# MAGMA IN A NUTSHELL

DON TAYLOR AND DANIEL TUBBENHAUER

ABSTRACT. These are lecture notes from the "MAGMA Mondays Workshop 2023" held at the University of Sydney.

## CONTENTS



2020 *Mathematics Subject Classification.* Primary 18M30, 20C08; Secondary 05E10, 17B10, 18N25.
*Key words and phrases.* KLR and weighted KLRW algebras, (sandwich) cellular algebras, crystal graphs.

## 1. Introduction

These lecture notes are an introduction to Magma. They grew out of the "Magma Mondays Workshop 2023" held at the University of Sydney. We assume no knowledge of Magma or any other computer algebra system (of course, having background knowledge will not hurt), and our examples are chosen to illustrate language and algorithmic features as simply as possible.

*Remark* 1.1. There is a YouTube channel associated to these lecture notes with videos of the lectures, see [Tub23]. ◇

We are not trying to give a comprehensive coverage of Magma, but we have rather several selected topics that we will explore. After a bit of background on the language, we will cover the combinatorics of groups and related objects. Our goal is to introduce the reader to the role Magma can play in mathematical research.

References that accomplish this text are:

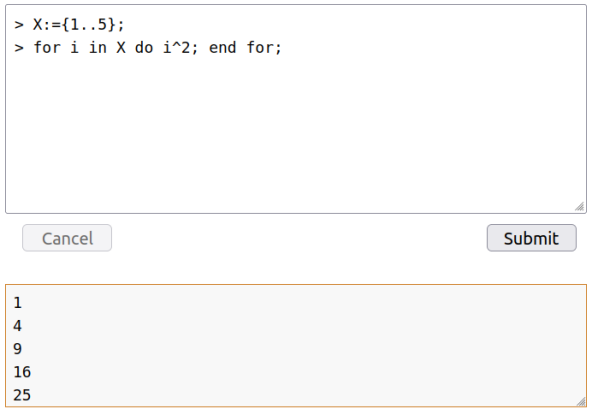
**(a)** The Handbook of Magma Functions (short: the handbook) [BC23].

**(b)** Discovering Magma via examples [BC06].

**(c)** An Introduction to Algebraic Programming with Magma [CP01].

**(d)** And shorter texts such as [Cra08].

*Remark* 1.2. The reader can directly copy the code given below and try themselves in the online calculator. Conveniently, there is no need to remove the greater symbols:




This is highly encouraged. ◇

**Acknowledgments.** We like to thank Bregje Pauwels and Alexander Sherman who organized "Magma Mondays Workshop 2023". You did a wonderful job.

## 2. Magma – what, why, how?

We now briefly explain the main features of Magma.

2A. **What?** Magma is a computer algebra system designed to solve problems in algebra, number theory, geometry and combinatorics. It was developed by Cannon and team at the University of Sydney and version 1 was first released in August 1993.

*Remark* 2A.1. Computer algebra system perform exact calculations which go under the umbrella of ***symbolic computation***: mathematical expressions are manipulated in a way similar to the traditional manual computations. This should not be confused with numerical computations; in particular, one can use Magma output in papers or theses without loosing the exactness. ◇

The name "MAGMA" comes from Bourbaki (and Serre) where it is used to denote a set with one or more binary operations without any additional axioms. In their language a ***magma*** is the most basic of algebraic structures.

Today, MAGMA is a huge system with several thousand pages of documentation. MAGMA can be used both interactively and as a programming language. The core of MAGMA is programmed in C but a large part of its functionality resides in package files written in the MAGMA user language.

The design principles underpinning both the user language and system architecture are based on ideas from universal algebra and category theory. The MAGMA language attempts to approximate the usual mathematical modes of thought and notation as closely as possible. In particular, the principal constructs in the user language are sets, algebraic structures such as groups, rings and fields and morphisms.

2B. **Why?** The main features of MAGMA are (this is taken from the handbook [BC23]):

- *Algebraic Design Philosophy* The design principles underpinning both the user language and system architecture are based on ideas from universal algebra and category theory. The language attempts to approximate as closely as possible the usual mathematical modes of thought and notation. In particular, the principal constructs in the user language are set, (algebraic) structure and morphism.
- *Explicit Typing* The user is required to explicitly define most of the algebraic structures in which calculations are to take place. Each object arising in the computation is then defined in terms of these structures.
- *Integration* The facilities for each area are designed in a similar manner using generic constructors wherever possible. The uniform design makes it a simple matter to program calculations that span different classes of mathematical structures or which involve the interaction of structures.
- *Relationships* MAGMA provides a mechanism that manages "relationships" between complex bodies of information. For example, when substructures and quotient structures are created by the system, the natural homomorphisms that arise are always stored. These are then used to support automatic coercion between parent and child structures.
- *Mathematical Databases* MAGMA has access to a large number of databases containing information that may be used in searches for interesting examples or which form an integral part of certain algorithms. Examples of current databases include factorizations of integers of the form $pn \pm 1$, p a prime; modular equations; strongly regular graphs; maximal subgroups of simple groups; integral lattices; K3 surfaces; best known linear codes and many others.
- *Performance* The intention is that MAGMA provide the best possible performance both in terms of the algorithms used and their implementation. The design philosophy permits the kernel implementor to choose optimal data structures at the machine level. Most of the major algorithms currently installed in the MAGMA kernel are state-of-the-art and give performance similar to, or better than, specialized programs.

What makes MAGMA also ***attractive for the working mathematician*** are many build in functions spanning the following fields:

**(a)** The MAGMA Language and System;

**(b)** Groups;

**(c)** Semigroups and monoids;

**(d)** Rings and fields;

**(e)** Commutative rings;

- **(f)** Linear algebra and module theory;
- **(g)** Lattices and quadratic forms;
- **(h)** Algebras;
- **(i)** Representation theory;
- **(j)** Homological algebra;
- **(k)** Lie theory;
- **(l)** Algebraic geometry and commutative algebra;
- **(m)** Arithmetic geometry and modular arithmetic geometry;
- **(n)** Combinatorics and graph theory;
- **(o)** Finite incidence geometry;
- **(p)** Differential Galois theory;
- **(q)** Error-correcting codes;
- **(r)** Cryptography;
- **(s)** Mathematical databases.

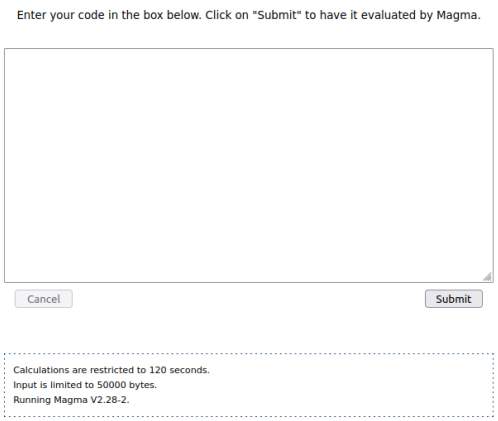
Thus, MAGMA spans a wide range of topics in algebra and combinatorics.

MAGMA is an ***imperative, call by value, lexically scoped, dynamically typed programming language***, with an essentially functional subset. This means:

- An imperative language manipulates data directly by assignment statements and control structures such as loops and if-else constructions. You can do this in MAGMA.
- A functional programming language achieves its goals by composing functions without side-effects. In MAGMA you can do this too.
- Functions are a fundamental part of MAGMA programming and they are first-class. That is, they can be assigned to variables, stored in data structures and returned from other functions. They can invoke themselves recursively and participate in mutual recursion.
- MAGMA also has procedures and unlike a function a procedure does not return a value but it may modify its arguments (provided they are declared as reference variables).

2C. **How?** MAGMA is a non-commercial system, but the costs (such as preparation of user documentation, the fixing of bugs, and the provision of of user support) need to be recovered. So MAGMA is ***non-commercial but not free***, and the distribution is organized on a subscription basis. In order to get MAGMA on your machine use this site: http://magma.maths.usyd.edu.au/magma/ordering/

Free, very useful, and completely enough for this course, is the ***online calculator*** http://magma.maths.usyd.edu.au/calc/:



There are mild restriction: calculations are restricted to 120 seconds, Input is limited to 50000 bytes, and you cannot run additional packages. However, experience tells us that the online calculator is enough most of the time.

## 3. Lecture 0 – Types, conditionals and loops

Let us get started with a few basics.

3A. **Basic arithmetic and symbols as placeholders.** Here is a basic syntax:

```
> 2+5;
  -------result-------
> 7
> 2*5;
  -------result-------
> 10
> 2/5;
  -------result-------
> 2/5
> 2^5;
  -------result-------
> 32
> 5 mod 2;
  -------result-------
> 1
```

Note the semicolon at the end of each line: this is crucial and tells MAGMA that the command ends. A common mistake is to forget the semicolon, so beware.

Eventually we want to use symbols instead of actual numbers, so here is how we do that:

```
> x:=3;
> x^3;
  -------result-------
> 27
> x:=x+1;
> x;
  -------result-------
> 4
> x+:=1;
> x;
  -------result-------
> 5
> 3*x;
> x;
  -------result-------
> 15
> 3x;
  -------result-------
> User error: Invalid hexadecimal integer
```

Be careful to write $3 * x$ and not $3x$ which would be another variable. Similarly:

```
> x:=12; x-:=4; x;
  -------result-------
> 8
> x/:=4; x;
  -------result-------
> 2
> y:=7;
> x*:=y; x;
  -------result-------
> 14
```

3B. **Data types.** Let us play with 2/5:

```
> 2/5;
  -------result-------
> 2/5
```

In the above code we see that we get 2/5 back as it is. This is because MAGMA ***treats it as a rational number***. To get a decimal expression we need to tell MAGMA to see 2/5 ***as a real number***:

```
> 2/5.0;
  -------result-------
> 0.40000000000000000000000000000000
```

The operation of changing the type of an element is crucial for MAGMA computations and called ***casting***. But what is a type anyway? Let us look at some examples:

```
> x:=2; Type(x);
  -------result-------
> RngIntElt
```

This is an element of the ring of integers.

```
> x:=2/5; Type(x);
  -------result-------
> FldRatElt
```

Thus, we have an element from the field of rational number.

```
> x:=2/5.0; Type(x);
  -------result-------
> FldReElt
```

Now we have casted 2/5 into the field of real numbers.

These examples illustrate what a ***type*** is: it is the category MAGMA stores elements in. We have also seen our first ***function***: the `Type` function.

When programming in MAGMA it is crucial to keep track of the type. Here is an easy example why this is so important.

```
> x:=4/3; y:=3/2; x*y;
  -------result-------
> 2
```

The output is 2, but MAGMA thinks this is a rational number since we have multiplied two rational numbers:

```
> x:=4/3; y:=3/2; x*y; Type(x*y);
```

```
-------result-------
> 2
> FldRatElt
```

This makes a difference when applying functions:

```
> IsEven(2); IsEven(x*y);
-------result-------
> True
> Runtime error in 'IsEven': Bad argument types
> Argument types given: FldRatElt
```

IsEven does what you think it does. It however is only defined for integers, so we cannot use it for the rational number $x * y$.

Similar problems can occur anytime, so we need to be aware of this. Here is another example, and a first way around:

```
> 3 mod 2; x:=6; y:=2; x/y; Type(x/y); x/y mod 2;
-------result-------
> 1
> 3
> FldRatElt
> Runtime error in 'mod': Bad argument types
> Argument types given: FldRatElt, FldRatElt
```

A first way around is to use div:

```
> 3 mod 2; x:=6; y:=2; x div y; Type(x div y); (x div y) mod 2;
-------result-------
> 1
> 3
> RngIntElt
> 1
```

A better way around, and something one always does is to cast the numbers into the correct rings. Let us do this. First, we setup the rings:

```
> Z:=IntegerRing(); Q:=Rationals(); R:=RealField(); C:=ComplexField();
> Type(Z); Type(Q); Type(R); Type(C);
-------result-------
> RngInt
> FldRat
> FldRe
> FldCom
```

Anyway, try to avoid working with real or complex numbers. They are always tricky in exact calculations, even the easiest ones:

```
> Sqrt(-1); Type(Sqrt(-1));
-------result-------
> 1.0000000000000000000000000000*$.1
> FldComElt
```

For completeness:

```
> Z:=IntegerRing(); Z; Type(Z); Type(2); Type(RngIntElt); Type(RngInt);
-------result-------
```

```
> Integer Ring
> RngInt
> RngIntElt
> Cat
> Cat
```

Let us now check a few elements and where the live according to MAGMA:

```
> 2 in Z; (11/4 * 4/1) in Q; (11/4 * 4/1) in Z; 11/4 in Z;
  -------result-------
> true
> true
> true
> false
```

Note that $(11/4 \cdot 4/1) \in \mathbb{Z}$ but still we get:

```
> (11/4 * 4/1) mod 2;
  -------result-------
> Runtime error in 'mod': Bad argument types
> Argument types given: FldRatElt, FldRatElt
```

This is where casting comes in. We need to tell MAGMA to consider $(11/4*4/3)$ as an integer. So we cast it to the integers!

```
> Z!(11/4 * 4/1) mod 2;
  -------result-------
> 1
```

Here is another example:

```
> x:=6; y:=2; Type(x/y); Type(Z!(x/y));
  -------result-------
> FldRatElt
> RngIntElt
```

We can cast $x/y$ into any of the previous rings:

```
> Type(Z!(x/y)); Type(Q!(x/y)); Type(R!(x/y)); Type(C!(x/y));
  -------result-------
> RngIntElt
> FldRatElt
> FldReElt
> FldComElt
```

Casting only works when it is supposed to work. Use `IsCoercible` to see whether casting is possible.

```
> Type(Z!11/4); IsCoercible(Integers(),11/2); IsCoercible(Integers(),12/2);
  -------result-------
> FldRatElt
> false
> true 6
```

Note that two outputs are given in the final command. To access them separately use:

```
> bool,val:=IsCoercible(Integers(),12/2); bool; val;
  -------result-------
> true
> 6
```

3C. **Sets and sequences.** Next, we explain how to setup ***sets*** and, more importantly, ***sequences***. First, setting up sets is straightforward and follows the usual syntax:

```
> X:={1,2,3,4}; X;
  -------result-------
> { 1, 2, 3, 4}
```

The operations on sets have the expected syntax as well:

```
> X:={1,2,3}; Y:={1,2,3,3}; Z:={2,3,4};
> X eq Y; X eq Z; X join Z; X meet Z; X diff Z; X sdiff Z; 1 in X;
  -------result-------
> true
> false
> { 1, 2, 3, 4 }
> { 2, 3 }
> { 1 }
> { 1, 4 }
> true
```

The size of sets is encoded using #, and $a..b$ stands for arithmetic progression:

```
> #X;
  -------result-------
> 3
> A:={1..4}; A eq X;
  -------result-------
> true
```

Here are a few more ways to construct and work with sets, all with a self-explaining syntax:

```
> Y:={1..9 by 2}; Z:={x^2: x in {1..10}}; 4 in Y; Z;
  -------result-------
> false
> { 1, 4, 9, 16, 25, 36, 49, 64, 81, 100 }
> A:={4,2,16,-3,0}; Maximum(A); Minimum(A); Random(A);
  -------result-------
> 16
> -3
> 4
> B:={1,4,6,10,-3}; Include(B,17); Exclude(B,10);
  -------result-------
> { -3, 1, 4, 6, 10, 17 }
> { -3, 1, 4, 6 }
```

There are three ways to generalize this: to ordered lists; to unordered lists allowing repetitions; and to ordered lists allowing repetitions. Let us go through all of them.

***Multisets*** can be defined by using {**}:

```
> X:={* 1,1,2,3 *}; X;
  -------result-------
> {* 1^2, 2, 3 *}
```

Note the types:

```
> A:={1,2,3}; B:={* 4,5,6 *}; Type(A); Type(B);
  -------result-------
```

```
> SetEnum
> SetMulti
```

So these cannot be e.g. joined together. But most of the above operations still work within the multiset type. For example:

```
> A join A;
  -------result-------
> {* 1^2, 2^2, 3^2 *}
> > D:={* x^2: x in {-2..2} *}; D; Maximum(D); Multiplicity(D,4);
  -------result-------
> {* 0, 1^2, 4^2 *}
> 4
> 2
```

The `Multiplicity` here is new and does what it suggests. Next, one uses {@@} for ***ordered sets***. Most of the above still works, e.g.:

```
> X:={@1,2,3,4,5@}; Y:={@8,7,6,5,4@}; X join Y; Y join X;
  -------result-------
> {@ 1, 2, 3, 4, 5, 8, 7, 6 @}
> {@ 8, 7, 6, 5, 4, 1, 2, 3 @}
```

What is fantastic about this is that now ***taking the nth element makes sense***:

```
> X:={1,2,3,4,5}; Y:={@6,7,8,9,10@}; Y[2]; X[2];
  -------result-------
> 7
> Runtime error in '[]': Bad argument types
```

The probably most important notion is an ordered multiset, called ***sequence*** in MAGMA.

```
> X:=[1..10]; X; #X; Y:=[3,6,6,2]; Y[2];
  -------result-------
> [ 1 .. 10 ]
> 10
> 6
```

Sequences are essentially the same as sets, but treated very differently, be careful. Here are some examples.

```
> X:=[1,6,4,3,9]; X[3]:=5; X;
  -------result-------
> [ 1, 6, 5, 3, 9 ]
> Append(X,8);
  -------result-------
> [ 1, 6, 5, 3, 9, 8 ]
> X:=Append(X,8); X;
  -------result-------
> [ 1, 6, 5, 3, 9, 8 ]
> X:=Prune(X); X;
  -------result-------
> [ 1, 6, 5, 3, 9 ]
> Remove(X,3);
  -------result-------
```

```
> [ 1, 6, 3, 9 ]
> Insert(X,2,17);
  -------result-------
> [ 1, 17, 6, 5, 3, 9 ]
> X;
  -------result-------
> [ 1, 6, 5, 3, 9 ]
> Reverse(X);
  -------result-------
> [ 9, 3, 5, 6, 1 ]
> X:=Reverse(X); X:=Sort(X); X;
  -------result-------
> [ 1, 3, 5, 6, 9 ]
> X:=[2,5,10,10,4,10,3]; Maximum(X);
  -------result-------
> 10 3
> A:=[2,5,3]; B:=[7,7,9,1,14]; C:=A cat B; C;
  -------result-------
> [ 2, 5, 3, 7, 7, 9, 1, 14 ]
> A1:=Append(A,5); A2:=A cat [5]; A1 eq A2;
  -------result-------
> true
```

It is often important to cast sets into list and back. This is how this works:

```
> A:=[2,5,3,3]; A;
  -------result-------
> [ 2, 5, 3, 3 ]
> B:=SequenceToSet(A); B;
  -------result-------
> { 2, 3, 5 }
> SetToSequence(B);
  -------result-------
> [ 2, 3, 5 ]
```

3D. **Conditionals.** There are plenty of other questions we can ask about numbers and objects, here are the most common ones:

| *Operation* | MAGMA command |
|---|---|
| $x = y$ | `x eq y` |
| $x \neq y$ | `x ne y` |
| $x \geq y$ | `x ge y` |
| $x \leq y$ | `x le y` |
| $x > y$ | `x gt y` |
| $x < y$ | `x lt y` |
| $x \in y$ | `x in y` |

We also have the usual logic operations:

| *Operation* | MAGMA command |
|---|---|
| $A \cap B$ | `and` |
| $A \cup B$ | `or` |
| $(A \cup B)^c$ | `nor` |

These are even able to jump over types, but not always. For example,

```
> x:=2; y:=4/2; x eq y;
  -------result-------
> true
> 4 gt 5.0;
  -------result-------
> false
> X:=[1,2]; Y:={1,2}; X eq Y;
  -------result-------
> Runtime error in 'eq': Bad argument types
> Argument types given: SeqEnum[RngIntElt], SetEnum[RngIntElt]
> x:=5; x eq 5 or x eq 4 and x eq 2;
  -------result-------
> true
```

Let us use this to move to the ***conditional commands*** or ***conditionals***: "if some condition is true, then do this series of commands". These are among the most important commands one needs to know in order to program.

The first type is the `If` ("if (condition) then (commands); end if;") and it works as follows:

```
> a:=3; b:=4; c:=5; if(a gt b) then c:=a*b; end if; c;
  -------result-------
> 5
> a:=3; b:=4; c:=5; if(a lt b) then c:=a*b; end if; c;
  -------result-------
> 12
```

It is often useful to separate conditionals visually, and a bit nicer is:

```
> a:=3; b:=4; c:=5;
> if(a gt b) then
> c:=a*b;
> end if;
> c;
  -------result-------
> 5
```

We can also do "if (condition) then (commands); else (commands); end if;". For example:

```
> a:=3; b:=4; c:=5;
> if(a gt b) then
> c:=a*b;
> else
> c:=6;
> end if;
> c;
  -------result-------
> 6
```

We can also run parallel `If`:

```
> x:=5;
> if(x mod 4 eq 0) then
> print("x is congruent to 0 modulo 4");
> elif(x mod 4 eq 1) then
> print("x is congruent to 1 modulo 4");
> elif(x mod 4 eq 2) then
> print("x is congruent to 2 modulo 4");
> else
> print("x is congruent to 3 modulo 4");
> end if;
```

`select` ("(variable):= (condition) select (value if true) else (value if false)") is a bit nicer when setting variables.

```
> s := (1 gt 0) select 1 else -1; s;
  -------result-------
> 1
> a:=-5; s := (a gt 0) select 1 else ((a eq 0) select 0 else -1);
  -------result-------
> -1
```

Side note: This was for illustration only. There is of course a better way to get the sign:

```
> Sign(-5);
  -------result-------
> -1
```

In general, ***always use build-in functions if applicable***. Anyway, if we want the text we could use `case`:

```
> x := 5;
> case Sign(x):
> when 1:
> "x is positive";
> when 0:
> "x is zero";
> when -1:
> "x is negative";
> end case;
  -------result-------
> x is positive
```

3E. **Loops.** The `for` ("for i in A do COMMAND; end for;") loop is one of the most fundamental of all operations in MAGMA. Let us run a few examples.

```
> for i in {1..5} do
> i^2;
> end for;
  -------result-------
> 1
> 4
> 9
> 16
> 25
```

Here is another example. We calculate the numbers $x^3+x+12$ modulo 7 for $x \in \{1,\dots,10\}$. In order to avoid ten outputs in a row we collect them in one sequence:

```
> out:=[];
> for i in {1..10} do
> out:=Append(out,(i^3+i+12) mod 7);
> end for;
> out;
  -------result-------
> [ 0, 1, 0, 3, 2, 3, 5, 0, 1, 0 ]
```

There are a few other loops, which are similar in style to a for loop, but which have slightly different uses. Here is `while`:

```
> n:=453;
> while((n mod 5) ne 0) do
> n+:=1;
> end while;
> n;
  -------result-------
> 455
```

The `repeat` function ("repeat (commands) until (condition)") is the same as the while function, except that the particular condition is tested at the end of the loop, not at the beginning. Here is an example:

```
> X:={1..1000};
> repeat a:=Random(X);
> bool:=IsPrime(a);
> until bool eq true;
> a;
  -------result-------
> 563
```

Shortened:

```
> X:={1..1000};
> repeat a:=Random(X);
> until IsPrime(a);
> a;
  -------result-------
> 223
```

Let us briefly address one problem one should be aware of when using loops. Take for example the same code as before but with a different set:

```
> X:={24,25,26,27,28};
> repeat a:=Random(X);
> until IsPrime(a);
> a;
```

The set $X$ does not contain a prime number. So this will ***not terminate*** since its stuck in an infinite loop! With for one is on the safe side. In any case, the `break` function might come in handy:

```
> n:=453;
> while((n mod 5) ne 0) do
> n+:=5;
> if(n ge 1000) then break; end if;
```

```
> end while;
> n;
  -------result-------
> 1003
> n := 10037;
> for x in [1..100] do
> for y in [1..100] do
> if x^2 + y^2 eq n then
> print x, y;
> break x;
> end if;
> end for;
> end for;
  -------result-------
> 46 89
```

`break` usually ends the nearest loop, and here we tell MAGMA using "break x" to break the outer $x$ loop.

3F. **Handmade functions.** We can also write functions ourselves. The following function computes the ***Fibonacci numbers***:

```
> function Fib(n)
> if(n le 0) then return 0; end if;
> if(n le 1) then return 1; end if;
> return Fib(n-1)+Fib(n-2);
> end function;
```

We can use it as follows:

```
> Fib(14);
  -------result-------
> 377
```

The following functions has two inputs and two outputs:

```
> function PPart(n,p);
> if(not(IsPrime(p))) then return "p is not a prime"; end if;
> a:=0;
> while(n mod p eq 0) do
> a+:=1;
> n div:=p;
> end while;
> return n,p^a;
> end function;
```

We can more generally produce functions with arbitrary in and outputs suing the comma separator. Let us see what this functions does:

```
> PPart(60,2)
  -------result-------
> 15 4
> PPart(60,3)
  -------result-------
> 20 3
> PPart(60,4)
```

```
-------result-------
> p is not a prime
```

It thus returns $n = mp^a$ via $m$ $p^a$ if $p$ is a prime and a warning otherwise.

We leave it to the reader to figure out what the next function does:

```
> function RandomPoisson(x)
> k:=0;
> max_k:=1000;
> p:= Random([1..10^5])/10^5;
> P:= Exp(-x);
> sum:=P;
> if(sum ge p) then return 0; end if;
> for k in [1..max_k] do
> P*:=x/k;
> sum+:=P;
> if (sum ge p) then return k; end if;
> end for;
> return k;
> end function;
> RandomPoisson(10);
-------result-------
> 8
```

A `procedure` is similar to a function, in that it takes inputs. However, it differs from a function in that it changes the inputs, rather than produces outputs.

```
> procedure Multiplies(~x,y)
> x:=x*y;
> end procedure;
> x:=2; y:=3;
> Multiplies(~x,y);
> x;
-------result-------
> 6
```

## 4. Lecture 1 – A few little games and finite groups

This section covers four examples of interactive programming, including a word game, Catalan numbers and a bit about finite groups.

