, there are two main algorithms. The first is the Berlekamp algorithm, which uses linear algebra and is applicable to very small K and particularly when the input polynomial is sparse. Otherwise, the general von zur Gathen–Kaltofen–Shoup algorithm is used, which is the state of the art. This is a major extension of the Cantor–Zassenhaus algorithm (which involves taking GCDs of the input polynomial with $x^{q^i} - x$ for successive i) and includes many optimisations directly related to the FFT algorithm, such as multiplying by the inverse of various fixed moduli during the course of the algorithm with the precomputation of the FFT images of such inverses. (Note that these special components are also explicitly used in the elliptic curve point counting algorithm over finite fields.)

Magma also contains a special optimised test for irreducibility of high-degree sparse polynomials over small finite fields. In 2009, R. Brent and P. Zimmerman discovered a series of record high-degree primitive trinomials over $GF(2)$ and A. Steel used this algorithm in Magma to verify irreducibility of the polynomials for them. A typical example is $x^{42643801} + x^{3706066} + 1 \in GF(2)[x]$ which was proven primitive in about 18 days by Magma.

Factorisation in $\mathbb{Z}[x]$ is of critical importance because it underpins all forms of polynomial factorisation in characteristic zero. The basic technique is to factor the input polynomial modulo some prime p, lift the partial factorisation to be modulo p^k for suitable k via Hensel lifting (which maps to FFT-based multiplication), and finally combine the lifted factors to obtain the true factors over $\mathbb{Z}$. The standard combination phase had exponential worst case complexity until M. van Hoeij's novel practical algorithm (2001) which uses the LLL basis reduction algorithm to solve a Knapsack problem derived from the traces of the lifted factors. Magma has an optimised version of this algorithm, originally developed by A. Steel in discussion with van Hoeij in 2001. As a consequence, the combination phase rarely dominates the computation. Factorisation in $\mathbb{Q}[x]$ is also trivially reduced to the case of $\mathbb{Z}[x]$.

Factorisation in $K[x]$, where $K = \mathbb{Q}(\alpha)$ is a number field, is achieved via Trager's algorithm, which relies on computing resultants (to eliminate α) and then an associated factorisation in $\mathbb{Q}[x]$. Prior to 2001, this algorithm was impractical in many cases where the combination phase was too difficult, but again van Hoeij's algorithm for $\mathbb{Z}[x]$ has removed the difficulty; Trager's complete algorithm now runs in polynomial time and is very effective in practice. Within Magma, Trager's

algorithm has also been generalised to allow factorisation in $K[x]$ where K is any algebraic field of characteristic zero with an arbitrary number of transcendental and algebraic generators.

A general algorithm for factoring polynomials over non-perfect function fields of small characteristic (with an arbitrary number of algebraic and transcendental generators) was developed within Magma by A. Steel in 2005. Problems of inseparability make computations with such fields (e.g., $GF(2)(t)$) difficult. The algorithm works by lifting to purely inseparable extension fields over which the relevant polynomials become separable and Trager's algorithm can be applied. Finally, a Gröbner basis elimination technique is used to compute the result over the original field. No other Computer Algebra System has support for factorisation over such general function fields.

6.5.5 Factorisation of Multivariate Polynomials

Factorisation in the bivariate polynomial ring $R = GF(q)[x, y]$ is performed by an algorithm developed by M. van Hoeij and A. Steel in 2002. It uses ideas similar to the van Hoeij $\mathbb{Z}[x]$ combination algorithm with relations built on traces, but it uses direct exact linear algebra (no LLL or approximation needed). The time is dominated by asymptotically-fast Hensel lifting over power series, which maps down to multiplication in $GF(q^k)[[y]][x]$. As an example, let $f \in GF(5)[x, t]$ be the following bivariate polynomial (arising in the representation theory of Dickson groups):

- `4*t^3750*x^3125 + 4*t^3500*x^625 + 3*t^3350*x^625 + t^3350*x^25 + t^3300*x^125 + t^3270*x^125 + 4*t^3270*x^5 + t^3230*x^25 + t^3224*x^25 + 4*t^3224*x + 2*t^3200*x^625 + 3*t^3200*x^25 + 4*t^3150*x^3125 + 2*t^3150*x^125 + t^3120*x^125 + 4*t^3120*x^5 + 2*t^3000*x^3125 + 3*t^3000*x^125 + t^2750*x^15625 + 4*t^2750*x^625 + t^2600*x^15625 + 4*t^2600*x^625 + x^78125 + 4*x^3125`

The polynomial has very high degrees (78125 and 3750 in x, t respectively), but is extremely sparse (24 terms). Magma factors f in about 14 minutes and there are 6 irreducible factors which are dense (about 6000 terms each) and have x-degrees: 1, 15624, 15750, 15750, 15500, 15500 respectively. A typical large step during the Hensel lifting phase is reduced to multiplication of two polynomials in $GF(5^3)[[t]][x]$, which itself is reduced to multiplication of two integers, each having about *45 million decimal digits* and which are multiplied in 2.7 seconds. This shows how Magma can exploit FFT-based multiplication in characteristic zero even when factoring polynomials in very small prime characteristic!

General multivariate factorisation (for an arbitrary number of variables) is performed in Magma by mapping the input to bivariate images and then using sparse ideal-adic Hensel lifting. The final combination phase is in general easy, since the bivariate evaluation factorisation pattern is 'generic' and so typically reflects the final pattern closely; thus there is rarely a large number of combinations to consider.

6.6 General Commutative Rings in Magma

6.6.1 Introduction

This section lists the commutative rings that are currently supported in Magma.

The calculations that can be performed with a given ring depend upon the abstract properties of a ring (whether or not a field, for example) and the availability of effective algorithms. The various rings are grouped in this section according to their key algebraic properties.

6.6.2 Exact and Non-exact Rings

It is important to note at the outset that commutative rings in Magma are divided into two classes: *exact rings* and *non-exact rings*. Exact rings are characterised as rings whose elements have finite representations and so calculations with them produce an exact result. An example is the ring $\mathbb{Q}[x]$ of univariate polynomials with rational numbers as coefficients. Non-exact rings, on the other hand, are rings whose elements have infinite representations in general. The real field is an example of a non-exact ring, recalling that real numbers such as π or $\sqrt{2}$, given as real numbers, have infinite representations.

At present all of the non-exact rings are fields: the real and complex fields and the p-adic and formal power series fields. So with these four exceptions, the reader should assume all other rings discussed in this section are exact.

6.6.3 Standard Operations for Commutative Rings

All commutative rings support the following *standard operations*:

- Internal algorithms for the basic ring arithmetic operations: addition, subtraction, negation, and multiplication.
- A test as to whether an element is zero. This implies a test for the equality of elements.
- A *normalization* algorithm: for any non-zero $a \in R$, this computes a unique *canonical associate* b of a such that $b = ua$, with u a unit.

In the case of exact rings, the zero has to be a specific element. In the case of non-exact rings, the test for zero may be weakened to require that the value being tested lies in a small disk or interval.

6.6.4 Exact Fields

A *computable field* in Magma is a field which contains internal algorithms for the standard ring arithmetic operations together with division. The following list summarises each of the computable fields that are supported:

- A finite field $GF(q)$
- The rational field $\mathbb{Q}$
- A quotient ring of the polynomial ring $K[x_1, ..., x_n]/I$, where K is a computable field and I is a maximal ideal
- A number field $\mathbb{Q}(\alpha_1, \ldots, \alpha_k)$ (finite degree extension of $\mathbb{Q}$)
- A rational function field $K(t_1, \ldots, t_k)$, where K is a computable field, and $t_i, i = 1, \ldots, k$ are transcendental
- An algebraic function field $(K(t)[x_1, ..., x_n])/I$, where K is a computable field and I is a maximal ideal
- The algebraically closed field (ACF) over $GF(q)$ or $\mathbb{Q}$

6.6.5 Exact Euclidean Domains

A commutative ring R is said to have an *euclidean function* if:

- There exists a function $\phi : R \to \mathbb{Z}_{\geq 0} \cup \{-\infty\}$, such that for all $a, b \in R$ with $a, b \neq 0$: $\phi(a) \leq \phi(ab)$
- There exists a *division with remainder* algorithm such that for all $a, b \in R$ with $b \neq 0$, there exist $q, r \in R$ with $a = qb + r, \quad \phi(r) < \phi(b)$

A *computable euclidean domain* is a computable integral domain R having a euclidean function. The computable euclidean domains supported in Magma are:

- The ring of integers $\mathbb{Z}$
- A polynomial ring $K[x]$, where K is any computable field
- A quotient ring O_K/I, where O_K is the maximal order of a number field K, and I is an integral prime ideal of O_K
- A valuation ring (with a discrete non-Archimedean valuation)

6.6.6 Exact Integral Domains

The integral domains in Magma that are not euclidean domains are:

- A ring $R[x]$, where R is a computable integral domain but not a field
- A ring $R[x_1, \ldots, x_n]$, where R is a computable integral domain and $n > 1$
- A quotient ring $K[x_1, ..., x_n]/I$, where K is a computable field and I is a prime ideal
- A subring of the maximal order O_K, where K is a number field
- A subring of a maximal order O_K, where K is a function field
- A quotient ring O/I, where O is a subring of the maximal order O_K having coefficient ring $\mathbb{Z}$, K is a number field, and I is an integral prime ideal

6.6.7 Exact Principal Rings having an Euclidean Function

The rings in this section may have zero divisors and hence are not integral domains in general, but they do satisfy two useful conditions:

- The ideals are principal, so that the ring is a principal ideal ring or, more concisely, a *principal ring*.
- The ring has a euclidean function as defined above. In other words there is a division algorithm which can be used to find a generator for any ideal.

The list below contains the computable rings in Magma which are not integral domains, are principal and which have an euclidean function:

- A quotient ring $\mathbb{Z}/<m>$, for $m \in \mathbb{Z}$
- A quotient ring $K[x]/<f>$, for $f \in K[x]$, where K is a computable field
- A quotient ring of a p-adic ring $\mathbb{Z}_p/<p^d>$ (equivalent to using a fixed precision d)
- A quotient ring of a local ring equivalent to using fixed precision d
- A galois ring $GR(p^a, d)$. i.e., the unique galois extension of $\mathbb{Z}/p^a\mathbb{Z}$ of degree d.

6.6.8 Other Exact Commutative Rings

The commutative rings which are neither integral domains nor have an euclidean function are as follows:

- The rings $R[x]$ and $R[x_1, \ldots, x_n]$, where R is neither an integral domain nor has a euclidean function
- A quotient ring $K[x_1, ..., x_n]/I$ for a non-prime ideal I, where K is a computable field
- A quotient ring O/I, where O is a subring of O_K, K is a number field or function field and I is a non-prime ideal

6.6.9 Non-exact Fields

Recall that non-exact fields are those whose elements have infinite representations and so calculations have to be carried out using a finite approximation to the element and any answer is returned as a finite approximation. The various non-exact fields in Magma allow the user to work at any precision desired. Thus, when computing in the real and complex fields, the user sets the precision and the calculations will be performed to that precision.

When computing in a non-exact field, obvious difficulties arise when deciding whether an element is a identity or whether two elements are equal. If the precision chosen for the calculation is insufficient, it may well result in the calculation crashing or returning meaningless results. In fact a number of algorithms used in Magma that return exact results perform some part of the calculation in non-exact structures. Just as is the case in numerical analysis, careful error analysis has to be undertaken to ensure that the results of a non-exact calculation fall into a bounded region. However, for complex algebraic algorithms the error analysis is often very difficult and has been undertaken in the case of very few algorithms. In the case of Magma, it was decided to avoid developing numerical versions of exact algebraic algorithms because of the very large amount of work that would be needed with an uncertain outcome.

If K is a non-exact computable field then rings defined over K will also be non-exact. For example, the polynomial ring $K[x]$ will also be non-exact. However, basic operations such as addition, subtraction, and multiplication of elements of polynomial rings and matrix rings defined over a non-exact ring or field K are supported.

The six families of non-exact fields are as follows:

- The real field
- The complex field
- A p-adic field $\mathbb{Q}_p$
- A local field (a finite degree extension of a p-adic field)
- A field of formal Laurent series, $K((x))$, where K is a field
- A field of Puiseux series over a field K

6.6.10 Non-exact Commutative Rings

All of the non-exact commutative rings that are not fields are listed in this section.

- A power series ring $R[[x]]$, where R is a integral domain
- An extension of a power series ring $K[[x]]$, where K is a finite field
- A ring of formal Laurent series, $R((x))$, where R is a ring
- A power series ring being the ring of integers of a Laurent series field K
- A ring of Puiseux series over a ring R

Chapter 7

Error-Correcting Codes

7.1 Linear Codes over Finite Fields

7.1.1 Introduction

Let K be a finite field and let V be the vector space of n-tuples over K. The *Hamming distance* between elements x and y of V, denoted $d(x, y)$, is defined by

$$d(x, y) = \#\{\ 1 \leq i \leq n \mid x_i \neq y_i\ \}.$$

The *minimum distance* d for a subset C of V is then

$$d = \min\{\ d(x, y) \mid x \in C, y \in C, x \neq y\ \}.$$

The subset C of V is called an (n, M, d) *code* if the minimum distance for the subset C is d and $|C| = M$. The code C is called a $[n, k, d]$ *linear code* if C is a k-dimensional subspace of V.

In this section, the term "code" will refer to a linear code. Machinery is provided for studying linear codes over finite fields $GF(q)$, over integer residue classes $\mathbb{Z}_m = \mathbb{Z}/m\mathbb{Z}$, and over galois rings $GR(p^n, k)$. Magma supports not only linear codes, but also codes over finite fields which are only linear over some subfield. These are known as *additive codes* and are covered in a subsequent section. At present, Magma provides support for *block codes* but not *convolutional codes*.

7.1.2 Constructions for Codes

One of the strengths of the Magma coding theory package is the provision of a huge number of tools for constructing linear codes. Firstly, all of the standard ways of defining a linear code are present: generator matrix, parity check matrix, set of vectors belonging to V, generator polynomial, etc. Secondly, a very large number of constructions are provided which construct a new code from one or more existing codes. Examples are direct sum and product of codes, Plotkin sum, concatenated code, Construction X, and many others. The third group of construction functions implement families of interesting codes such as cyclic codes, BCH codes, Hamming codes, Reed–Muller codes, and Reed–Solomon codes as well as more exotic codes such as quasi-twisted cyclic codes, Chien–Choy codes, alternant codes, etc. Practically all of the constructions found in *The Handbook of Coding Theory* are available.

Functions are provided which give lower and upper bounds for the parameters n, k, and d associated with codes. There are also tables of the best known bounds for linear codes. Of particular interest are databases of the best known linear codes. An $[n, k]$ linear code C is said to be a *best known linear* $[n, k]$ *code* (BKLC) if C has the highest minimum weight among all

known $[n, k]$ linear codes. An $[n, k]$ linear code C is said to be an *optimal linear* $[n, k]$ *code* if the minimum weight of C achieves the theoretical upper bound on the minimum weight of $[n, k]$ linear codes. Magma currently has databases for best known linear codes over $GF(q)$ for $q = 2, 3, 4, 5, 7, 8, 9$. Given any two of the parameters length, dimension, and minimum weight, then Magma will return the code with the best possible value of the omitted parameter. Giving a specified length and minimum weight, for example, will result in a corresponding code of maximal possible dimension.

The database for codes over $GF(2)$ contains constructions of best codes of every dimension in F_2-vector spaces of length up to 256. The codes of length up to 31 are optimal. The $GF(2)$-database is complete in the sense that it contains a construction for every set of parameters. Thus the user has access to the 33,152 best-known binary codes covering every dimension and block length up to 256. Partial lists of best known codes are available for the other $GF(q)$ listed above. The databases are virtual in the sense that most codes are created when requested from a small number of "seed codes". These databases were constructed over the period 2000–2008 by J. Cannon (Magma), M. Grassl (Karlsruhe), and G. White (Magma).

A small amount of machinery is provided for constructing Algebraic Geometric codes. The most important tools are functions implementing the Goppa method for constructing a code from an irreducible plane curve defined over $GF(q)$. Other functions enable the user to construct a code based on toric geometry, specifically from the lattice points of a polygon.

7.1.3 Minimum Weight and Weight Distribution

Perhaps the key invariant of a linear code is its minimum weight as this is the same as its minimum distance which, in turn, determines the number of errors the code is guaranteed to be able to correct. Much effort has been invested into designing and implementing fast algorithms for minimum weight determination. For non-cyclic linear codes an algorithm of A.E. Brouwer and K.H. Zimmermann is used. If the code is cyclic a more efficient variant of the Brouwer–Zimmermann algorithm due to A.E. Brouwer is used. G. White generalised this improvement to quasicyclic codes and any codes whose known automorphisms have large cycles.