4A. **A simple word game.** How can you use Magma to find a word composed of the letters $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$, $\boldsymbol{t}$, $\boldsymbol{r}$? Well, you could look at all 120 permutations of the letters and hope to recognize which ones (if any) are English words.

First of all, enter the list into Magma as a sequence

```
> letters := ["a","b","c","t","r"];
```

Use the symmetric group on $\{1, 2, 3, 4, 5\}$ to generate all permutations of the letters, then ***concatenate*** them to form 'words'.

```
> for p in Sym(5) do &*[ letters[i^p] : i in [1..5] ]; end for;
> // Sym is the symmetric group
-------result-------
> abctr
> bctra
> (118 more)
```

(Many commands in MAGMA have ***synonyms***: e.g., `Sym` is a synonym of `SymmetricGroup`, and `SL` is a synonym of `SpecialLinearGroup`.)

What is happening here? Well:

- `//` introduces a comment.
- We use a for-loop: `for ... do ... end for;` to iterate over the elements of the symmetric group.
- `letters[i]` refers to the `i`-th element of the sequence; indexing begins at 1 (not 0).
- `i^p` applies the permutation `p` to `i`.
- Strings are a monoid with binary operation `*`. If `X` is a sequence, `&*X` concatenates the elements of `X`. (You can also use `&cat X`.)
- `[1..5]` is the sequence `[1,2,3,4,5]`.
- `letters := ["a","b","c","t","r"];` is an ***assignment statement***.
- MAGMA prints the results of ***expressions*** (such as `&*[letters[i^p]`) that are not statements. Sometimes, for clarity, it is better to use the keyword `print`.

*Remark* 4A.1. The part below does not run in the online calculator since we are loading a file from the system. This is the only example in this file that cannot be run without installing MAGMA. However, the below illustrates how to load files which eventually might be important for the reader. ◇

Looking through 120 possible 'words' and then checking the dictionary does not seem like much fun.

Let us see how MAGMA can help. In this case it will depend on the operating system. Unfortunately, because of the `System` call, this won't run in the online calculator.

If MAGMA is running on MacOS or Linux you can do the following.

```
> letters := ["a","b","c","t","r"];
> for p in Sym(5) do
> word := &*[ letters[i^p] : i in [1..5] ];
> cmd := "grep -w " cat word cat " /usr/share/dict/words";
> System(cmd);
> end for;
```

```
-------result-------
> bract
```

On Windows, find a word list somewhere (say `words.txt`), then use

```
> cmd := "findstr \\"\\\\\\<" cat word cat "\\\\\\>\\" words.txt";
```

Suppose you modify the problem and ask for the ***five*** letter words composed of the letters ***h***, ***r***, ***s***, ***u***, ***k***, ***n***.

For each subset of five letters, apply the previous solution.

```
> letters :=["h","r","s","u","k","n"];
> S := Sym(5);
> for n := 1 to 6 do
> X := Remove(letters,n);
> for p in Sym(5) do
> word := &*[ X[i^p] : i in [1..5] ];
> cmd := "grep -w " cat word cat " /usr/share/dict/words";
> System(cmd);
> end for;
> end for;
```

```
-------result-------
> hunks
```

Now suppose you want to solve the word puzzle for other combinations of letters. Instead of typing ever more variations of the code into the REPL the thing to do is to create a function or procedure, store it in a file, then load the file whenever you need it.

In MAGMA a ***function*** takes ***arguments*** and returns one or more ***values***. A ***procedure*** is similar except that it does not return any values.

The first step will be to write a procedure `findwds` (details below) in a file called `wordgame.m`. (You only need to print the result, so use a procedure, not a function.)

To load the file and use the procedure, type the commands

```
> load "wordgame.m";
> findwds("abctr");
-------result-------
> bract
```

```
> findwds := procedure(str)
> letters := Eltseq(str);
> m := #letters;
> for p in Sym(m) do
> word := &*[ letters[i^p] : i in [1..m] ];
> cmd := "grep -w " cat word cat " /usr/share/dict/words";
> System(cmd);
> end for;
> end procedure;
```

MAGMA has a 'flat' namespace. The MAGMA kernel and all packages are loaded at startup. The functions and procedures (such as `Eltseq`) in these packages are called ***intrinsics***; they are always available.

```
> load "wordgame.m";
> findwds("torecv");
-------result-------
> vector
> covert
```

Modify the procedure to take both a ***string*** argument `str` and an ***integer*** argument `k` giving the length of the words we are looking for.

```
> findwds := procedure(str,k)
> letters := Eltseq(str);
> m := #letters;
> if k gt m then k := m; end if;
> R := Subsets({1..m},m-k);
> S := Sym(k);
> for A in R do
> X := [ letters[i] : i in [1..m] | i notin A ];
> for p in S do
> word := &*[ X[i^p] : i in [1..k] ];
> cmd := "grep -w " cat word cat " /usr/share/dict/words";
> System(cmd);
> end for;
> end for;
> end procedure;
```

```
> findwds("torxcv",4);
```

```
-------result-------
> torc
```

Here is what is going on:

- if `ss` is a sequence, a set, a group, etc., `#ss` is its length.
- `if ... then ... else ... end if;`.
- `if ... then ... elif ... else ... end if;`.
- Sequence constructor: `[ f(x) : x in O | condition on x ]`.
- Set constructor: `f(x) : x in O | condition on x .`

What happens if there are repeated letters?

```
> findwds("tocrxc");
-------result-------
> torc
> torc
> croc
> croc
```

*Exercise* 4A.1. Suppose that `letters` is a sequence of letters. The following code produces all 'words' made from these letters.

```
> [&*[ letters[i^p] : i in [1..n]] : p in Sym(n)]
> where n is #letters ;
```

If you first type

```
> letters := ElementToSequence("aact");
```

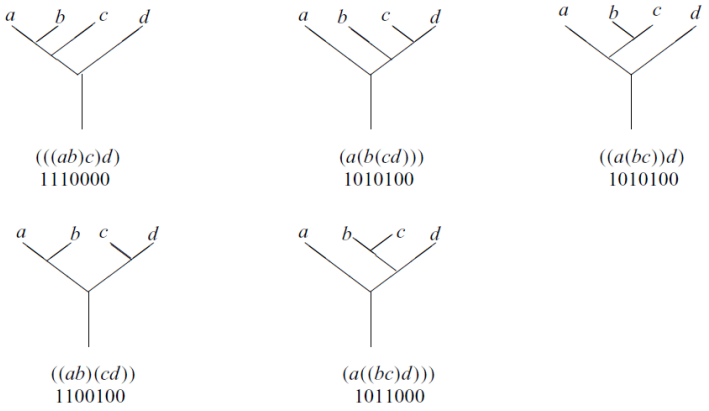
and then use the code above you will see that some 'words' appear twice.

(a) Write a few lines of code that produce a sequence of words without duplicates.

(b) Change the code so that it produces a sequence of three letter 'words'.

Of course, one can replace `"aact"` by anything else. ◇

4B. **The Catalan numbers.** Among many other things, the $n$th ***Catalan number*** is the number of balanced strings of $n$ left and $n$ right brackets, where 'balanced' means that each left bracket has a matching right bracket (to its right) and the string in between is balanced. The empty string is balanced. This can be illustrated using binary trees, here the third Catalan number 5:




We shall use MAGMA to construct sets of balanced strings. It turns out that

$$c_n = \frac{1}{n+1}\binom{2n}{n}.$$

Using MAGMA the 20th Catalan number is

```
> print Binomial(40,20) div 21;
  -------result-------
> 6564120420
```

Note that `div` is used for integer division and MAGMA can deal with very large numbers. Using ***Stirling's approximation*** for $n!$ we have an asymptotic approximation

$$c_n \sim 2^{2n}/n\sqrt{\pi\, n}.$$

To use $\pi$ in MAGMA we specify the precision.

```
> pi := Pi(RealField(10)); pi;
  -------result-------
> 3.141592654
```

Here is a function which returns Stirling's approximation to the $n$th Catalan number.

```
> approxCat := func< n | 2^(2*n)/(n*Sqrt(pi*n)) >;
> print approxCat(15);
  -------result-------
> 10427688.40
```

`func⟨ x, y, z | expression ⟩` defines a function which returns the value of the expression in the arguments $x$, $y$, $z$.

Let us look at a recursion for $c_n$. A balanced string of brackets is either empty or has the form $(S)T$, where $S$ and $T$ are themselves balanced. Therefore the Catalan numbers satisfy the recurrence relation (for $n > 0$)

$$c_{n+1} = \sum_{k=0}^{n} c_k\, c_{n-k} \quad \text{where } c_0 = 1.$$

You can use `$$` to refer to a MAGMA function within its own body. The following function uses recursion to return the sequence of the first $n$ Catalan numbers.

```
> CatSeq := function(n);
> if n eq 0 then   seq := [1];
> elif n eq 1 then seq := [1,1];
> else
> seq := $$(n-1);
> print seq;
> Append(~seq, &+[Integers()| seq[k+1]*seq[n-k] : k in [0..n-1]]);
> end if;
> return seq;
> end function;
```

Instead of `$$` you can place the directive

```
> forward CatSeq;
```

in your file, somewhere before the function definition. Then you can refer to `CatSeq` within its own definition. We can use this now as follows:

```
> CatSeq(5);
  -------result-------
> [ 1, 1 ]
> [ 1, 1, 2 ]
> [ 1, 1, 2, 5 ]
> [ 1, 1, 2, 5, 14 ]
> [ 1, 1, 2, 5, 14, 42 ]
```

A bit of more explanation:

- Procedures can modify their arguments. Such an argument is prefixed with a tilde ~ both in the definition and when called.
- The command `Append(~seq,num)` is a call to the intrinsic procedure `Append` that modifies `seq` by including `num` in the set.
- Sequences can be indexed by other sequences. `X := letters[Setseq(A)];`
- Errors

```
> CatSeq(A)
  -------result-------
> User error: Identifier 'A' has not been declared or assigned
```

*Exercise* 4B.1. Write a function expression `CatNum` in MAGMA that returns the $n$th Catalan number. ◇

*Exercise* 4B.2. Recall the function `CatSeq` from above:

```
> CatSeq := function(n);
> if n eq 0 then   seq := [1];
> elif n eq 1 then seq := [1,1];
> else
> seq := $$(n-1);
> print seq;
> Append(~seq, &+[Integers()| seq[k+1]*seq[n-k] : k in [0..n-1]]);
> end if;
> return seq;
> end function;
```

Rewrite `CatSeq` as a function expression using `select`. ◇

*Exercise* 4B.3. What happen if you replace the `Append` line of the `CatSeq` function with

```
> Append(~seq, &+[seq[k+1]*seq[n-k] : k in [0..n-1]]);
```

Test this! If you get the same result, then can you explain why? ◇

*Exercise* 4B.4. Write a MAGMA ***expression*** that returns the sequence of the first 20 Catalan numbers.

(The MAGMA function `Self(n)` refers to entry `s[n]` of a sequence `s` inside its constructor. Remember that sequences are indexed from 1 not 0.) ◇

To wrap-up, let us discuss balanced strings.

$$S \to (S)S$$
$$S \to \varepsilon$$

is a grammar that describes balances strings. We can use this to design a procedure that displays a random balanced string.

```
> produce := procedure()
> seq := ["S"];
> rhs := ["(","S",")","S"];
> X := { 1 };
> repeat
> i := Random(X);
> if Random(1) gt 0 then Insert(~seq,i,i+1,rhs);
> else Remove(~seq,i);
> end if;
> X := { i : i in [1..#seq] | seq[i] eq "S"};
```

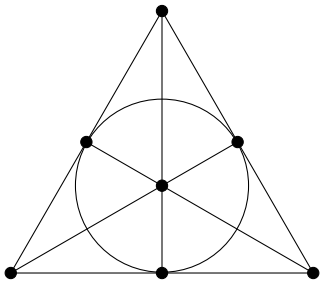
```
> until IsEmpty(X);
> print #seq gt 0 select &*seq else "eps";
> end procedure;
> produce()
```

```
-------result-------
> ()(((())(()()
```

*Exercise* 4B.5. Write a function that outputs the ratio and the difference between the sequence of Catalan numbers and the approximation given above. ◇

4C. **Projective planes, graphs, automorphism groups.** A ***projective plane*** consists of a set of points and a set of lines such that every pair of distinct points lies on a unique line and every pair of distinct lines meet in a unique point.

The smallest projective plane is the ***Fano plane*** whose points are the 1-dimensional subspaces and whose lines are the 2-dimensional subspaces of a vector space of dimension 3 over the field of 2 elements.



If the plane is finite, there is an integer $n$ (the ***order*** of the plane) such that every line has $n+1$ points and every point lies on $n+1$ lines. Thus there are $n^2+n+1$ points and $n^2+n+1$ lines. The Fano plane is the unique projective plane of order two, so there are $4+2+1=7$ points and lines.

```
> fano := FiniteProjectivePlane(2);
> P := Points(fano);
> L := Lines(fano);
```

The points and lines are represented by their (normalized) homogeneous coordinates.

```
> p := P[2];
> l := L[3];
> p; l, p in l;
```

```
-------result-------
> ( 0 : 1 : 0 )
> < 0 : 0 : 1 >
> true
```

A ***flag*** in a projective plane is a point-line pair $(p,\ell)$ such that $p$ lies on $\ell$.

We shall construct a graph whose vertices are the points $P$, the lines $L$ and the flags $F$ of the Fano plane plus an additional vertex $\star$.

**(a)** Join $\star$ to all of $P$ and $L$.

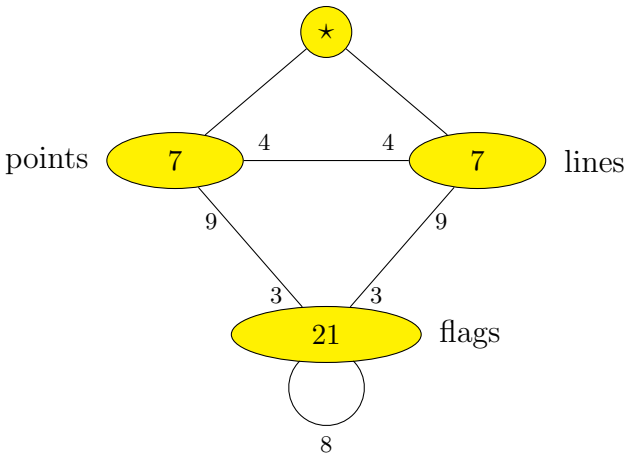
**(b)** Join a point to the 4 lines not through it.

**(c)** Join a point to the 9 flags which have their line through it.

**(d)** Join a line to the 9 flags which have their point on it.

**(e)** Join flags $(p_1,\ell_1)$ and $(p_2,\ell_2)$ if $p_1\neq p_2$, $\ell_1\neq\ell_2$, $p_1\in\ell_2$ and $p_2\in\ell_1$.

One gets: 36 vertices, degree 14, 252 edges




To build a graph in MAGMA we need a set of ***vertices*** and a set of ***edges***. In an undirected graph the edges are pairs of vertices.

For the graph from the Fano plane we represent the vertices as pairs of integers, as follows:

**(a)** ⋆ by the pair `0, 0`,

**(b)** the point `P[i]` by `-1, i`,

**(c)** the line `L[j]` by `-2, j`,

**(d)** the flag `(P[i],L[j])` by `i, j`.

Here we go:

```
> vertices := { <0,0> }
> join { <-1,i> : i in [1..7] }
> join { <-2,j> : j in [1..7] }
> join { <i,j> : i,j in [1..7] | P[i] in L[j] };
```

Ok, let us now create the graph in MAGMA.

```
> F := [ <i,j> : i,j in [1..7] | P[i] in L[j] ];
> edges := {{<0,0>,<-1,i>} : i in [1..7] }
> join { {<0,0>, <-2,i>} : i in [1..7] }
> join { {<-1,i>,<-2,j>} : i,j in [1..7] | P[i] notin L[j]}
> join { {<-1,i>,<j,k>} : i,j,k in [1..7] | P[i] in L[k]
> and P[j] in L[k] }
> join { {<-2,i>,<j,k>} : i,j,k in [1..7] | P[j] in L[k]
> and P[j] in L[i] }
> join { {f,g} : f, g in F | f[1] ne g[1] and f[2] ne g[2]
> and (P[f[1]] in L[g[2]] or P[g[1]] in L[f[2]]) };
```

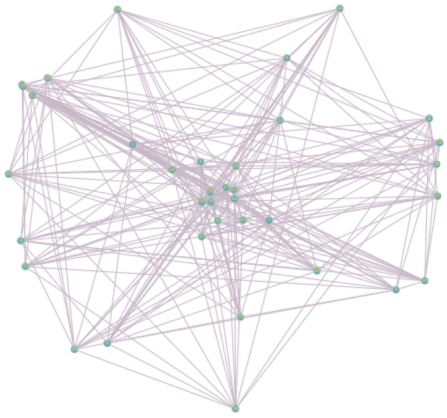
The graph constructor returns three values: the graph, the vertex set (type `GrphVertSet`) and the edge set (type `GrphEdgeSet`).

```
> Gr, V, E := Graph< vertices | edges >;
> AdjacencyMatrix(Gr);
```

*Exercise* 4C.1. Let MAGMA check whether `Gr` is planar using `IsPlanar`. Implement your favorite graph in MAGMA and check whether it is planar. In both case use `time` to check how long the calculation takes. (Fun fact: `IsPlanar` runs in linear time in the number of vertices.) ◇

The final command gives the adjacency matrix of the graph. The matrix is quite large. Copy it and put it into https://graphonline.ru/en/ using the "Adjacency matrix" tab on

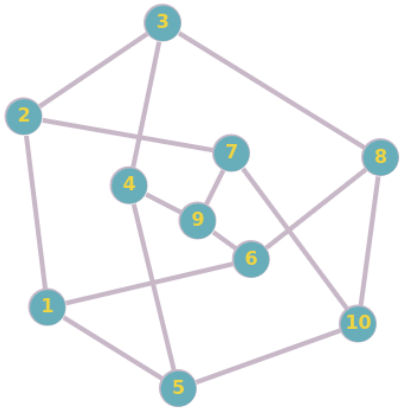
that side. We get:



Well, its quite large. For smaller graph, say the ***Petersen graph*** given by

```
> P := Graph< 10 | { 1, 2 }, { 1, 5 }, { 1, 6 }, { 2, 3 }, { 2, 7 },
> { 3, 4 }, { 3, 8 }, { 4, 5 }, { 4, 9 }, { 5, 10 },
> { 6, 8 }, { 6, 9 }, { 7, 9 }, { 7, 10 }, { 8, 10 } >;
> AdjacencyMatrix(P);
```

the same strategy gives (using "Arrange the graph" under "Algorithms"):




*Exercise* 4C.2. Implement the complete graphs $C_n$ on $n$ vertices in MAGMA and display them for small $n$. ◇

Back to the graph obtained from the Fano plane. Let us calculate its automorphism group:

```
> A := AutomorphismGroup(Gr);
> IsTransitive(A);
  -------result-------
> true
> CompositionFactors(A);
  -------result-------
> G
> |  Cyclic(2)
> *
> |  2A(2, 3) = U(3, 3)
> 1
> H := Stabiliser(A,1);
> CompositionFactors(H);
  -------result-------
```

```
> G
> |  Cyclic(2)
> *
> |  A(1, 7) = L(2, 7)
> 1
```

```
> check, _ := IsIsomorphic(SU(3,3),DerivedGroup(A)); check;
  -------result-------
> true
```

And finally, the group itself is:

```
> A;
  -------result-------
> Permutation group A acting on a set of cardinality 36
> Order = 12096 = 2^6 * 3^3 * 7
> (2, 20)(3, 24)(5, 36)(7, 11)(9, 29)(10, 31)(12, 33)(18, 26)(19, 23)(21,
> 34)(27, 28)(32, 35)
> (2, 27)(5, 29)(7, 11)(8, 17)(9, 36)(10, 12)(14, 22)(18, 32)(19, 23)(20,
> 28)(26, 35)(31, 33)
> (2, 14)(4, 24)(5, 30)(8, 11)(10, 22)(12, 19)(13, 18)(15, 21)(16, 29)(23,
> 27)(25, 32)(28, 33)
> (3, 20)(4, 27)(5, 30)(6, 7)(8, 13)(10, 28)(11, 18)(12, 16)(15, 32)(17,
> 35)(19, 29)(21, 25)(22, 33)(23, 24)(26, 34)(31, 36)
> (1, 2, 12, 19, 9, 3, 14)(4, 6, 5, 8, 18, 22, 29)(7, 33, 28, 15, 17, 21,
> 36)(10, 26, 31, 13, 11, 23, 16)(20, 27, 34, 30, 32, 25, 24)
```

MAGMA realizes groups as permutation groups, that is, as subgroups of some symmetric group. The permutation in the output are the generators of the group $A$.

*Exercise* 4C.3. In MAGMA the command `RandomGraph(n, 1/2)` creates a random graph on $n$ vertices by flipping a coin for each pair of vertices to determine whether MAGMA puts an edge or not. Compute the automorphism groups of random graphs for 'large' $n$ in MAGMA. ($n = 20$ is a good compromise between 'large' and 'fast computation'.) ◇

*Exercise* 4C.4. A ***hyperoval*** in a projective plane of even order $q$ is a set of $q + 2$ points, no three of which are on a line.

(a) Find an example of a hyperoval in the 21-point projective plane. You can begin with the command

```
> plane, points, lines := FiniteProjectivePlane(4);
```

(b) Write a function `IsHyperoval(P,X)` to test whether $X$ is a hyperoval in a projective plane $P$.

(c) Find all the hyperovals in the 21-point projective plane.

(d) Find the orbits of the groups `PGL(3,4)` and `PSL(3,4)` on the set of hyperovals.

Hint 1. What are `points.1` and `points.2`? What is `lines.3`? Hint 2. `Exclude(S,v)` removes the element $v$ from the set $S$. If you want to remove a representative from $S$ and assign it to a variable $x$, use `ExtractRep(S,x)`. ◇

*Exercise* 4C.5. The points and lines of the 21-point plane can be identified with the 1- and 2-dimensional subspaces of a vector space of dimension 3 over the field of 4 elements. In this representation an example of a hyperoval is the set of singular points of a quadratic

form together with its radical. You can use the following code to construct the form and the quadratic space.

```
> P<x,y,z> := PolynomialRing(GaloisField(4),3);
> f := x*y+z^2;
> V := QuadraticSpace(f);
```

Find 6 vectors that represent the points of the hyperoval. Check that they do indeed form a hyperoval.

(Hint. `Radical(V)` is the radical of $V$ and `QuadraticNorm(v)` is the value of the quadratic form at the vector $v$.) ◇

4D. **Exploring small groups: the Small Groups Database.** Let $G$ be a group. A non-empty subset $S$ of $G$ is ***product-free*** if $ab \notin S$ for all $a, b \in S$.

Which finite groups have a maximal (by inclusion) product-free set of size 1, of size 2, of size 3, ...?

Checking if a set is product-free is easy.

```
> prodfree := func< S | forall{<a,b> : a,b in S | a*b notin S } >;
> prodfree({2});
> prodfree({1,2});
```

```
-------result-------
```

```
> true
> false
```

Finding ***all*** the groups with a product-free set of size 1, 2 or 3 is harder.

We will start with maximal product-free sets of size 1.

```
> checkmax1 := function(G)
> for a in G do
> if a eq One(G) then continue; end if;
> found := true;
> for b in G do
> if b eq One(G) or b eq a then continue; end if;
> if prodfree({a,b}) then found := false; continue; end if;
> end for;
> if found then return true, a; end if;
> end for;
> return false, _;
> end function;
```

Let us check a few cyclic groups.

```
> [checkmax1(CyclicGroup(n)) : n in [2 .. 10]];
```

```
-------result-------
```

```
> [ true, true, true, false, false, false, false, false, false ]
```

The results so far suggest that the only cyclic groups containing a product-free set of size 1 are $C_2$, $C_3$ and $C_4$. (Actually, it is quite easy to prove this directly.)

Are there any other groups with a product-free set of size 1?

```
> [checkmax1(DihedralGroup(n)) : n in [3 .. 10]];
```

```
-------result-------
```

```
> [ false, false, false, false, false, false, false, false ]
```

None there. So where else can we look?

MAGMA has a large number of databases containing information that may be used in searches for interesting examples or which form an integral part of certain algorithms.

Examples of current databases include factorizations of integers of the form $pn \pm 1$, $p$ a prime; modular equations; strongly regular graphs; maximal subgroups of simple groups; integral lattices; K3 surfaces; best known linear codes and many others.

We shall use MAGMA's Small Groups Database to get a supply of groups to check for small product-free sets.

Perhaps we can spot a pattern that will lead to a proof of their classification.

The number of groups of order $n$ in the database is returned by (here $n = 32$):

```
> n:=32; NumberOfSmallGroups(n);
  -------result-------
> 51
```

To extract a copy of the $j$th group of order $n$ use (here $j = 3$):

```
> n:=32; j:=3; G := SmallGroup(n,j); G;
  -------result-------
> GrpPC : G of order 32 = 2^5
> PC-Relations:
> G.1^2 = G.3,
> G.2^2 = G.4,
> G.3^2 = G.5
```

Find the groups of order at most 50 that contain a product-free set of size 1.