Both of the above minimum weight algorithms are very easy to parallelise in such a way that the speed-up factor approaches n (the number of processors) as n becomes large. As an example, the minimum weight for the hull of the binary [273, 101] code generated by the lines of the dual Dempwolff projective plane of order 16 was found to be 24 (a [273, 102] code). It is sometimes desirable to list all of the code words of minimum weight and for this a variation of the Brouwer–Zimmermann minimum weight algorithm is used.

Algorithms are also provided for determining the weight distribution of a linear code. A general algorithm for this problem involves enumerating the code words so it only useful for codes of small dimension. For example, the weight distribution of a [32210, 10] linear code over $GF(11)$ was found in 63,950 seconds running on a single processor. Various functions are provided for computing the weight enumerator or complete weight enumerator.

7.1.4 Equivalence and Automorphisms

The equivalence of two linear codes may be determined for codes defined over finite fields $GF(q)$, where q is a small prime or 4. The algorithm is due to J. Leon. An algorithm developed by W. Unger (Magma) is capable of handling much larger codes (especially codes having large block lengths). At present it only applies to binary codes but it will be extended to other finite fields in the near future.

The automorphism group of the linear code C over the field K is the group A of all monomial-action permutations which preserve C. Thus, both permutation of coordinates and multiplication

of components by non-zero elements of K is allowed. Variations of both the Leon and Unger equivalence algorithms are provided for computing the automorphism group of a linear code. The restrictions on the finite field that apply for testing equivalence also apply here.

7.1.5 Decoding

Currently, two methods are provided: syndrome decoding and a Euclidean decoding algorithm. The syndrome algorithm applies to any code but requires the storage for a set of coset representatives for the code with respect to its parent vector space. Consequently, it is restricted to linear codes having small codimension. The Euclidean algorithm decoder applies to any alternant code which includes BCH, Goppa, and Reed–Solomon codes amongst others.

7.1.6 Low Density Parity Check Codes

As their name suggests, low density parity check codes (LDPC codes) are block codes with parity-check matrices H that contain only a very small number of non-zero entries. It is the sparseness of H which guarantees both a decoding complexity which increases only linearly with the code length and a minimum distance which also increases linearly with the code length. LDPC codes are designed by first constructing a sparse parity-check matrix and then determining a generator matrix for the code. The biggest difference between LDPC codes and classical block codes is the manner in which they are decoded. Classical block codes are generally decoded using maximum likelihood decoding (decoding a vector into a closest codeword) and so are usually short. LDPC codes, however, are decoded iteratively using a representation of their parity-check matrix as a graph.

Magma provides machinery for constructing a LDPC code C either by choosing a parity-check matrix from some random distribution or from a standard construction such as the group-based construction of Margulis. All of the standard operations for linear codes over $GF(q)$ apply to LDPC codes. Special operations implemented for an LDPC code include the construction of its Tanner graph and the calculation of its theoretical rate from an ensemble used to construct C.

LDPC codes of practical interest are usually chosen to have very large block lengths so that it is not possible to determine the minimum weight and related properties. Instead, their usefulness is determined by simulation. Magma supports two different channel models: the binary symmetric channel (BSC) and the Gaussian channel. The standard iterative decoding algorithm for LDPC codes is provided. For a given LDPC code, a simulation function is provided which simulates N transmissions across the given channel and returns the accumulated bit-error rate and word-error rate. The asymptotic performance of ensembles of LDPC codes can be determined using *density evolution*. An ensemble of LDPC codes is defined by a pair of degree distributions, corresponding to the degrees at the variable and check nodes of the Tanner graph. Density evolution may be applied to an ensemble of codes with respect to either the BSC or the Gaussian channel models.

7.1.7 The McEliece Cryptosystem

The McEliece cryptosystem is a public-key cryptosystem whose security is based on the difficulty of decoding a general linear code. Interest in this cryptosystem has grown recently due to the belief that it is one of the few current PKCs that are immune from attack by quantum computers. An attack on the McEliece cryptosystem must determine the coset leader (of known weight) given an error coset. Magma contains implementations of a number of attacks on McEliece. These include the attacks of McEliece, Lee–Brickell, Leon, Stern and Canteaut–Chabaud. In addition, the user may choose any combination of the implemented methods, which include improvements on the standard attacks.

7.1.8 Use and Applications

The Magma facility for algebraic coding theory is superior to other packages in this field (noting that it is not intended to compete with packages designed for the engineering side of ECC). Most groups working in algebraic coding theory use Magma for a great diversity of applications. Typical uses involve the construction of special codes, the construction of combinatorial objects from codes, the search for better codes (here the minimum weight tools are fundamental), certain attacks on cryptosystems that reduce to finding low weight codewords, and the classification of linear codes of a certain type.

Approximately 400 published papers on coding theory that cite the use of Magma have been identified (the exact number is unknown). A major project that has been carried out by Japanese mathematicians over the past decade has been to classify all self-dual codes of various lengths. Here one of the roles played by Magma was to test codes for equivalence. Another project to which Magma has contributed over many years has been research in finite geometry. For example, it was used in 2008 to show that none of the non-desarguesian affine planes of order 16 are tame.

7.2 Linear Codes over Finite Rings

7.2.1 Codes Over Galois Rings

Interest in codes that are defined over rings rather than fields arose following the discovery by A.R. Hammons, P.V. Kumar, A.R. Calderbank, N.J.A. Sloane and P. Solé in 1994 that the nonlinear Preparata and Kerdock binary codes can be derived in a natural manner from *linear codes* over $\mathbb{Z}_4$. The Preparata and Kerdock binary codes are among the best binary codes known — the Preparata code has more codewords for its length than any known linear code.

Magma provides basic facilities for computing with codes over Galois rings, including the construction of cyclic codes, the calculation of minimum weight, and the complete weight enumerator. Substantial additional functionality is available for the special case of codes over $\mathbb{Z}_4$.

For modules defined over rings with zero divisors, the concept of dimension does not exist as the modules are not free. But in Magma each code over such a ring has a *unique* generator matrix corresponding to the *Howell form*. The number of rows k in this unique generator matrix will be called the *pseudo-dimension* of the code. It should be noted that this pseudo-dimension is not invariant with respect to equivalent codes, and so does not play the same role as the dimension of a code over $GF(q)$. The rank of the generator matrix is always well-defined and unique (being based on the Smith form which is well-defined over principal ideal rings (PIRs), but k may sometimes be larger than the rank. Without a concept of dimension, codes over finite rings are referenced by their cardinality. A code C is called an (n, M, d) *code* if it has length n, cardinality M, and minimum Hamming weight d. In this section, as in the case of codes over finite fields, the term "code" will refer to a linear code unless otherwise specified.

7.2.2 Constructions for Codes

Let R be the Galois ring over which a code C is to be defined; thus C is a submodule of the R-module $M = R^{(n)}$. A basic approach to defining C is to specify a list of generators for C. Just as in the case of codes over finite fields, cyclic codes are an important family of linear codes over finite rings and a number of special constructions are provided. New codes can be formed from an existing code C by extending, padding, shortening, and puncturing C. Codes can also be created by taking two existing codes defined over the same ring R and combining them in some way such as forming their direct sum, direct product, concatenation, Plotkin sum, etc.

Constructions are provided for a number of important families of codes over the ring $\mathbb{Z}_4$. These

include the Kerdock, Preparata, Reed–Muller codes, the Goethals, Delsarte–Goethals, Quadratic Residue codes, the Hadamard code, and the simplex alpha and beta codes. Magma provides functions for constructing three important binary codes which can be derived from a $\mathbb{Z}_4$-code. Perhaps the most important of these is the binary image of a quaternary code under the Gray map. Functions are also provided that construct codes over $\mathbb{Z}_4$ obtained by modifying the codewords of an existing code over $\mathbb{Z}_4$. The main operations are various forms of the Plotkin sum.

The usual basic operations for submodules of $M = R^{(n)}$ are supported for codes C belonging to $R^{(n)}$. In the case in which $R = \mathbb{Z}_4$, functions are available which compute the image and kernel of C with respect to the Gray map. The image will be a binary linear code while the kernel will be code with parent module M.

7.2.3 Minimum Weight and Weight Distribution

In the case of codes over Galois rings, three metrics are used: the *Hamming metric*, the *Lee metric*, and the *Euclidean metric*. For each metric algorithms are provided for computing the minimum weight, the weight distribution, and the dual weight distribution. The minimum weight is computed using a variant of the highly efficient minimum weight algorithm developed for codes over finite fields and so minimum weight can be determined in quite large codes. The weight enumerator, weight distribution, and dual weight distribution can be computed with respect to each of the metrics. The complete weight enumerator is available. For codes over $\mathbb{Z}_4$ the symmetric weight enumerator can also be computed.

The Octacode is a $(8, 256, 4)$ code generated by the vectors $(1, 0, 0, 0, 3, 1, 2, 1)$, $(0, 1, 0, 0, 1, 2, 3, 1)$, $(0, 0, 1, 0, 3, 3, 3, 2)$, and $(0, 0, 0, 1, 2, 3, 1, 1)$ over $\mathbb{Z}_4$. The Hamming weight enumerator, the Lee weight enumerator and the Euclidean weight enumerator are, respectively:

$$x^8 + 14x^4y^4 + 112x^3y^5 + 112xy^7 + 17y^8$$

$$x^{16} + 112x^{10}y^6 + 30x^8y^8 + 112x^6y^{10} + y^{16}$$

$$x^{32} + 128x^{24}y^8 + 126x^{16}y^{16} + y^{32}$$

7.3 Additive Codes

7.3.1 Introduction

The concept of a linear code over a finite field can be generalised to the notion of an *additive* code. Given a finite field F and the space of all n-tuples over F, an additive code is a subset of $F^{(n)}$ which is a K-linear subspace for some subfield $K \subseteq F$. Additive codes have become increasingly important recently due to their application to the construction of quantum error-correcting codes, though they are also of interest in their own right. A Magma package for quantum error-correcting codes is built on the machinery for additive codes.

In Magma an additive code has both an *alphabet* F and a *coefficient field* K, which is a subfield of F. An error-correcting code is considered to be defined by its wordset, so there may be several different ways of presenting a given code using different coefficient fields. Since a given code may be presented over different coefficient rings, the *dimension* k of an additive code is defined relative to the alphabet of the code, $\#C = (\#F)^k$, leading to possibility of *fractional* dimensions. Consequently, the number of generators of an additive code will not equal its dimension, there being $[F : K]$ times as many generators. So a length n K-additive code over F has between zero and $n \times [F : K]$ generators.

For example, consider the two length 3 vectors over F_4: $(1, 0, \omega^2), (0, \omega, 0)$. The linear code generated by these vectors consists of all scalar multiples and sums, resulting in a total of $4^2 = 16$

vectors. But the F_2-additive code generated by these two vectors contains *only their sums*, resulting in a total of $2^2 = 4$ vectors. These vectors are $(0,0,0), (1,0,\omega^2), (0,\omega,0), (1,\omega,\omega^2)$. The alphabet of this code is F_4, its coefficient field is F_2, it has 2 generators and is of dimension 1.

A length n, dimension k K-additive code over F with k_g generators is represented in Magma as an $[n, k : k_g]$ K-additive code over F. The concepts of weight, distance and their respective distributions and enumerators transfer directly from linear codes. An $[n,\ k : k_g,\ d]$ K-additive code over F is a K-linear subset of $F^{(n)}$ which has fractional dimension k, k_g generators, and a minimum weight of d.

7.3.2 Constructions for Additive Codes

Let $F = GF(q)$ and take K to be a subfield of F. A length n additive code C with alphabet F and coefficient field K consists of codewords which form a K-linear subspace of $F^{(n)}$. A straightforward way of defining C is to specify a set of elements of $F^{(n)}$ which generates C. Thus additive codes can be constructed from any linear code defined over $GF(q)$ provided that $GF(q)$ has a suitable subfield K.

The usual vector and submodule operations for $M = F^{(n)}$ are supported for additive codes C belonging to $F^{(n)}$. Familiar operations are provided for producing a new code from one or more existing codes: augmentation, complement, concatenation, juxtaposition, direct sum, direct product, extension, padding, Plotkin sum, puncturing, and shortening. Constructions are provided for two important families of additive codes: additive cyclic codes and additive quasicyclic codes.

Various properties of an additive code can be determined: self-dual, self-orthogonal, perfect, projective, and additive projective.

7.3.3 Minimum Weight and Weight Distribution

An adaptation of the minimum weight algorithm for linear codes over finite fields has been developed for additive codes by M. Grassl and G. White. The algorithm is very efficient. For example, taking a random $GF(2)$-additive code with alphabet $GF(4)$, length 82, and 39 generators, the minimum weight of 32 was found in less than 7 seconds. The weight enumerator, weight distribution, and dual weight distribution can be computed. The complete weight enumerator and symmetric weight enumerator are also available.

7.3.4 Automorphism Group

The automorphism group of an additive code can be computed for codes over $GF(4)$. This will be extended to other fields in the near future.

7.4 Quantum Error-correcting Codes

7.4.1 Quantum Error-correction

Interest in quantum computing has grown rapidly following the discovery by Peter Shor in 1994 of a polynomial-time algorithm for integer factorisation on a quantum computer. In a classical computer a sequence of N binary digits defines one specific configuration among the 2^N possible values. However, in a quantum computer a collection of N "qubits" has a state function (in 'ket' notation) $ket(\psi)$ in a *Hilbert space*, which can be in a superposition of all 2^N possible values:

$$ket(\psi) = \sum_{\mathbf{v}\in\mathbb{Z}_2^N} \alpha_{\mathbf{v}} ket(\mathbf{v}), \qquad \alpha_{\mathbf{v}} \in \mathbb{C}, \quad \sum_{\mathbf{v}} |\alpha_{\mathbf{v}}|^2 = 1$$

A basic theorem in quantum information theory states that it is impossible to clone a quantum state. Since this implies that it is not possible to copy quantum information, it was initially believed that error-correction would be impossible on a quantum computer. However, in 1995 Shor showed that it *was* possible to encode quantum information in such a way that errors can be corrected, assuming an error model in which errors occur independently in distinct qubits. Following this discovery, a class of quantum error-correcting codes known as *stabiliser codes* were developed. In 1996, Calderbank, Rains, Shor, and Sloane showed that stabiliser codes can be represented in terms of additive codes over finite fields. This remarkable result reduces the problem of constructing fault-tolerant encodings on a continuous Hilbert space to that of constructing certain discrete codes, allowing the use of many of the tools developed in classical coding theory.

7.4.2 Stabiliser Codes and the Error Group

The current Magma package for quantum codes deals exclusively with finite field representations of stabiliser codes. It is important to keep in mind that, although a quantum code is *represented* by a code over a finite field, an actual quantum code is in fact a totally different object. The reduction of the problem of continuous errors on a Hilbert space to a problem employing a discrete finite field representation is achieved by confining attention to a finite *error group*. An element of the error group, acting on the N qubits, is expressed as a combination of bit flip errors, designated by the operator X, and phase shift errors, designated by the operator Z (as well as an overall phase factor that will be ignored here):

$$X(\mathbf{a})Z(\mathbf{b})ket(\mathbf{v}) = (-1)^{\mathbf{v}\cdot\mathbf{b}}ket(\mathbf{v}+\mathbf{a})$$

The error group is given by the set $\{X(\mathbf{a})Z(\mathbf{b}) : \mathbf{a},\mathbf{b} \in \mathbb{Z}_2^n\}$ and its elements can be written as length $2N$ binary vectors $(\mathbf{a}|\mathbf{b})$. An error represented by such a vector in Magma is said to be in *extended format* which is distinct from the default representation. A more common (and practical) representation is as the element $\mathbf{w}$ of $GF(4)^{(N)}$ given by $\mathbf{w} = \mathbf{a} + \omega\mathbf{b}$, where ω is a primitive element of $GF(4)$. This representation is referred to as the *compact format*, and is the default format used in Magma for quantum codes.

The Magma package also supports non-binary quantum codes, which are obtained by generalising from the binary case in a natural way. For quantum codes based on qubits over $GF(q)$, the compact format in $GF(q^2)$ will be $\mathbf{w} = \mathbf{a} + \lambda\mathbf{b}$, where λ is a fixed element of $GF(q^2)$ which defines the $GF(q^2)$ representation for quantum codes over $GF(q)$.

A symplectic inner product is defined on the group of errors in its representation as a set of $GF(q)$-vectors. For vectors in extended format this is defined by

$$(\mathbf{a}_1|\mathbf{b}_1) * (\mathbf{a}_2|\mathbf{b}_2) = \mathbf{a}_1 \cdot \mathbf{b}_2 - \mathbf{a}_2 \cdot \mathbf{b}_1$$

In compact format (over $GF(4)$) the equivalent inner product is defined by

$$\mathbf{w}_1 * \mathbf{w}_2 = \mathrm{Trace}(\mathbf{w}_1 \cdot \overline{\mathbf{w}}_2).$$

Evaluating the commutator of two errors we see that errors commute if and only if their finite field representations are orthogonal with respect to the symplectic inner product.