*Remark* 4D.1. For efficiency, first open the database, using:

```
> SGD := SmallGroupDatabase();
```

then pass the reference as the first argument to the database functions. ◇

*Exercise* 4D.1. Recall the following function:

```
> checkmax1 := function(G)
> for a in G do
> if a eq One(G) then continue; end if;
> found := true;
> for b in G do
> if b eq One(G) or b eq a then continue; end if;
> if prodfree({a,b}) then found := false; continue; end if;
> end for;
> if found then return true, a; end if;
> end for;
> return false, _;
> end function;
```

Find all the groups (of order strictly smaller than 1024 if necessary due to computation limitation) in `SmallGroupDatabase()` that contain a maximal product-free set of size 1. ◇

*Exercise* 4D.2. Modify `checkmax1` and write a function `checkmax2` that can be used to find the groups in `SmallGroupDatabase()` that contain a maximal product-free set of size 2. Make a conjecture about the classification of all finite group with a maximal product-free set of size 2. ◇

```
> for n := 2 to 50 do
> for j := 1 to NumberOfSmallGroups(n) do
> G := SmallGroup(n,j);
> found, witness := checkmax1(G);
> if found then print n,j,witness; end if;
> end for;
```

```
> end for;
```

```
-------result-------
```

```
> 2 1 G.1
> 3 1 G.1
> 4 1 G.2
> 8 4 G.3
```

We know that the groups of orders 2, 3 and 4 are cyclic. What is the structure of the group of order 8?

```
> G := SmallGroup(8,4);
> IsAbelian(G);
```

```
-------result-------
```

```
> false
```

```
> G;
```

```
-------result-------
```

```
> GrpPC of order 8 = 2^3
> PC-Relations:
> $.1^2 = $.3,
> $.2^2 = $.3,
> $.2^$.1 = $.2 * $.3
```

To convert $G$ to a permutation group we can look at its regular representation. This is equivalent to its action on the cosets of the identity subgroup.

```
> f, H, K := CosetAction(G, sub<G | >);
> f;
```

```
-------result-------
```

```
> Homomorphism of GrpPC : G into GrpPerm: H, Degree 8 induced by
> G.1 |--> (1, 6, 2, 5)(3, 8, 4, 7)
> G.2 |--> (1, 4, 2, 3)(5, 7, 6, 8)
> G.3 |--> (1, 2)(3, 4)(5, 6)(7, 8)
```

```
> H;
```

```
-------result-------
```

```
> Permutation group H acting on a set of cardinality 8
> (1, 6, 2, 5)(3, 8, 4, 7)
> (1, 4, 2, 3)(5, 7, 6, 8)
> (1, 2)(3, 4)(5, 6)(7, 8)
```

```
> K;
```

```
-------result-------
```

```
> GrpPC : K of order 1
> PC-Relations:
```

We are almost there!

```
> #{ x : x in H | Order(x) eq 2 };
```

```
-------result-------
```

```
> 1
```

So $H$ is the quaternion group. Let us test this:

```
> Q := Group< a, b | a^2 = b^2, b^a = b^-1 >;
> IsIsomorphic(PCGroup(Q),H);
```

```
-------result-------
```

```
> true Mapping from: GrpPC to GrpPerm: H
> Composition of Mapping from: GrpPC to GrpPC and
> Mapping from: GrpPC to GrpPerm: H
```

Let us wrap-up with three exercises:

*Exercise* 4D.3. For a group $G$ and $m \geq 1$ the ***power*** subgroup $G^m$ is the subgroup generated by the elements $g^m$ where $g$ runs through $G$.

Here are some MAGMA functions to compute the power subgroups and the number of nonpower subgroups.

```
> psg := func< G | { sub<G | [g^m : g in G] > :
> m in Divisors(Exponent(G)) } >;
> nps := func< G | &+[ x`length : x in Subgroups(G) ] - #psg(G) >;
```

Apply this to a few dihedral groups. Is there a pattern?

```
> [<n,nps(DihedralGroup(n))> : n in [3..8]];
  -------result-------
> [ <3, 3>, <4, 7>, <5, 5>, <6, 13>, <7, 7>, <8, 15> ]
```

Find the number of nonpower subgroups in a group of order $n$.

```
> for n := 4 to 10 do
> for j := 1 to NumberOfSmallGroups(n) do
> print [n,j,nps(SmallGroup(n,j))];
> end for;
> end for;
  -------result-------
> [ 4, 1, 0 ]
> [ 4, 2, 3 ]
> [ 5, 1, 0 ]
> .  .  .
```

Describe the results and play around with this as much as you like. For example, increase $n$ above. ◇

*Exercise* 4D.4. Let $G$ be a group. Write a function that returns exactly one representative of $\{x, x^{-1}\}$ for all $x \in G$. ◇

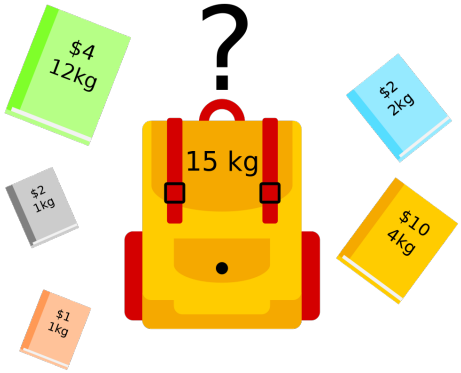
*Exercise* 4D.5. The ***group number function***, also called GNU, $GNU\colon \mathbb{Z}_{\geq 1} \to \mathbb{Z}_{\geq 1}$ outputs the number of groups of a given order. Use MAGMA to compute $GNU(n)$ for all $n < 1000$. An open question is whether the GNU function is surjective, i.e. whether all numbers are group numbers. Write MAGMA code that shows that all numbers strictly smaller than 19 appear as group numbers. ◇

## 4E. **Extended exercise: knapsack.**

*Exercise* 4E.1. Try to understand what the code below is doing. ◇

The ***knapsack problem*** is the following problem in combinatorial optimization: Given a set of items, each with a weight and a value, determine which items to include in the collection so that the total weight is less than or equal to a given limit and the total value is as large as

possible.




Let us modify the question a bit.

*Question* Given a number like 151 (the knapsack), fill it up with items (primes) until it is full. Which items do you have to take?

Here is the code:

```
> knapsack := function(Q, t)
> R := [IntegerRing() | x : x in Q | x le t];
> if &+R lt t then
> return [ ];
> elif t in R then
> return [t];
> else
> for x in R do
> RR := Exclude(R, x);
> s := $$(RR, t - x);
> if not IsEmpty(s) then
> return Append(s, x);
> end if;
> end for;
> return [ ];
> end if;
> end function;
> primes := [ p : p in [1..100] | IsPrime(p) ];
> k := knapsack(primes, 151);
> k, &+k;
```

```
-------result-------
> [ 43, 31, 19, 17, 13, 11, 7, 5, 3, 2 ]
> 151
```

*Exercise* 4E.2. Modify the Knapsack algorithm so that we fill the Knapsack with square numbers instead of primes. ◇

What is happening? The key is that the function `function(Q,t)` calls itself. Moreover:

- We first setup a function `function(Q,t)` taking a subset of the integers $Q$ and a bound $t$. The function environment ends at `end function;`. The remaining three lines just applies the function.
- After the `else` the real code starts; before we just output trivial cases.
- `R` is the set of allowed elements we want to put in the knapsack. In the code above `R` is the number of primes between 1 and 100.

- The code defines `s` recursively by using `function(RR,t-x)`. Then it appends a nonempty `s` to the knapsack.
- We point out that `return` will terminate the function with the given output.
- `&+R;` and `&+k;` sums over the elements of $R$ and $k$. (Careful: this does not work if the sets are empty.)

*Exercise* 4E.3. Output the knapsacks and the sum of their elements from 2 to 151. ◇

What we have seen above is an example of a ***recursion*** – and the machine likes recursions. Here are two ways to compute $n!$:

```
function fac(n)
if n eq 1 then return 1;
else return n*$$(n-1); end if;
end function;
```

```
function factwo(n)
j:=1;
for i in [1..n] do
j:=j*i;
end for;
return j;
end function;
```

The first function is recursive.

*Exercise* 4E.4. Compare the two functions to compute $n!$ using `time`. Hint: The numbers get very large. You can use

```
T:=Time(); x := fac(6000); Time(T);
T:=Time(); y := factwo(6000); Time(T);
```

to hide the output of the factorial. Does MAGMA like recursions? ◇

*Remark* 4E.1. Computing factorials is an intrinsic MAGMA function that one should use instead of the above. In general, always use intrinsic MAGMA functions instead of self-coded ones as MAGMA goes to the kernel where calculations are always faster. ◇

## 5. Lecture 2 – Group theory examples

In this section we give three example from the realm of groups. Before that, let us go a bit more into the type system of MAGMA.

5A. **The type system and coercion.** (Almost) every object in MAGMA belongs to a ***category***, also known as the ***type*** of the object. In addition, every object has a ***parent***.

```
> A := Alt(4); // the alternating group on {1,2,3,4}
> A;
  -------result-------
> Permutation group G acting on a set of cardinality 4
> Order = 12 = 2^2 * 3
> (1, 2)(3, 4)
> (1, 2, 3)
```

```
> Type(A), Type(A.1);
  -------result-------
> GrpPerm GrpPermElt
```

```
> Parent(A.1):Minimal;
```

```
-------result-------
> GrpPerm: A, Degree 4, Order 2^2 * 3
> Generic(A);
-------result-------
> Symmetric group acting on a set of cardinality 4
> Order = 24 = 2^3 * 3
```

There are a large number of built-in functions (intrinsics) in MAGMA with the same name. So the name alone is not enough to determine which function MAGMA will use. The ***signature*** of the function (the number and types of the arguments) will also be used.

```
> G := Sym(4);
> Order(G.1);
-------result-------
> 4
> P := FiniteProjectivePlane(5);
> Order(P);
-------result-------
> 5
```

To see the signatures, type the function name followed by a semicolon.

To see all functions with a given prefix, type the first few letters followed by typing the ***tab*** key once or twice.

```
> Vector
-------result-------
> Vector                    VectorSpaceOverQ        VectorsLimit
> VectorAction              VectorSpaceWithBasis
> VectorSpace               Vectors
```

Suppose that $V$ is a vector space of dimension 3 over the rational numbers. In MAGMA the elements of $V$ are triples of rational numbers; i.e., row vectors. However, a triple `[2,3,7]` represented as a sequence will not be recognized as an element of $V$.

```
> V := VectorSpace(Rationals(),3);
> v := [2,3,7];
> v in V;
-------result-------
> >> v in V;
> Runtime error in 'in': Bad argument types
```

In order to have MAGMA recognize $v$ as an element of $V$ it must be ***coerced*** into $V$.

```
> V!v in V;
-------result-------
> true
> Type(v), Type(V), Type(V!v);
-------result-------
> SeqEnum ModTupFld ModTupFldElt
```

*Exercise* 5A.1. Suppose that $X$ is an invertible 2-by-2 matrix over the finite field $F$ of 11 elements. The function $\theta_M\colon M \mapsto X^{-1}MX$ is a linear transformation of the vector space of all 2-by-2 matrices over $F$. Furthermore $\theta$ is a homomorphism from the general linear group `GL(2,F)` to `GL(4,F)`.

**(a)** Let `F:=GaloisField(11)` and write a MAGMA function that returns the matrix of X with respect to the 'standard basis' of the vector space `KMatrixSpace(F,2,2)`.

**(b)** Find the image of the generators of `GL(2,F)` under the homomorphism $\theta$ and thereby find the order of the images of `GL(2,F)` and `SL(2,F)` in `GL(2,F)`.

Note that `KMatrixSpace(F,2,2).1` returns the basis vectors of `KMatrixSpace(F,2,2)`. ◇

*Exercise* 5A.2. Let $s_1$, $s_2$ and $s_3$ be the ***Pauli matrices*** defined over the Gaussian field $\mathbb{Q}[i]$.

```
> K<i> := QuadraticField(-1);
> s1 := Matrix(K,[[0,1],[1,0]]);
> s2 := Matrix(K,[[0,i],[-i,0]]);
> s3 := Matrix(K,[[1,0],[0,-1]]);
```

and put

```
> t := Matrix(K,[[i,0],[0,i]]);
```

Let $E$ be the subgroup of `GL(2,K)` generated by the above four elements. Show that the matrices $ts_1$, $ts_2$ and $ts_3$ generate the quaternion group $Q$ and $E$ is the central product of a cyclic group of order four and $Q$. ◇

5B. **The Hall–Janko Group.** In 1968 Zvonimir Janko announced the possible existence of two new finite simple groups. Janko assumed (i) center of a Sylow 2-subgroup is cyclic and (ii) the centralizer of the central ***involution*** (i.e., an element of order 2) has a normal subgroup of order $2^5$ whose quotient is the alternating group Alt(5).

If there is one class of involutions, the group order is 50 232 960; otherwise there are two classes of involutions, the order is 604 800 and the group is called either $J_2$ or the ***Hall–Janko group*** HaJ.

The existence of HaJ was established by Marshall Hall and David Wales. They produced three permutations on 100 vertices. Peter Swinnerton-Dyer verified by computer that the permutations generate a simple group satisfying Janko's conditions.

The group HaJ is a subgroup of index 2 in the automorphism group of a graph on 100 points. This is the construction we investigate in the reminder of this section.

The first step is to revisit the construction of the graph built from the points, lines and flags of the 7-point plane.

```
> fano := FiniteProjectivePlane(2);
> P := Points(fano);
> L := Lines(fano);
```

Using just the points and lines, construct a graph with 14 vertices and 28 edges. This time we use an ***indexed sets*** `{@ ··· @}` of vertices.

```
> vertices1 := {@ <-1,i> : i in [1..7] @} join {@ <-2,j> : j in [1..7] @};
> edges1 := { {<-1,i>,<-2,j>} : i,j in [1..7] | P[i] notin L[j] };
> G1 := Graph< vertices1 | edges1 >;
> M1 := AutomorphismGroup(G1);
> CompositionFactors(M1);
```

```
-------result-------

> G
> |  Cyclic(2)
> *
> |  A(1, 7)                    = L(2, 7)
> 1
```

The output of `CompositionFactors(M1)` shows that the automorphism group of `G1` has a normal subgroup which is isomorphic to the simple group `L(2,7)` of linear fraction transformations of the projective line over the field of 7 elements. The quotient is the cyclic group of order 2. (In fact `M1` $\simeq \mathrm{PGL}(2,7)$.)

`L(2,7)` is often written as $\mathrm{PSL}(2,7)$. It is isomorphic to the group $\mathrm{SL}(3,2)$ of $3\times 3$ matrices over the field of 2 elements.

```
> IsIsomorphic(SL(3,2),PSL(2,7));
  -------result-------
> true Homomorphism of SL(3, GF(2)) into GrpPerm: $, Degree 8,
> Order 2^3 * 3 * 7
> induced by
> [1 1 0]
> [0 1 0]
> [0 0 1] |--> (1, 5)(2, 8)(3, 6)(4, 7)
> [0 0 1]
> [1 0 0]
> [0 1 0] |--> (1, 5, 8)(2, 6, 7)
```

Composition factors are simple groups and therefore $\mathrm{SL}(3,2)$ is the derived group of `M1`.

```
> D1 := DerivedGroup(M1);
> tf, _ := IsIsomorphic(D1,SL(3,2)); tf;
  -------result-------
> true
```

The orbits of `D1` are the points and lines of the Fano plane.

```
> Orbits(D1);
  -------result-------
> [
> GSet {@ 1, 7, 4, 5, 6, 2, 3 @},
> GSet {@ 8, 14, 12, 13, 9, 11, 10 @}
> ]
```

A `GSet` is a set with a group action. If `G` is a permutation group, `GSet(G)` is the set on which it acts. Conversely, if `X` is a `GSet`, then `Group(X)` is the group acting on `X`.

*Exercise* 5B.1. Let ***fano*** be the 7-point plane, and as in the lecture, define a graph (call it $Gr1$) on the points and lines by joining each line to the points not on it.

**(a)** Use MAGMA to show that the automorphism group of $Gr1$ is isomorphic to the projective linear group `PGL(2,7)`.

**(b)** Let

```
> P2 := {1..7};
> L2 := {{1+n,1+(n+1) mod 7, 1+(n+3) mod 7} : n in [0..6]};
```

Define a graph $G2$ by joining each triple $X$ in $L2$ to the points in its complement in $P2$. Use MAGMA to show that $G1$ is isomorphic to $Gr2$.

A good start is to check what MAGMA claims the order of `PGL(2,7)` is. ◇

*Exercise* 5B.2. Let $Gr1$ be as in 5B.1, and let $M1$ be its automorphism group.

**(a)** Check that there are 28 involutions of $M1$ not in its derived group $D$.

**(b)** Check that the involutions form a single conjugacy class in $M1$ and that each involution interchanges the orbits of $D$.

**(c)** Check that there are 28 symmetric matrices in `SL(3,2)`. Find a connection between these 28 matrices and the conjugacy class of 28 involutions in $M1$.

**(d)** The stabilizer in $M1$ of a vertex v in the graph $Gr1$ is the subgroup

```
> H := Stabilizer(M1,1);
```

Find the orbits of the stabilizer on the vertices of the graph.

**(e)** By exploring the action of $H$ on its orbits (or otherwise) show that $H$ is isomorphic to `Sym(4)`.

Hint: `OrbitAction(H,orb)`, returns $f$, $S$, $K$, where $f$ is a homomorphism from $H$ to the group $S$ defined by the action of $H$ on *orb*, and $K$ is the kernel of $f$. ◇

*Exercise* 5B.3. Let $Gr2$ be the graph on 36 vertices defined above.

**(a)** Show that the automorphism group of $Gr2$ is isomorphic to the group `SU(3,3)` of 3-by-3 unitary matrices (with coefficients in the field with 9 elements) extended by the field automorphism $x \mapsto x^3$.

**(b)** Show that the automorphism group of the graph $Gr2$ is isomorphic to the group of Lie type $G_2(2)$.

Beware: these exercises are rather tricky. ◇

In the previous section we extended the graph on the points $P$ and lines $L$ of the Fano plane by including the flags $F$ and an additional vertex $\star$.

Recall that a flag is an incident point-line pair.

```
> F := [<i,j> : i,j in [1..7] | P[i] in L[j]];
```

To define the edges we joined

- $\star$ to all of $P$ and $L$,
- a point to the 4 lines not through it,
- a point to the 9 flags which have their line through it,
- a line to the 9 flags which have their point on it,
- flags $(p_1, \ell_1)$ and $(p_2, \ell_2)$ if $p_1 \neq p_2$, $\ell_1 \neq \ell_2$, $p_1 \in \ell_2$ and $p_2 \in \ell_1$.

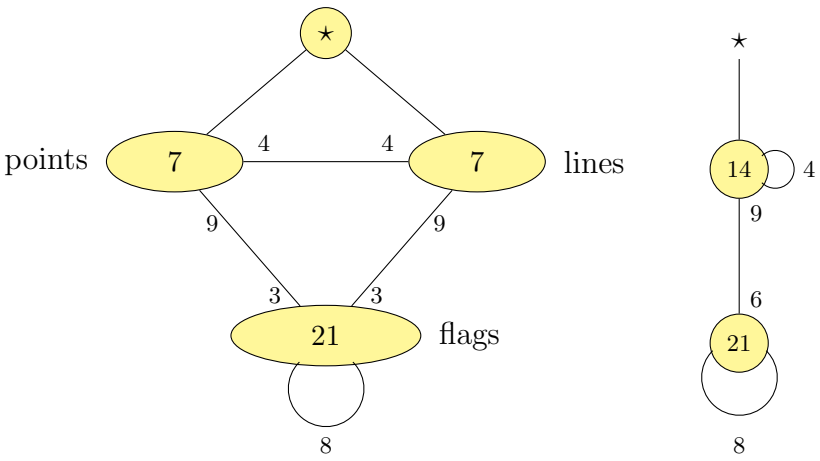
Represent $\star$ by the pair `0, 0`, the point `P[i]` by `-1, i`, the line `L[j]` by `-2, j`, the flag `(P[i],L[j])` by `i, j`.

```
> vertices2 := {@ <0,0> @} join vertices1
> join {@ <i,j> : i,j in [1..7] | P[i] in L[j] @};
> edges2 := {{<0,0>,<-1,i>} : i in [1..7] }
> join { {<0,0>, <-2,i>} : i in [1..7] } join edges1
> join { {<-1,i>,<j,k>} : i,j,k in [1..7] | P[i] in L[k]
> and P[j] in L[k] }
> join { {<-2,i>,<j,k>} : i,j,k in [1..7] | P[j] in L[k]
> and P[j] in L[i] }
> join { {f,g} : f, g in F | f[1] ne g[1] and f[2] ne g[2]
> and (P[f[1]] in L[g[2]] or P[g[1]] in L[f[2]]) };
```

The graph constructor returns the graph, the vertex set and the edge set but we ignore the vertex and edge sets.

```
> G2 := Graph< vertices2 | edges2 >;
```

This graph has 36 vertices, degree 14, 252 edges.




```
> M2 := AutomorphismGroup(G2);
> CompositionFactors(M2);
  -------result-------
>  G
>  |  Cyclic(2)
>  *
>  |  2A(2, 3)                    = U(3, 3)
>  1}
```

```
> D2 := DerivedGroup(M2);
```

The derived group `D2` of `M2` is a subgroup of index 2 isomorphic to the group $\mathrm{SU}(3,3)$ of $3\times 3$ unitary matrices with coefficients in the Galois field $\mathbb{F}_9$ of order 9.

*Exercise* 5B.4. Use MAGMA to show that `M2` is isomorphic to $\mathrm{SU}(3,3)$ extended by the automorphism $\sigma : x \mapsto x^3$ of $\mathbb{F}_9$. ◇

*Exercise* 5B.5. Show that `M2` is isomorphic to the group of Lie type $G_2$ over the field of two elements. ◇

The group $\mathrm{SU}(3,3)$ acts on a vector space of dimension 3 over $\mathbb{F}_9$ and preserves an hermitian form.

```
> J, sigma := StandardHermitianForm(3,3);
> J;
  -------result-------
> [    0    0    1]
> [    0    1    0]
> [    1    0    0]
```

```
> sigma;
  -------result-------
> Mapping from: GF(3^2) to GF(3^2) given by a rule [no inverse]
```

```
> V := UnitarySpace(J,sigma);
> U := SU(3,3);
> forall{ g : g in Generators(U) | IsIsometry(V,g)};
  -------result-------
> true
```

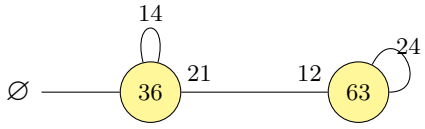
We see from `J` that $(1,0,0)$ is isotropic and $(0,1,0)$ is non-isotropic.

```
> u := V![1,0,0]; v := V![0,1,0];
> DotProduct(u,u), DotProduct(v,v);
  -------result-------
> 0 1
```

The isotropic and non-isotropic 1-dimensional subspaces (i.e., lines) of $V$ afford representations of degrees 28 and 63 of SU(3, 3).

```
> iso := sub<V|u>^U;
> noniso := sub<V|v>^U;
> #iso, #noniso, "= total number of 1-subspaces:",(9^3-1) div (9-1);
  -------result-------
> 28 63 = total number of 1-subspaces: 91
```

The graph `G2` constructed from the Fano plane has 36 vertices. It can be combined with the representation of degree 63 and a new point ∅ to create a regular graph of degree 36 on 100 vertices.




It will be more convenient to label the vertices with the integers 1, 2, . . . , 100.

Convert the edges of the graph on 36 points to the new labeling.

```
> edges := { {Index(vertices2,x) : x in edge} : edge in edges2 };
```

The 63 new vertices are the non-isotropic lines of the unitary space $V$. The stabilizer in SU(3, 3) of a non-isotropic line contains a unique central involution. These involutions are the elements of a conjugacy of size 63 in SU(3, 3). In MAGMA the conjugacy classes are represented by triples `order, size, representative` .

```
> exists(t) { c[3] : c in Classes(M2) | c[1] eq 2 and c[2] eq 63 };
  -------result-------
> true
```

```
> X := Conjugates(M2,t);
```

Convert `X` from a set to a sequence. This will allow us to refer to individual elements.

```
> X := SetToSequence(X);
```

The group `M1` is the stabilizer of a vertex of the graph `G2`. It contains a conjugacy class of 21 involutions that belong to `X`.

```
> edges join:= {{i,j+36} : i in [1..36],j in [1..63] | i^X[j] eq i};
```

We also need the edges between the elements of `X`. If $t \in$ `X`, the edges just defined join $t$ to 12 elements of `G1`. So we need to join $t$ to 24 elements of `X`.

```
> for i in { Order(s*t) : s in X } do
> i,#{ s : s in X | Order(s*t) eq i };
> end for;
  -------result-------
> 1 1
> 2 6
> 3 32
> 4 24
```

```
> edges join:= {{i+36,j+36} : i,j in [1..63] | Order(X[i]*X[j]) eq 4};
```

Let us now see the Wales graph for HaJ: Finally we add the edges from vertex 100 to `G1`, create the graph, check that it is regular and find its automorphism group.

```
> edges join:= { {i,100} : i in [1..36] };
> WalesGraph := Graph< 100 | edges >;
> IsRegular(WalesGraph);
> JJ2 := AutomorphismGroup(WalesGraph);
> CompositionFactors(JJ2);
  -------result-------
> G
> |  Cyclic(2)
> *
> |  J2
> 1
```

*Exercise* 5B.6. Check Janko's conditions for the derived group `J2` of `JJ2`: the center of a Sylow 2-subgroup is cyclic and the centralizer $C$ of a central involution has a normal subgroup $E$ such that $C/E \simeq \mathrm{Alt}(5)$.
Hint 1: `SylowSubgroup`, `Centre`, `Centraliser`.
Hint 2: to find $E$, check out `pCore(C,2)`. What is `C/E`? ◇

5C. **The group determinant.** Suppose that $G$ is a finite group of order $n$. For each $g \in G$ let $x_g$ be an indeterminate.

The determinant of the $n \times n$ matrix $\left(x_{gh^{-1}}\right)_{g,h\in G}$ is the ***group determinant*** of $G$.

What is the group determinant of the dihedral group of order 8?

There is a MAGMA intrinsic to compute dihedral groups. The default is to represent them as permutation groups.

```
> D8 := DihedralGroup(4);
> D8;
  -------result-------
> Permutation group D8 acting on a set of cardinality 4
> Order = 8 = 2^3
> (1, 2, 3, 4)
> (1, 4)(2, 3)
```

Here is a function for the group determinant:

```
> groupDet := function(G)
> n := #G;
> P := PolynomialRing(Integers(),n : Global);
> AssignNames(~P,["x" cat IntegerToString(i) : i in [1..n]]);
> L := Setseq(Set(G)); L := [h*g : g in L] where h is L[1]^-1;
> M := ZeroMatrix(P,n,n);
> for i -> x in L, j -> y in L do
> k := Index(L,x*y^-1);
> M[i,j] := P.k;
> end for;
> return M, Determinant(M);
> end function;
> B,D := groupDet(D8);
> Factorisation(D);
  -------result-------
> [
```

```
> <x1 + x2 - x3 - x4 - x5 - x6 + x7 + x8, 1>,
> <x1 + x2 - x3 - x4 + x5 + x6 - x7 - x8, 1>,
> <x1 + x2 + x3 + x4 - x5 - x6 - x7 - x8, 1>,
> <x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8, 1>,
> <x1^2 - 2*x1*x2 + x2^2 + x3^2 - 2*x3*x4 + x4^2 - x5^2 +
> 2*x5*x6 - x6^2 - x7^2 + 2*x7*x8 - x8^2, 2>
> ]
```

What is happening?

- `P := PolynomialRing(R,n)` — the ring of polynomials in $n$ indeterminates `P.1`, ..., `P.n` with coefficients in $R$.
- `AssignNames` — names for printing.
- `P[x] := PolynomialRing(R,n)` will assign names `x[1],x[2],...` which can be used for input as well as printing.
- `Setseq` is a synonym for `SetToSequence`.
- The `where ... is ...` clause introduces a variable local to the expression to its left.
- `for ... do` — this is ***dual iteration***; `i` is the index of the element `x` in `L`.
- `return` statements can return more than one value.
- use `_` to ignore a return value.

Let us move to another group. There are many ways to construct the quaternion group $Q_8$ in MAGMA. For example, by generators and relations.

```
> Q8<r,s> := Group< x,y | x^2 = y^2, x^y = x^-1 >;
```

The group $Q_8$ is the unique Sylow 2-subgroup and therefore the largest normal 2-subgroup of $\mathrm{SL}(2,3)$.

```
> S := SL(2,3);
> Q8 := pCore(S,2);
> M,gD := groupDet(Q8);
> Factorisation(gD);
  -------result-------
> [
> <x1 - x2 - x3 - x4 + x5 + x6 - x7 + x8, 1>,
> <x1 - x2 - x3 + x4 + x5 - x6 + x7 - x8, 1>,
> <x1 + x2 + x3 - x4 + x5 - x6 - x7 - x8, 1>,
> <x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8, 1>,
> <x1^2 - 2*x1*x5 + x2^2 - 2*x2*x3 + x3^2 + x4^2 - 2*x4*x7
> + x5^2 + x6^2 - 2*x6*x8 + x7^2 + x8^2, 2>
> ]
```

What happened above? Here is a brief curse on naming generators:

```
> S := SL(2,3);
> S.1;
  -------result-------
> [1 1]
> [0 1]
> S<a,b> := SL(2,3);
> print a, b;
  -------result-------
> [1 1]
> [0 1]
> [0 1]
> [2 0]
```

```
> P<x> := PolynomialRing(Rationals());
> F<a> := NumberField(x^2 - x - 1);
> a^2;
```

```
-------result-------
```

```
> a + 1
```

Back to the group determinant. Computing the group determinant quickly becomes very expensive in both time and space.

```
> G := Alt(4);
> time M,D := groupDet(G);
```

```
-------result-------
```

```
> Time: 36.010
```

```
> Factorial(#G), Length(D);
```

```
-------result-------
```

```
> 479001600 417577
```

From version 2.28 it is possible to find out how many results the caller of your function has requested. To avoid computing the determinant when it is not needed you could insert the following lines into the code for `groupDet`.

```
> if NumberOfResults() eq 1 then
> return M;
> else
> return M, Determinant(M);
> end if;
```

*Exercise* 5C.1. Factorize the group determinants of the five groups of order 12. You can get the groups from the Small Groups Database.

Warning. This can take rather a long time. Are there faster ways to factorize the group determinant? ◇

5D. **Central extensions.** A ***central extension*** of a group $G$ is a group $\Gamma$ with a homomorphism $\pi$ from $\Gamma$ onto $G$ such that the kernel of $\pi$ is contained in the center of $\Gamma$.

Let $\pi : \Gamma \to G$ be a central extension and let $A = \ker \pi$. Choose a set $T = \{ x_g \mid g \in G \}$ of coset representatives for $A$ in $\Gamma$ such that $\pi(x_g) = g$.

Then $x_g x_h = \alpha(g,h) x_{gh}$, for some $\alpha : G \times G \to A$. It follows from the associativity of $G$ that $\alpha(xy,z)\alpha(x,y) = \alpha(x,yz)\alpha(y,z)$. That is, $\alpha \in Z^2(G,A)$ is a 2-***cocycle***. The image of $\alpha$ in $H^2(G,A)$ does not depend on the choice of transversal.

Conversely, if $A$ is an abelian group and $\alpha \in Z^2(G,A)$, there exists a central extension $\pi : \Gamma \to G$ with $\ker \pi = A$ and a transversal $\{ x_g \mid g \in G \}$ with $\pi(x_g) = g$ such that $x_g x_h = \alpha(g,h) x_{gh}$.

To find the central extensions of Sym(5) by a group of order 2, for example, first construct the second cohomology group.

```
> G := Sym(5);
> CM := CohomologyModule(G,A) where A is TrivialModule(G,GF(2));
> H2 := CohomologyGroup(CM,2);
> Dimension(H2);
```

```
-------result-------
```

```
> 2
```

Thus `H2` $= H^2(\mathrm{Sym}(5), C_2)$ is a vector space of dimension 2 over the field $\mathbb{F}_2$. It has four elements, each of which defines a central extension.

```
> E0 := Extension(CM,Zero(H2));
> print Type(E0);
  -------result-------
> GrpFP
> P0 := CosetImage(E0,sub<E0|>);
> flag, phi := IsIsomorphic(P0,DirectProduct(CyclicGroup(2),G));
> flag;
  -------result-------
> true
```

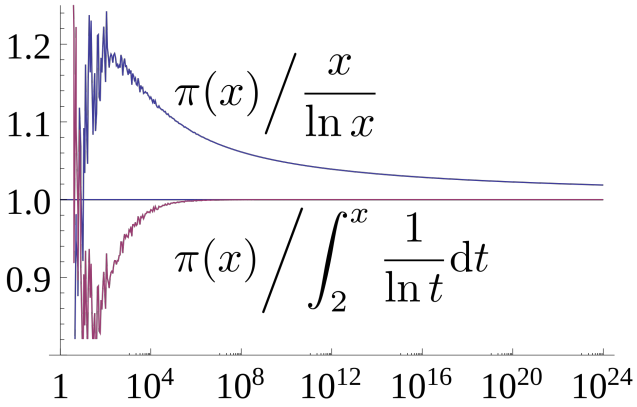
*Exercise* 5D.1. Find the other extensions and describe their structure. ◇

*Exercise* 5D.2. Using MAGMA's cohomology intrinsics find all central extensions of `Sym(5)` by the cyclic group of order two and describe their structure. ◇

## 5E. Extended exercise: prime number theorem.

*Exercise* 5E.1. As before, try to understand what the code below is doing. ◇

The ***prime number theorem*** is the remarkable statement that the number of primes $\leq n$, denoted by $\pi(n)$, is asymptotically given by the logarithmic integral $li(n) = \int_2^n 1/\ln(t)dt$.




The code to verify this is as follows:

```
> PrimeNumberTheorem:=procedure(limit)
> n := 10;
> final_prime := 7;
> true_count := 4;
> // low-precision fields for convenience of printing
> R8 := RealField(8); R4 := RealField(4);
> // the heading for the output
> print "n\tpi(n)\tli(n)\tpi(n)/li(n)\tR(n)\tpi(n)/R(n)";
> print "-"^58
> while n le limit do
> p := NextPrime(final_prime);
> while p le n do
> true_count +:= 1;
> final_prime := p;
> p := NextPrime(final_prime);
> end while;
> li_count   := LogIntegral(R8!n);
> true_on_li := true_count / li_count;
> rie_count  :=&+[LogIntegral(Root(R8!n, k))*MoebiusMu(k)/k : k in [1..15]];
```

```
> true_on_rie:= true_count / rie_count;
> printf "%o\t%o\t%o\t%o\t\t%o\t%o\n",
> n, true_count,
> Round(li_count), (true_on_li lt 1 select R4 else R8)!true_on_li,
> Round(rie_count), (true_on_rie lt 1 select R4 else R8)!true_on_rie;
> n *:= 10;
> end while;
> end procedure;
> PrimeNumberTheorem(1000000);
-------result-------
```

The output is:

```
n       pi(n)   li(n)   pi(n)/li(n)     R(n)    pi(n)/R(n)
-----------------------------------------------------------
10      4       6       0.6487          4       0.8911
100     25      30      0.8298          26      0.9751
1000    168     178     0.9459          168     0.9978
10000   1229    1246    0.9863          1227    1.0016540
100000  9592    9630    0.9961          9587    1.0004697
1000000 78498   78628   0.9984          78527   0.9996
```

Ok, let us go through some of the things happening:

- This time we have a procedure `procedure(limit)` with input being the maximal $n$ we want to run the prime number counting.
- The first interesting bit is the `while` which counts the primes by adding one to the counter `truecount` after finding a prime number.
- `LogIntegral` sets up a logarithmic integral, here with precision 8 determined by fixing `RealField(8)` at the beginning.
- `trueonli` is the ratio between the prime numbers and the logarithmic integral.
- The final step increases $n$ by a factor of ten using `n *:= 10`.

*Exercise* 5E.2. Note that the above does not consider the approximation given by $n/\log(n)$. Include $n/\log(n)$ into the program and determine what $R(n)$ is. ◇

*Exercise* 5E.3. Try to understand how the table output is created in the code. ◇

## 6. Lecture 3 – Lattices and Lie theory

This section is mostly about root systems and their associated Lie theory. We also discuss structure constants algebras.

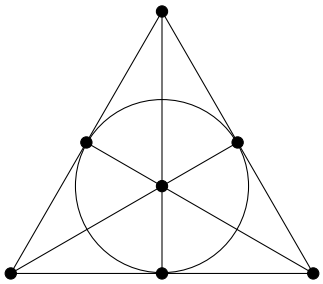
6A. **The octonions.** Let $R$ be a ring. The algebra $\mathbb{O}(R)$ of ***octonions*** over $R$ has a basis $1 = e_1, e_2, \dots, e_8$, such that

$$e_i^2 = -1 \quad \text{for } i \geq 2 \text{ and}$$
$$e_i e_j = \pm e_k \quad \text{for } i, j \geq 2 \text{ and } i \neq j,$$

where the triples $\{i, j, k\}$ form the lines of a 7-point projective plane on the set $\{2, 3, \dots, 8\}$. The signs are determined by setting $e_2e_3 = e_5 = -e_3e_2$ and using the fact that for $i, j \geq 2$ and

$i \neq j$, the elements $e_i$ and $e_j$ generate an associative algebra (quaternions) such that $e_i e_j = e_k$ implies $e_{i+1} e_{j+1} = e_{k+1}$ (subscripts modulo 7).

Fano again:



```
> fano := {@ <2 + n, 2 + (n+1) mod 7, 2 + (n+3) mod 7> : n in [0..6] @};
```

The code

```
> Algebra< R, n | T >;
```

creates a ***structure constant algebra*** with a basis $e_1, \ldots, e_n$ satisfying $e_i e_j = \sum_k a_{ij}^k e_k$, where the sequence `T` contains the 4-tuples $(i, j, k, a_{ij}^k)$ such that $a_{ij}^k \neq 0$.

The structure constant 4-tuple corresponding to $e_2 e_3 = e_5$ is `2,3,5,1` and from this we get five more by applying the symmetric group Sym(3) to the first three indices, taking account of the sign.

```
> T := [<f[1^g],f[2^g],f[3^g],Sign(g)> : g in Sym(3), f in fano];
```

Next add $e_i^2 = -1$ (for $2 \leq i \leq 8$), then the relations $e_1 e_i = e_i e_1 = e_i$.

```
> T cat:= [ <i,i,1,-1> : i in [2..8] ];
> T cat:= [ <1,i,i,1> : i in [1..8] ] cat [<i,1,i,1> : i in [2..8] ];
```

The octonions over the ring `R`:

```
> octonions := func< R | Algebra< R, 8 | T > >;
```

*Remark* 6A.1. There is a MAGMA intrinsic `OctonionAlgebra(K,a,b,c)`, where `K` is a field (of odd or zero characteristic) and `a`, `b` and `c` are parameters. ◇

To check that everything is working, let us print the multiplication table.

```
> OZ := octonions(Integers());
> PA<e1,e2,e3,e4,e5,e6,e7,e8> := PolynomialAlgebra(Integers(),8);
> print Matrix(PA,8,8,
> [&+[Eltseq(OZ.i*OZ.j)[h] * PA.h : h in [1..8]]: i,j in [1..8]]);
  -------result-------
> [ e1  e2  e3  e4  e5  e6  e7  e8]
> [ e2 -e1  e5  e8 -e3  e7 -e6 -e4]
> [ e3 -e5 -e1  e6  e2 -e4  e8 -e7]
> [ e4 -e8 -e6 -e1  e7  e3 -e5  e2]
> [ e5  e3 -e2 -e7 -e1  e8  e4 -e6]
> [ e6 -e7  e4 -e3 -e8 -e1  e2  e5]
> [ e7  e6 -e8  e5 -e4 -e2 -e1  e3]
> [ e8  e4  e7 -e2  e6 -e5 -e3 -e1]
```

For each line of the Fano plane there is a ***quaternion*** subalgebra. For example, the quaternion algebra $\mathbb{H}$ of the triple $[2, 3, 5]$ is the linear span of 1, $e_2$, $e_3$ and $e_5$ and the octonion algebra is $\mathbb{H} \oplus e_8 \mathbb{H}$.

The linear span of $e_2, \ldots, e_8$ is the space of ***pure*** octonions.

If $\xi = ae_1 + \eta$, where $a$ is a scalar, and $\eta$ is a pure octonion, the ***conjugate*** of $\xi$ is $\overline{\xi} = ae_1 - \eta$.

The ***norm*** of $\xi$ is defined by $\xi\overline{\xi} = \overline{\xi}\xi = \mathrm{norm}(\xi)e_1$.
The ***trace*** of $\xi$ is defined by $\xi + \overline{\xi} = \mathrm{trace}(\xi)e_1$.

$$\xi^2 - \mathrm{trace}(\xi)\xi + \mathrm{norm}(\xi)e_1 = 0.$$

```
> conj := func< xi | 2*xi[1]*Parent(xi)!1-xi>;
> norm := func< xi | (xi*conj(xi))[1] >;
> trace := func< xi | 2*xi[1] >;
> F<z1,z2,z3,z4,z5,z6,z7,z8> := FunctionField(Integers(),8);
> OF := octonions(F);
> x := OF![z1,z2,z3,z4,z5,z6,z7,z8];
> norm(x), trace(x), trace(x*OF.3);
  -------result-------
> z1^2 + z2^2 + z3^2 + z4^2 + z5^2 + z6^2 + z7^2 + z8^2
> 2*z1
> -2*z3
```

*Exercise* 6A.1. Let $A = \mathbb{O}(\mathbb{Q})$ denote the algebra of octonions over the field `Rational()`, see above.

**(a)** Let a be the matrix corresponding to the permutation $(2,3,4,5,6,7,8)$. Show that a is an automorphism of $A$ that permutes the vectors $\pm e_i$.

**(b)** Let $b0$ be the permutation $(2,7)(3,4)$. Show that $b0$ is an automorphism of the 7-point plane defined by

```
> fano := {@ <2+n, 2+(n+1) mod 7, 2+(n+3) mod 7> : n in [0..6] @};
```

Then find a diagonal matrix $d = diag(\pm 1, \ldots, , \pm 1)$ such that $db$ is an automorphism of $A$ that permutes the vectors $\pm e_i$, where $b$ is the permutation matrix of $b0$.

**(c)** Let $G$ be the subgroup generated by the matrices $a$ and $db$. Show that the order of $G$ is 1344 and that $G$ has a normal abelian subgroup $E$ of order 8 such that the quotient $G/E$ is isomorphic to `SL(3,2)`. Furthermore, this extension is non-split; that is, there is no subgroup of $G$ isomorphic to `SL(3,2)`.

Hint: `PermutationMatrix(...)` ◇

6B. **Lattices and root systems.** A ***lattice*** in MAGMA is a free $\mathbb{Z}$-module contained in $\mathbb{Q}^n$ or $\mathbb{R}^n$, with a positive definite inner product taking values in $\mathbb{Q}$ or $\mathbb{R}$.

A subring of a finite dimensional algebra $A$ over $\mathbb{Q}$ is an ***order*** if it is a lattice in $A$ (and contains a basis of $A$).

An order is ***integral*** over $\mathbb{Z}$ (i.e., every element is the root of polynomial with coefficients in $\mathbb{Z}$).

```
> B :=  Matrix([[1,2,3],[3,2,1]]);
> L := Lattice(B);
> AmbientSpace(L); //{\black returns two objects}
  -------result-------
> Full Vector space of degree 3 over Rational Field
> Mapping from: Lat: L to Full Vector space of degree 3 over
> Rational Field given by a rule [no inverse]
> Rank(L);
  -------result-------
> 2
```

An element of $\mathbb{O}(\mathbb{Q})$ is ***integral*** if its trace and norm are integers.

A subring of $\mathbb{O}(\mathbb{Q})$ is an order if its elements are integral; e.g. $\mathbb{O}(\mathbb{Z})$.

There are seven maximal orders in $\mathbb{O}(\mathbb{Q})$ that contain $\mathbb{O}(\mathbb{Z})$; they are pairwise isomorphic.

An order containing $\mathbb{O}(\mathbb{Z})$ is spanned by $e_i$ $(1 \le i \le 8)$ and elements of the form $\frac{1}{2}(\pm e_{h_1} \pm e_{h_2} \pm e_{h_3} \pm e_{h_4})$.

Let $\mathbb{O}_\mathbb{Z}$ denote the lattice spanned by $\mathbb{O}(\mathbb{Z})$ and $\frac{1}{2}(e_{h_1}+e_{h_2}+e_{h_3}+e_{h_4})$, where $\{h_1,h_2,h_3,h_4\}$ or its complement in $\{1,\dots,8\}$ has the form $\{1,i,j,k\}$ and $\{i,j,k\}$ is a line of the Fano plane with 1 and 2 swapped.

```
> X := { Include( {h^pi : h in line}, 2 ) : line in fano }
> where pi is Sym(8)!(1,2); X;
```

```
-------result-------
```

```
> {1,2,3,5},{1,2,4,8},{1,2,6,7},{1,5,7,8},
> {1,3,6,8},{1,3,4,7},{1,4,5,6}
```

Conway calls $\mathbb{O}_\mathbb{Z}$ the ***octavian integers***; it is a maximal order.

The units in $\mathbb{O}_Z$ are the elements of norm 1. They form a ***Moufang loop*** $\mathcal{M}$ of order 240.

```
> X join:= {{1..8} diff x : x in X };
> X := { SetToSequence(x) : x in X };
> OQ := octonions(Rationals());
> B := Basis(OQ);
> M := { a*x : x in B, a in {1,-1} };
> M join:= {(a*B[p[1]]+b*B[p[2]]+c*B[p[3]]+d*B[p[4]])/2 :
> a,b,c,d in {1,-1}, p in X};
> #M, forall{ <x,y> : x,y in M | x*y in M };
```

```
-------result-------
```

```
> 240 true
```

*Exercise* 6B.1. Show that the elements of $\mathcal{M}$ satisfy the alternative laws: $(xy)x = x(yx)$, $x(xy) = x^2y$, $(xy)y = xy^2$ but $\mathcal{M}$ is not associative. ◇

*Exercise* 6B.2. Show that every element of $\mathcal{M}$ has an inverse. ◇

Now let us move to root systems. The ***reflection*** $r_\alpha$ in the hyperplane orthogonal to a non-zero vector $\alpha$ in a vector space $V$ with inner product $(u,v)$ is given by

$$vr_\alpha = v - \frac{2(v,\alpha)}{(\alpha,\alpha)}\alpha.$$

In $\mathbb{O}(\mathbb{Q})$ we have $(u,v) = u\overline{v} + v\overline{u}$ and so $vr_\alpha = -\alpha\overline{v}\alpha/\alpha\overline{\alpha}$.

```
> ref := func< a, v | -a*conj(v)*a / norm(a) >;
> refmat := func< a | MatrixRing(BaseRing(P),Dimension(P))!
> [ref(a,x) : x in Basis(P)] where P is Parent(a) >;
```

We claim that the Moufang loop $\mathcal{M}$ is a root system. That is

- $0 \notin \mathcal{M}$.
- For all $\alpha \in \mathcal{M}$ the reflection $r_\alpha$ leaves $\mathcal{M}$ invariant.
- For all $\alpha, \beta \in \mathcal{M}$ the inner product $(\alpha,\beta)$ is an integer.

*Exercise* 6B.3. Use MAGMA to check the claim. ◇

First find a set of positive roots

```
> z := OQ![2^i : i in [1..8]];
> P := {@ v : v in M | InnerProduct(z,v) gt 0 @}; #P;
```

```
-------result-------
```

```
> 120
```

A ***simple root*** is a positive root that is not the sum of positive roots.

```
> S := P diff {@ u+v : u,v in P | u+v in P @};
> V := VectorSpace(OQ);
> SV := ChangeUniverse(S,V);
> C := Matrix(Integers(),8,8,[2*(a,b)/(b,b) : a,b in SV]);
> C; //\black Cartan matrix
  -------result-------
> [ 2 -1 -1 -1  0  0  0  0]
> [-1  2  0  0  0  0  0 -1]
> [-1  0  2  0  0  0  0  0]
> [-1  0  0  2  0  0 -1  0]
> [ 0  0  0  0  2 -1  0  0]
> [ 0  0  0  0 -1  2 -1  0]
> [ 0  0  0 -1  0 -1  2  0]
> [ 0 -1  0  0  0  0  0  2]
```

We are ready for $E_8$, Coxeter group, Dynkin diagram, automorphisms. The octavian ring $\mathbb{O}_{\mathbb{Z}}$ is the $E_8$ root lattice.

```
> W := CoxeterGroup(C);
> DynkinDiagram(W);
  -------result-------
> E8    8 - 2 - 1 - 4 - 7 - 6 - 5
> |
> 3
```

$w \in \mathbb{O}_{\mathbb{Z}}$ has order 3 if and only if its norm is 1 and trace is $-1$.

```
> M3 := [ x : x in M | trace(x) eq -1 ];
> forall{ w : w in M3 | w^3 eq 1 };
  -------result-------
> true
```

If $w$ has order 3, the map $x \mapsto \overline{w}xw$ is an automorphism of $\mathbb{O}_{\mathbb{Z}}$.

```
> aut := func< a, v | a^3 eq 1 select a^2*v*a else 0 >;
> autmat := func< a | MatrixRing(BaseRing(P),Dimension(P))!
> [aut(a,x) : x in Basis(P)] where P is Parent(a) >;
> forall{ <s,t,w> : s,t in S, w in M3 | aut(w,s*t) eq aut(w,s)*aut(w,t)};
  -------result-------
> true
```

The automorphism group of $\mathbb{O}_{\mathbb{Z}}$ is as follows.

```
> reps := [ Rep(Q) : Q in {{x,x^-1} : x in M3}];
> gens := [ autmat(w) : w in reps ];
> G := sub<GL(8,Rationals()) | gens >;
> CompositionFactors(G); #G;
  -------result-------
> G
> |  2A(2, 3) = U(3, 3)
> 1
> 6048
```

*Exercise* 6B.4. Show that the elements `gens` are involutions and that `G` can be generated by three of them. ◇

*Exercise* 6B.5. Find the orbits of `G` on $\mathcal{M}$ and their lengths. ◇

The map $x \mapsto \overline{x}$ is an automorphism of $\mathbb{O}_{\mathbb{Z}}$; its matrix is

```
> conjmat := MatrixRing(Rationals(),8)![ conj(b) : b in Basis(OQ) ];
> A := sub<GL(8,Rationals()) | G, conjmat >; #A;
  -------result-------
> 12096
```

*Exercise* 6B.6. Show that `A` is the full automorphism group of $\mathbb{O}_{\mathbb{Z}}$. ◇

6C. **Reductive groups.** A reductive group is defined by a ***root datum*** and a field.

A ***root datum*** is a 4-tuple $\mathcal{R} = (X, \Phi, Y, \Phi^\star)$ where $X$ and $Y$ are lattices in duality with respect to a pairing $(-,-) : X \times Y \to \mathbb{Z}$, and $\Phi \subset X$ and $\Phi^\star \subset Y$ are root systems with a bijection $\Phi \to \Phi^\star : \alpha \mapsto \alpha^\star$ such that $(\alpha, \alpha^\star) = 2$. For $\alpha \in \Phi$, the ***reflections***

$$s_\alpha : X \to X : x \mapsto x - (x, \alpha^\star)\alpha \quad \text{and}$$
$$s_\alpha^\star : Y \to Y : y \mapsto y - (\alpha, y)\alpha^\star$$

satisfy $\Phi s_\alpha = \Phi$ and $\Phi^\star s_\alpha^\star = \Phi^\star$.