A quantum stabiliser code is defined by an abelian subgroup of the error group. In the context of its finite field representation this translates to a self-orthogonal additive code under the symplectic inner product. So a quantum stabiliser code Q is defined by a symplectic self-orthogonal additive code S, which is (with some redundancy) termed the stabiliser code of Q. The error-correcting capability of a code is determined by the set of errors which cannot be detected. For classical linear codes these undetectable errors are precisely the non-zero codewords of the code, while for

a quantum code the undetectable errors are given by the set $S^{\perp}\backslash S$, where $S^{\perp}$ is the symplectic dual of S.

The most important measure of the ability of a quantum code to correct errors is its *minimum weight*; that is, the minimum of the weights of the words of $S^{\perp}\backslash S$. An exception to this definition occurs in the case of quantum codes having dimension zero, which are defined by symplectic self-dual stabiliser codes. These are termed "self-dual quantum codes" and are defined to have a minimum weight equal to the (classical) minimum weight of their stabiliser code.

7.4.3 Constructions for Quantum Codes

A quantum code of length n over $GF(q)$ is defined in terms of a symplectic self-orthogonal stabiliser code, which is given either as a length n additive code over $GF(q^2)$ (compact format) or as a length $2n$ additive code over $GF(q)$ (extended format). If Q is a quantum code with generator matrix G_1 in compact format, and generator matrix $G_2 = (A|B)$ in extended format, then

$$G_1 = A + \lambda B$$

where λ is a fixed element of $GF(q^2)$ which defines the $GF(q^2)$ representation for quantum codes over $GF(q)$.

An $[n, k]$ symplectic self-orthogonal linear code over $GF(q^2)$ will generate an $[[n, n/2-k]]$ quantum stabiliser code. A (compact format) additive symplectic self-orthogonal code C over $GF(q^2)$ will give a quantum code of the same length and "dimension" $\log_q(N)$, where N is the number of code words in C.

The basic construction takes an additive code S which is self-orthogonal with respect to the symplectic inner product and returns the quantum code defined by S. The usual constructions for producing new codes by modifying an existing code are provided: direct sum, extension, puncture, shorten. Given linear binary codes C_1 and C_2 such that C_2 is a subcode of C_1, the construction due to Calderbank, Shor, and Steane may be used to construct a quantum code.

Cyclic quantum codes are those having cyclic stabiliser codes. The conditions which generating polynomials must satisfy in order to yield symplectic self-orthogonal stabiliser codes have been published by Calderbank, Rains, Shor, and Sloane. A number of functions are provided to construct both cyclic quantum codes and quasicyclic quantum codes.

Given a quantum code, functions are provided to return the stabiliser group and various versions of the quantum error group. The error group is an extraspecial p-group and so is returned in the form of a pc-group. The stabiliser group and normaliser matrix can also be accessed.

An $[[n, k]]$ quantum stabiliser code Q is said to be a *best known* $[[n, k]]$ *quantum code* (BKQC) if C has the highest minimum weight among all known $[[n, k]]$ quantum codes. Magma currently has a database of binary quantum codes, though it should be noted that these codes are considered to be over the alphabet $GF(4)$, not $GF(2)$. The database for codes over $GF(4)$ currently contains constructions of all best known quantum codes of length 35, including self-dual quantum codes up to length 35. Quantum codes of length up to 12 are optimal, in the sense that their minimum weights meet the upper bound. Thus, the user has access to 665 best-known binary quantum codes. The database was compiled by M. Grassl (Karlsruhe).

7.4.4 Minimum Weight and Weight Distribution

The weight distribution of a quantum code Q consists of three separate distributions:

- The weight distribution of the stabiliser code S.
- The weight distribution of the symplectic dual $S^{\perp}$ of S.

- The weight distribution of the codewords in $S^{\perp} \backslash S$. Note that this set is not a linear space.

For a quantum code Q with stabiliser code S, the weights of the undetectable errors are the weights of the codewords in $S^{\perp} \backslash S$. The weight distribution function for a quantum code returns each of these distributions as a separate value. The value returned by the minimum weight function is the minimum weight of the codewords in $S^{\perp} \backslash S$. The default algorithm is based on the minimum weight algorithm for classical linear codes.

7.4.5 Automorphism Group

Automorphisms acting on a quantum code are a slight generalisation of those which act on the underlying additive stabiliser code. Thus, automorphisms comprise a permutation action on the columns of a stabiliser code, combined with a monomial action on the individual columns which permute the values. The automorphism group of a length n additive stabiliser code over $GF(4)$ is a subgroup of $Z_3 \wr Sym(n)$ of order $3 \times n!$. However, the automorphism group of the quantum code it generates is a subgroup of $Sym(3) \wr Sym(n)$ of order $3! \times n!$ because of the more general action on the values in the columns. At present the computation of the automorphism group is restricted to binary quantum codes.

7.4.6 Simulation on Hilbert Spaces

The QECC machinery includes code that allow the construction and analysis of quantum Hilbert spaces. Such a space is constructed from a complex field and an integer giving the number of qubits which the space is to model. A set of basic operators allow new states to be constructed from existing states. A set of functions are provided that allow the user to determine probabilities of quantum states. For example, given a binary vector v of length equal to the number of qubits in the quantum state e, a function returns the probability of the basis state corresponding to v being returned as the result of a measurement on e. A further function returns the probability distribution of a given quantum state. A small set of unitary transformations on quantum states are available. These include *bit flip*, *phase flip*, *controlled not*, and the *Hadamard transformation*.

7.4.7 Applications

A number of groups working in the field of Quantum Information Theory make heavy use Magma in their work. A very incomplete list includes:

- Centre for Quantum Technologies (National University of Singapore)
- Institute of Cryptography and Security (Karlsruhe Institute of Technology)
- Institute for Quantum Computing (Waterloo)
- NEC Laboratories America, Inc. (Princeton)
- Perimeter Institute (Waterloo)
- Quantum Information Group (Innsbruck)

Some relevant publications up to 2010 that cite Magma may be found at

- `http://magma.maths.usyd.edu.au/magma/citations/topic/QntCmp/`

Chapter 8

Group Theory

8.1 Finitely Presented Groups

8.1.1 Introduction

A *presentation* for a group G consists of a set X such that every element can be written as a word in G and a set R of relations such that the equivalence classes of words with respect to R correspond to the elements of G. A group G is said to be *finitely-presented* if both X and R are finite. In that case, G is said to be a finitely-presented group (fp-group).

Two basic properties of an fp-group G frequently sought are (a) whether G is the identity group or a non-trivial group and (b) whether G is finite or infinite. Novikov and Boone have shown that all such questions are undecidable in general. However, Magma has a range of tools that can frequently answer these questions.

8.1.2 Coset Enumeration: Todd–Coxeter Procedure

The Todd–Coxeter procedure takes a group G and a set of words defining a subgroup H of G and attempts to enumerate the cosets of H in G. The Todd–Coxeter implementation used in Magma is essentially the ACE program of G. Havas and C. Ramsay. The version of this program installed in Magma allows about 10^9 cosets to be defined in the case of a 2-generator group.

If the procedure terminates the result is a table known as the coset table, which gives the action of G on the cosets and thus provides a permutation representation of G. The coset table of H in G makes it possible to show that G is non-trivial (true if the index of H is greater than 1)), test membership of elements of G in H, determine if H is normal in G, construct a homomorphism ϕ of G onto a subgroup of $\mathrm{Sym}(n)$ defined by the action of G on the cosets of H, and compute the kernel of ϕ.

If the group G is finite and not too large, then the Todd–Coxeter procedure can be used to enumerate the cosets of the identity and hence determine the order of the group. A much better approach is to try to find a subgroup H of known order and attempt to enumerate its cosets. Enumerating the cosets of the identity subgroup of G gives us the regular permutation representation of G. Tools are available in Magma which will attempt to transform this representation into a faithful permutation representation for G of much smaller degree.

The procedure will not terminate if H has infinite index in G, or if the index is much too large for the computation to fit into the computer's memory. However, even if the index is small, there exist presentations that cause the Todd–Coxeter procedure to define such a large number of redundant cosets that the enumeration will never complete in practice.

8.1.3 Low Index Subgroups

It is possible to enumerate the conjugacy classes of subgroups for an fp-group having small index. This employs an algorithm of C. Sims which enumerates the coset tables corresponding to distinct conjugacy classes of subgroups of G. The algorithm constructs the coset tables by using a backtrack algorithm to run through the possible tables but otherwise builds the tables in a similar manner to the Todd–Coxeter procedure. Although it sometimes cannot complete the enumeration for indices as small as 10, it can get much further for many groups. For example, in the case of the infinite triangle group $\langle x, y \mid x^2, y^3, (xy)^7 \rangle$ the low index algorithm in Magma finds the 3,037,698 conjugacy classes of subgroups having index not greater than 100 in 414 seconds.

The low index algorithm is very useful. For example, it is a good first choice when attempting to prove that a group is non-trivial. Also it can be used to search for subgroups with certain properties such as having an infinite abelian quotient. Overall, it is a key tool in heuristic algorithms which attempt to prove that a group is infinite.

In 2006 an algorithm was developed by D. Holt and D. Firth which searches for low index normal subgroups (LINS). The algorithm searches for small quotients of G by a systematic enumeration of the possibilities for a composition series for the quotient. It was implemented in Magma by Holt. In favourable circumstances it can search for normal subgroups of index up to $1,000,000$. In the case of the triangle group mentioned above, LINS finds the 35 normal subgroups of index not exceeding 100,000 in 1344 seconds.

8.1.4 Presentations for Subgroups

In the previous subsection techniques have been described for locating subgroups of small index in an fp-group G. The Reidemeister–Schreier algorithm enables the user to construct a presentation for a subgroup H on Schreier generators for H. The main drawback is that the number of generators and the number of relations produced for H are each a multiple of $[G, H]$. This problem can be reduced somewhat by performing simplification (see below) on the Reidemeister–Schreier presentation. However, once the index $[G, H]$ gets up towards 100 even the simplified presentation may be too huge for some applications.

An alternative function attempts to construct a presentation for H in terms of the generators originally used to define H. This has the advantage that they may be more meaningful to the user and it is likely that there are many fewer of them. This uses a variant of ACE called *extended coset enumeration* whereby information such as coset definitions are stored as words as the Todd–Coxeter proceeds.

8.1.5 Simplification of Presentations

In various situations it is desirable to simplify a presentation. In particular, presentations constructed by the Reidemeister–Schreier algorithm typically have a huge number of unnecessary generators and relations. Tietze transformations comprise four basic operations that can be performed on any presentations with the property that the group defined does not change. Tietze showed that any presentation for G can be obtained from any other presentation for G using only Tietze transformations. Magma contains code due to G. Havas and C. Ramsay for performing simplification of presentations using Tietze transformations.

The simplification program often produces a good result when applied to presentations with many redundant generators and relations; the number of generators is typically reduced to less than 10. However, some of the relations become exceedingly long and this can make subsequent computations perform badly. Two typical operations one wants to do with a subgroup of G is obtain its abelian quotient or some more general quotient such as a p-quotient. Both of these are somewhat forgiving of large presentations.

8.1.6 Quotients of a Finitely Presented Group

A tactic often used when investigating an fp-group G is to construct homomorphisms onto groups with solvable word problems and where the group structure can easily be computed. Note that a coset table defines a homomorphism of G onto a permutation group. Below other quotient constructions are listed.

- Magma has very powerful methods for finding the maximal abelian quotient of G. An integer matrix is created by writing the relations of G additively and its Smith Normal Form is computed. The method is capable of handing presentations involving tens of thousands of generators and relations such as those arising from an Reidemeister–Schreier presentation of a subgroup.
- Magma includes the ANU p-Quotient program for computing quotients of an fp-group that are p-groups. The algorithm returns the largest p-quotient of G which is returned as a pc-group, a special form of finite presentation where the word problem is solvable. The program was developed by M. Newman and E. O'Brien.
- A Nilpotent Quotient algorithm has been developed by W. Nickel and his implementation is used in Magma. It computes nilpotent quotients of G and, if it terminates, returns the largest nilpotent quotient of G. It is a generalisation of the p-Quotient algorithm. The nilpotent quotient is returned as a polycyclic group (a generalisation of pc-groups to infinite groups) which again has a solvable word problem.
- Another quotient method in Magma is a version of W. Plesken's Soluble Quotient algorithm which is designed to return the largest solvable quotient of G (providing it is finite) as a pc-group. The version installed in Magma is due to H. Brückner (Aachen), a student of Plesken.
- All the above quotient methods return soluble quotients, which only exist if the abelian quotient of G is non-trivial. Outside this situation G is perfect, and so it may be possible to find homomorphisms of G onto finite simple groups. Magma's simple quotient algorithm, designed and implemented by D. Holt, performs such searches.
- In 2009 Plesken and Fabiànska (both in Aachen) developed an algorithm which computes all quotients of G which are isomorphic to $PSL(2,q)$ or $PGL(2,q)$, were q is a prime power. The algorithm finds all possible q and can determine whether G has an infinite number of $PSL(2,q)$ or $PGL(2,q)$ quotients. The program used in Magma is an improved version developed by S. Jambor.

8.1.7 Finite or Infinite?

Perhaps the most common question a user wants answered about an fp-group G is whether the group is finite or infinite. As with most computational problems relating to general fp-groups there is no algorithmic solution to the problem. Instead, various procedures are applied in the hope that one will provide an answer. If there is no strong information suggesting one possibility is more likely than the other then equal effort has to be invested in trying to establish each of the possibilities.

The standard approach to proving that a group is finite is to examine the presentation for G and try to identify a subgroup H which is clearly finite. Dihedral groups are often found in a group having some generators of order 2. Then one tries to enumerate the cosets of H in G. If this fails then other choices of H are sought. If the direct enumeration of the cosets of a finite subgroup fails then the techniques of the previous section are applied to try to find non-trivial quotients. The quotients may directly show that G is infinite but if that fails, they can be used to provide a lower bound on the order of G.

There are few useful methods for proving that a group is infinite. By far the most effective is to demonstrate the existence of a subgroup H of G that has an infinite section. The easiest way of doing this is to locate a subgroup of G that has an infinite abelian quotient. The low index

subgroups and low index normal subgroups algorithms are an important source of subgroups. If the low index searches are unsuccessful (or there are no subgroups of small enough index to find) quotient groups and their subgroups are another important source. A program of J. Cannon has automated the systematic application of these and other tests to try to establish that an fp-group is infinite. Applied to the fundamental groups of the 11126 hyperbolic 3-manifolds having small volume compiled by J. Weeks the program was able to produce a proof that all but 10 of the these fundamental groups are infinite.

8.1.8 Classes of FP-Groups with Solvable Word Problem

For certain families of fp-groups, presentations are known which provide a solution to the word problem for all groups in that family. A simple example is the class of abelian fp-groups. Five important families supported in Magma are introduced below but will not be discussed further in this document.

A solution to the word problem for finite soluble fp-groups is provided by the *power-commutator presentation* which defines a normal form for each element and an algorithm (collection) that will normalise an arbitrary word. A group given by a power-commutator presentation is known as a *pc-group*. A large collection of efficient algorithms has been developed for pc-groups.

The appropriate generalisation of pc-group to an infinite soluble group is the *polycyclic group* which is a group admitting a subnormal series with cyclic factors. While no practical method is known for deciding whether an infinite fp-group is polycyclic, W. Plesken has described an algorithm for constructing a polycyclic quotient from a general fp-group. Some structural algorithms have been developed for the subclass of polycyclic groups that are nilpotent.

A *Coxeter group* is one given by a presentation where all the relations have either the form $x^2 = 1$ or $(xy)^k = 1$. They are abstract reflection groups. An algorithm due to B. Brinck and R. Howlett gives a solution to the word problem for these groups. Hence element arithmetic and the calculation of growth functions are supported. Computation with parabolic subgroups, their cosets, and double cosets is also implemented.

The family of infinite groups known as *braid groups* B_n, $1 < n$, can be defined as fundamental groups of a configuration space. They also have an intuitive definition in terms of bunches of strings with a composition operation. The groups have a nice presentation due to Artin which has a solvable word problem. A substantial facility for computing with braid groups has been developed by V. Gebhardt (Magma).

An *automatic group* is an finitely generated group G together a finite state automata and associated regular languages. Let G have a finite monoid generating set A. Then G is an automatic group if there is a language L over A such that the natural map $\pi : L \mapsto G$ is onto and certain associated languages are regular. Automatic groups have a soluble word problem and a word can be put into normal form in quadratic time. Based on D. Holt's `kbmag` package, Magma can attempt to find an automatic structure and, if successful, determine finiteness, perform element arithmetic, and compute growth functions.

8.2 Permutation Groups

8.2.1 Constructions for Permutation Groups

Permutation groups in Magma may be defined by giving the support set and a generating set for the group. In addition, Magma provides a large number of standard constructions for permutation groups, as well as databases of permutation groups.

Standard constructions include the symmetric and alternating groups, as well as projective linear groups and the other families (unitary, symplectic, orthogonal) of projective classical groups, affine linear groups, projective Suzuki groups, cyclic groups, abelian groups, dihedral groups, and extraspecial groups. Standard functions for constructing permutation groups from other groups include direct products, wreath products, and primitive wreath products. Coset enumeration in arbitrary groups (including permutation groups) also leads to permutation groups.

Magma includes a database of all transitive groups of degree up to 48 (Butler, Hulpke, Cannon, Holt) and a database containing all primitive permutation groups of degree up to 4095 (Sims, Roney-Dougal, Unger, Coutts). Magma also contains a version of the online finite simple groups database of Wilson *et al.*, which contains many permutation representations of simple and nearly simple groups.

8.2.2 Base and Strong Generating Set

To determine the order of a permutation group and most other structural information it is necessary to first construct a suitable representation of the set of group elements. In 1970 C. Sims introduced the notion of a *base and strong generating set* (BSGS) for a permutation group G. The base is a sequence $B = \beta_1, \ldots, \beta_k$ of points chosen from the set on which G acts such that the only element of G that fixes B pointwise is the identity. Thus, B defines a chain of stabilisers $G \supset G_{\beta_1} \supset G_{\beta_1,\beta_2}, \supset \ldots, \supset G_{\beta_1,\beta_2,\ldots,\beta_k} = 1$ from G down to the trivial subgroup. The strong generating set associated with B is a subset of G containing a generating set for each of these stabilisers.

The construction of a BSGS for a large group of high degree is a challenging task. Magma contains a number of different algorithms for this task. The usual strategy for a hard example is to apply the Random Schreier–Sims to get a probable BSGS and then verify its correctness using the Sims–Cannon–Brownie algorithm. This has been successfully applied to short base groups of degree up to 10^8.

Having a BSGS for a group G immediately gives the order of G and this suffices for some applications, such as completion of the proof of the Hall–Paige conjecture. The availability of a BSGS also provides an efficient method for testing membership of G. Having a BSGS for G allows a BSGS for any subgroup H of G to be constructed efficiently. These capabilities make efficient the computation of the normal closure of any subgroup H. This in turn enables various series, such as the derived series, to be computed efficiently.

8.2.3 Natural Actions of a Permutation Group

Fundamental to the study of a permutation group G are various actions of the group on the support Ω itself as well as subsets and partitions of Ω. We first consider the action on Ω. Subgroups stabilising points and sequences of points of Ω are efficiently computed applying a technique known as *base change* to the group's BSGS. The next class of groups related to Ω are the stabilisers of subsets and partitions (both ordered and unordered). These are harder to compute and in most cases they have to found using a backtrack search.

Magma's functionality for determining the G-invariant subsets and partitions of Ω is highly optimised. The corresponding actions on orbits and block systems may be constructed, along with the corresponding homomorphisms onto quotients of G. These homomorphisms support calculation of the image and preimage of elements and subgroups. While these actions will seldom be faithful, these actions can be used to design algorithms which work by reducing the problem down to a primitive group. In the current version of Magma this approach is used to find the Sylow p-subgroup for any prime divisor p of $|G|$.

8.2.4 Conjugacy Classes

Many applications need the conjugacy classes of elements of a group G. For example, this information is needed when computing irreducible characters or representations. The central algorithm used is a lifting algorithm which reduces the problem to determining the conjugacy classes for each non-abelian composition factor of G. In the case of the classical groups, theory and code have been developed by E. O'Brien and D. Taylor for all types of classical groups. For the other types of simple group an inductive method devised by G. Butler is used. For soluble groups a lifting method is used and can be very efficient, particularly in the case of p-groups.

8.2.5 Normal Structure

The BSGS makes it straightforward to compute the normal closure N_H of a subgroup H of G. One simply initialises the subgroup N_H to be equal to H and then forms its closure under the operation of conjugating its generators by those of G. Being able to compute normal closures makes it straightforward to compute the derived subgroup, the derived series, and lower central series of G.

Many advanced permutation group functions rely on constructing certain normal subgroups of the group and representations of the corresponding group quotients. These include methods by W. Unger (Magma) for computing the soluble radical of the group and the method of J. Cannon and D. Holt (Warwick) for computing the socle of a permutation group, which W. Unger has recently extended to degree 2^{32}. The socle is found using a divide-and-conquer method using natural homomorphisms which reduce down to a primitive group. The socle of a primitive group is then found by application of the O'Nan–Scott Theorem.

Other standard normal subgroup constructions available in Magma and based on normal closure are derived series, lower central series, p-cores, Fitting subgroup, and minimal normal subgroups. Magma can also name the composition factors and chief factors of a permutation group, and find corresponding normal and subnormal series in the group.

8.2.6 Socle and Composition Series

In this section we consider the construction of several important normal subgroups of G as well as series. The key subgroup is the socle (subgroup generated by all the minimal normal subgroups). This is found by a divide-and-conquer method using natural homomorphisms which reduce down to a primitive group. The socle of a primitive group is then found by application of the O'Nan–Scott Theorem. Once the socle is available it is straightforward to find the composition series, composition factors, chief series, and chief factors.

A normal subgroup that is important for algorithmic purposes is the solvable radical, the largest solvable normal subgroup. The algorithm used in Magma is due to W. Unger (Magma).

8.2.7 Backtrack Search Algorithms

There are a number of subgroup (and conjugacy) functions which are are currently hard to compute. The more important include:

- Intersection of two subgroups
- Centraliser of an element; test two elements for conjugacy
- Centraliser of a subgroup
- Normaliser of a subgroup; test two subgroups for conjugacy
- Stabiliser of a subset of Ω
- Stabiliser of a partition (ordered or unordered) of Ω

The non-existence of polynomial-time algorithms for the general case of some of these problems has been established. In the case of permutation groups, J. Cannon and G. Butler developed a large family of backtrack search algorithms for this type of problem. Recently the partition refinement method developed by J. Leon was slightly improved and implemented by W. Unger. While it is often quite fast it is not hard to find examples where the complexity of the search leads to slower runtimes than desired.

8.2.8 Subgroups

A key capability for the analysis of finite groups is the ability to determine the maximal subgroups of a group G. In 2004, J. Cannon and D. Holt described an algorithm which reduced the problem to that of determining the maximal subgroups of the composition factors of G. Algorithms were then devised which can quickly produce the maximal subgroups of those simple groups which can arise in practice.

A function devised by J. Cannon and D. Holt can construct the subgroups of a permutation group up to conjugacy. A simpler version efficiently computes the normal subgroups. Subgroups satisfying various restrictions such abelian, nilpotent, soluble, regular, having a specific order, etc., can be obtained using the subgroups function together with an appropriate filter. As an example, the 111,004 conjugacy classes of subgroups of the sporadic simple group F_{22} of order 64,561,751,654,400 are computed in 231,300 seconds.

8.2.9 Automorphism Group and Isomorphism

The automorphism group of a finite group is normally computed using a lifting algorithm using a generalised elementary series. An algorithm due to J. Cannon and D. Holt is used for permutation groups. In the case of soluble groups, a version of the lifting algorithm developed by Michael Smith is used, while for p-groups an algorithm designed and implemented by E. O'Brien is employed. A new approach to computing the automorphism group of soluble groups has recently been developed by D. Howden. For each of the above automorphism group algorithms a slight variant has been produced which tests a pair of groups for isomorphism.

8.3 Matrix Groups

8.3.1 Introduction

A matrix group G may be defined over any ring R for which Magma has a method for computing the inverse of a matrix. The methods used to compute structural information of matrix groups depend upon the coefficient ring R as well as whether G is finite or infinite. Until quite recently, our techniques for computing with matrix groups of large degree were limited. However, the *Large Matrix Group* project led by C. Leedham-Green and E. O'Brien has developed a computational model known as Composition Tree which makes it possible to compute with matrix groups over finite fields that are too large for a useful BSGS to be constructed. The current approaches for computing structural properties of matrix groups that exist within Magma include the following:

1. If the ring R is a finite field then the group must be finite. If the group has moderate degree and it is possible to find a low dimensional subspace of the natural vector space for G whose orbit under G has length bounded by a million or so, then it is possible to construct a stabiliser chain representation for the group similar to that used for permutation groups (the *BSGS representation*). A similar procedure applies if R is the Euclidean Ring $\mathbb{Z}/m\mathbb{Z}$.
2. If the coefficient ring R is a finite field but the degree and size of the group are such that it is not possible to construct a useful BSGS representation then the group may be investigated using the Composition Tree method developed by Leedham-Green, O'Brien and others (the

Composition Tree representation). At present only a subset of the standard group structure operations are available for this type of group.
3. If the ring R is $\mathbb{Q}$, a number field, or a function field, then Magma can usually determine whether the group is finite or infinite. If G is found to be finite, then an isomorphic subgroup H of $GL(n, F_p)$ can be constructed putting us into cases (1) or (2).
4. If G is a finite group and the ring R is either a field different to the field types in (3) or a Euclidean Domain then it may be possible to construct a *BSGS representation* as above and thereby perform structural computations.

8.3.2 Construction of Matrix Groups

Matrix groups are usually defined by giving a set of generating matrices for the group. In addition to this, Magma provides a large number of constructions for standard matrix groups. Standard constructions include most families of finite groups of Lie type. Functions for constructing matrix groups from other groups include direct products, wreath products, and tensor wreath products.

Databases of matrix groups in Magma include:

- Matrix representations of the quasisimple groups in all characteristics (including zero) having degree up to 100. Eventually the database will include all such groups of degree up to 248.
- A version of the finite simple groups database, edited and managed by Robert Wilson *et al.*, which contains many matrix representations of nearly simple groups.
- The irreducible subgroups of $GL(n, p)$, where p is prime and $p^n \leq 2499$.
- The maximal finite subgroups of $GL(n, \mathbb{Q})$, for n up to 31.

8.3.3 Natural Actions of a Matrix Group

Let G be a matrix group defined over some reasonable ring (field or Euclidean ring). Let M be the natural module for G. Then G has a natural action on elements and submodules of M and functions exist to to compute their orbits. For groups defined over $GF(q)$ a number of special tools are provided for constructing or estimating (the length of) very large orbits.

Using the G-set machinery a G-action can be defined on a range of objects associated with M. However, at present the calculation of stabilisers is restricted to BSGS groups and the stabiliser request must be of a single object from some G-set. Working with the action of G on the cosets of a subgroup of G is supported.

8.3.4 BSGS Model

The BSGS approach was extended to matrix groups by G. Butler and J. Cannon. The orbits here are the G-orbits obtained by the action of G on the vectors of the natural module associated with G. Given that suitable orbits can be found, a BSGS is constructed using similar algorithms to those employed in the permutation group case. Given a BSGS most of the algorithms used for the permutation group case can be carried across to matrix groups.

The matrix group BSGS method works well provided that the degree of the matrix group is not too large, that suitable orbits can be found, and that the group is not extremely large. Consequently, BSGS is the preferred method when applicable.

When BSGS is not applicable and the matrix group is defined over a finite field, the Composition Tree method can be used. However, there are many computations available under BSGS that are not yet available under Composition Tree.

8.3.5 Composition Tree Model

Matrix groups present a challenge for computational group theorists since the base and strong generating set (BSGS) paradigm, which is the basis for permutation group computations, is restricted to those matrix groups having relatively small order and degree. A new paradigm, the *composition tree* (CT) method, developed specifically to overcome this problem, has only recently matured to the point where it can handle matrix groups beyond the limit of the BSGS method. The method applies to permutation groups and matrix groups over a finite field.

The CT is a binary tree whose vertices are sections of the group. The leaves of the tree will correspond to the composition factors of G. If the vertex labelled H is not a leaf then the left child of H will correspond to a normal subgroup N of H and the right child will correspond to the section H/N. The algorithm to construct a composition tree is extremely complex and it has only recently been made generally available in Magma. The CT datastructure includes the order of G and the composition factors of G; it also supports testing membership of matrices in G.

For CT groups, *normal closure* type algorithms exist for calculating the normal closure of a subgroup and the derived group. Various algorithms that work off the CT datastructure have been implemented for composition series, composition factors, chief series, chief factors, soluble radical, Fitting subgroup, centre, and Sylow subgroup.

The following CT group *lifting type* algorithms will work only if a permutation representation can be computed for G/L, where L is the soluble radical of G: centraliser, conjugacy of elements, normaliser, conjugacy of subgroups, conjugacy classes of elements, and maximal subgroups.

Consider a 40-dimensional integer matrix group of order 7,172,259,840. It takes Magma 1081 seconds to compute the lattice of 6 normal subgroups of the group.

Unlike the BSGS datastructure, the CT scheme is still under development. For example, when constructing a subgroup of a CT group it does not come equipped with a compatible CT structure (or any CT structure unless the user asks for it). The BSGS scheme in Magma automatically creates a BSGS for any subgroup in such a way that subgroups and homomorphisms can seamlessly pass between subgroup and group.

8.4 Abelian Groups

8.4.1 Finitely-Presented Abelian Groups

Abelian groups can be defined in a variety of ways but the preferred representation is in terms of generators and relations. Given an abelian group in terms of generators and relations then it is usually desirable to to present the torsion part as a direct sum of cyclic groups. This is achieved by computing the Smith Normal Form (SNF) of its relation matrix. The SNF code is capable of handling groups having tens of thousand generators and relations. Functions are provided that convert an abelian group given in any of Magma's representations into Magma's standard abelian group representation (generators and relations). The Hermite and Smith normal form algorithms are used to construct a normal form for subgroups and quotient groups.

In the case of finite abelian groups, the functions provided for general finite groups such as composition series, maximal subgroups, subgroup lattice, and automorphism group apply to abelian groups. Determining the order of an element or computing discrete logarithms can be difficult in the case of very large groups. The algorithms used for the discrete logarithm problem are improvements of those described by J. Buchmann, M.J. Jacobson, and E. Teske.

8.4.2 Generic Abelian Groups

The term generic abelian group refers to the situation in which a group A is described by giving a set X together with functions acting on X that perform the fundamental group operations for A. Specifically, the user must provide functions which compute the product, the inverse, and the identity. For efficiency, the user may also optionally specify the order and a generating set for the group.

Going from such a definition of an abelian group A to a presentation for A will often be very expensive and so a small number of operations are implemented so as to not require this information. The two key operations are finding the order of an element and computing the discrete logarithm of an element to a given base. For many abelian group operations, knowledge of a presentation is required and if such an operation is invoked, the structure of A (and hence a presentation) will be automatically constructed.

There are two possible ways of computing the structure of the group. If the order of the group is known (or can be computed) then it is relatively easy to construct each of the p-Sylow subgroups. If the order is not available, the structure is computed from a set of generators supplied by the user.

Chapter 9

Modules and Representation Theory

9.1 Module Theory

9.1.1 Vector Spaces

Representation of Subspaces

Magma contains broad range of facilities for computing with vector spaces. Let K be a field. In Magma, the standard K-vector space is taken to be the set of n-tuples over the field K, which is written as K^n.

There are two different ways for handling standard vector spaces, depending on how subspaces are presented to the user. Suppose V is the vector space K^n and S is a subspace of V of dimension d. The two ways of representing S are as follows:

- *Embedded*: Here S is presented as its natural embedding into V so elements of S are represented as elements of V explicitly (i.e., of length n) and the basis of S consists of d vectors in V.
- *Reduced*: Here S is simply the abstract vector space K^d so elements of S are represented as vectors of length d and the basis of S is the standard basis of cardinality d. At the same time, an embedding ϕ from S into V is maintained in the background so that elements of S can be mapped into V and related to other subspaces of V; many functions in Magma automatically use these ϕ mappings without the user needing to access them explicitly. The embedded representation is the usual way vector spaces are regarded in elementary linear algebra and is typically more convenient when working with vectors from various subspaces which are all to be related to the original K^n (particularly when working with linear codes and their words). On the other hand, the reduced representation is useful when working in a more abstract setting where operations are relevant to a reduced basis and so every vector space is to be considered simply as K^d (this representation is also necessary when working with modules with matrix action; see below).

To provide the user with the maximum flexibility, Magma supports functions for creating spaces with both forms of presentation: vector spaces with the embedded form are called *K-spaces*, while those with the reduced form are called *K-modules*. Usually a calculation commences with the definition of one or two vector spaces from which further spaces are created by the operations of forming subspaces or quotient spaces. Magma provides parallel creation functions which allow the user to choose the form of presentation. Once that choice has been made, all spaces derived from the initial spaces will follow the same presentation convention.

Operations with Subspaces

All standard operations on vector spaces are supported by Magma. One non-trivial operation is to compute the intersection of two subspaces; here Magma can use two different methods (depending on the dimensions), both of which map to asymptotically fast matrix multiplication. One can also create a vector space V with a given basis B (not necessarily a standard basis) and then repeatedly ask for the *coordinates* w.r.t. B of a vector $v \in V$ (expressing v as a linear combination of the elements of B).