The ***Weyl group*** of $\mathcal{R}$ is $(s_\alpha \mid \alpha \in \Phi)$.

The root datum is completely determined by its ***simple roots*** and ***simple coroots***.

```
> RD := RootDatum("E7" : Isogeny := "SC");
  -------result-------
> RD: Simply connected root datum of dimension 7 of type E7
```

Let $e_1, e_2, \ldots, e_d$ be a basis for $X$, let $f_1, f_2, \ldots, f_d$ be the dual basis for $Y$ and use these bases to identify $X$ and $Y$ with the standard lattice $\mathbb{Z}^d$.

Choose a base of simple roots $\alpha_1, \ldots, \alpha_\ell$ for $\Phi$.
Then $\alpha_i = \sum_{j=1}^d a_{ij} e_j$ and $\alpha_i^\star = \sum_{j=1}^d b_{ij} f_j$ and $C = (\alpha_i, \alpha_j^\star) = AB^\top$, where $A = \big(a_{ij}\big)$ and $B = \big(b_{ij}\big)$.

Conversely, a pair of $\ell \times d$ matrices $A$ and $B$ such that $AB^\top$ is a Cartan matrix determines a root datum $\mathcal{R}$. The rows of $A$ are the simple roots and the rows of $B$ are the corresponding coroots.

The ***semisimple rank*** of $\mathcal{R}$ is $\ell$, the number of simple roots; the ***reductive rank*** is $d$, the rank $d$ of $X$.

Isogeny: the root datum is ***semisimple*** if $\ell = d$; it is ***adjoint*** if $X = \mathbb{Z}\Phi$;
it is ***simply connected*** if $Y = \mathbb{Z}\Phi^\star$.

Adjoint and simply connected root data are necessarily semisimple.

Now, a MAGMA example:

```
> RD := RootDatum("G2");
> A := SimpleRoots(RD); A;
  -------result-------
> [1 0]
> [0 1]
> B := SimpleCoroots(RD); B;
  -------result-------
> [ 2 -3]
> [-1  2]
> CartanMatrix(RD) eq A*Transpose(B);
```

```
-------result-------
> true

> RD eq RootDatum(A,B);
-------result-------
> true
```

*Exercise* 6C.1. Find all semisimple root data (up to isomorphism) of type $A_3$. (Hint: Let $C$ be a Cartan matrix of type $A_3$ and consider factorizations $C = AB^\top$.) ◇

Suppose that `RD` is a root datum $(X, \Phi, Y, \Phi^\star)$

If `A` is a ring, `GroupOfLieType(RD,A)` creates a group of ***Lie type***.

The generators are ***root elements*** $x_\alpha(a)$ and ***torus elements*** $y \otimes t$, where $\alpha \in \Phi$, $a \in A$, $y \in Y$ and $t \in A$ $(t \neq 0)$.

```
> RD := RootDatum("G2");
> F := GaloisField(5);
> G := GroupOfLieType(RD,F);
> Random(G);
-------result-------
> x2(2) x3(2) x6(1) x4(4) x5(3) x1(3)
> (2 1)
> n1 n2 n1 n2 n1
> x3(2) x6(3) x4(3) x5(3) x1(1)
```

$(2, 1)$ is the torus element $(f_1 \otimes 2)(f_2 \otimes 1)$; $elt < G|Vector(F, [2, 1]) >$.

`n1 n2 n1 n2 n1` is the Weyl group element corresponding to the product of reflections $s_{\alpha_1} s_{\alpha_2} s_{\alpha_1} s_{\alpha_2} s_{\alpha_1}$; $elt < G|1, 2, 1, 2, 1 >$.

The ***weight lattice*** is $\Lambda = \{ x \in \mathbb{Q}\Phi \mid (x, \alpha^\star) \in \mathbb{Z} \text{ for all } \alpha \in \Phi \}$. It has a basis $\varpi_1, \ldots, \varpi_\ell$ of fundamental weights dual to the simple coroots. A weight $\lambda \in \Lambda$ is ***dominant*** if $(\lambda, \alpha^\star) \geq 0$ for all simple roots $\alpha$; i.e., a non-negative linear combination of the fundamental weights.

Let $L$ be a finite-dimensional rational $\mathbf{G}$-module, where $\mathbf{G}$ is a reductive group. Then $L = \bigoplus_\lambda L_{\lambda \in \Lambda}$, where

$$L_\lambda = \{ v \in L \mid v(y \otimes t) = t^{(\lambda, y)} v \text{ for all } y \in Y, t \in K^\times \}$$

and $\lambda$ is a ***weight of*** $L$ if $L_\lambda \neq 0$. If $\mathbf{G}$ is semisimple and $\lambda$ is a dominant weight, there is an irreducible $\mathbf{G}$-module whose ***highest weight*** is $\lambda$. The restriction to a finite group of Lie type need not be irreducible.

```
> G := GroupOfLieType(RD,GF(3));
> rho := HighestWeightRepresentation(G,[3,0]); rho;
-------result-------
> Mapping from: GrpLie: G to GL(77, GF(3)) given by a rule
> [no inverse]

> IsIrreducible(Image(rho));
-------result-------
> false

> RD := RootDatum("G2" : Isogeny := "SC");
```

If $t$ is a field element, the MAGMA code for $x_{\alpha_i}(t)$, where $\alpha_i$ is the $i$th root in the group `G` of Lie type is $elt < G| < i, t > >$.

Using a the function field (i.e., the ring of fractions of the polynomial ring) of the finite field $\mathbb{F}_5$ we can carry out symbolic calculations.

```
> FF<w,z> := FunctionField(GF(5),2);
> G := GroupOfLieType(RD,FF);
> elt<G| <1,w>> * elt<G|<2,z>>;
  -------result-------
x2(z) x3(w*z) x6(w^3*z^2) x4(w^2*z) x5(w^3*z) x1(w)
> std := StandardRepresentation(G); std(TorusTerm(G,3,z));
  -------result-------
> [    z     0     0     0     0     0     0]
> [    0   z^2     0     0     0     0     0]
> [    0     0   1/z     0     0     0     0]
> [    0     0     0     1     0     0     0]
> [    0     0     0     0     z     0     0]
> [    0     0     0     0     0 1/z^2     0]
> [    0     0     0     0     0     0   1/z]
```

Let us now discuss an application: constructive recognition

Given a group $H$ with generators $Y$, construct an isomorphism between $H$ and a 'standard copy'. Use this this to write an arbitrary element of $H$ as a ***straight-line program*** (SLP) in $Y$.

If we know that $H$ is a homomorphic image of a simply connected finite group of Lie type $G(q)$ we can do the following.

- Identify the Lie type of $H$.
- Use the Liebeck–O'Brien algorithm to construct a Curtis–Steinberg–Tits presentation for $H$.
- Construct a homomorphism $\rho : G(q) \to H$ using the CST generators of $G(q)$.
- Construct $\varphi : H \to G(q)$ such that $\rho(\varphi(h)) = h$. For $h \in H$, $\varphi(h)$ will be a word in the Steinberg generators of $G(q)$.

Let $C$ be the algebra of octonions over the finite field $\mathbb{F}_q$ of $q$ elements and suppose that $q$ is odd. We shall construct $A = \mathrm{Aut}(C)$ as a matrix group and then find an explicit isomorphism with a group of Lie type defined by Chevalley–Steinberg generators.

```
> q := 5;
> C := octonions(GF(q));
```

In order to proceed we need some automorphisms.

An ***orthogonal pair*** is an ordered pair $(a, b)$ of elements of norm 1 in $C$ such that $a$ and $b$ are orthogonal to 1 and to each other. Equivalently, $(a, b)$ is an orthogonal pair if $a^2 = b^2 = -1$ and $ab = -ba$. Thus the linear span of 1, $a$, $b$ and $ab$ is a 'quaternion algebra'.

Theorem. The automorphism group of $C$ acts transitively on the set of orthogonal pairs.

For a proof, read the function coming up!

Given orthogonal pairs `p1` and `p2`, the following function returns the matrix of an automorphism of $\mathbb{O}(q)$ transforming `p1` to `p2`.

```
> orthogPairAut := function(p1,p2)
> a1, b1 := Explode(p1);
> a2, b2 := Explode(p2);
> C := Parent(a1);
> V := VectorSpace(C);
> B1 := [V| One(C), a1, b1, a1*b1 ];
> B1perp := OrthogonalComplement(V,sub<V|B1>);
> assert exists(c1){ c : v in B1perp | norm(c) ne 0 where c is C!v};
> mu := norm(c1);
> B1 cat:= [V| c1, c1*a1, c1*b1, c1*(a1*b1) ];
```

```
> B2 := [V| One(C), a2, b2, a2*b2 ];
> B2perp := OrthogonalComplement(V,sub<V|B2>);
> assert exists(c2){ d : v in B2perp | norm(d) eq mu where d is C!v};
> B2 cat:= [V| c2, c2*a2, c2*b2, c2*(a2*b2) ];
> return Matrix(B1)^-1*Matrix(B2);
> end function;
```

Here is another version of `orthogPairAut`:

```
> orthogPairAut2 := function(p1,p2)
> extendBasis := function(p : lambda := 0) //{\black local function}
> a, b := Explode(p);
> assert a^2 eq -1 and b^2 eq -1 and a*b eq -b*a; //{\black error check}
> C := Parent(a);
> V := VectorSpace(C);
> B := [V| One(C), a, b, a*b];
> Bperp := OrthogonalComplement(V,sub<V|B>);
> c := (lambda eq 0) select rep{c : v in Bperp | norm(c) ne 0
> where c is C!v}
> else rep{c : v in Bperp | norm(c) eq lambda where c is C!v};
> return B cat [V| c*C!x : x in B], norm(c);
> end function;
> B1, lambda := extendBasis(p1);
> B2, _ := extendBasis(p2 : lambda := lambda);
> return Matrix(B1)^-1*Matrix(B2);
> end function;
```

The lines of the Fano plane provide a supply of orthogonal pairs.

```
> p1 := <C.2,C.3>;
> auts := [orthogPairAut(p1,<C.i,C.j>) : pp in fano[2..7] |
> true where i,j is Explode(pp)];
> L := sub< GL(8,q) | auts >; #L;
  -------result-------
> 1344
```

Not quite large enough. Let us find another automorphism.

```
\magmacode
> a := &+[C.i : i in [3..8]];
> b := C![0,0,3,2,3,0,2,0];
> a^2 eq -1, b^2 eq -1, a*b + b*a eq 0;
  -------result-------
> true true true
> g := orthogPairAut(p1,<a,b>);
> A := sub<GL(8,q) | L, g >;
> LieType(A,5);
  -------result-------
> true <G, 2, 5>
```

*Exercise* 6C.2. Use MAGMA to find `b` (or equivalent). ◇

*Exercise* 6C.3. The MAGMA code

```
> P<x> := PolynomialRing(Rationals());
> F<t> := NumberField(x^2-x-1);
```

creates the field $F$ generated over the rationals by the element $t$ such that $t^2 - t - 1 = 0$. Then the code

```
> H<i,j,k> := QuaternionAlgebra< F | -1,-1 >;
```

creates the algebra of quaternions over $F$ with basis $1, i, j, k$ such that

$$i^2 = j^2 = k^2 = ijk = -1.$$

Let

```
> p := (1/2)*(-1+i+j+k);
> s := (1/2)*(t-1+i+t*j);
> X := {H ! 1,p,s};
```

and let $I$ be the smallest multiplicatively closed subset of $H$ containing $X$.

Show that $I$ is isomorphic to `SL(2,5)`. Moreover, show that $I$ is a root system (when considered as a subset of $H$). What is its Cartan type? ◇

6D. **G2.** Here is the group $G_2(q)$:

```
> G := GroupOfLieType("G2",q : Isogeny := "SC");
> flag, _, _, _, _, X, _ :=
>    ExceptionalConstructiveRecognition(A,G,2,5);
> rho := Morphism(G,X[1],X[2] : GS);
> rho(elt<G|<1,2>>);
  -------result-------
> [1 0 0 0 0 0 0]
> [0 4 3 0 3 3 2 2]
> [0 1 4 4 3 2 4 3]
> [0 4 2 4 3 4 2 4]
> [0 2 2 2 1 0 2 2]
> [0 3 2 0 0 4 1 4]
> [0 4 0 2 3 0 4 1]
> [0 3 2 1 3 1 4 1]

> f := Inverse(rho);
> f(A.1);
  -------result-------
> x2(1) x3(2) x6(3) x5(3) n2 n1 n2 n1 n2 x2(4) x3(3) x5(2)
```

Miscellaneous properties of $\mathrm{Aut}(\mathbb{O}(q))$ are:

```
> FactoredOrder(A);
  -------result-------
> [ <2, 6>, <3, 3>, <5, 6>, <7, 1>, <31, 1> ]

> M := GModule(A);
> DirectSumDecomposition(M);
  -------result-------
> [
> GModule of dimension 1 over GF(5),
> GModule of dimension 7 over GF(5)
> ]
```

Borel subgroup

```
> bgens := [ elt<G| <1,1>> ,elt<G|<2,1>> ];
> borel := sub<A | [rho(x) : x in bgens] >;
> FactoredOrder(borel);
```

```
-------result-------
> [ <5, 6> ]
```

Torus

```
> tgens := [TorusTerm(G,i,2) : i in [1,2]];
> torus := sub< A | [rho(x) : x in tgens] >;
> FactoredOrder(torus);
-------result-------
> [ <2, 4> ]
```

MAGMA cannot compute the stabilizer of `C.2` directly nor can `C.2` be coerced directly into the module `M`. Instead, we do the following.

```
> A1 := Stabiliser(A,Vector(C.2));
> CompositionFactors(A1);
-------result-------
> G
> |  A(2, 5)  = L(3, 5)
> 1
```

The group `A1` $\simeq \mathrm{PSL}(3,5)$ is not maximal. It has index 2 in its normalizer.

```
> N1 := Normaliser(A,A1);
> Index(N1,A1);
-------result-------
> 2
```

However, `N1` is maximal because the action of `A` on the cosets of `N1` is primitive.

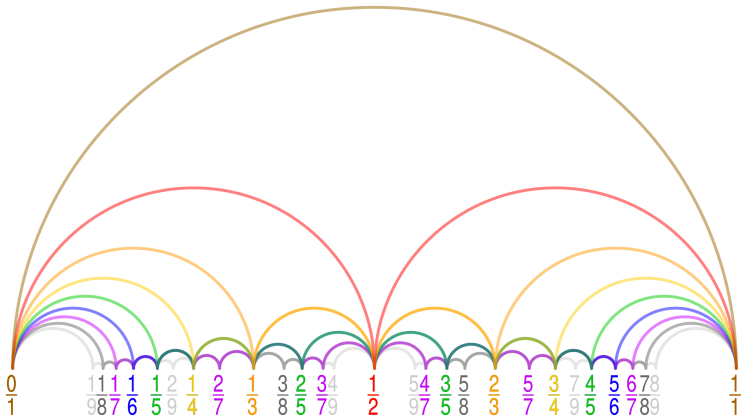
```
> B := CosetImage(A,N1);
> IsPrimitive(B);
-------result-------
> true
```

*Exercise* 6D.1. Let $p$ be a prime and let $S$ be the simply connected group of Lie type A and rank 1 over the finite field of $p$ elements. For $p = 2, 3, 5$ find the dimensions of the highest weight representations of $S$ (as computed by MAGMA)? ◇

## 6E. **Extended exercise: Farey fractions.**

*Exercise* 6E.1. As already twice before, try to understand what the code below is doing. ◇

The ***Farey sequence*** of order $n$ is the sequence of completely reduced fractions, say between 0 and 1, which when in lowest terms have denominators less than or equal to $n$, arranged in order of increasing size.




Code producing these fractions is:

```
> D := Denominator;
> N := Numerator;
> function farey(n)
> if n eq 1 then
> return [RationalField() | 0, 1 ];
> end if;
> f := farey(n-1);
> i := 0;
> while i lt #f-1 do
> i +:= 1;
> if D(f[i]) + D(f[i+1]) eq n then
> Insert(~f, i+1, (N(f[i]) + N(f[i+1]))/(D(f[i]) + D(f[i+1])));
> end if;
> end while;
> return f;
> end function;
> time f100 := farey(100);
> farey(10);
  -------result-------
> Time: 0.130
> [ 0, 1/10, 1/9, 1/8, 1/7, 1/6, 1/5, 2/9, 1/4, 2/7, 3/10, 1/3, 3/8, 2/5,
> 3/7, 4/9, 1/2, 5/9, 4/7, 3/5, 5/8, 2/3, 7/10, 5/7, 3/4, 7/9, 4/5, 5/6,
> 6/7, 7/8, 8/9, 9/10, 1 ]
```

Instead of spoiling the exercise, let us keep it short:

- This is again a recursion and the function calls itself for a lower value.
- `n eq 1` sets the recursion minimum.
- `farey(n-1)` is the same function for smaller $n$.

*Exercise* 6E.2. Modify the program so that it outputs Farey sequences for the interval $[a, b]$ where $a < b$ are two input values. ◇

## 7. Lecture 4 – Representation theory of finite dimensional algebras

MAGMA has two different ways to study representation theory of finite dimensional algebras: by directly constructing the representations, or using character theory. The latter is for finite groups only, and we will start with it.

7A. **Character theory.** Throughout we consider finite groups. Recall that the ***character*** of a group representation is a function on the group that associates to each group element the trace of the corresponding matrix. The character carries the essential information about the representation in a more condensed form, and the easiest way to study representations of a finite group is to look at their character.

Recall that characters are constant on conjugacy classes, so it is enough to record them in a sequence $(c_1, \dots, c_k)$ where $c_i$ is the value of the character on the $i$th conjugacy class.

The first thing to try is:

```
> G:=SymmetricGroup(3);
> X:=CharacterTable(G);
> X
  -------result-------
> Character Table of Group G
>  -------------------------
```

```
>
>
> -----------------
> Class |   1  2  3
> Size  |   1  3  2
> Order |   1  2  3
> -----------------
> p  =  2   1  1  3
> p  =  3   1  2  1
> -----------------
> X.1   +   1  1  1
> X.2   +   1 -1  1
> X.3   +   2  0 -1
```

What do we see? Well, let us ignore finite characteristic and different fields for now, that is let us ignore

```
> -----------------
> p  =  2   1  1  3
> p  =  3   1  2  1
> -----------------
```

and the sign + in the table.

Then what we see is from top to bottom, a numbering for the conjugacy classes, their sizes, their order (that is, the order of any element in the conjugacy class). Finally, the square matrix of characters, with the rows being the ***character sequences*** as above.

Let us double check the conjugacy classes:

```
> ConjugacyClasses(G);
  -------result-------
> Conjugacy Classes of group G
> ----------------------------
> [1]     Order 1       Length 1
> Id(G)
>
> [2]     Order 2       Length 3
> (1, 2)
>
> [3]     Order 3       Length 2
> (1, 2, 3)
```

which works out with

```
> -----------------
> Class |   1  2  3
> Size  |   1  3  2
> Order |   1  2  3
> -----------------
```

MAGMA can do character computations really fast. Here are slightly bigger examples, using the group databases `SmallGroup(n,j)` and `Group("X")`:

```
> G := SmallGroup(512,11600);
> time X := CharacterTable(G);
> #X;
> Degree(X[14]);
  -------result-------
```

```
> Time: 0.030
> 44
> 2
```

```
> G := Group("HS");
> #G;
> time X := CharacterTable(G);
> #X;
> Degree(X[14]);
  -------result-------
> Time: 0.160
> 44352000
> 24
> 896
```

Back to `SymmetricGroup(3)`. To get a specific row = character, or even a specific entry, we can use for example:

```
> X[3]; X[3][2];
  -------result-------
> ( 2, 0, -1 )
> 0
```

Note however that a character is not a sequence. And that is good, because we can add and multiply characters:

```
> Type(X[3]); X[3]+X[2]; X[3]*X[2];
  -------result-------
> AlgChtrElt
> ( 3, -1, 0 )
> ( 2, 0, -1 )
```

In fact, characters live in the character ring, so there are some other (not quite representation theoretical) operations, e.g.:

```
> X[3]-X[2]; 4*X[3];
> IsCharacter(X[3]-X[2]); IsGeneralizedCharacter(X[3]-X[2]);
  -------result-------
> ( 1, 1, -2 )
> ( 8, 0, -4 )
> false
> true
```

Some more operations (hopefully self-explanatory) on characters are:

```
> IsIrreducible(X[3]); IsIrreducible(X[3]*X[3]);
> IsFaithful(X[2]); IsFaithful(X[3]);
> IsLinear(X[2]); IsLinear(X[3]);
  -------result-------
> true
> false
> false
> true
> true
> false
```

*Remark* 7A.1. Let us add the following known but maybe not well-known fact which explains why MAGMA can check faithfulness efficiently from the character table: a finite dimensional complex representation $V$ of a finite group is faithful $\Leftrightarrow$ $\dim V$ appears only in the character entry of the unit. ◇

*Exercise* 7A.1. Run the following code

```
> for n in [2..14] do
> G:=SymmetricGroup(n);
> X:=CharacterTable(G);
> M:=[1..#X];
> for i in [1..#X] do
> if(IsFaithful(X[i]) eq true) then M[i]:=1; else M[i]:=0; end if;
> end for;
> M;
> end for;
```

and interpret the output. What happens for `SpecialLinearGroup(2,p)` ($p$ is a prime) instead of the symmetric group?

◇

Also, recall that characters are traces, so we can evaluate them:

```
> G.2 @ X[2]
  -------result-------
> -1
```

Numerical values associated to characters:

```
> Degree(X[3]);
> InnerProduct(X[3],X[3]*X[3]);
> Norm(X[3]*X[3]);
> Indicator(X[3]);
  -------result-------
> 2
> 1
> 3
> 1
```

We will come back to the ***Schur indicator*** `Indicator()` later. Otherwise, the ***norm*** is the inner product with itself, and the ***inner product*** `InnerProduct(x,y);` measures how often $x$ appears in $y$. `Degree` is the dimension.

With the inner product we can decompose characters into ***simple*** characters (simple means irreducible):

```
> G:=SymmetricGroup(3);
> X:=CharacterTable(G);
> y:=X[3]*X[3];
> M:=[1..#X];
> for i in [1..#X] do
> M[i]:=InnerProduct(y,X[i]);
> end for;
> M;
  -------result-------
> [ 1, 1, 1 ]
```

*Exercise* 7A.2. Consider following code:

```
> G:=SymmetricGroup(3);
> X:=CharacterTable(G);
> for n in [1..10] do
> y:=X[3]^(n);
> M:=[1..#X];
> for i in [1..#X] do
> M[i]:=InnerProduct(y,X[i]);
> end for;
> RealField(10)!(&+M)^(1/n);
> end for;
  -------result-------
> 1.000000000
> 1.732050808
> 1.709975947
> 1.821160287
> 1.838416287
> 1.871736643
> 1.886389086
> 1.901623404
> 1.911688613
> 1.920622757
```

What is the code doing? Modify the code so that it works for power between 90 to 100, and for any representation of `SymmetricGroup(5)`. What is the limit $n \to \infty$? ◇

Let us ***restrict and induce***:

```
> G:=SymmetricGroup(5);
> H:=sub< SymmetricGroup(5) | (1,2), (2,3), (3,4)>;
> X:=CharacterTable(G);
> Y:=CharacterTable(H);
> Induction(Y[3],G);
> Restriction(X[3],H);
  -------result-------
> ( 10, 0, 2, -2, 0, 0, 0 )
> ( 4, 0, -2, 1, 0 )
```

***Frobenius reciprocity*** (for the symmetric group) is then:

```
> G:=SymmetricGroup(5);
> H:=sub< SymmetricGroup(5) | (1,2), (2,3), (3,4)>;
> K:=sub< G | (1,2), (2,3)>;
> Y:=CharacterTable(H);
> Restriction(Induction(Y[3],G),H);
> Induction(Restriction(Y[3],K),H);
> Restriction(Induction(Y[3],G),H)-Induction(Restriction(Y[3],K),H)-Y[3];
  -------result-------
> ( 10, 2, 0, -2, 0 )
> ( 8, 0, 0, -1, 0 )
> ( 0, 0, 0, 0, 0 )
```

Restriction and induction can be setup in many ways:

```
> G<x,y>:=PermutationGroup<23|
```

```
> [2,1,4,3,5,6,8,7,10,9,11,12,14,13,16,15,17,18,20,19,22,21,23],
> [16,9,1,5,8,22,7,23,21,10,3,2,20,18,17,11,15,6,19,13,12,14,4]>;
> CompositionFactors(G);
> #G;
> Factorization(#G);
> H:=SylowSubgroup(G,2);
> #H;
  -------result-------
> G
> |  M23
> 1
> 10200960
> [ <2, 7>, <3, 2>, <5, 1>, <7, 1>, <11, 1>, <23, 1> ]
> 128
```

This is the ***Mathieu group*** $M_{23}$, and we took a 2-Sylow subgroup of it. We can now induce and restrict between $G$ and $H$, say the `PermutationCharacter(G,H)` (obtained from the right cosets action of $G$ on $H$):

```
> x:=PermutationCharacter(G,H);
> x;
> Restriction(x,H);
> Induction(Restriction(x,H),G);
  -------result-------
> ( 79695, 735, 0, 19, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0 )
> ( 79695, 735, 735, 735, 735, 735, 735, 735,
> 19, 19, 19, 19, 19, 19, 19, 19, 1 )
> ( 6351293025, 540225, 0, 361, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0 )
```

Finally, let us say again that MAGMA is really fast when it comes to characters. The following computes the character table of `SymmetricGroup(15)` from scratch.

```
> G:=sub< SymmetricGroup(16) | (1,2), (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15)>;
> #G;
> Factorial(15);
> time X:=CharacterTable(G);
  -------result-------
> 1307674368000
> 1307674368000
> Time: 4.090
```

*Remark* 7A.2. Fun fact, the symmetric group `SymmetricGroup(15)` can be generated by two elements: a ***simple transition***, say $(1,2)$, and the ***long cycle***, here $(1,\dots,15)$. ◇

*Exercise* 7A.3. What is the largest $n$ so that MAGMA's online calculator can output the character table of `SymmetricGroup(n)`? Compare the from scratch construction above with `time X:=CharacterTable(SymmetricGroup(n))`. ◇

Let us end with an instance of ***the law of small numbers***: on first sight the characters of the symmetric group seems to have small entries. But that is really only the case for small $n$; here is MAGMA code that computes the average character values of symmetric groups `SymmetricGroup(n)` up to `SymmetricGroup(15)`:

```
> for n in [1..15] do
> G:=SymmetricGroup(n);
> X:=CharacterTable(G);
```

```
> Av:=0;
> for i in [1..#X] do
> for j in [1..#X] do
> Av+:=Abs(X[i][j]);
> end for;
> end for;
> RealField(10)!(Av/(#X)^(2));
> end for;
```

```
-------result-------
> 1.000000000
> 1.000000000
> 1.000000000
> 1.080000000
> 1.244897959
> 1.528925620
> 2.084444444
> 2.950413223
> 4.768888889
> 8.195011338
> 15.95663265
> 31.45977399
> 70.97568866
> 161.9799726
> 402.7822831
```

*Exercise* 7A.4. Write MAGMA code that gives the average degree of characters of symmetric groups. Compare to the average character values given above. ◇

7B. **Working with representations.** One can work with ***representations*** instead of characters. The advantage is that this works for any finite dimensional algebra, but at the cost of a slower performance. Let us look at an example to get started:

```
> G:=SymmetricGroup(7);
> X:=CharacterTable(G);
> y:=PermutationCharacter(G);
> y;
> M:=[1..#X];
> for i in [1..#X] do
> M[i]:=InnerProduct(y^5,X[i]);
> end for;
> time M;
```

```
-------result-------
> ( 7, 5, 1, 3, 4, 1, 3, 1, 2, 0, 2, 1, 0, 0, 0 )
> [ 52, 0, 1, 151, 160, 4, 5, 74, 160, 15, 75, 41, 30, 45, 150 ]
> Time: 0.000
```

```
> G:=SymmetricGroup(7);
> M:=PermutationModule(G,RationalField());
> M;
> time IndecomposableSummands(TensorPower(M,5));
```

```
-------result-------
```

**The computation exceeded the memory limit and so was terminated prematurely.**

```
GModule M of dimension 7 over Rational Field

Current total memory usage: 76.1MB, failed memory request: 1077.6MB
System Error: User memory limit has been reached
```

For the symmetric group, `PermutationModule` and `PermutationCharacter` are the representation respectively character obtained by the action of `SymmetricGroup(n)` on a vector space (using complex numbers for the character and a field that we specify for the representation) of dimension $n$ by permutation of a fixed basis.