Inner Product Spaces

Magma also allows the user to construct a vector space $V = K^n$ with an attached inner product matrix F. Here F is an $n \times n$ symmetric matrix which is used for computing the inner product of elements of V as follows: $(u, v) = u \cdot F \cdot v^{tr}$ for $u, v \in V$. This inner product for V is automatically inherited by subspaces of V and automatically used for elements of such by all relevant functions in Magma.

$Hom_K(U, V)$

A special type of vector space are the *Hom Spaces*: These are vector spaces of the form $K^{m \times n}$, consisting of all $m \times n$ matrices over K (for any $m, n \geq 0$) and their subspaces. A subspace of this type having dimension d is always represented in embedded form (since the reduced form would just be the standard K^d) and the basis consists of d matrices of size $m \times n$. All functions relevant to vector spaces are supported for Hom spaces (sometimes with some subtle modifications because of the form of elements). This includes the ability to express a matrix in terms of a user-provided basis for a Hom space.

Let U and V be K-vector spaces of dimensions m and n, respectively (which may be explicitly embedded in a parent space of higher dimension). The set of all linear transformations with domain U and codomain V is denoted by $Hom_K(U, V)$. Once bases have been chosen for U and V, $Hom_K(U, V)$ can be identified with $K^{(m \times n)}$. Magma contains a non-trivial algorithm to compute $Hom_K(U, V)$ for arbitrary U and V. If U, V are reduced then the algorithm is trivial, but if U, V are embedded then the space of matrices which give valid linear mappings from U to V is not easy to construct.

9.1.2 Modules with Scalar Action

Introduction

Let R be a general Euclidean ring (not necessarily a field). Magma provides extensive facilities for computing with the *scalar R-module* R^n, whose elements are just n-tuples over R, where the module action is scalar multiplication by elements of R. This is essentially a generalisation of a vector space where the base ring is no longer assumed to be a field.

Submodules of R-modules over Euclidean Domains

As for vector spaces, Magma provides both embedded and reduced representations. All fundamental operations which are applicable to vector spaces are also supported for scalar modules. The most significant difference is that bases of subspaces must be handled differently when R is not a field. If R is a domain (such as $\mathbb{Z}$ or $K[x]$ for a field K), then the Hermite Normal Form (HNF) is used to compute a basis of a submodule (see the section on matrices over Euclidean rings). This form gives a unique canonical basis for any submodule of R^n. As for vector spaces, the user can also construct a submodule with a given basis so that vectors can be expressed as a linear combination w.r.t. this basis.

Submodules of R-modules over Rings with Zero Divisors

If R is a ring with zero divisors (such as the residue class rings $\mathbb{Z}_m = \mathbb{Z}/m\mathbb{Z}$ with m composite, or $K[x]/<f>$ with f reducible), then submodules of R^n may not possess a basis in general (i.e., it may be the case that there does not exist a generating set of minimal cardinality which is linearly independent) and the standard HNF algorithm is inadequate. In this situation, Magma uses an algorithm to compute the *Howell Normal Form*, which is like the HNF but which potentially adds extra rows related to annihilator ideals; the result yields a unique canonical generating set for a submodule. This special form is needed even for a simple test as to whether a vector is in a submodule and also allows submodules to be compared for equality, etc.

Quotients of R-modules

Quotient modules are handled by the Smith Normal Form (SNF) algorithm. The SNF itself gives an abstract presentation of the quotient module and several transformation matrices are kept in the background to facilitate the natural mapping (and its inverse) between the original module and the quotient module. Handling this in general is quite non-trivial, since a quotient module may have both free and torsion parts; even if R is a domain (such as $\mathbb{Z}$), a quotient module may have torsion and so be equivalent to a module over a residue class ring with zero divisors.

Similarly to vector spaces, Magma also provides support for Hom Spaces over a Euclidean ring R: these are scalar R-modules of the form $R^{m\times n}$, consisting of all $m \times n$ matrices over R and their subspaces. The user can also compute $Hom_R(U, V)$ for arbitrary submodules U, V of R^n.

9.1.3 Modules with Matrix Action

Introduction

Let A be a subalgebra of the matrix algebra $M_{n\times n}(K)$, where K is a field. An A-module M can be regarded as the vector space K^n equipped with an *action* by A (multiplication on the right by elements of A). Magma provides a major package for computations with such A-modules. The main feature is a suite of algorithms for structural analysis of A-modules and several types of related finite-dimensional associative algebras. This has very many applications in areas such as representation theory as well as in the theory of algebras and number theory.

Let M be an A-module, where A is a subalgebra of $M_{n\times n}(K)$. A submodule of M is a vector subspace V of K^n which is invariant under the action of A (i.e., such that $v \cdot a \in V$ for all $v \in V, a \in A$). An A-module M is called *simple* or *irreducible* if there are no proper submodules of M (i.e., other than the zero submodule and M itself). If M does possess a proper submodule S then M is called *reducible* and one can form the quotient module $Q = M/S$. Proceeding with constructing proper submodules and quotient modules of these derived modules as much as possible, one can form a *composition series* for M as a chain of submodules:

$$0 \subset S_1 \subset S_2 \subset \ldots \subset S_{k-1} \subset M,$$

where the quotient of each pair of adjacent terms is simple.

Splitting Modules with the Meataxe

The fundamental computational tool for analysis of A-modules over fields is the so-called *Meataxe* algorithm of Parker (1984), which was later improved by Holt & Rees (1994). The Meataxe algorithm takes an A-module M and determines whether M is simple; if M is not simple then the algorithm also returns a proper submodule S of M. The algorithm works by constructing submodules generated by vectors from suitable generalised eigenspaces of random elements of A; either this yields a proper submodule with reasonable probability, or gives information whereby

M can be proved to be simple. If the Meataxe shows that a given submodule M is not simple and thus also returns a proper submodule S, then one can reapply the Meataxe recursively on S and the quotient module M/S and so on to construct a complete composition series for M.

Magma contains a highly optimised implementation of the Meataxe algorithm, which carefully minimises the memory usage as it proceeds (several dense matrices need to be managed in large examples). The eigenspaces of elements of A are formed via the asymptotically-fast matrix algorithms for multiplication and echelonisation. Furthermore, a key subroutine of the algorithm computes the invariant submodule of M generated by a given vector (traditionally called the 'Spin' algorithm); Magma's novel implementation of this algorithm also maps this problem to matrix multiplication, so this takes advantage of the optimised fast matrix operations for a wide range of fields (see the section on matrix operations). Thus Magma contains the only implementation of the Meataxe algorithm which maps the whole algorithm completely to fast matrix multiplication.

The original Meataxe algorithm was only designed for the case that A is an algebra over a finite field and is technically Las Vegas (with a very small probability of failure) but terminates on virtually all inputs in practice. Magma contains an extension of the Meataxe to fields of characteristic zero; this is not guaranteed to terminate on large examples for general A-modules because of the difficulty of finding singular elements of A when the coefficients are large, but works very well on many classes of inputs (see below).

$Hom_A(M_1, M_2)$ and Isomorphism

Suppose M_1 and M_2 are A-modules (for the same fixed algebra A). The *module of homomorphisms* $H = Hom_A(M_1, M_2)$ can be viewed as the set of all homomorphisms from the underlying vector space of M_1 to the underlying vector space of M_2 which are compatible with the action of A (i.e., H is the set of all homomorphisms h from M_1 to M_2 such that $h(v \cdot a) = h(v) \cdot a$ for all $v \in M_1, a \in A$). The *endomorphism ring* of an A-module M is simply $Hom_A(M, M)$; i.e., the set of homomorphisms of M into itself which are compatible with A.

An important problem in module theory is the testing of two modules for isomorphism. Suppose M_1 and M_2 are A-modules. If one of the modules is irreducible (which is easily determined first by the Meataxe) then Magma uses a straight-forward algorithm of Holt and Rees, using some information computed by the Meataxe for the irreducible module. Otherwise, Magma uses an advanced algorithm based on the general Hom-algorithm above, which searches for an invertible homomorphism or proves non-isomorphism via various criteria.

Decomposing Modules

Suppose that an A-module M possesses a proper submodule S. If there also exists a direct summand T such that $M = S \oplus T$, then M is called *decomposable*. Conversely, if M cannot be written as a direct sum of proper submodules, then M is called *indecomposable*. Magma contains an algorithm developed by C. Leedham-Green and A. Steel for determining whether such a module is decomposable or not. This algorithm depends on an analysis of the endomorphism ring of M (which is first computed by the above Hom-module).

Applications

The characteristic-zero Meataxe algorithm is used heavily in the number theory package of Magma when decomposing an ideal of an order O of a number field $K = \mathbb{Q}(\alpha)$ into the product of prime ideals of O.

9.2 Representation Theory

9.2.1 Character Theory

The general character theory machinery is currently restricted to characters defined over the complex field. Operations on characters include the following:

- Definition of class functions
- Construction of permutation characters
- Arithmetic on class functions: sum, difference, tensor product
- Frobenius–Schur indicator
- Norm, order, kernel, centre of a character
- Properties: generalized character, character, irreducible, faithful, linear
- Induction and restriction of a character
- Decomposition of a tensor power: orthogonal components, symmetric components
- Action of a group on the characters of a normal subgroup
- Decomposition of characters
- Class matrix, structure constants for centre of group algebra
- Table of ordinary irreducible characters (Dixon–Schneider algorithm, Unger's algorithm)
- Partition of character table into blocks modulo p.
- Schur indices of characters
- Representation corresponding to an irreducible character

The table of ordinary (complex) irreducible characters of a finite group is of critical importance. The algorithm of W. Unger (2006) is very efficient and allows Magma to compute character tables of very large groups. For example, J. Cannon used it to computer the 62 ordinary irreducible characters for the simple group J_4 of order $86,775,571,046,077,562,880$ given as a permutation group of degree $173,067,389$. This was the response of Cannon and Unger to a talk given by J.-P. Serre at Harvard in March 2015 where he pointed out that many recent theorems etc. depend upon information taken from the Atlas character tables but these tables lack proofs of correctness. We are able to compute the character tables of all but five of the 450 groups given in the Atlas. One table was found to be incorrect. Since the Unger algorithm uses a completely different method for computing character tables this is a completely independent verification.

9.2.2 Modular Representations

Introduction

Let A be the group algebra $K[G]$ for a finite group G and a finite field K. Unless otherwise stated, the A-modules discussed in this section will be defined over a finite field of characteristic p, where p divides $|G|$. In this section the machinery provided for investigating the irreducible A-modules for G is outlined. The issues are finding the irreducible constituents of a given A-module, finding the irreducible Brauer characters, determining all the irreducible A-modules, and finally, given an irreducible A-module N, find the projective indecomposable A-module U such that $U/Rad(U) \cong N$, where $Rad(U)$ is the radical of U.

Algorithms Based on the Meataxe

A highly efficient Meataxe algorithm for modules over $\mathbb{F}_q$ has been developed by A. Steel over a long period. The following tools, based on the Meataxe, are available:

- Split a G-module, irreducible constituents.
- Test irreducibility, absolute irreducibility
- Determine whether two G-modules are isomorphic

Brauer Characters

A Brauer character modulo p in Magma is represented as a class function (that is, element of a character ring) which is zero on p-singular group elements. In this format the standard character operations of addition, multiplication, induction, and restriction all apply directly to Brauer characters as they do to other class functions. Given the table T of ordinary characters of G and a prime p, a basic function partitions T into *p-blocks*.

A function is provided to construct the table of Brauer characters modulo the prime p. If G is soluble then its Brauer characters can be derived directly form the table of ordinary characters. In the case in which G is non-soluble, work is underway to produce many of the Brauer characters without first constructing the corresponding irreducible A-modules. However, at present irreducibles are first computed. The *decomposition matrix* D is also returned. The (i, j)-th entry of D gives the number of times the absolutely irreducible $K'[G]$-module j ($K <= K'$) occurs as a constituent of the mod-p restriction of the ordinary character i.

Irreducible G-modules

The method for constructing the irreducible $K[G]$-modules proceeds as follows. Let S be the list of irreducibles that have been constructed so far. A $K[G]$-module N that contains an irreducible not already in the list S is identified. Then N is split using the Meataxe and the new irreducibles added to S. This is the same general approach taken by Parker and others when constructing particular irreducible modules for simple groups. However, we have made this process entirely automatic as distinct from earlier work where intervention was needed at various points (particularly in regard to choosing a suitable module N).

Projective Indecomposable Modules

The indecomposable direct summands of A are projective as A-modules; they are known as the *projective indecomposable* A-modules. Furthermore, any finitely generated projective A-module is isomorphic to a direct sum of projective indecomposables.

The algorithm for computing projective indecomposable modules assumes that representatives of the irreducible $K[G]$-modules are known. The principal step of the algorithm for computing projective indecomposables is, for a given irreducible $K[G]$-module $L \in \mathcal{L}$, compute the unique projective indecomposable module $U \in \mathcal{U}$ with $U/Rad(U) \cong L$.

We can successfully compute difficult examples with p small for groups G up to order about 10^7.

9.2.3 Group Representations in Characteristic Zero

Splitting Modules with the Meataxe

A characteristic zero Meataxe algorithm for modules was developed by A. Steel in 2009 and is used for splitting general A-modules, where A is a finite dimensional matrix algebra defined over the rational field. This yields an effective algorithm for decomposing a module into indecomposable summands. If the module is a G-module for some group G, extensive use is also made of character theory. Representations associated with characters having non-trivial Schur indices are properly handled. The difficult problem of splitting homogeneous modules (direct sums of the same indecomposable) is handled by decomposing the endomorphism ring of the module via a maximal order. Modules having dimensions in the several hundreds are routinely split into indecomposable modules.

Splitting Homogeneous Modules

Critical to the success of the characteristic zero Meataxe algorithm is an effective method to split a *homogeneous* module M into a direct sum of k isomorphic copies of a single indecomposable module S. This is achieved by first computing a maximal order O of the endomorphism ring R of M via an algorithm developed by G. Nebe and A. Steel (2009) and by then finding a singular element of O via a LLL-based heuristic search algorithm. Repeating the process gives the direct sum into the multiple copies of the indecomposable S. This also yields a practical algorithm for proving a homogeneous module to be indecomposable. Finally, the main decomposition algorithm first uses asymptotically-fast modular algorithms to split the module into homogeneous components and then splits each component by the above method.

Condensation

The technique of *condensation* allows one to analyse modules and algebras with dimensions much larger than possible with the direct Meataxe algorithm. The condensation of an algebra A involves working with a smaller associated algebra eAe (where e is a suitable idempotent in A) which is *Morita equivalent* to A, which means that the algebras have exactly the same representation theory. For algebras which are group algebras, there are specific methods for producing suitable idempotents, coming from group theory, and consequently there has been much development of theory and practical algorithms for condensation and applications. Magma provides tools for the construction and analysis of the condensation of:

- Permutation modules;
- Tensor products of existing modules;
- The induction of existing modules to a supergroup. These tools handle modules defined either over finite fields or characteristic zero fields.

Constructing Irreducible Rational Representations

An algorithm for computing irreducible $\mathbb{Q}[G]$-modules for a finite group G was developed by A. Steel in 2009. Given a rational character of G, the algorithm proceeds by locating a (typically reducible) module that contains the desired module; this module may arise from permutation, induced or tensor representations. Then using the Meataxe described above, the module M is split thereby, yielding the required irreducible module. A novel component of this algorithm is an automatic search for a suitable condensed module so as to reduce the dimensions of the modules that have to be split. The algorithm controls the growth of coefficients at every stage, thus returning modules whose actions are usually defined by matrices with very small integral entries. A variant of the algorithm is also provided which determines all irreducible $\mathbb{Q}[G]$-modules for G. The machinery has been used to construct irreducible $\mathbb{Q}[G]$-modules having dimension well over a thousand in favourable circumstances.

As an example, the irreducible degree-2024 rational representation of the Conway sporadic simple group Co_2 can be computed in 4350 seconds. This involves condensing a degree-2300 permutation representation of the group down to a dimension-82 module and decomposing this via the above Meataxe. The resulting representation has integral entries with at most 2 digits.

Chapter 10

Number Theory

10.1 Elementary Number Theory

10.1.1 Introduction

It is essential for a general computer algebra system to include good algorithms for the fundamental problems of integer factorisation and primality testing. The latter breaks into two separate problems: (fast) probabilistic primality testing, and primality proving (providing a certificate). All three problems are very frequently needed (for integers of moderate size) in algorithms in all areas of the system. Magma does not aim to be state-of-the-art but to provide fast reliable routines suitable for general purpose use. For record factorisations and so on, one would naturally use custom software.

10.1.2 Factorisation

Magma's automated factorisation routine, by default, uses the following techniques controlled by parameters depending on the size of the input: trial division, checking for Cunningham factors, SQUFOF, Pollard methods, elliptic curve factorisation, and (finally) multiple polynomial quadratic sieve. The implementation of ECM (together with the Pollard methods) is a version of Zimmerman's GMP-ECM which has been installed in Magma. Magma also contains a number field sieve implementation (Contini); this must be called explicitly as it is not incorporared in the automated routine. The quadratic sieve and number field sieve are able to use multiple cores (the user must start the additional processes manually).

The factoristion of integers of the form $n = b^k \pm 1$, where b and k are not too big, uses R. Brent's `factor` algorithm, which employs a combination of table-lookups and attempts at "algebraic" factorisation (Aurifeuillian techniques). The tables contain 385,251 factors f of numbers $b^k \pm 1$, where $b < 10000$, $k < 10000$, and $f > 109$. For $a < 12$ these are mainly from the Cunningham tables (for k up to 1200), while for $13 < a < 99$ they are mainly from the Brent–Montgomery–te Riele extension of the Cunningham tables. For more information consult Richard Brent's website. The ability to factor integers of the form $b^k - 1$ is of considerable importance when working with the multiplicative group of a finite field.

10.1.3 Primality

Primality testing/proving is hard coded for reasonably small integers (it has been established by brute force computation that a certain procedure suffices). For other integers, the default probabilistic test consists of twenty iterations of the Miller–Rabin test. When a proof is desired, the probabilistic test is followed by calling an implementation of elliptic curve primality proving

(Morain). A primality certificate can be obtained from the same routine (in terse form, then another function can translate this to a human-readable mathematical proof.) An ingredient in the ECPP algorithm is a table of precomputed "Weber" polynomials (related to modular curves). The Magma routine calls on Morain's extended (1998) table.

10.1.4 The Ring $\mathbb{Z}/N\mathbb{Z}$

The ring $\mathbb{Z}/N\mathbb{Z}$ (for composite N) is a fundamental object which holds some nontrivial (computational) problems. Many algorithms require one to calculate square roots mod N or a primitive root mod N. Efficient algorithms for these tasks are provided in Magma. Dirichlet characters modulo N are essentially characters on the multiplicative group of $\mathbb{Z}/N\mathbb{Z}$. They are basic objects used in constructions in all areas of number theory, from elementary problems to complicated objects such as L-functions and modular forms, as well as group theory. Magma contains a well developed package for dealing with Dirichlet characters with all these purposes in mind.

10.1.5 Solving Conics

One of the simplest Diophantine equations encounted is Pell's equation: to find integer solutions to $x^2 - dy^2 = 1$. The theory is well known; this can be solved by calculating the continued fraction of $\sqrt{d}$ (and Magma can perform standard continued fraction calculations), or it can be solved by calculating units in the ring $\mathbb{Z}[\sqrt{d}]$.

Another Diophantine problem is to find an integer solution to a conic equation $Ax^2+By^2+Cz^2 = 0$. (Finding one solution yields a parametrisation of all solutions.) As it happens, conics are the most important Diophantine equations from the point of computational algebra generally; solving conics is a necessary step in algorithms for decomposing algebras and constructing group representations. Magma contains a good implementation (by M. Watkins) of Simon's algorithm for solving conics with integer (or rational) coefficients, and also Simon's algorithm for the more general problem of finding an isotropic subspace for a quadratic form in more variables.

10.2 Algebraic Number Theory

10.2.1 Introduction

Algebraic number theory is concerned with number fields and univariate function fields. A *number field* is a finite degree field extension of the field of rationals while a *univariate function field* is a finite degree field extension of the field $k(t)$ of rational functions over k. For function fields whose constant field k is a finite field, the arithmetic theory is largely parallel to that of number fields.

10.2.2 Basic Arithmetic

Magma provides an environment for performing calculations in number fields and function fields efficiently, but at the same time in a very flexible manner. Fields may be constructed as extensions defined by one or more polynomials, or as towers of extensions. Alternative representations of these may then be constructed; for instance, as a single extension or with an optimal polynomial. Relationships between related fields and their orders are carefully maintained so that elements and ideals can be easily moved between related structures.

10.2.3 Local Rings and Fields

The most commonly encountered local fields are the field of p-adic numbers and the field $k((t))$ of power series over a (finite) field k. In addition, one may form finite extensions of these. In algebraic number theory, local fields arise as completions of "global" number fields and function

fields. The most familiar example is for the field of rational numbers: its completions consist of the field of real numbers and the field of p-adic numbers $\mathbb{Q}_p$ for each prime p.

Magma includes several implementations of local rings and fields ($\mathbb{Z}_p$, $\mathbb{Q}_p$, $k((t))$, and finite extensions of these). The lowest-level implementation is for speed, while the highest level is for convenience, allowing calculations to be performed in the most user-friendly way, where each element has its own relative precision. Relative precisions are also handled smoothly when passing between extensions. Beyond field arithmetic, the most important algorithms provided are those for factorisation (Pauli) and roots of polynomials over local fields.

10.2.4 Subfields

Magma contains an implementation of the algorithms developed by Klüners, van Hoeij, and Novocin for calculating all subfields of a given number field or the field defined by a given polynomial. The algorithms rely mainly on polynomial factorisation rather than heavy computations concerning the structure of the field such as maximal order. Some variants of the algorithm use LLL techniques. They are effective for fields of rather large degree. For instance, the subfields of $x^{40} - 2$ can be obtained in a few seconds.

10.2.5 Maximal Order

The maximal order (also known as the ring of integers) of a field is the starting point when it comes to understanding the number theoretic properties of the field — after all, one must first have the integers before one has the primes, and completions, and so on. Magma has efficient implementations of the Zassenhaus "Round 2" and "Round 4" algorithms which are generally quite adequate for fields arising in applications. In addition Magma has special algorithms for fields that are expressed as Kummer or Artin–Schreier extensions.

10.2.6 Ideal Class Group

The class group is a key invariant of a number field, along with the regulator which measures the size of the unit group. For general number fields, the basic method is Buchmann's subexponential algorithm. One obtains a presentation of the class group where the generators are the primes up to some bound, by searching for enough elements for which all prime factors are among those primes; these elements are then the relations. Magma's main routine is a recent implementation (S. Donnelly) which boasts a number of novel features which serve to optimise the strategy.

10.2.7 The Galois Group

Let f be a polynomial whose coefficients belong to one of the fields $\mathbb{Q}$, a number field, a finite extension of $k(t)$ where k is a finite field, or $\mathbb{Q}(t)$. Then the Galois group of the *splitting polynomial* of f can be computed in Magma. The general approach is based on ideas of Stauduhar published in 1973. A much more efficient version of that algorithm was developed around 2008–2009 by C. Fieker and J. Klüners, with further work undertaken recently by S. Elsenhans. In contrast to earlier algorithms, there is no *a priori* limit on the degree, and no requirement for large precomputed tables. The Galois group is determined not only as an abstract group, but also as a permutation group acting on the set of roots of the polynomial in some splitting field.

10.2.8 Class Field Theory

Class field theory classifies the abelian extensions of a global field by giving an isomorphism between their Galois groups and ray class groups of the original field. While the theory was completed in the first decades of the twentieth century, the explicit construction of the field extensions remains a major open problem. Explicit construction means finding an explicit polynomial that defines

the field. There is a complete solution, in terms of values of transcendental functions, only for quadratic number fields. This method has an analogue for function fields using Drinfeld modular curves.

In Magma, abelian extensions can be constructed first in an abstract form, using ray class data rather than explicit polynomials. To construct them explicitly, an algorithm based on the standard approach using Kummer theory is implemented for number fields and function fields. For function fields, there is an additional algorithm based on Drinfeld modules. These methods were all developed by C. Fieker.

10.3 Local Rings and Fields

10.3.1 Introduction

The most commonly encountered local fields are the field of p-adic numbers and the field $k((t))$ of power series over a finite field k. In addition, finite extensions of these fields may be constructed. In algebraic number theory, local fields arise as completions of "global" number fields and function fields. The most familiar example occurs in the case of the field of rational numbers; its completions consist of the field of real numbers and the field of p-adic numbers $\mathbb{Q}_p$ for each prime p. Local fields are an indispensable tool in the theory of fields, much of which is developed by analysing their "local properties". This approach is mirrored in many algorithms.

10.3.2 Implementation

Magma includes several implementations of local rings and fields ($\mathbb{Z}_p$, $\mathbb{Q}_p$, $k((t))$, and finite extensions of these). The lowest level implementation is for speed, while the highest level is for convenience, allowing calculations to be performed in the most user-friendly way, where each element has its own relative precision. Relative precisions are also handled smoothly when passing between extensions. Beyond field arithmetic, the most important algorithms provided are those for factorisation (S. Pauli) and roots of polynomials over local fields. One of the implementations of p-adic rings and fields is discussed in the next subsection.

10.3.3 Exact p-adic Rings and Fields

Exact p-adic rings and fields have been implemented by C. Doris. Such structures and their elements can be approximated to have as much precision as the objects they depend upon allow, sometimes to arbitrary precision. A "lazy" representation of elements is used whereby a record of their construction is kept but an approximation of an element is only computed when needed.

Elements of exact p-adic rings and fields have additional intrinsics which apply to them but not to elements of ordinary p-adic rings and fields. These intrinsics mitigate the infinite nature of the valuation intrinsic for exact zeros.

A polynomial ring type defined over an exact p-adic ring or field is available. Its elements are referred to as "exact polynomials". Also an ordinary polynomial ring can be created over these exact p-adic rings and fields. Polynomials can be converted to "exact polynomials" where the properties of being weakly zero or weakly equal are recognized. The factors and roots of an exact polynomial can be computed.

10.3.4 Applications

An important application of local fields is to the calculation of a maximal order for a number field or a function field. Once the discriminant of the field has been factored, finding a maximal order can be reduced to "local calculations". Local rings and fields have many applications outside

algebraic number theory. Localisation is an important tool in algebra, especially in algebraic geometry. In practice, this means performing calculations in (some finite approximation of) a complete ring. For example, analysis of singularities on curves and surfaces typically involves calculating in power series rings. In a different direction, the most efficient methods for counting points on curves are p-adic in nature and are implemented in Magma using p-adic rings. For this application the speed of the p-adic arithmetic is critical.

10.4 Lattices

10.4.1 Introduction

Integral lattices are a fundamental topic in computational algebra, both for their own sake and also as the basis for numerous applications. For example, the LLL algorithm is used in such diverse places as polynomial factorisation, cryptography, solution of conics (and higher dimensional quadratic equations), computation of character tables of groups, and recognition of algebraic numbers from real, complex, or p-adic approximations. Indeed, these are all applications based on an ability to find *short vectors* in a lattice, while LLL itself actually has a number of additional uses. Magma additionally provides routines for other (stronger) types of reduction, such as the HKZ or BKZ algorithms, which can be of use in specific problems.

Magma also includes the best available algorithm for computing automorphism groups of lattices. Another novel feature is the ability to use the short vector enumeration routines to recognise the Θ-series of a lattice as a modular form, exploiting Atkin–Lehner involutions when applicable.

Additional functionality for lattices includes the computation of Voronoi cells, genus and neighbour theory, and Hermitian structures (including over quaternion algebras). Finally, Magma provides access to standard lattice constructions, and contains a version of the Nebe–Sloane lattice database.

10.4.2 Reduction of Lattices

As noted in the Introduction, the workhorse algorithm for lattice applications is typically the LLL algorithm of Lenstra, Lenstra, and Lovász. This works by finding a lattice basis for which an explicit (and small) bound can be given regarding the size of vectors in terms of linear combinations of the basis vectors. Although, in general, finding a vector having the *smallest norm* is an NP-complete problem, the LLL algorithm usually succeeds in finding lattice vectors that are nearly as short, and much closer to the true minimum than the typical worst-case bounds would imply.

The Magma LLL implementation is a modern robust version of the LLL^2 algorithm by Nguyen and Stehlé, which handles underflow and numerical stability problems at very little cost of overhead, and provides completely rigourous bounds on the floating-point operations when desired. This was implemented by Stehlé during a two year secondment to the Magma group.

Magma also contains stronger forms of lattice reduction, most particularly BKZ (block Korkine–Zolotarev) and HKZ (a Hermite version, corresponding to the whole lattice as a block). These can be useful when searching for small vectors in a lattice (as described in the next subsection), as the enumeration cost can be notably smaller. Again the implementation is due to Stehlé.

Finally, Magma has routines for pair and Seysen reduction, the former being a much simplified reduction scheme (only pairs of vectors are considered in the conditions for a lattice to be reduced, rather than blocks as in LLL), while the latter is useful for (say) reducing a lattice and its dual simultaneously.

10.4.3 Enumeration of Short Vectors

Magma contains a state-of-the-art implementation of short vector enumeration due to Stehlé. The method is completely rigorous (using explicit error bounds of Pujol and Stehlé), and additionally is more efficient than previous methods of Kannan and Fincke–Pohst or Schnorr–Euchner. Furthermore, in applications where an exact count of short vectors is not needed, one can use the idea of pruning arrays (essentially, search in the parts of the ellipsoids which contains the most volume) to expedite the process in a probabilistic manner.

These methods for finding short vectors were vital in the proofs of lattice extremality for 80-dimensional lattices that are described below. One would typically find, say, 10^5 short vectors of norm 10 in a given basis, apply a random transformation and re-apply LLL (and possibly BKZ), then find 10^5 short vectors in the next basis, etc. In the presence of a sufficiently large (known) automorphism group, this is a viable path to finding all vectors of a given length in a suitable manner.

10.4.4 Theta Series as Modular Forms

The Θ-series of a lattice (usually taken to be integral) is defined as $\Theta(L) = \sum_v q^{|v|}$, or possibly by replacing $q \to \sqrt{q}$ when the lattice is even. This is a modular form of weight $d/2$ where d is the lattice dimension (or rank), and thus specifying finitely many coefficients suffices to identify the Θ-series in the modular forms space. The level of the modular form is related to the determinant of lattice, essentially having the same prime divisors.

Watkins and Donnelly have worked out an efficient scheme to compute the modular form for a wide variety of lattices. In particular, their method uses the Atkin–Lehner involutions on the modular forms, as transferred to partial duals of the lattice, to reduce the amount of coefficients needed by a factor of 2^w, where w is the number of such involutions (essentially the number of prime divisors of the level or lattice determinant). Given that the enumeration methods themselves scale in a manner dependent on the dimension, this is a tremendous time-gain in practice (easily a factor of a million or more in typical 32-dimensional cases). The method is most robust in dimensions divisible by 4 with lattices of square determinant; in this case, the modular form is for a Γ_0 congruence subgroup, and the Atkin–Lehner involutions take a simpler form. In general, one can append a form of small dimension to reduce to this case, removing the effect of its Θ-series by taking a quotient in the final result.

In this manner, at the request of Neil Sloane, a modular form has been produced for almost all the lattices of dimension up to 36 in the Nebe–Sloane database.

10.4.5 Automorphism Group

Magma contains a hightly efficient backtrack search program to compute the automorphism group of a lattice. This was developed and implemented by W. Unger (Magma), and operates by finding images of basis vectors and using invariants (such as Bacher polynomials) which must be preserved by automorphisms. The input to the program is a (complete) list of short vectors, computing as previously described.

This has recently been used by G. Nebe to show that her new extremal unimodular 48-dimensional lattice has automorphism group of size 1200 (and no larger). This computation took two months and involved the 26,208,000 (pairs of) minimal vectors of norm 6. This example is particularly hard because there are very few useful invariants that are easy to compute, and so many images of the minimal vectors must be typically be considered.

Additional work of automorphism groups and extremal unimodular lattices has been done in Magma group by D. Stehlé and M. Watkins. For instance, they showed that a 80-dimensional

lattice associated to the extended binary quadratic code of length 79 was indeed extremal (a long outstanding question), via enumerating 7,541,401,190,400 vectors of norm 10 and using the nonnegativity of the Θ-series. The process was expedited by use of the automorphism group, which here is $SL_2(\mathbf{F}_{79})$. Watkins was later able to show extremality for another 80-dimensional lattice, and computed (provably) the full automorphism group for these examples and others of similar size.

Magma can also find automorphisms which fix an additional quadratic form (other than just the Gram matrix). This allows one to determine Hermitian automorphism groups over imaginary quadratic fields, and also over quaternionic structures. For instance, Magma takes 25 seconds to compute that the quaternionic automorphism group of the Leech lattice over the standard Hamiltonians has order 503,193,600. One can also invert the process; for instance, one can take the irreducible 6-dimensional representation of the group $SU(3,3)$ and compute the invariant form for its G-module over the quaternion algebra ramified at 3 and ∞, realising the Coxeter–Todd lattice in this manner.

10.5 Elliptic Curves over $\mathbb{Q}$ and Number Fields

10.5.1 Introduction

The study of elliptic curves over the rationals and number fields are motivated by their relevance to major theoretical questions such as the conjecture of Birch and Swinnerton-Dyer which is one of the millennium problems. A practical application of computational techniques is to find the solutions of certain classes of diophantine equation.

Much of arithmetic geometry grew out of the study of Diophantine equations, with the basic problem being to describe the set of integral or rational solutions to a system of polynomial equations. Since the mid twentieth century, Diophantine equations have been studied more and more from the point of view of the geometry of the curve or variety they define, as this approach has yielded more powerful results than elementary arguments.