*Remark* 7B.1. Note that MAGMA actually works with ***modules***, i.e. vector spaces with actions, and not with ***representations***, i.e. the homomorphisms defining the action. We will see that in more details below. In any case, we abuse language and say representations. ◇

*Exercise* 7B.1. Compare the third tensor powers, via characters and representations, and their timing. ◇

Ok, computational aspects aside, we can now setup nonsemisimple representations. We start with `PGL(3,4)`, the ***projective linear group*** $PGL(\mathbb{F}_4^3) = GL(\mathbb{F}_4^3)/\text{center}$ over the Galois field `GF(4)` with four elements and look at representations over `GF(3)`:

```
> G:=PGL(3,4);
> M:=PermutationModule(G,GF(3));
> M; IsSemisimple(M);
-------result-------
> GModule M of dimension 21 over GF(3)
> false
```

Here we get a 21 dimensional representation using `PermutationModule` because `PGL(3,4)` is realized in MAGMA within `SymmetricGroup(21)`:

```
> PGL(3,4);
-------result-------
> Permutation group acting on a set of cardinality 21
> Order = 60480 = 2^6 * 3^3 * 5 * 7
> (4, 14, 21)(5, 15, 17)(7, 10, 20)(8, 9, 12)(11, 13, 19)
> (1, 8, 21, 16, 15, 3, 2)(4, 10, 20, 18, 17, 9, 7)
> (5, 12, 11, 14, 19, 13, 6)
```

To move on, we note that this is indeed not semisimple, so the composition factors are different from the direct summands:

```
> CompositionSeries(M);
> CompositionFactors(M);
> IsIrreducible(M);
> IndecomposableSummands(M);
-------result-------
> [
> GModule of dimension 1 over GF(3),
> GModule of dimension 20 over GF(3),
> GModule M of dimension 21 over GF(3)
```

```
> ]
> [
> GModule of dimension 1 over GF(3),
> GModule of dimension 19 over GF(3),
> GModule of dimension 1 over GF(3)
> false
> ]
> [
> GModule M of dimension 21 over GF(3)
> ]
```

Let us test whether the composition factors are the same or not:

```
> X:=CompositionFactors(M); X;
> IsIsomorphic(X[1],X[3]);
> ConstituentsWithMultiplicities(M);
```

```
-------result-------
```

```
> true
> [
> <GModule of dimension 1 over GF(3), 2>,
> <GModule of dimension 19 over GF(3), 1>
> ]
> [ 1, 2, 1 ]
```

Many other tricks can be played as soon as a representation is setup, e.g.:

```
> IndecomposableSummands(TensorPower(M,2));
> IndecomposableSummands(ExteriorPower(M,2));
> IndecomposableSummands(SymmetricPower(M,2));
```

```
-------result-------
```

```
> [
> GModule of dimension 21 over GF(3),
> GModule of dimension 21 over GF(3),
> GModule of dimension 84 over GF(3),
> GModule of dimension 126 over GF(3),
> GModule of dimension 189 over GF(3)
> ]
> [
> GModule of dimension 84 over GF(3),
> GModule of dimension 126 over GF(3)
> ]
> [
> GModule of dimension 21 over GF(3),
> GModule of dimension 21 over GF(3),
> GModule of dimension 189 over GF(3)
> ]
```

Or submodules can be constructed:

```
> G:=PGL(3,4);
> M:=PermutationModule(G,GF(3));
> N:=JacobsonRadical(M);
> I:=IndecomposableSummands(TensorProduct(M,N));
> I; IsIsomorphic(I[1],M);
```

```
-------result-------
```

```
> [
> GModule of dimension 21 over GF(3),
> GModule of dimension 84 over GF(3),
> GModule of dimension 126 over GF(3),
> GModule of dimension 189 over GF(3)
> ]
> true
```

So, since the usual algebra can just be run in MAGMA with only the memory being a problem, the crucial question remains how to construct representations to begin with.

For groups one can construct all simple representation and all projective indecomposables over any finite field in one go, say for the 3-by-3 general linear group with coefficients in `GF(2)`:

```
> G:=PGL(3,2);
> X:=IrreducibleModules(G,GF(2));
> Y:=IrreducibleModules(G,GF(3));
> X; Y;
  -------result-------
> [
> GModule of dimension 1 over GF(2),
> GModule of dimension 3 over GF(2),
> GModule of dimension 3 over GF(2),
> GModule of dimension 8 over GF(2)
> ]
> [
> GModule of dimension 1 over GF(3),
> GModule of dimension 6 over GF(3),
> GModule of dimension 6 over GF(3),
> GModule of dimension 7 over GF(3)
> ]
```

```
> A:=ProjectiveIndecomposableModules(G,GF(2));
> B:=ProjectiveIndecomposableModules(G,GF(3));
> A; B;
  -------result-------
> [
> GModule of dimension 8 over GF(2),
> GModule of dimension 16 over GF(2),
> GModule of dimension 16 over GF(2),
> GModule of dimension 8 over GF(2)
> ]
> [
> GModule of dimension 9 over GF(3),
> GModule of dimension 6 over GF(3),
> GModule of dimension 6 over GF(3),
> GModule of dimension 15 over GF(3)
> ]
```

***Projective covers*** can also be constructed easily:

```
> ProjectiveCover(X[1]);
  -------result-------
> GModule of dimension 8 over GF(2)
>
```

```
> [1]
> [1]
> [1]
> [1]
> [1]
> [1]
> [1]
> [1]
```

Similarly, one can use `InjectiveHull()` to construct ***injective hulls***.

*Remark* 7B.2. Note that MAGMA has a bias and likes projective things over injective things. However, for a group this does not make a difference since the projective covers and injective hulls coincide. ◇

Let us look at a slightly bigger example. We consider the cyclic group `CyclicGroup(5)`, this is $\mathbb{Z}/5\mathbb{Z}$, and its representations over `GF(5)`. This group has five indecomposables (called $Z$ below) and only one simple (called $L$ below) representation. The indecomposables are the representations where the generator of $\mathbb{Z}/5\mathbb{Z}$ acts by the following ***Jordan blocks***:

$$
\begin{aligned}
Z_1 = L_1 &: \begin{pmatrix} 1 \end{pmatrix}, \text{is simple} \\
Z_2 &: \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}, \text{filtration } 0 - L_1 - Z_2, \\
Z_3 &: \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}, \text{filtration } 0 - L_1 - L_1 - Z_3, \\
Z_4 &: \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}, \text{filtration } 0 - L_1 - L_1 - L_1 - Z_4, \\
Z_5 = P_1 &: \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}, \text{filtration } 0 - L_1 - L_1 - L_1 - L_1 - Z_5,
\end{aligned}
$$

We construct the three dimensional $\mathbb{Z}/5\mathbb{Z}$-representation $Z3 = Z_3$ that is indecomposable but not simple:

```
G:=CyclicGroup(5);
Z3:=CompositionSeries(ProjectiveIndecomposables(G,GF(5))[1])[3];
IsIrreducible(Z3);
IndecomposableSummands(Z3);
```

```
-------result-------
> false GModule of dimension 1 over GF(5)
> GModule of dimension 2 over GF(5)
> [
> GModule X of dimension 3 over GF(5)
> ]
```

There are additionally $Z1$ and $Z5$ of dimensions one and five, respectively, which are the trivial $\mathbb{Z}/5\mathbb{Z}$-representation and the regular $\mathbb{Z}/5\mathbb{Z}$-representation. We have $Z3 \otimes Z3 \cong Z1 \oplus Z3 \oplus Z5$, with only $Z5$ being projective:

```
> P:=IndecomposableSummands(TensorPower(Z3,2));
> P;
> IsProjective(P[1]);
> IsProjective(P[2]);
> IsProjective(P[3]);
```

```
-------result-------
```

```
> [
> GModule of dimension 1 over GF(5),
> GModule of dimension 3 over GF(5),
> GModule of dimension 5 over GF(5)
> ]
> false
> false
> true
```

*Exercise* 7B.2. Check with MAGMA that $Z3 \otimes Z5 \cong Z5^{\oplus 3}$. ◇

Since $Z1$ is the trivial $\mathbb{Z}/5\mathbb{Z}$-representation, if we annihilate $Z5$, then we have that $Z3^{\otimes 2}$ satisfies $X^2 = 1 + X$ whose roots are the ***golden ratio*** and its Galois conjugate. Thus, tensor powers of $Z3$ give the ***Fibonacci sequence***:

```
> G:=CyclicGroup(5);
> Z3:=CompositionSeries(ProjectiveIndecomposables(G,GF(5))[1])[3];
> Y:=[Z3];
> n:=10;
> 1;
> for i in [2..n] do
> X:=[];
> for y in Y do
> P:=IndecomposableSummands(TensorProduct(Z3,y));
> for p in P do
> if(Dimension(p) ne 5) then
> X:=Append(X,p);
> end if;
> end for;
> Y:=X;
> end for;
> #X;
> end for;
  -------result-------
> 1
> 2
> 3
> 5
> 8
> 13
> 21
> 34
> 55
> 89
```

*Exercise* 7B.3. Confirm in MAGMA that the $n$th root of the sequence above gives the golden ratio. What happens if one includes $Z5$ and takes the $n$th root?

(Careful: the convergence is slow, and you might not be able to 100% verify what the limit is in the online calculator.) ◇

In general, one can construct representations by specifying the ***acting matrices***. This is in particularly useful when one has a generator-relation presentations. For example:

```
> G<x,y>:=PermutationGroup<11|[1,10,3,11,7,6,5,9,8,2,4],
> [4,5,8,3,6,9,7,1,2,10,11]>;
```

```
> F:=GF(3);
> x:=Matrix(F, 5, 5,
> [[0,2,0,1,1],[2,1,1,0,2],[1,1,1,2,2],
> [0,2,2,2,2],[0,2,2,1,0]]);
> y:=Matrix(F, 5, 5,
> [[0,1,1,0,1],[2,0,0,1,0],[0,0,1,2,2],
> [2,1,0,0,0],[0,1,2,2,1]]);
> M:=GModule(G,[x,y]);
> M;
  -------result-------
> GModule M2 of dimension 5 over GF(3)
```

*Exercise* 7B.4. Check what happens if there would be a typo in the action matrices. ◇

We can run the same syntax as above, for example:

```
> IndecomposableSummands(TensorProduct(M,N));
  -------result-------
> [
> GModule of dimension 10 over GF(3),
> GModule of dimension 15 over GF(3)
> ]
```

For a general finite dimensional algebra, let us setup an algebra given by matrices:

```
> K:=GF(2);
> A := MatrixAlgebra<K, 4 |
> [1,0,0,0, 0,1,0,0, 0,0,1,0, 0,0,0,1],
> [0,0,0,0, 1,0,0,0, 0,0,0,0, 0,0,1,0],
> [0,0,0,0, 0,0,0,0, 1,0,0,0, 0,1,0,0] >;
> Dimension(A);
  -------result-------
> 4;
```

The algebra $A$ above is $A = \mathbb{F}_2[X,Y]/(X^2, Y^2)$, which is the group ring of the ***Klein four group*** $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z}$ in characteristic two. We can now tell MAGMA to work with its ***regular representation***:

```
> V:=RModule(K,4);
> m:=map< CartesianProduct(V, A) -> V | t :-> t[1]*t[2] >;
> M:=Module(A,m);
> IndecomposableSummands(M);
  -------result-------
> [
> Right A-module of dimension 4,
> where A is Matrix Algebra of degree 4 with 3
> generators over GF(2)
> ]
```

*Remark* 7B.3. With constructions of the form

```
> V:=RModule(K,4);
> m:=map< CartesianProduct(V, A) -> V | t :-> t[1]*t[2] >;
> M:=Module(A,m);
> IndecomposableSummands(M);
```

one can construct any representation as long as one has a good control over the action map encoded by $m$ in this example. ◇

```
> CompositionFactors(M);
  -------result-------
> [
> RModule of dimension 1 over GF(2),
> RModule of dimension 1 over GF(2),
> RModule of dimension 1 over GF(2),
> RModule of dimension 1 over GF(2)
> ]
```

*Remark* 7B.4. Sometimes one gets bugs. The code

```
> K:=GF(2);
> A := MatrixAlgebra<K, 4 |
> [1,0,0,0, 0,1,0,0, 0,0,1,0, 0,0,0,1],
> [0,0,0,0, 1,0,0,0, 0,0,0,0, 0,0,1,0],
> [0,0,0,0, 0,0,0,0, 1,0,0,0, 0,1,0,0] >;
> V:=RModule(K,4);
> m:=map< CartesianProduct(V, A) -> V | t :-> t[1]*t[2] >;
> M:=Module(A,m);
> CompositionSeries(M);
```

which is not much different from the code above, produced:

```
[
    A-module of dimension 1, where A is

Magma: Internal error

Please mail this entire run [*** WITH THE DETAILS BELOW ***]
    to magma-bugs@maths.usyd.edu.au

You can print the entire input by entering:
%P

Version: 2.28-2
Initial seed: 553935780
Time to this point: 0.01
Memory usage: 32.09MB
Segmentation fault
```

For a different way to setup `CompositionSeries` for the Klein four group see below. ◇

Here are a few construction to build new representations:

```
> N:=DirectSum([M,M]);
> ActionMatrix(N,A.2);
> W:= ModuleWithBasis([ M.1+M.2+M.3, M.2+M.3, M.3 ]);
> ActionMatrix(W,A.2);
> IndecomposableSummands(W);
  -------result-------
> [0 0 0 0 0 0 0 0]
> [1 0 0 0 0 0 0 0]
> [0 0 0 0 0 0 0 0]
> [0 0 1 0 0 0 0 0]
> [0 0 0 0 0 0 0 0]
> [0 0 0 0 1 0 0 0]
> [0 0 0 0 0 0 0 0]
```

```
> [0 0 0 0 0 0 1 0]
> [1 1 0]
> [1 1 0]
> [0 0 0]
> [
> Right A-module of dimension 3, where A is
> Matrix Algebra of degree 4 and
> dimension 4 with 3 generators over GF(2)
> ]
```

Let us setup another example. Again, take the ***Klein four group*** $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z}$ but setup differently:

```
> G:=SmallGroup(4,2);
> IsCyclic(G);
  -------result-------
> false
```

Since there are only two groups of order four, and the one we took is not cyclic, it must be the Klein four group. Let us pick an indecomposable three dimensional representation over `GF(2)`:

```
> X:=CompositionSeries(ProjectiveCover(IrreducibleModules(G,GF(2))[1]))[3];
> IndecomposableSummands(X);
  -------result-------
> [
> GModule X of dimension 3 over GF(2)
> ]
```

Now we take tensor products:

```
> for i in [1..5] do
> IndecomposableSummands(TensorPower(X,i))
> [#IndecomposableSummands(TensorPower(X,i))];
> end for;
  -------result-------
> GModule X of dimension 3 over GF(2)
> GModule of dimension 5 over GF(2)
> GModule of dimension 7 over GF(2)
> GModule of dimension 9 over GF(2)
> GModule of dimension 11 over GF(2)
```

In each step we get a new indecomposable of the Klein four group! (The dimension verifies that these are really new.) One can in fact check that `SmallGroup(4,2)` has infinitely many indecomposables over `GF(2)`.

To extract the acting matrices, say for the representation of dimension five, we can run the following:

```
> M:=IndecomposableSummands(TensorPower(X,2))
[#IndecomposableSummands(TensorPower(X,2))];
> phi := Representation(M);
> phi(G.1)+phi(G.2);
  -------result-------
> [0 0 1 0 0]
> [0 0 1 0 0]
> [0 0 0 0 0]
> [0 0 0 0 0]
```

```
> [1 1 0 0 0]
```

This outputs the sum of the action matrices of $(1,0)$ and $(0,1)$ in $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z}$. Note that the decomposition algorithm underlying `IndecomposableSummands` is randomized and we (can) get different basis, hence action matrices, in every evaluation.

*Exercise* 7B.5. Compare the constructions of representations of the Klein four group via `MatrixAlgebra` and `SmallGroup(4,2)` above. What are the differences? ◇

Finally, MAGMA is really awesome when working with ***basic algebras*** (quivers if you want). Here is one example: we setup a "Temperley–Lieb algebra" and compute the dimensions of the projective representations and their composition factors:

```
> K:=GF(2);
> G:=SL(2,K);
> X:=IrreducibleModules(G,K);
> V:=TensorPower(X[2],4);
> A:=EndomorphismAlgebra(V);
> B:=BasicAlgebra(A);
> DimensionsOfProjectiveModules(B);
> DimensionsOfProjectiveModules(B);
> CompositionFactors(ProjectiveModule(B,1));
> CompositionFactors(ProjectiveModule(B,2));
  -------result-------
> [ 1, 2 ]
> [
> AModule of dimension 1 over GF(2)
> ]
> [
> AModule of dimension 1 over GF(2),
> AModule of dimension 1 over GF(2)
> ]
```

7C. **Representation theory over different fields.** Let us have a look at the character table of $\mathbb{Z}/5\mathbb{Z}$:

```
> X:=CharacterTable(CyclicGroup(5)); X;
  -------result-------
> Character Table
> ---------------
>
>
> -------------------------------
> Class |   1    2    3    4    5
> Size  |   1    1    1    1    1
> Order |   1    5    5    5    5
> -------------------------------
> p  =  5   1    1    1    1    1
> -------------------------------
> X.1   +   1    1    1    1    1
> X.2   0   1   Z1 Z1#2 Z1#3 Z1#4
> X.3   0   1 Z1#4 Z1#3 Z1#2   Z1
> X.4   0   1 Z1#3   Z1 Z1#4 Z1#2
> X.5   0   1 Z1#2 Z1#4   Z1 Z1#3
>
```

```
>
> Explanation of Character Value Symbols
> -------------------------------------
>
> # denotes algebraic conjugation, that is,
> #k indicates replacing the root of unity w by w^k
>
> Z1     = (CyclotomicField(5: Sparse := true)) ! [
> RationalField() | 0, 0, 0, 1 ]
```

One immediately hits the problem of working with roots of unity. Thus, working over $\mathbb{R}$ instead of $\mathbb{C}$ is not completely straightforward. However, the ***Schur indicator*** (or ***Frobenius–Schur indicator***) comes to our rescue. This is the entry $+$, sometimes written 1, and 0 in the table (it could also be $-$ or $-a$, but not in this example), and we can check this as follows:

```
> X:=CharacterTable(CyclicGroup(5));
> for i in [1..#X] do Indicator(X[i]); end for;
  -------result-------
> 1
> 0
> 0
> 0
> 0
```

The indicator gives us all the information we need to work over $\mathbb{R}$:

**(a)** If its 1, then there is nothing to do and the complex simple character is also a real simple character.

**(b)** If its 0, then we need to add two complex simple characters together (one plus its conjugate) to get a real simple character.

**(c)** The case $-1$ is the exercise below.

For example, to get the simple real characters in this case we do:

```
> X:=CharacterTable(CyclicGroup(5));
> IsReal(X[1]);
> IsReal(X[2]); IsReal(X[2]+X[3]);
> IsReal(X[4]); IsReal(X[4]+X[5]);
  -------result-------
> true
> false
> true
> false
> true
```

*Exercise* 7C.1. Use MAGMA to find a group with Schur indicator $-1$. What happens for the representations when you use `IsReal`? ◇

The second thing that comes to mind is character theory over the algebraic closure of the Galois fields. That is easy in MAGMA can controlled by `BrauerCharacterTable(G,p)` with $p$ being the characteristic, e.g.:

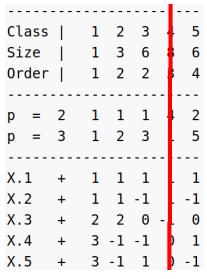
```
> G:=SymmetricGroup(4);
> X:=BrauerCharacterTable(G,3);
> X;
> X[3];
> CharacterTable(G);
```

```
  -------result-------
> [
> ( 1, 1, 1, 1 ),
> ( 1, 1, -1, -1 ),
> ( 3, -1, -1, 1 ),
> ( 3, -1, 1, -1 )
> ]
> ( 3, -1, -1, 1 )
>
> Character Table of Group G
> --------------------------
>
>
> ----------------------
> Class |   1  2  3  4  5
> Size  |   1  3  6  8  6
> Order |   1  2  2  3  4
> ----------------------
> p  =  2   1  1  1  4  2
> p  =  3   1  2  3  1  5
> ----------------------
> X.1   +   1  1  1  1  1
> X.2   +   1  1 -1  1 -1
> X.3   +   2  2  0 -1  0
> X.4   +   3 -1 -1  0  1
> X.5   +   3 -1  1  0 -1
```

The Brauer character table is a square matrix of size being the $p$-irregular conjugacy classes. The the example above the 3-irregular conjugacy classes are $1, 2, 3, 5$, while the 3-irregular conjugacy classes are $1, 4, 5$, as indicated by the orders coprime to $p$ in this row:

```
> Order |   1  2  2  3  4
```

*Remark* 7C.1. For soluble groups the Brauer characters can be deduced from the complex character table, and this is what MAGMA does. Without going into details, if one deletes the column four for $p = 3$ one gets:



Now $X.3 = X.1 + X.2$ and that is why this is dropped from the Brauer character table.

For non-soluble groups things are trickier and MAGMA constructs the simple representations. ◇

Essentially the same syntax as for characters works, so we skip that part. But there are also new interesting feature, such as `Blocks` (giving as output the blocks and their ***defect***) and `DefectGroup` (returning the ***defect group*** of a block):

```
> G:=SymmetricGroup(4);
> Y:=CharacterTable(G);
> Blocks(Y,3);
```

```
> DefectGroup(Y,Blocks(Y,3)[1],3);
  -------result-------
> [
> { 1, 2, 3 },
> { 4 },
> { 5 }
> ]
> [ 1, 0, 0 ]
> Permutation group acting on a set of cardinality 4
> Order = 3
> (1, 3, 2)
```

These calculations are efficient and fast, for example:

```
> G:=SymmetricGroup(12);
> Y:=CharacterTable(G);
> time IsAbelian(DefectGroup(Y,Blocks(Y,3)[1],3));
  -------result-------
> false
> Time: 0.020
```

Another option for characters are ***rational characters***, e.g.:

```
> G:=AlternatingGroup(4);
> CharacterTable(G);
> RationalCharacterTable(G);
  -------result-------
> Character Table of Group G
> --------------------------
>
>
> ------------------------
> Class |   1  2    3    4
> Size  |   1  3    4    4
> Order |   1  2    3    3
> ------------------------
> p  =  2   1  1    4    3
> p  =  3   1  2    1    1
> ------------------------
> X.1   +   1  1    1    1
> X.2   0   1  1    J -1-J
> X.3   0   1  1 -1-J    J
> X.4   +   3 -1    0    0
>
>
> Explanation of Character Value Symbols
> --------------------------------------
>
> J = RootOfUnity(3)
>
> [
> ( 1, 1, 1, 1 ),
> ( 2, 2, -1, -1 ),
> ( 3, -1, 0, 0 )
```

```
> ]
> [
> [ 1 ],
> [ 2, 3 ],
> [ 4 ]
> ]
```

In this example the characters two and three are not rational, but their direct sum is.