Elliptic curves in Magma can be defined over $\mathbb{Z}$, $\mathbb{Q}$, a number field, a function field, a p-adic field or a finite field. The tools for creating elliptic curves and for computing basic information about them are largely common for each of the above rings. In this chapter we concentrate on curves over $\mathbb{Q}$ and number fields.

10.5.2 Curves over the Rationals

Below is a list of some information that can be obtained from an elliptic curve over $\mathbb{Q}$:-

- Local invariants: Conductor, primes of bad reduction, Tamagawa number, trace of Frobenius.
- Kodaira symbols.
- Determine whether the curve has complex multiplication or not and, if so, also returns the discriminant of the CM quadratic order.
- Isogenous Curves: The set of curves $\mathbb{Q}$-isogenous to a rational elliptic curve.
- Heights and Height Pairing: Naive height, canonical height, height pairing, regulator, Silverman bound, Siksek bound, p-adic height.
- Analytic Information: analytic rank, periods, elliptic exponential, elliptic logarithm, root number, conjectural regulator, modular degree, others.
- Integral points and S-integral points.

10.5.3 Curves over a Number Field

Below is a list of some information that can be obtained from an elliptic curve over a number field K:-

- Local invariants: Conductor, places of bad reduction, local reduction data at a prime ideal, the reduction of an elliptic curve at a prime.
- Determine whether the curve has complex multiplication or not and, if so, also returns the discriminant of the CM quadratic order.
- Heights and height pairing: Naive height, Néron-Tate height, Local height.
- Integral points.
- Analytic information: Analytic rank, periods, elliptic exponential, root number, others.
- Elliptic Curve Chabauty: This refers to a method for finding the rational points on a curve, if the curve admits a suitable map to an elliptic curve over some extension field. The method was developed by Nils Bruin.
- Analytic Information: Root number, analytic rank, conjectural regulator and Conjectural Sha.

10.5.4 Elliptic Curves of Given Conductor

A routine is provided for finding elliptic curves with given conductor, or with good reduction outside a given set of primes. The aim is not to provably find all such curves; in most cases, this would be a very difficult task using current algorithms. Rather the aim is to efficiently search for these curves, using a variety of techniques, taking advantage of all available information, and taking advantage of all the tools available in Magma which can be applied to the problem.

The routine is very much more effective when some traces of Frobenius are known for the desired curve. The principal application of this tool is for finding elliptic curves that match known modular forms.

10.5.5 Mordell–Weil group

The Mordell–Weil theorem states that for an elliptic curve E defined over a global field F (such as $\mathbb{Q}$ or a number field), the set E(F) of points on E with F-rational coordinates forms a finitely generated abelian group. This is called the Mordell–Weil group, and the $\mathbb{Z}$-rank of the free part is called the Mordell–Weil rank of E.

It is an open problem to give an explicit algorithm which can determine the Mordell-Weil group of any elliptic curve over $\mathbb{Q}$ (or any number field). The general approach is to use the method of n-descent, for one or more suitable integers n. There are two main problems with this. Firstly, n-descent cannot determine the rank of E if the Tate-Shafarevich group Sha(E) has elements of order n. It is an open conjecture that, for fixed E, this occurs for only finitely many values of n. The second problem is more practical: one cannot implement n-descent for general n.

Magma has implementations of descent over $\mathbb{Q}$ for n = 2, 4, 8, 3, 9, 5, 6, 12 as well as for isogenies of various degrees. It is the only software that has a complete implementation of 2-descent over number fields (in the sense that 2-coverings are reduced and therefore can be used to search for points). In addition, the Cassels-Tate pairing on the 2-Selmer group is implemented over all global fields, and also on the 4-Selmer group over $\mathbb{Q}$. All these techniques (and some others) are combined in the main functions to determine ranks and generators.

The method of descent furnishes a "meta-algorithm" which, if standard conjectures hold, must succeed for all curves. It would be hard to encode this entire method in a finite program. Nevertheless, Magma goes much further than any other software, both for curves over $\mathbb{Q}$ and also over general number fields; it contains by far the most powerful array of tools available anywhere for obtaining rank bounds and for finding generators.

The first and simplest form of descent is 2-descent (implementations by G. Bailey, N. Bruin, M. Watkins, S. Donnelly). Magma contains the only implementation of *full* 2-descent over general number fields; that is, it has routines to express the 2-descendents as hyperelliptic curves and to find models of these that are nice enough to be well-suited for searching for points. The next simplest form is 4-descent (M. Watkins, N. Bruin). The remaining techniques are implemented only for curves over $\mathbb{Q}$: 3-descent (M. Stoll, S. Donnelly, T. A. Fisher); 8-descent (Stamminger, T. A. Fisher); 6- and 12- descent by combining the earlier descents (T. A. Fisher); 9-descent (B. Creutz). Excluding 9-descent, these implementations all produce nice models of covering curves which are useful for searching for generators; the most powerful of them for this purpose is 12-descent, which is often as effective as the complementary Heegner point method (M. Watkins) where both are applicable.

10.5.6 Points

A feature is the algorithm developed by S. Donnelly for finding integral points on an elliptic curve over either $\mathbb{Q}$ or a totally real number field. This assumes that generators for at least part of the Mordell–Weil group can be found.

A Heegner point package developed by M. Watkins (Magma) is designed to compute the generator of a rank 1 elliptic curve over $\mathbb{Q}$. S. Donnelly has added additional functions that make it possible to compute the Heegner point directly over the Hilbert class field, rather than its trace to $\mathbb{Q}$. In addition, this machinery allows a real approximation of the Heegner point to be computed on a 2-cover or 4-cover of the elliptic curve, greatly expediting the process in some cases.

10.5.7 Heights of Points

Magma is able to compute the p-adic height of a point on an elliptic curve over $\mathbb{Q}$ for a good ordinary prime $p \geq 5$. This is achieved using Kedlaya's algorithm to compute the Monsky–Washnitzer cohomology, and then applying an algorithm of Mazur, Stein, and Tate, as improved by Harvey. This has been used by various people, including J-S. Mueller, in computations with p-adic variants of the BSD conjecture, and can also be used in conjunction with Iwasawa theory to bound the p-part of the Tate–Shafarevich group.

10.5.8 Morphisms

Four types of maps between elliptic curves may be constructed: isogenies, isomorphisms, translations, and rational maps. A large number of constructions for isogenies are provided together with tools for working with them. Given two elliptic curves defined over the same field, a function is provided which determines whether the curves are isomorphic and if so returns the isomorphism. The endomorphism ring and automorphism group of an elliptic curve can be computed.

10.6 Hyperelliptic Curves over $\mathbb{Q}$ and Number Fields

10.6.1 Introduction

A hyperelliptic curve, which is taken to include the genus one case, is given by a nonsingular generalized Weierstrass equation $y^2 + h(x)y = f(x)$, where h(x) and f(x) are polynomials over a field K. The curve is viewed as embedded in a weighted projective space, with weights 1, g + 1, and 1, in which the points at infinity are nonsingular.

Functionality for hyperelliptic curves includes optimized algorithms for working on genus two curves over $\mathbb{Q}$, including heights on the Jacobian, and a datatype for the Kummer surface of

the Jacobian. For Jacobians of curves over finite fields, there exist specialized algorithms for computing the group structure of the set of rational points.

Much of the code for hyperelliptic curves, both initially and continuing, has been developed by M. Stoll.

10.6.2 Points

The hyperelliptic curve is embedded in a weighted projective space, with weights 1, g + 1, and 1, respectively on x, y and z. Therefore point triples satisfy the equivalence relation

$(x : y : z) = (\mu x : \mu^{g+1} y : \mu z)$,

and the points at infinity are then normalized to take the form (1 : y : 0).

For a curve C over $\mathbb{Q}$ of the form $y^2 = f(x)$ with integral coefficients, Magma can find the set of rational points such that the naive height of the x-coordinate is less than a specified bound. For a curve C over a number field, the set of rational points in a search region defined by two parameters can be found. The algorithm used is a sieve method described by N. Bruin.

10.6.3 Jacobians over Number Fields or $\mathbb{Q}$

The Jacobian of a hyperelliptic curve is implemented as the divisor class group of the curve. In particular, no equations giving the Jacobian as a variety ever appear. The Jacobian of any hyperelliptic curve can be created, but most of the interesting functionality is over finite fields, or for genus 2 over number fields or $\mathbb{Q}$.

Points on Jac(C) are represented as divisors on C. They can be specified simply by giving points on C, or divisors on C, or in the Mumford representation, which is the way Magma returns them.

For the remainder of this section the hyperelliptic curve C is assumed to have genus 2.

Let K be a field of characteristic different from 2. Consider a curve of genus 2, given by an equation $C : y^2 = f(x)$ where f(x) is a square-free polynomial of degree 5 or 6. Let J be the Jacobian of C. Here a *Richelot isogeny* is defined to be a polarized isogeny $\Phi : J -> A$ between principally polarized abelian surfaces, such that the kernel of Φ over the algebraic closure has group structure $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z}$. Magma can construct Richelot isogenous surfaces of J, double Richelot isogenies and isogenies having degree a power of 2.

10.6.4 Kummer Surfaces

The Kummer surface K associated to the Jacobian J of a genus 2 curve is the quotient of J by the inverse map. It can be embedded as a quartic hypersurface in projective 3-space whose only singularities are 16 ordinary double points. The Jacobian is a double cover of K ramified at these double points; they are the images of the two-torsion points on J. Resolving the singularities on K yields a K3-surface.

Currently, Kummer surfaces in Magma are not schemes, but are of type SrfKum. They can be used to perform arithmetic on the Jacobian without the need for reduction of divisors. The other nontrivial functionality that uses them is point searching.

The Kummer surface and arithmetic on it are implemented for Jacobians in arbitrary characteristic.

10.6.5 Mordell–Weil Group

The key problem now is to compute the finitely generated Mordell–Weil group of the Jacobian of the curve. For both problems, 2-descent algorithms implemented by N. Bruin (Magma) provide

bounds. Complementing this, points can often be found using efficient tools for searching on curves using algorithms implemented by N. Bruin, G. Bailey, and S. Donnelly.

Given a Jacobian J defined by an integral model over the rationals, Magma can determine all rational points on the Jacobian J whose naive height on the associated Kummer surface is less than or equal a given bound. If in addition the curve J has the form $y^2 = f(x)$ with integral coefficients then the rational torsion subgroup of J can be computed.

Height functions are provided for working with points on the Mordell–Weil group of the Jacobian of hyperelliptic curves over a number field k. In the case of genus two curves defined over $\mathbb{Q}$, the canonical height is computed using an algorithm of Müller and Stoll. The algorithm uses local error functions on the associated Kummer surface.

The 2-Selmer group of the Jacobian J of a hyperelliptic curve defined over $\mathbb{Q}$ or a number field gives an upper bound of the rank of the Mordell–Weil group and can be computed. A function is provided which determines the parity of the 2-rank of the Shafarevich–Tate group. Given the Jacobian J of a genus 2 curve these and other tools are the basis of a routine which attempts to compute the Mordell-Weil group J($\mathbb{Q}$).

10.6.6 Chabauty's Method

Chabauty's method is an algorithm for finding the rational points on a curve C of genus at least 2, that applies when the Mordell-Weil group of J(C) has rank less than the genus of C. It involves doing local calculations at some prime where C has good reduction.

For hyperelliptic curves of genus 2 over Q, two separate routines are available in Magma. The newer routine is far more powerful, with Chabauty calculations being combined with a "Mordell-Weil sieve" which combines information obtained using many different primes, to determine precisely the set of rational points on the curve. However, this routine does not apply to curves with no rational points. Both routines are applicable only when the Chabauty condition holds: for a genus 2 curve, the Jacobian must have Mordell-Weil rank 1. In addition a separate routine is provided for the easy case where the Jacobian has rank 0.

10.6.7 Analytic Jacobians

Suppose C is a curve of genus g defined over the complex numbers. The analytic Jacobian of the curve is an abelian torus and is constructed as follows: We view the complex points on the curve, $C(\mathbb{C})$, as a compact Riemann surface of genus g. Define ϕ to be a certain mapping from C to $\mathbb{C}^g$ and define L to be the image of the map from the first homology group, $H_1(C, \mathbb{Z})$, of C to $\mathbb{C}^g$. Then L turns out to be a lattice in $\mathbb{C}^g$ and so we get the complex torus $\mathbb{C}^g/L$. We then see that ϕ is a well-defined mapping from C into $\mathbb{C}^g/L$. We extend this mapping additively to the divisors of degree zero on C. The Abel-Jacobi theorem states that two divisors map to the same image under ϕ if and only if they are linearly equivalent and that ϕ is onto. So there is a bijection between the points of $\mathbb{C}^g/L$ and the points of the (algebraic) Jacobian.

So far the analytic Jacobian $\mathbb{C}^g/L$ is just a torus, but it can be made into an an abelian manifold by choosing a polarization and this makes possible a range of computations. Firstly, the small period matrix and the large period matrix can be computed. Then there is a function that maps the point (x, y) on the curve to the analytic Jacobian and a second function that makes it possible to move points between the analytic and algebraic Jacobians. A faster version of these mappings was provided in 2022 by T. Keller and M. Stoll.

Functionality is provided for finding isomorphisms and isogenies between different analytic Jacobians and also for computing the endomorphism ring of an analytic Jacobian.

10.6.8 Invariants

Over an algebraically closed field K of characteristic p not equal to 2, the isomorphism class of a genus g hyperelliptic curve $y^2 = f(x)$ with $f \in K[x]$ corresponds to the orbit of the binary form $z^2 g + 2f(x/z)$ under the classical action of GL(2, K).

For genus 2 Igusa managed to find a "universal set of invariants", which works in every characteristic, including 2. Given a genus 2 curve, Magma can compute its Igusa invariants and also Igusa-Clebsch invariants, Clebsch invariants and absolute invariants. The Magma package implementing the functions was written by Everett W. Howe with some advice from Michael Stoll and is based on some GP routines written by Fernando Rodriguez–Villegas.

In addition, a package of functions written by Reynald Lercier, Christophe Ritzenthaler and Jeroen Sijsling has been added. It contains, amongst other things, functionality for working with a different set of absolute (as opposed to weighted projective) invariants referred to as Cardona-Quer-Nart-Pujola invariants. These work in characteristic 2 as well as other characteristics, although the definitions are different in the two cases.

For curves of genus 3, the Shioda invariants may be computed for curves in characteristic 0 or characteristic at least 11. Also for genus 3, Maeda invariants can be computed.

The construction of a curve of genus 2 from a given set of Igusa–Clebsch invariants is not always possible, even in theory. But over a finite field or the rationals, Mestre gives a method for deciding whether it is possible and if it is, for finding such a curve. Mestre's algorithm was implemented by P. Gaudry. In any characteristic, the package of Lercier, Ritzenthaler and Sijsling produces a genus 2 curve from a given set of Cardona–Quer–Nart–Pujola invariants. They have also contributed a package for genus 3 hyperelliptic curves that produces a curve from a given set of Shioda invariants.

10.6.9 Automorphism Group and Isomorphism Testing

The genus 2 part of Lercier, Ritzenthaler and Sijsling cpode provides a function to determine the geometric automorphism group of a genus 2 curve in any characteristic by working with Cardona–Quer–Nart–Pujola invariants. They also provide a function that returns a list of all possible geometric automorphism groups for genus 2 curves over a given finite field and the number of isomorphism classes of curves with each possible group.

The genus 3 part of Lercier, Ritzenthaler and Sijsling provides the same for genus 3 hyperelliptic curves, working with Shioda invariants. Here, the characteristic has to be 0 or greater than 7. It is also possible to compute (geometric, reduced) isomorphisms and automorphisms by a fast direct computation in all characteristic different from 2 and genus greater than 1.

10.6.10 Superelliptic Curves

Magma also contains analogues of each of these hyperelliptic algorithms for the generalisation to superelliptic rather than hyperelliptic curves. The theory and implementations were undertaken by B. Creutz (Magma), and M. Mourao. A number of methods are provided which can provably determine all rational points on a curve of genus 2. These include Chabauty's method, implemented by N. Bruin, and a combination of Chabauty with the "Mordell–Weil sieve" method implemented by M. Stoll.

10.7 Curves Over Finite Fields

10.7.1 Introduction

This chapter describes some of the specialised facilities for elliptic and hyperelliptic curves defined over finite fields. Details concerning their construction, arithmetic, and basic properties may be found in Chapters Elliptic Curves and Hyperelliptic Curves.

10.7.2 Point Counting on Elliptic Curves

A problem of considerable importance is the calculation of the number of points on a curve over a finite field. For elliptic and hyperelliptic curves, Magma contains a range of algorithms adapted for different sizes of field and characteristic.