*Remark* 7C.2. In general, the rational characters, are the sums of the Galois orbits on the complex character table. Hence, the story over $\mathbb{Q}$ is similar, but more complicated, than over $\mathbb{R}$. ◇

Working with algebraically closed fields is not easy on a machine. The characters avoid that problem by using a combinatorial approach. For representations we cannot cheat so let us have a look at different fields, starting with algebraically closed ones.

One can construct, e.g., $\overline{\mathbb{Q}}$ or $\overline{\mathbb{F}_p}$, but a lot of operations will not work. For example:

```
> AlgebraicClosure(GF(2));
> AlgebraicClosure(RationalField());
> SpecialLinearGroup(2,AlgebraicClosure(GF(2)));
  -------result-------
> Algebraically closed field with no variables over GF(2)
> Algebraically closed field with no variables over Rational Field
>
> >> SpecialLinearGroup(2,AlgebraicClosure(GF(2)));
> ^
> Runtime error in 'SpecialLinearGroup': Cannot compute generators for matrix
> group
```

In finite characteristic it is better to simulate $\overline{\mathbb{F}_p}$ by making the underlying field big enough. This can be done afterwards, for example by ***base change***:

```
> G:=SpecialLinearGroup(2,4);
> X:=IrreducibleModules(G,GF(2));
> Y:=TensorPower(X[2],2);
> for k in [1..6] do
> #IndecomposableSummands(ChangeRing(Y,GF(2^k)));
> end for;
  -------result-------
> 3
> 4
> 3
> 4
> 3
> 4
```

In characteristic zero one can use ***cyclotomic fields*** as approximations. For example:

```
> K:=RationalField();
> F:=CyclotomicField(3);
> G:=AlternatingGroup(4);
> A:=GroupAlgebra(K,G);
> V:=RegularRepresentation(A);
> X, _:=IndecomposableSummands(V);
> Y, _:=IndecomposableSummands(ChangeRing(V,F));
> X; Y;
```

```
-------result-------
> [
> Matrix Algebra [ideal of V] of degree 12 and dimension 1 over
> Rational Field,
> Matrix Algebra [ideal of V] of degree 12 and dimension 9 over
> Rational Field,
> Matrix Algebra [ideal of V] of degree 12 and dimension 2 over
> Rational Field
> ]
> [
> Matrix Algebra [ideal] of degree 12 and dimension 1 over F,
> Matrix Algebra [ideal] of degree 12 and dimension 9 over F,
> Matrix Algebra [ideal] of degree 12 and dimension 1 over F,
> Matrix Algebra [ideal] of degree 12 and dimension 1 over F
> ]
```

Compare this to the rational and complex character tables of `AlternatingGroup(4)` above.

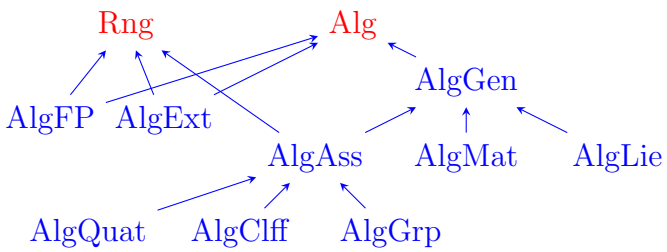
*Exercise* 7C.2. The above code gives the decomposition into bimodules, and that is why the $9 = 3 \cdot 3$ is appearing. Ask MAGMA to give the decomposition into representations. For example, you could setup `AlternatingGroup(4)` as a twelve dimensional matrix algebra, and define its regular representation by acting on a twelve dimensional vector space. ◇

## 8. Lecture 5 – Noncommutative algebras

We will now explore how MAGMA can handle (finite dimensional) algebras.

8A. **Types of algebras.** There are many types of algebras in MAGMA. For example:

```
> M := MatrixAlgebra(GF(5),4); Type(M), IsAssociative(M);
-------result-------
> AlgMat true
```




An example of what these are is copied from the MAGMA Handbook:

In MAGMA a ***finitely-presented algebra*** (FPA) is a quotient of a free associative algebra by an ideal of relations.

To compute with these ideals of relations, one constructs noncommutative Gröbner bases, which have many parallels with commutative Gröbner bases.

At the heart of the theory is a noncommutative version of the Buchberger algorithm which computes a Gröbner basis of an ideal of an algebra starting from an arbitrary basis (generating set) of the ideal.

One significant difference with the commutative case is that a noncommutative Gröbner basis may not be finite for a finitely-generated ideal.

```
> ISA(AlgMat,AlgAss), ISA(AlgClff,AlgAss), ISA(AlgAss,Rng);
-------result-------
> false true true
```

Other types: Heck algebras, universal enveloping algebras (this is `AlgUE`), quantized universal enveloping algebras (this is `AlgQUE`) and many more.

We will now explore some of these types of algebras.

8B. **Clifford algebras.** Let $V$ be a finite dimensional vector space over a field $\mathbb{F}$ and let $Q : V \to \mathbb{F}$ be a quadratic form with ***polar form*** $\beta$; i.e., $\beta(u, v) = Q(u + v) - Q(u) - Q(v)$.

The ***Clifford algebra*** of $Q$ is an $\mathbb{F}$-algebra $C$ with identity $\mathbf{1}$ and a linear map $f : V \to C$ such that

$$f(v)^2 = Q(v)\mathbf{1} \quad \text{for all } v \in V.$$

Then $f(u)f(v) + f(v)f(u) = \beta(u, v)\mathbf{1}$.

The dimension of a Clifford algebra is $2^n$, where $n = \dim V$. For example:

```
> Q := StandardQuadraticForm(4,GF(11));
> C, V, f := CliffordAlgebra(Q);
> Dimension(C), One(C);
```

```
-------result-------
```

```
> 16 ( 1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0)
```

An ***exterior algebra*** is the Clifford of a quadratic form that is identically zero. MAGMA returns a ***structure constant algebra*** of type `AlgClff`.

However, MAGMA's intrinsic `ExteriorAlgebra` returns a quotient of a free algebra and even though `AlgExt` does not inherit from `AlgFP` most of the operations applicable to ***finitely presented algebras*** can be used. Thus the Gröbner basis machinery applies to algebras of type `AlgExt`.

```
> E<w,x,y,z> := ExteriorAlgebra(GF(11),4);
> I := ideal< E | w*x + y*z >;
> B := quo< E | I>; B;
```

```
-------result-------
```

```
> Affine Algebra of rank 4 over GF(11)
> Graded Reverse Lexicographical (exterior algebra) Order
> Variables: w, x, y, z
> Quotient relations:
> [
> w*x + y*z
> ]
```

To construct a homomorphism from an exterior algebra of type `AlgExt` to another algebra we only need to supply the images of the basis elements.

```
> C := CliffordAlgebra(ZeroMatrix(GF(11),4,4));
> E<w,x,y,z> := ExteriorAlgebra(GF(11),4);
```

The vector space $V$ and the embedding $f : V \to C$ can be obtained as attributes of $C$; namely `C‘space` and `C‘embedding` (the quotation marks do not come out nicely in the font – they should be backwards, so be careful when copying the code).

```
> h := hom< E -> C | [C‘embedding(v) : v in Basis(C‘space)] >;
```

The constructor `hom` returns a linear map but MAGMA makes no attempt to check whether it preserves multiplication. But we can check directly.

```
> forall{ <s,t> : s, t in [w,x,y,z] | h(s*t) eq h(s)*h(t) };
```

```
-------result-------
```

```
> true
```

Here are two Clifford algebras of forms in dimension 2:

```
> I := IdentityMatrix(Rationals(),2);
> C1<e1,e2>, V1, f1 := CliffordAlgebra(I);
> C2<i,j>, V2, f2 := CliffordAlgebra(-I);
> J := Matrix(Rationals(),[[1,0],[0,-1]]);
> C3<u1,u2>, V3, f3 := CliffordAlgebra(J);
```

C1 is the algebra of $2 \times 2$ ***matrices*** over the rationals. C2 is the algebra of ***quaternions*** with rational coefficients.

```
> U := [e1,e1*e2];
> Matrix(2,2,[ (U[s]*U[t] + U[t]*U[s])[1]/2 : s,t in [1,2] ]);
{\result
[ 1  0]
[ 0 -1]
}
> phi := hom< C3 -> C1 | [One(C1),e1,e1*e2,e2 ] >;
> forall{<s,t> : s,t in Basis(C3) | phi(s*t) eq phi(s)*phi(t) };
\result
true
```

```
-------result-------
> [ 1  0]
> [ 0 -1]
```

```
> phi := hom< C3 -> C1 | [One(C1),e1,e1*e2,e2 ] >;
> forall{<s,t> : s,t in Basis(C3) | phi(s*t) eq phi(s)*phi(t) };
```

```
-------result-------
> true
```

Therefore C3 is isomorphic to C1.

8C. **Finitely presented and Clifford algebras.** The ***free associative algebra*** of rank $n$ over a field $K$ is the set of $K$-linear combinations of noncommutative polynomials in $n$ indeterminates. This is the ***tensor algebra*** of the vector space $K^n$.

A ***finitely presented algebra*** is the quotient of a free algebra by an ideal.

The Clifford algebra C1 of the previous slide can be constructed as a finitely presented algebra.

```
> F<x1,x2> := FreeAlgebra(Rationals(),2);
> I := ideal< F | x1^2 - 1, x2^2 - 1, x1*x2 + x2*x1 >;
> C<e1,e2> := quo< F | I >;
> Rank(C), Dimension(C), Type(C);
```

```
-------result-------
> 2 4 AlgFP
```

```
> LeadingTerm(f);
```

```
-------result-------
> -4*e2*e1
```

Suppose that $f : V \to C$ is the Clifford algebra of a quadratic form $Q$.

If $a \in f(V)$ is invertible, the map $f(V) \to f(V) : b \mapsto -a^{-1}ba$ is the reflection in the hyperplane orthogonal to $a$.

The reflections generate the orthogonal group of $Q$.

```
> F<z> := GF(9);
> Q := StandardQuadraticForm(4,F);
> C,V,f := CliffordAlgebra(Q);
> I := IsometryGroup(V);
```

```
> a := f(z*V.2 + V.3);
> M := -Matrix([(a^-1*f(b)*a) @@ f : b in Basis(V)]);
> M in I, IsReflection(M);
  -------result-------
> true
> true (  0   z   1   0)   (  0 z^7   1   0)
```

The ***Clifford group*** of the Clifford algebra $f : V \to C$ is

$$\Gamma = \{\, s : s \in C \mid s \text{ is invertible and } s^{-1} f(v) s \in f(V) \text{ for all } v \in V \,\}.$$

The map $\chi : \Gamma \to \mathrm{GL}(V)$ such that $f(v\chi(s)) = s^{-1} f(v) s$ is the ***vector representation*** of $\Gamma$. If $s \in \Gamma \cap f(V)$, then $-\chi(s)$ is a reflection.

If $\dim V$ is even, the image of $\chi$ is an orthogonal group (the isometry group of the quadratic space $V$).

```
> H := sub<GL(4,F)|
>    [VectorAction(f(g)) : g in V | QuadraticNorm(g) eq 0] >;
> H eq I;
  -------result-------
> true
```

*Exercise* 8C.1. Suppose that $a, b \in f(V)$ and $a$ is invertible. Show that $a^{-1}ba \in f(V)$. ◇

*Exercise* 8C.2. Find the image of $\chi$ when $\dim V$ is odd. ◇

Let $C_+$ (resp. $C_-$) be the subspace spanned by products of an even (resp. odd) number of basis elements. Then $C_+$ is a subalgebra and $C = C_+ \oplus C_-$.

The ***main involution*** of $C$ is the linear map $J : C \to C$ such that $J(u) = u$ for $u \in C_+$ and $J(u) = -u$ for $u \in C_-$. It is an automorphism.

```
> J := MainInvolution(C);
> Cplus := EvenSubalgebra(C);
> forall\{<u,v> : u,v in Basis(V) |  J(f(u)*f(v)) eq J(f(u))*J(f(v))\};
```

*Exercise* 8C.3. Suppose that $f : V \to C$ is a Clifford algebra over $\mathbb{F}$. Write a MAGMA function `derivation(C,lambda)` that takes a linear functional $\lambda : V \to \mathbb{F}$ and returns a derivation $d : C \to C$ such that $d(f(v)) = \lambda(v)\mathbf{1}$ and $d(xy) = d(x)y + J(x)d(y)$ for all $x, y \in C$. ◇

Let us have a few more exercises.

*Exercise* 8C.4. Consider the following code:

```
> F := RationalField();
> Q := DiagonalMatrix(F,[1,-2,-5]);
> C,V,f := CliffordAlgebra(Q);
> E, h := EvenSubalgebra(C);
```

Show that $E$ is a generalized quaternion algebra. ◇

*Exercise* 8C.5. Consider the following code:

```
> Q := StandardQuadraticForm(4,GF(25));
> C := CliffordAlgebra(Q);
> E := EvenSubalgebra(C);
```

Show that `E` is not simple. Find orthogonal central idempotents that generate its ideals. (Hint. Check out `DirectSumDecomposition`.) ◇

*Exercise* 8C.6. Consider the following code:

```
> F<w> := GF(25);
> Q := StandardQuadraticForm(5,F);
> C := CliffordAlgebra(w*Q);
```

Show that `C` is the algebra of $4 \times 4$ matrices over the field $F_{625}$. ◇

8D. **Spin groups.** The mapping that reverses the multiplication is the ***main antiautomorphism*** of $C$; its square is the identity.

The ***special Clifford group*** is $\Gamma^+ = \Gamma \cap C_+$.

The ***spin group*** is $\mathrm{Spin}(V,Q) = \{\, s \in \Gamma^+ \mid \alpha(s)s = 1 \,\}$, where $\alpha$ is the main antiautomorphism of $C$.

Suppose that $s = f(u)$ and $t = f(v)$ where $u, v \in V$ are orthogonal and $Q(u) = 0$. Then $st - 1 \in \mathrm{Spin}(V,Q)$ and $\chi(uv - 1)$ is a Siegel transformation.

```
> s := f(V.1);
> t := f(V.2);
> VectorAction(s*t - One(C)) eq SiegelTransformation(V.1,V.2);
  -------result-------
> true
```

The Siegel transformations generate the group $\Omega(V,Q)$.

If the dimension of $V$ is even, the Clifford algebra $C$ of $Q$ is simple. A minimal right ideal of $C$ is a ***spin representation*** and its elements are ***spinors***. The minimal right ideals of $C_+$ are the ***half spin*** spaces.

The restrictions to the groups $\Gamma$, $\Gamma^+$ and $\mathrm{Spin}(V,Q)$ are also called spin representations.

```
> F<z> := GF(9);
> Q := StandardQuadraticForm(6,F);
> C,V,f := CliffordAlgebra(Q);
> S := MinimalRightIdeals(C : Limit := 1)[1];
> Dimension(S);
> 8
> s := f(V.1); t := f(V.2); g := s*t - One(C);
> m := VectorAction(g); n := ActionMatrix(S,g);
> IsUnipotent(m), IsUnipotent(n), IsUnipotent(-n);
  -------result-------
> true 2
> false
> true 2
```

Collect 6 random Siegel elements of $\mathrm{Spin}^+(6,9)$ and find the group they generate in the spin representation.

```
> X := { };
> for random u in V do
>   if u eq 0 or QuadraticNorm(u) ne 0 then continue; end if;
>   for random v in V do
>     if v ne 0 and DotProduct(u,v) eq 0 then Include(~X,<u,v>);
>       break;
>     end if;
>   end for;
>   if #X ge 6 then break; end if;
> end for;
> H := sub<GL(Dimension(S),F) | [ActionMatrix(S,f(u)*f(v) - One(C))
>      : p in X | true where u,v is Explode(p) ]>;
>   LMGFactoredOrder(H), FactoredOrder(OmegaPlus(6,F));
```

```
-------result-------
> [ <2, 12>, <3, 12>, <5, 2>, <7, 1>, <13, 1>, <41, 1> ]
> [ <2, 11>, <3, 12>, <5, 2>, <7, 1>, <13, 1>, <41, 1> ]
```

Let us have a look at ***Minkowski space***.

```
> Q := DiagonalMatrix(Rationals(),[1,1,1,-1]);
> C<e1,e2,e3,e4>, V, f := CliffordAlgebra(Q);
> IsSimple(C), Dimension(Centre(C));
-------result-------
> true 1
```

`C` is the central simple algebra of $4 \times 4$ matrices over $\mathbb{Q}$.

```
> E, h := EvenSubalgebra(C);
> Z := Centre(E); i := Z.2;
> IsSimple(E), Dimension(E), Dimension(Z), i^2;
-------result-------
> true 8 2 (-1  0)
> AsPolynomial(h(i));
-------result-------
> e1*e2*e3*e4
```

`E` is the central simple algebra of $2 \times 2$ matrices over $\mathbb{Q}[i]$.

```
> ee := (1/2)*(1 - e1*e4);
> ff := (1/2)*(1 + e1*e4);
> R, r := rideal< E | ee >;
> S, s := rideal< E | ff >;
> Dimension(R), Dimension(S);
-------result-------
> 4 4
```

*Exercise* 8D.1. Let `E` be the even subalgebra of the Clifford algebra of the quadratic form `Q` over the rationals with signature $(3, 1)$.

```
> Q := DiagonalMatrix(Rationals(),[1,1,1,-1]);
> C<e1,e2,e3,e4>, V, f := CliffordAlgebra(Q);
> Z := Centre(E);
```

Let `R` be the right ideal

```
> R, r := rideal< E | (1/2)*(1 - e1*e4) >;
```

Observe that `Z` is isomorphic to the Gaussian field $\mathbb{Q}[i]$ and that `R` is a vector space of dimension 2 over `Z`. Check that `{-e1*e2*e3*e4+e2*e3, e1*e2-e2*e4}` is a `Z`-basis for `R`.

Identifying `Z` with $\mathbb{Q}[i]$, write a MAGMA function that returns the matrix in $\mathrm{Mat}(2, \mathbb{Q}[i])$ of an element of `E` acting on `R`.

Show that the matrices of `e4*e3`, `e4*e2` and `e4*e1` are the ***Pauli*** matrices. ◇

8E. **Group algebras.** The ***group algebra*** of a finite group $G$ with coefficients from a field (or ring) $K$ is the $K$-space $K[G]$ of formal sums $\sum_{g \in G} a_g g$ with coefficients $a_g \in K$ and multiplication inherited from $G$.

Let $\chi_j$ $(1 \leq j \leq m)$ be the irreducible complex characters of $G$, let $\rho_j : G \to \mathrm{GL}(W_j)$ be a representation corresponding to $\chi_j$ and put $n_j = \dim(W_j)$.

Define $\widetilde{\rho}_j : \mathbb{C}[G] \to \mathrm{End}(W_j) : \sum_{g\in G} a_g g \mapsto \sum_{g\in G} a_g \rho_j(g)$. The family $\left(\widetilde{\rho}_j\right)_{1\le j\le m}$ defines the ***Fourier transform***

$$\widetilde{\rho} : \mathbb{C}[G] \to \prod_{j=1}^{m} \mathrm{End}(W_j) \simeq \prod_{j=1}^{m} \mathrm{Mat}(n_j, \mathbb{C}).$$

The group algebra $\mathbb{C}[G]$ is semisimple and $\widetilde{\rho}$ is an isomorphism.

In MAGMA $\mathbb{C}$ is not an 'exact field'. However, the irreducible representations of $G$ can always be written over the field of $n$th roots of unity, where $n$ is the exponent of $G$ (Richard Brauer).

For example, $\mathbb{Q}[w]$, where $w^3 = 1$ is a splitting field for Alt(4).

```
> G := AlternatingGroup(4);
> F<w> := CyclotomicField(3 : Sparse);
> A := GroupAlgebra(F,G);
> R, rho := RegularRepresentation(A);
> V := GModule(sub<GL(#G,F) | [rho(G.i) : i in [1..Ngens(G)]]>);
> dsd := DirectSumDecomposition(V);
> [Dimension(X) : X in dsd];
```

```
-------result-------
> [ 1, 1, 1, 3, 3, 3 ]
```

```
> ActionGenerators(dsd[4]);
```

```
-------result-------
> [
> [-1  0  0]       [ 0  1  1]
> [ 0  0  1]       [ 1  0 -1]
> [ 0  1  0],      [ 0  1  0]
> ]
```

Collect the representations $G \to \mathrm{GL}(W_j)$.

```
> irreps := IrreducibleModules(G,F);
> sigma := [hom< G -> GL(Dimension(W),F) | ActionGenerators(W) >
>   : W in irreps];
```

Let $\widetilde{\rho}_j$ $(1 \le j \le m)$ be the irreducible representations of $F[G]$. Suppose that $U = [u_1, \ldots, u_m]$ where $u_j \in \mathrm{im}\widetilde{\rho}_j$. The following function returns $u \in F[G]$ such that $\widetilde{\rho}_j(u) = u_j$ for all $j$.

```
> fourierInv := func< A,sigma,U |
>   &+[ &+[Nrows(u)*Trace(sigma[i](s^-1)*u) : i -> u in U]*A!s
>      : s in Group(A) ] >;
```

Check:

```
> U := < rho(Random(G)) : rho in sigma >;
> fourierInv(A,sigma,U);
```

```
-------result-------
> (-w + 1)*Id(G) + (-w + 1)*(1, 2)(3, 4)
> + (-w + 1)*(1, 3, 2) + > (-w + 1)*(1, 4, 3)
> + (-w + 1)*(2, 3, 4) + (-w + 1)*(1, 2, 4)
> + (2*w - 2)*(1, 3, 4) + (2*w - 2)*(1, 4, 2)
> + (2*w + 10)*(2, 4, 3) + (2*w - 2)*(1, 2, 3)
> + (-w + 1)*(1, 3)(2, 4) + (-w + 1)*(1, 4)(2, 3)
```

8F. **Basic algebras.** Let us have a look at the handbook again:

A ***basic algebra*** is a finite dimensional algebra over a field, all of whose simple modules have dimension one.

In the literature such an algebra is known as a "split" basic algebra.

Every algebra is Morita equivalent to a basic algebra, though a field extension may be necessary to obtain the split basic algebra.

MAGMA has several functions that create the basic algebras corresponding to algebras of different types.

The type `AlgBas` in Magma is optimized for the purposes of doing homological calculations.

Suppose $A$ is a finite dimensional algebra over a field $F$. If $e_1, \dots, e_s$ are primitive orthogonal idempotents such that $1 = e_1 + \cdots + e_s$, then $A$ is the direct sum of the indecomposable (a.k.a. projective) right $A$-modules $e_iA$. If $A$ is basic, then $e_iA \simeq e_jA$ if and only if $i = j$.

Number the $e_i$ so that $e_1A$, $\dots$, $e_tA$ represent the isomorphism classes of projective indecomposable modules. Then $eAe$ is a basic algebra for $A$.

```
> P := PermutationGroup(ATLASGroup(2A7));
> M := PermutationModule(P,Stabiliser(P,1),GF(2));
> A := Action(M);
  -------result-------
> Matrix Algebra of degree 240 with 2 generators over GF(2)
> C := CondensedAlgebra(A); C;
  -------result-------
> Matrix Algebra of degree 36 with 15 generators over GF(2)
> CartanMatrix(A);
  -------result-------
> [3 2 0 0 0 4]
> [2 3 0 0 0 4]
> [0 0 7 4 4 0]
> [0 0 4 4 2 0]
> [0 0 4 2 4 0]
> [4 4 0 0 0 8]
```

Here is an example from Jon Carlson showing that the basic algebra of the ***principal*** block of the double cover of Alt(7) is isomorphic to the basic algebra of the ***second*** block of the double cover of Alt(9).

```
> A := BasicAlgebraFromGroup("2A7",2,1); A;
  -------result-------
> Basic algebra of dimension 38 over GF(2)
> Number of projective modules: 3
> Number of generators: 8
> B := BasicAlgebraFromGroup("2A9",2,2); B;
  -------result-------
> Basic algebra of dimension 38 over GF(2)
> Number of projective modules: 3
> Number of generators: 8
> IsIsomorphic(A,B);
  -------result-------
> true Mapping from: AlgBas: A to AlgBas: B
```

## 9. A few additional examples

We will cover one example for some of the topics mentioned in Section 2B.

9A. **Rings and fields.** ***Algebraic integers*** are solutions to polynomial equations with integer coefficients. An easy example is $\sqrt{2}$, which is a solution of $x^2 - 2 = 0$.

One could ask the question how difficult is it to write down algebraic integers explicitly. Are they all easy expressions such as $\sqrt{2}$ or $\sqrt{\sqrt{2}+2}$? Expressions of this form a called radical expressions: expressions that can be obtained by finite concatenations of addition, subtraction, multiplication, division and taking roots.

The ***Galois group*** of a polynomial can be seen as a measurement of how far away an algebraic integer is from being a radical expression. To simplify our story, the Galois group $G(f)$ of a polynomial of degree $n$ is a subgroup of the symmetric group $S_n = \mathrm{Aut}(\{1, \ldots, n\})$, and we say an algebraic integer $x$ is difficult if $G(f)$ is close to $S_n$, where $f$ is the minimal polynomial of $x$.