Magma contains an efficient implementation of the Schoof–Elkies–Atkin (SEA) algorithm, with R. Lercier's extension to base fields of characteristic 2. This is used for the case where the characteristic is not small. For small characteristic, canonical lift algorithms are used, while for moderate characteristic a deformation algorithm is used. The canonical lift methods use a range of techniques to lift a modular parameter of the curve to an unramified p-adic ring.

If a suitable Gaussian normal basis exists then the method of R. Lercier and D. Lubicz is used with some adaptations by M. Harrison (Magma) to increase the speed for fields of moderate size. Otherwise, the MSST algorithm as described by P. Gaudry is used for fields up to a certain size, and a recursive version due to R. Harley is used for larger fields. When $X_0(p)$ is hyperelliptic, but excluding $p = 37$, Harrison's method involves lifting a *pair* of modular parameters, specifically, the coordinates on a Weierstrass model for $X_0(p)$.

As a consequence of point counting, the Euler factor and zeta function can be computed for elliptic curves. The points and rational points can also be enumerated.

10.7.3 Point Counting on Hperelliptic Jacobians

For Jacobians of curves of genus 2, all the techniques described by P. Gaudry and P. Harley are available in Magma. The best algorithms are those based on p-adic liftings when the characteristic is not too large. In odd characteristic Kedlaya's algorithm is used, while in characteristic 2 Mestre's canonical lift method as adapted by Lercier and Lubicz is used. An implementation of Vercauteren's characteristic 2 version of Kedlaya's algorithm is also available.

The deformation methods for both elliptic and hyperelliptic curves are due to A. Lauder and were implemented by H. Hubrechts. The p-adic methods also provide the Euler factor and zeta function, in addition to the number of points.

The set of rational points of the Jacobian can be constructed. Also the structure of the abelian group of rational points of the Jacobian can be obtained.

10.7.4 Pairings

Pairings on elliptic curves over finite fields have many uses, ranging from checking independence of torsion points to more practical applications in cryptography. Both destructive applications (such as the MOV attack) and constructive applications (such as pairing-based cryptography) are worth mentioning.

Magma contains an implementation of the most popular pairings on elliptic curves over finite fields, including the Weil pairing, the Tate pairing, and various versions of the Eta and Ate pairings. These were implemented by F. Vercauteren. The Weil pairing provides the basis of an

efficient algorithm for determining the structure of the abelian group of rational points of the elliptic curve.

In the case of hyperelliptic Jacobians the Weil pairing of two m-torsion points on a 2-dimensional Jacobian can be computed.

10.7.5 Discrete Logarithms

Several algorithms are provided to solve the discrete log problem (DLP) which work by reducing the DLP on a curve to a DLP in a finite field. An implementation of Weil descent in characteristic 2 was undertaken by F. Hess (Magma) for use in the GHS attack on the DLP for elliptic curves.

In the other direction, a routine is provided to generate a cryptographically secure curve together with a certificate of its security.

10.8 Artin Representations

An Artin representation is a continuous homomorphism $\rho : \mathrm{Gal}(\bar{\mathbb{Q}}/\mathbb{Q}) \to GL(V)$ from the absolute Galois group of $\mathbb{Q}$ to a finite dimensional complex vector space V. By continuity, this factors through the Galois group of some finite extension $K/\mathbb{Q}$. The Euler factors L_p can be computed by taking completions of K, while the computations for the places of ramification are more difficult.

The main Magma function takes a number field K and computes all irreducible Artin representations that factor through the normal closure F of K. Various properties of the representations can be obtained. These include the conductor, kernel and ramification at a prime. Most importantly the local polynomial (Euler factor) of an Artin representation can be computed at a prime p.

Through the use of representation theory, the Dedekind ζ-function of K can then be decomposed into the corresponding Artin L-functions. As the Artin L-series have smaller conductors, it is easier to compute with them. By way of contrast, PARI/GP has to work directly with the Dedekind ζ-function. In this sense, the Artin L-series are the real primitive building blocks for number field computations, rather than the Dedekind ζ-functions.

The Magma implementation of Artin representations was originally developed by T. Dokchitser and then later expanded by Tim and Vladimir Dokchitser, and it relies heavily on the permutation group machinery available in Magma for identifying Frobenius elements in Galois groups. For instance, they have an example where the Galois group $PGSp_4(GF(3))$ has order 51840 (coming from the action on 3-torsion of a genus 2 Jacobian), where the Dokchitser brothers can compute various special L-values to 10 digits in a few hours.

10.9 L-Functions

10.9.1 Introduction

An L-series or an L-function is an infinite sum $L(s) = \sum_{n=1}^{\infty} a_n/n^s$ in the complex variable s with complex coefficients a_n. Such functions arise in many places in mathematics and they are usually naturally associated with some kind of mathematical object, for instance a character, a number field, a curve, a modular form or a cohomology group of an algebraic variety. The coefficients a_n are certain invariants associated with that object. For example, in the case of a character $\chi : (\mathbb{Z}/m\mathbb{Z})^* \to \mathbb{C}^*$ they are simply its values $a_n = \chi(n)$ when $\gcd(n, m) = 1$ and 0 otherwise.

Magma is able to associate an L-series to various types of object. The heart of the analytic number theory package in Magma is an L-series package of routines originally due to T. Dokchitser. In this context an L-series can be thought of as a Dirichlet series with specified coefficients a_n given

by $L(s) = \sum_n a_n/n^s$ for the complex variable s. Almost all applications require $L(s)$ to have an Euler product — namely $L(s) = \prod_p L_p(1/p^s)$ where each L_p is a finite degree polynomial.

Such an L-series must also satisfy a functional equation $\Lambda(s) = \Lambda(w+1-s)$ for some (nonnegative, integral) weight w upon completing the L-series via $\Lambda(s) = \gamma(s)L(s)\mathbb{Q}^s$, where $\gamma(s)$ is an appropriate product of Γ-functions and the *conductor* $\mathbb{Q}$ is an arithmetically-defined integer. In many cases, the most difficult computations arise when determining the Euler factors L_p at primes p dividing $\mathbb{Q}$; such primes are called *bad primes*. A prototypical example of an L-series is the Riemann zeta function.

The barebones structure of this implementation makes possible the definition of general L-series and subsequent calculations with them. These include checking their functional equation and determining any special values to a given precision. T. Dokchitser first implemented such a framework in PARI/GP, but his versions of Magma from 2014 onwards contain a greatly expanded spectrum of possible L-series as listed below:

10.9.2 Sources of L-Functions

- Number Fields
- Elliptic curves over $\mathbb{Q}$and number fields
- Hyperelliptic curves over $\mathbb{Q}$and number fields
- Artin representations
- Dirichlet characters over
  QQ
- Hecke Grossencharacters
- Hypergeometric motives
- Jacobi motives
- Classical modular forms
- Hilbert modular forms
- Symmetric powers
- Tensor products

Dirichlet characters, number fields, elliptic curves over $\mathbb{Q}$, and classical modular forms are relatively standard. The remainder are basically unique to Magma. Elliptic curves, hyperelliptic curves, Artin representations, hypergeometric motives and Jacobi motives are discussed in other chapters while the remaining cases are discussed briefly in the following subsections.

10.9.3 Artin Representations

An Artin representation is a continuous homomorphism $\rho : \mathrm{Gal}(\bar{\mathbb{Q}}/\mathbb{Q}) \to GL(V)$ from the absolute Galois group of $\mathbb{Q}$ to a finite dimensional complex vector space V. By continuity, this factors through the Galois group of some finite extension $K/\mathbb{Q}$. The Euler factors L_p can be computed by taking completions of K, while the computations for the places of ramification are more difficult.

10.9.4 Elliptic Curves over $\mathbb{Q}$ and Number Fields

The Dokchitser brothers have also implemented functionality in Magma which makes it possible to twist an elliptic curve over $\mathbb{Q}$ by an Artin representation (for instance), allowing faster computation. For example, this makes it possible to fix an elliptic curve over $\mathbb{Q}$ and study how the arithmetic varies when extending the field, the computations being expedited by the Artin representation machinery.

10.9.5 Hyperelliptic Curves over $\mathbb{Q}$ and Number Fields

A hyperelliptic curve can be defined as $y^2 = f(x)$ where f is a polynomial of degree at least 5. (When f has degree 2 or less one obtains a conic, while degrees 3 and 4 give a genus 1 curve and so elliptic curve methods are more relevant). The determination of the Euler factors at good primes can be accomplished via Kedlaya's algorithm. However, to determine the Euler factors at bad primes, Magma relies on the integral model package of S. Donnelly (Magma), whereas an analogous PARI/GP version uses only Liu's algorithm.

Magma also allows one to work with hyperelliptic curves over number fields. Here the functionality seems not to be available elsewhere.

10.9.6 Hecke Grossencharacters

A Dirichlet character over a number field modulo an ideal I is a homomorphism from O_K/I to $\mathbb{C}$, the image necessarily consisting of roots of unity. This defines a map on field elements; correspondingly, a Hecke character is defined on ideals, which means in particular that it must be trivial on the units. A Hecke character can be thought of as a Dirichlet character that is trivial on the units and is then extended by the class group, corresponding to the non-principal ideals. This is indeed essentially how the implementation proceeds as it dualises the ray class group machinery due to F. Hess and C. Fieker (Magma). The Hecke characters have associated L-functions, which is not directly the case for Dirichlet characters.

A Hecke Grossencharacter additionally has a nontrivial size component, which arises from embeddings into the complex numbers. For instance over $\mathbb{Q}(i)$, the ideal $(2 + i)$ can be mapped to the complex number $2 + i$ (or $2 - i$), and similarly at other principal ideals. The embeddings are determined by requiring a specific residue class for a generator of the ideal. This makes the subject matter much richer, although also much more delicate to work with, as it is necessary to ensure that the embeddings for each prime ideal are globally coherent. The theory behind computing with Grossencharacters was worked out by M. Watkins (Magma) over a three month period in 2009.

The Magma Grossencharacters package has been used by students of F. Calegari in their construction of a half-integral weight Hilbert modular form; in particular, they computed the values of the L-series at negative integers.

10.9.7 Hilbert Modular Forms

The generality of Dokchitser's package makes it easy to implement L-series for Hilbert modular forms once information from the Hecke operator computations is available. As noted in the previous section, this was used to construct a half-integral weight Hilbert modular form.

10.9.8 Hypergeometric Motives

Magma provides a number of ways of specifying hypergeometric data of which the product of cyclotomic polynomials is one. Functions are provided for computing the hypergeometric trace according to the p-adic Γ-function definition. The Euler factor function computes the p-th Euler factor of the hypergeometric motive at t, a rational number. Finally, an L-series function tries to construct the L-series of the associated motive.

Functions are also provided that try to determine the associated Artin representation and the associated elliptic curve, as catalogued by Cohen.

10.9.9 Jacobi Motives

A topic related to hypergeometric motives is that of Jacobi sum motives. These are indeed simpler, and in fact the tame prime information for hypergeometric motives can be determined from Jacobi motives, possibly twisted by Kummer and Tate characters.

The classical Jacobi sums were indicated by Weil to come from Grössencharacters and this functionality is also included, with it indeed being the preferred method to compute Euler factors and the L-series, once the reciprocity correspondence has been established and the Grössencharacter identified.

The Euler factor of Jacobi motive can be computed at a good prime. Also given a Jacobi motive, it can be identified as a Grössencharacter. This uses the Weil bound on the conductor, and then tries enough good primes to distinguish the character. This is now the preferred way to compute the LSeries of a Jacobi motive (though the latter still exists).

10.9.10 Tensor Products

The creation of L-series from smaller ones, via symmetric powers and tensor products, is well developed in Magma. In particular, the case of symmetric powers of L-series for elliptic curves over $\mathbb{Q}$ is completely handled. This is based on theoretical work due to Dummigan, Martin, and Watkins. The Dokchitser brothers have recently developed methods for handling many cases of bad Euler factors for tensor products.

The symmetric square L-function of an elliptic curve plays a major role in the computation of its degree of modular parametrisation (the existence of this being a result coming out of Andrew Wiles' work on proving Fermat's Last Theorem). Watkins (Magma), in joint work with Dummigan and also Martin, has produced extensive numerical data for higher symmetric powers, including analysis of central vanishings (for odd powers) and divisibility of special values.

10.9.11 Applications

One of the principal directions of inquiry in the area of L-functions concerns their order of vanishing at the central point. This is conjecturally related to the rank of a finitely generated abelian group in great generality; the simplest case is the Birch and Swinnerton-Dyer conjecture (a Clay Milliennial problem), which states that the rank of an elliptic curve should be equal to the order of vanishing of its L-function at the central point. Knowing how large the rank of an elliptic curve (or the order of vanishing) can be is also a significant open problem, having been hotly disputed over the years. M. Watkins has undertaken many experiments, both for elliptic curves and other types of L-functions, which have led to a refined understanding. Two specific problems studied include (a), predicting the growth rate for rank 3 curves in a quadratic twist family and (b), whether rank 7 is the maximal rank for quadratic twists of the congruent number curve. The congruent number curve is the elliptic curve $y^2 = x^3 - n^2x$, where n is a positive integer.

10.10 Modular Forms and Modular Curves

10.10.1 Classical Modular Forms

By definition, modular forms are analytic functions on the upper half of the complex plane that possess certain symmetries under unimodular transformations. A family of commuting operators, called Hecke operators, act on modular forms. It is the simultaneous eigenforms under these operators that have special significance in number theory: An eigenform gives rise to an L-function and this is often how the correspondence with other number theoretic objects is expressed.

To compute modular forms for some specified weight or level means firstly to calculate the dimension of the space (which is finite), and secondly to calculate Hecke operators as matrices with respect to some chosen basis of the space. After calculating sufficient Hecke operators, the decomposition of the space into simultaneous eigenforms for the Hecke action can be determined.

Magma contains a package originally due to W. Stein for classical modular forms. The core of this package is an implementation of modular symbols, since the standard algorithm for obtaining the cuspidal part of spaces of modular forms for weight $k \geq 2$ is based on calculations with modular symbols. The package provides fast computation of many Hecke eigenvalues for a given eigenform.

There are some further features. Eisenstein series, which generate the complement of the cuspidal part of a space, can be obtained by direct construction. Sturm bounds are available, and are used to determine the $\mathbb{Z}$-lattice of all forms whose q-expansions have integer coefficients. These features are also implemented for weight 1 and half-integral weights. A major application of q-expansion computation is to the solution of Diophantine-type problems using arguments in the vein of Fermat's last theorem. Results of this type are due to Siksek and many others. Another application is to an algorithm for computing the theta series of an integral lattice.

10.10.2 Representations from Modular Forms

In the Langlands conjectures, and in other work, the relationships between modular forms and other number theoretic objects are expressed in terms of relationships between several different sorts of representations. Some of these representations can be constructed in Magma. Given a modular eigenform and a prime p, the p-adic "admissible representation" can be constructed, and also the corresponding local Galois representation. The algorithms are due to Loeffler and Weinstein. The purpose is to make accessible some information about an eigenform that is not readily accessed from its eigenvalues. This information might, for instance, determine the behaviour mod p of an object that (conjecturally) corresponds to the eigenform.

10.10.3 Modular Abelian Varieties

A package dealing with the Jacobians of modular curves $X_0(N)$ and $X_1(N)$, as well as their subvarieties and isogenous varieties, is built on top of the core modular symbols algorithms. Included are functions for decomposing modular abelian varieties, isomorphism testing, computing endomorphism rings, arithmetic with finite subgroups, and computing torsion subgroups, special values of L-functions, and Tamagawa numbers. This package was developed by W. Stein.

10.10.4 Modular Forms over Number Fields

Classical modular forms can be generalised in many ways of which one of the most important is to pass from $\mathbb{Q}$ to a number field. Hilbert modular forms correspond to the case where the number field is totally real, and a fair part of the theory of classical modular forms carries over to this case. However, the standard modular symbols algorithm does not carry over; instead, several very different alternative algorithms have been devised. Magma contains the only available implementation of general Hilbert modular forms. In fact, it contains implementations of two completely separate algorithms: one developed primarily by Dembele and Donnelly (Magma), the other by Greenberg and Voight. The first of these, which covers most fields and levels, is highly developed and efficient.

The case of imaginary quadratic fields (sometimes called Bianchi modular forms) is dealt with by algorithms developed by Gunnells and Yasaki. Earlier, several authors had studied and provided software for individual fields but Magma is the only software available which includes a general algorithm handling all quadratic fields.

10.10.5 Modular Curves

Closely related to modular forms are modular curves which are the basis of a huge range of constructions in mathematics. One crucial application is in the standard algorithm (ECPP) for proving primality of larger integers. Magma contains several databases of modular curves to serve these needs. Equations in the standard forms — classical, canonical, and Atkin (as in ECPP) — are listed for appropriate levels. A more interesting database, computed by M. Harrison (Magma), gives particularly nice models over $\mathbb{Q}$ for the curves and, in addition, all the modular data including the cusps, standard functions, maps, etc., in terms of those models. Thus, it provides a dictionary between the analytic and $\mathbb{Q}$-rational descriptions for each included modular curve.