It turns out that almost all algebraic integers are difficult, and we can convince ourselves that this is true using MAGMA:

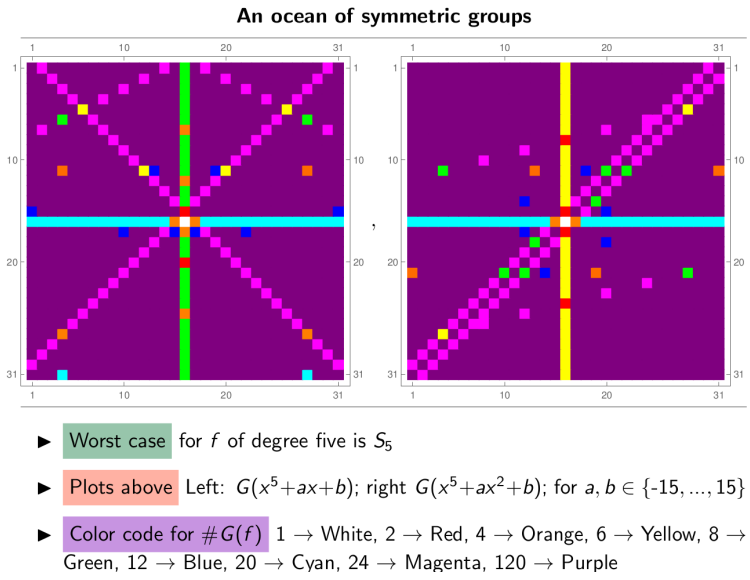
```
> Z:= Integers();
> P<x>:= PolynomialRing(Z);
> n:=5;
> X:=[];
> for a in [-n..n] do
> for b in [-n..n] do
> G, R, S := GaloisGroup(x^5+a*x^2+b);
> X:=Append(X,[Order(G),a,b]);
> end for;
> end for;
> print(X);
```

```
-------result-------
```

```
> [
> [ 120, -1, -1 ],
> [ 2, -1, 0 ],
> [ 120, -1, 1 ],
> [ 4, 0, -1 ],
> [ 1, 0, 0 ],
> [ 4, 0, 1 ],
> [ 120, 1, -1 ],
> [ 2, 1, 0 ],
> [ 120, 1, 1 ]
> ]
```

Here we took the polynomial $f = x^5 + ax^2 + b$, varied $a$ and $b$ between $-n$ and $n$ for $n = 1$ and let MAGMA output the order of $G(f)$. The maximal order in this case is 120 and we have already quite a few appearances of it. If we make the $n$ larger, say $n = 15$, the pattern becomes

clear:




(The labels on the x and y axis need to be shifted by $-n$ so that 0 is in the middle.) Here we did the same as above for $f$ and also for $g = x^5 + ax + b$. We get an ocean of symmetric groups.

*Exercise* 9A.1. Instead of just the order of $G(f)$, let MAGMA compute the group itself. Identify the groups that one gets. ◇

9B. **Algebras.** Let $V = \mathbb{C}^2$ be the vector representation of $\mathrm{SL}_2(\mathbb{C})$ The ***Temperley–Lieb algebra*** is the endomorphism algebra $TL_n = \mathrm{End}_{\mathrm{SL}_2(\mathbb{C})}(V^{\otimes n})$. The dimension of the Temperley–Lieb algebra is given by the Catalan numbers as in Section 4B.

Changing from $\mathbb{C}$ to $\overline{\mathbb{F}}_p = \bigcup_{k\in\mathbb{Z}_{\geq 0}} \mathbb{F}_{p^k}$ does not change much – one still obtains the Temperley–Lieb algebra as $TL_n = \mathrm{End}_{\mathrm{SL}_2(\overline{\mathbb{F}}_p)}(V^{\otimes n})$ for $V = \overline{\mathbb{F}}_p^2$ (just the ground field is different).

However, one needs to work with the algebraic closure of the finite fields as the following code reveals:

```
> p:=5;
> k:=1;
> G:=SL(2,p^k);
> Irr:=IrreducibleModules(G, GF(p^k));
> M:=Irr[2];
> for i in [1..6] do
> N:=TensorPower(M,i);
> Z:=EndomorphismAlgebra(N);
> print Dimension(Z);
> print Binomial(2*i,i) div (i+1);
> end for;
  -------result-------
> 1
> 1
> 2
> 2
> 5
> 5
> 14
> 14
```

```
> 42
> 42
> 133
> 132
```

This code computes $\mathrm{End}_{\mathrm{SL}_2(\mathbb{F}_{p^k})}((\mathbb{F}_{p^k}^2)^{\otimes n})$ and prints its dimension. It also prints the Catalan numbers, so the dimension of the Temperley–Lieb algebra. Observe that these numbers eventually differ. The corresponding algebra over $\mathbb{F}_{p^k}$ is called the ***finite Temperley–Lieb algebra***.

*Exercise* 9B.1. Write code that verifies that the finite Temperley–Lieb algebra as above is not semisimple most of the time. ◇

9C. **Representation theory.** Let us produce the ***character table*** of the symmetric group `Sym(3)`:

```
> G:=Sym(3); CharacterTable(G);
  -------result-------
> Character Table of Group G
> -------------------------
>
>
> -----------------
> Class |   1  2  3
> Size  |   1  3  2
> Order |   1  2  3
> -----------------
> p  =  2   1  1  3
> p  =  3   1  2  1
> -----------------
> X.1   +   1  1  1
> X.2   +   1 -1  1
> X.3   +   2  0 -1
```

Note the zero in the table; it appears for a conjugacy class of size three. There are two things we can count with respect to the zeros:

**(a)** The number of zeros in the table itself. Here we have $1/9 \approx 0.111$ entries that are zero.

**(b)** The number of weighted zero, where we weight by the number of elements in the character table. The number of weighted entries are $3(1+3+2) = 18$ and the one zero appears with weight 3 so that the ratio is $1/6 \approx 0.167$

It appears for a conjugacy class of size three, so three elements of the symmetric group have character zero.

Let us count the number of weighted appearance of zero for larger $n$:

```
> R := RealField(10);
> for n in [1 .. 12] do
> G:=Sym(n);
> X:=CharacterTable(G);
> Y:=Classes(G);
> count:=0;
> for i in [1 .. #X] do
> for j in [1 .. #X] do
> if X[i,j] eq 0 then
> count+:=Y[j,2];
```

```
> end if;
> end for;
> end for;
> R!count/(#X*#G);
> end for;
  -------result-------
> 0.0000000000
> 0.0000000000
> 0.1666666667
> 0.2333333333
> 0.2595238095
> 0.3584595960
> 0.3723015873
> 0.4692302489
> 0.5008948780
> 0.5319339989
> 0.5545011186
> 0.6065162365
```

We get the expected $1/6$ as the third output. And now the number of zeros in the character table itself:

```
> R := RealField(10);
> for n in [1 .. 12] do
> G:=Sym(n);
> X:=CharacterTable(G);
> count:=0;
> for i in [1 .. #X] do
> for j in [1 .. #X] do
> if X[i,j] eq 0 then
> count+:=1;
> end if;
> end for;
> end for;
> R!count/(#X^2);
> end for;
  -------result-------
> 0.0000000000
> 0.0000000000
> 0.1111111111
> 0.1600000000
> 0.2040816326
> 0.2396694215
> 0.2444444444
> 0.3161157025
> 0.3411111111
> 0.3333333333
> 0.3246173470
> 0.3761173891
```

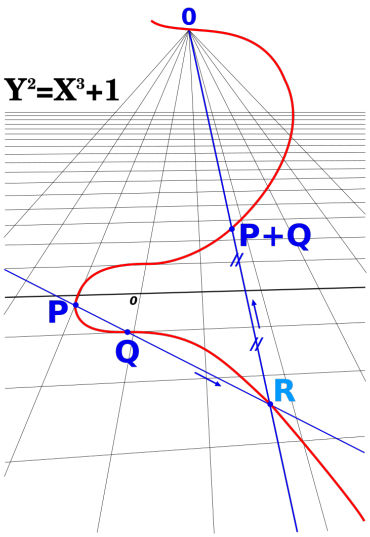
The third entry is `Sym(3)` where we had 1 out of 9 zeros in the character table

*Exercise* 9C.1. Can you read of the limit $n \to \infty$ of the above? Also try other groups and study the limit. ◇

9D. **Algebraic geometry and commutative algebra.** The following is a modification of [BC06, Graded rings and special K3 surfaces].

Recall that a ***projective variety*** $X$ over $\mathbb{C}$ is a subset of complex projective space $\mathbb{P}^n$ that is the zero-locus of some finite family of homogeneous polynomials of $n+1$ variables that generate a prime ideal.

An example the reader can keep in mind is the elliptic curve $Y^2 = X^3+1$ which is illustrated as




Note that the point at infinity is included since we view this variety as part of $\mathbb{P}^2$. To make $Y^2 = X^3 + 1$ homogeneous we modify it to $Y^2Z = X^3 + Z^3$.

A projective variety $X$ has an associated graded ring $R(X)$, called its ***homogeneous coordinate ring***. For example,

$$X = (Y^2Z - X^3 - Z^3) \subset \mathbb{P}^1 \hookleftarrow \mathbb{C}[X,Y]/(Y^2Z - X^3 - Z^3).$$

Using rational coefficients, this can be setup in MAGMA as follows:

```
> P3<X,Y,Z>:=ProjectiveSpace(Rationals(),2);
> X:=Scheme(P3,Y^2*Z-X^3-Z^3); X;
  -------result-------
> Scheme over Rational Field defined by
> -X^3 + Y^2*Z - Z^3
```

The ***Hilbert series*** $P_{R(X)}(t)$ records the vector space structure of $R(X)$: if $r_n$ denotes the dimension of the vector space of homogeneous polynomials of degree $n$ in $R(X)$, then

$$P_{R(X)}(t) = 1 + r_1t + r_2t^2 + r_3t^3 + \dots,$$

so that $P_{R(X)}(t)$ is the generating function of the $r_n$.

This is how MAGMA computes the Hilbert series:

```
> R:=CoordinateRing(X);
> P<t>:=HilbertSeries(R);
> P;
  -------result-------
> (t^2 + t + 1)/(t^2 - 2*t + 1)
```

Viewing this as a power series works as follows:

```
> S<s>:=PowerSeriesRing(Rationals());
> S!P;
  -------result-------
```

```
> 1 + 3*s + 6*s^2 + 9*s^3 + 12*s^4 + 15*s^5 +
> 18*s^6 + 21*s^7 + 24*s^8 + 27*s^9 +
> 30*s^10 + 33*s^11 + 36*s^12 + 39*s^13 + 42*s^14 + 45*s^15 + 48*s^16 +
> 51*s^17 + 54*s^18 + 57*s^19 + O(s^20)
```

Let us describe a family of graded rings with a given Hilbert series. Consider the following power series $P = P(t)$:

```
> T<t>:=PowerSeriesRing(Rationals():Precision:=50);
> P:=1 + t + t^2 + t^3 + 2*t^4 + 3*t^5 + 4*t^6+
> &+[ (n-3)*t^n : n in [7..49] ]
> + O(t^50);
```

Multiplying $P$ by $1-t$ cancels some small powers of $t$ in the expansion, in the sense that

```
> (1-t)*P;
  -------result-------
> 1 + t^4 + t^5 + t^6 + t^8 + t^9 + t^10
> + t^11 + t^12 + t^13 + t^14 + t^15 + t^16
> + t^17 + t^18 + t^19 + t^20 + t^21
> + t^22 + t^23 + t^24 + t^25 + t^26 + t^27
> + t^28 + t^29 + t^30 + t^31 + t^32
> + t^33 + t^34 + t^35 + t^36 + t^37 + t^38
> + t^39 + t^40 + t^41 + t^42 + t^43
> + t^44 + t^45 + t^46 + t^47 + t^48 + t^49
> + O(t^50)
```

Multiplying this by $1-t^4$ gives:

```
> (1-t)*(1-t^4)*P;
  -------result-------
> 1 + t^5 + t^6 + t^11 + O(t^50)
```

That completes the task (at least up to the given precision) since the last display can be rewritten as

$$P = \frac{1 + t^5 + t^6 + t^{11}}{(1-t)(1-t^4)}.$$

One can go further:

```
> (1-t)*(1-t^4)*(1-t^5)*(1-t^6)*P;
  -------result-------
> 1 - t^10 - t^12 + t^22 + O(t^50)
```

And so on. Expressed in this way, the power series $P$ is the Hilbert series of any variety

$$X = (f_{10} = g_{12} = 0) \subset \mathbb{P}^3(1,4,5,6),$$

where $f$ and $g$ are homogeneous polynomials, of the indicated degrees, in variables $X, Y, Z, W$ of weights $1, 4, 5, 6$.

*Exercise* 9D.1. Prove that, for suitable choice of equations, the variety $X$ as above is a nonsingular curve of genus 4 polarized by a subcanonical divisor D with $\deg D = 1$ and $6D = K_X$ where $K_X$ is a canonical divisor. ◇

9E. **Arithmetic geometry and modular arithmetic geometry.** This example is strongly motivated by [BC06, Some ternary Diophantine equations of signature (n,n,2)].

Say we want to find integer solutions for $D(x^4+7) = y^2$ where $D \in \{2,3,5,7,11,13\}$.

We first check whether there are any points over $\mathbb{Q}_2$:

```
> _<x>:=PolynomialRing(Rationals());
> Dset:={2,3,5,7,11,13};
> {D:D in Dset|IsLocallySolvable(HyperellipticCurve(D*(x^4+7)),2)};
```

```
-------result-------
```

```
> { 2, 7 }
```

In other words, we have proven that there are no integer solutions for all values except $D = 2$ and $D = 7$.

For $D = 2$ one has the solution $(x, y) = (1, 4)$. This implies that our curve is isomorphic to an elliptic curve. The rational points of an ***elliptic curve*** form a finitely generated group, and Magma can compute an upper bound on the free rank of that group:

```
> _<x>:=PolynomialRing(Rationals());
> C2:=HyperellipticCurve(2*(x^4+7));
> p0:=C2![1,4];
> E,C2toE:=EllipticCurve(C2,p0);
> RankBound(E);
```

```
-------result-------
```

```
> 1 true
```

Let us compute what this group is.

```
> G,GtoE:=MordellWeilGroup(E);
> G;
```

```
-------result-------
```

```
> Abelian Group isomorphic to Z/2 + Z
> Defined on 2 generators
> Relations:
> 2*G.1 = 0
```

We can even get solutions:

```
> [Inverse(C2toE)(GtoE(g)):g in OrderedGenerators(G)];
```

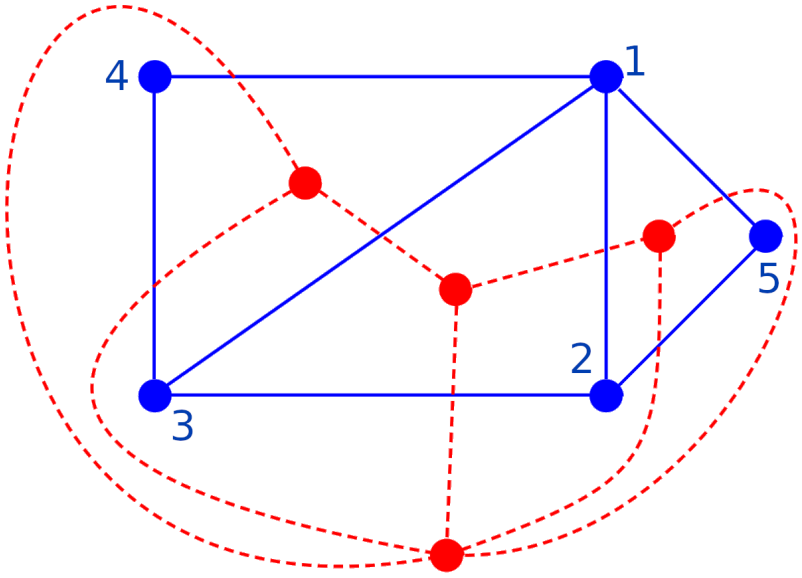
```
-------result-------
```

```
> [ (-1 : -4 : 1), (1 : -4 : 1) ]
```

This gives us the solutions $(x, y) = (-1, \pm 4)$ and $(x, y) = (1, \pm 4)$. Using the group law, arbitrarily many solutions can be constructed from these.

*Exercise* 9E.1. Do the case $D = 7$. ◇

9F. **Combinatorics and graph theory.** Let us set up the following graph (blue, undashed) in MAGMA:




This works as follows:

```
> G := Graph< 5 | { 1, 2 }, { 2, 3 }, { 3, 4 }, { 4, 1 },
{ 1, 5 }, { 2, 5 }, { 1, 3 } >;
```

The graph $P$ has five vertices, and the above given edges.

An amazing fact is that testing whether a graph is ***planar*** (can be drawn in the plane without intersecting edges) is easy to check: the algorithm used by MAGMA runs in $O(n)$, where $n$ is the number of vertices.

MAGMA can check this as follows:

```
> IsPlanar(G);
   -------result-------
> true
```

MAGMA also knows the set of faces:

```
> F:=Faces(G);
> F;
   -------result-------
> [
> [ {1, 5}, {5, 2}, {2, 1} ],
> [ {1, 2}, {2, 3}, {3, 1} ],
> [ {1, 3}, {3, 4}, {4, 1} ],
> [ {1, 4}, {4, 3}, {3, 2}, {2, 5}, {5, 1} ]
> ]
```

We can use this to record the orientation of edges defining a face saying $\{a, b\}$ is 1 if $a < b$ and $\{a, b\}$ is $-1$ if $a > b$. The following code does that for us:

```
> Ds := [[1 : x in [1..#F[i]]] : i in [1..#F]];
> for i in [1..#F] do
> for j in [1..#F[i]] do
> if InitialVertex(F[i][j]) gt TerminalVertex(F[i][j]) then
> Ds[i][j] := -1;
> end if;
> end for;
> end for;
```

```
> Ds;
```

```
-------result-------
```

```
> [
> [ 1, -1, -1 ],
> [ 1, 1, -1 ],
> [ 1, 1, -1 ],
> [ 1, -1, -1, 1, -1 ]
> ]
```

The first three faces are triangles, and the final displayed face is the outside one in the picture above. We can access the faces and their edges by using:

```
> F[1];
> F[2][1];
```

```
-------result-------
```

```
> [ {1, 5}, {5, 2}, {2, 1} ]
> {1, 2}
```

The ***dual graph*** $D(G)$ of a planar graph $G$ is the graph with vertices being faces of $G$, and edges between touching faces (there will be a loop when the same face appears on both sides of an edge). Is there are way to get the dual graph in MAGMA? Well, here we go:

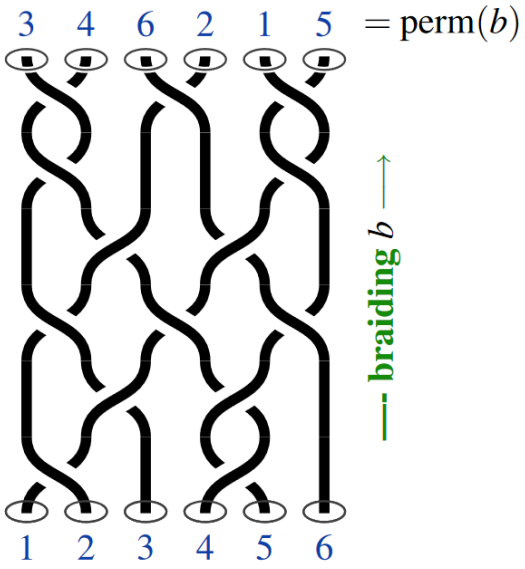
```
> nstar := #F;
> Gstar := MultiDigraph< nstar | >;
> Fs := [ SequenceToSet(f) : f in F ];
> for u in [1..nstar-1] do
> for v in [u+1..nstar] do
> M := Fs[u] meet Fs[v];
> for e in M do
> p := Position(F[u],e);
> q := Position(Edges(G),e);
> if Ds[u][p] eq 1 then
> Gstar,edge := AddEdge(Gstar,VertexSet(Gstar)!u, VertexSet(Gstar)!v);
> else
> Gstar,edge := AddEdge(Gstar,VertexSet(Gstar)!v,VertexSet(Gstar)!u);
> end if;
> end for;
> end for;
> end for;
> for i := 1 to Size(G) do
> , EdgeSet(G).i , , EdgeSet(Gstar).i;
> end for;
```

```
-------result-------
```

```
> {1, 2}   < [2, 1], 1 >
> {1, 3}   < [1, 4], 2 >
> {1, 4}   < [4, 1], 3 >
> {1, 5}   < [3, 2], 4 >
> {2, 3}   < [2, 4], 5 >
> {2, 5}   < [4, 3], 6 >
> {3, 4}   < [3, 4], 7 >
```

*Exercise* 9F.1. Interpret the output above. ◇

9G. **Cryptography.** The ***braid groups*** are groups that describe real world braids such as




algebraically. They are (almost always) infinite and their internal structure is a bit delicate.

Fix a group $G$, the platform group. A classical way to encrypt a message is to use conjugation. This works as follows. Write $g^a = aga^{-1}$ for conjugation. Fix a public element $g \in G$. Then party A chooses privately $a \in G$ and party B chooses privately $b \in G$. Then A communicates $g^a$ and $b$ communicates $g^b$, and the common secret is $g^{ab} = g^{ba}$ A third party C has access to $g$, $g^a$ and $g^b$, but finding $g^{ab}$ from the known data is difficult as long as the conjugacy problem is difficult to solve.

Braids groups were proposed as platform groups since it is not immediately clear how to solve the conjugacy problem for them. However, a fast way to solve conjugacy problems was actually discovered using MAGMA! See [**BC06**, Conjugacy problem in braid groups]. Let us discuss how that works.

First, we set up the braid group in seven strands:

```
> B := BraidGroup(7);
> f := FundamentalElement(B);
> f eq LCM({B.i : i in [1..6]});
> InducedPermutation(d);
  -------result-------
> true
> (1, 7)(2, 6)(3, 5)
```

What is happening? The `B.i` are crossing generators that correspond to the simple transposition $(i, i+1)$ and pull, say, the left string atop the right. `f` is the lowest common multiple of the `B.i`, often called the ***half twist***. Its induced permutation is the longest possible.

The key now is the existence of a unique way of writing any given braid as product of ***simple elements*** of a certain form. By definition, the simple elements of our choice are the nontrivial divisors of `f`.

Now we get a ***normal form***: every positive braid $x$ (from now on we restrict to positive braids for simplicity) can be written as

$$x = f^k A_1 \dots A_r$$

for simple elements $A_i$ satisfying $A_{i-1}^{-1} f \wedge A_i = 1$.

We compute this normal form using `GCD` repeatedly:

```
> function MyNormalForm(x)
> f := FundamentalElement(Parent(x));
```

```
> k := 0;
> seq := [ ];
> while not IsId(x) do
> d := GCD(x,f);
> if d eq f then
> k +:= 1;
> else
> Append(~seq,InducedPermutation(d)^(-1));
> end if;
> x := d^(-1)*x;
> end while;
> return k,seq;
> end function;
> B := BraidGroup(2);
> MyNormalForm(B.1*B.2*B.1*B.2*B.1*B.1*B.2*B.1);
  -------result-------
> [
> (1, 2, 3)
> ]
> 2
```

This function computes the normal form with the output being $k$, the exponent of `f`, and the induced permutation for $A_1 \dots A_r$.

We call the number $k$ the ***infimum*** of $x$ and denote it by $\inf(x)$. Similarly, $\sup(x) = k + r$ is the ***supremum*** of $x$, while $r$ is the ***canonical length***. Using these define the ***super summit set*** $S_x$ of $x$ as:

$$S_x = \{y \text{ conjugate to } x | \inf(y) = \inf_s(x), \sup(y) = \sup_s(x)\},$$

where $\inf_s(x) = \max\{\inf(y) | y \text{ conjugate to } x\}$ and $\sup_s(x) = \min\{\sup(y) | y \text{ conjugate to } x\}$.

It turns out that ($y$ conjugate to $x \Leftrightarrow S_x \cap S_y \neq \emptyset$), and the latter is a fairly easy to check condition. At least with MAGMA!

We first setup a function that gives a representative of $S_x$:

```
> function MySuperSummitRepresentative(x)
> n := NumberOfStrings(Parent(x));
> conj := Id(Parent(x));
> count := n*(n-1)/2-1;
> inf := Infimum(x);
> while count gt 0 and CanonicalLength(x) gt 0 do
> x,c := Cycle(x); count-:=1;
> conj := conj*c;
> if Infimum(x) gt inf then
> count := n*(n-1)/2-1;
> inf := Infimum(x);
> end if;
> end while;
> count := n*(n-1)/2-1;
> sup := Supremum(x);
> while count gt 0 and CanonicalLength(x) gt 0 do
> x,c := Decycle(x); count-:=1;
> conj := conj*c;
> if Supremum(x) lt sup then
> count := n*(n-1)/2-1;
```

```
> sup := Supremum(x);
> end if;
> end while;
> return x,conj;
> end function;
> B := BraidGroup(3);
> MySuperSummitRepresentative(B.1*B.2*B.1*B.2*B.1*B.1*B.2*B.1);
  -------result-------
> B.2 * B.1 * B.2^2 * B.1 * B.2 * B.1 * B.2
> <Artin, 2, [
> (1, 2, 3)
> ], 0> B.1 * B.2 * B.1 * B.2^-1 * B.1^-1 * B.2^-1
> <Artin, -1, [
> (1, 3, 2),
> (1, 3, 2),
> (2, 3),
> (1, 2)
> ], -1>
```

*Exercise* 9G.1. What is the function `MySuperSummitRepresentative(x)` doing? List the main commands and what they do. ◇

```
> function MyIsConjugate(x,y)
> x,c_x := MySuperSummitRepresentative(x);
> S := {@ x @};
> conj := {@ c_x @};
> y,c_y := MySuperSummitRepresentative(y);
> if y eq x then
> return true, 1, c_x*c_y^(-1);
> end if;
> pos := 1;
> while pos le #S do
> for s in { MinimalElementConjugatingToSuperSummit(S[pos],a)
>  : a in Generators(Parent(x))} do
> ns := LeftNormalForm(s^(-1)*S[pos]*s);
> if ns notin S then
> Include(~S,ns);
> Include(~conj,LeftNormalForm(conj[pos]*s));
> if y eq ns then
> return true, #S, conj[pos]*s*c_y^(-1);
> end if;
> end if;
> end for;
> pos +:=1;
> end while;
> return false, #S, _;
> end function;
> B := BraidGroup(3);
> MyIsConjugate(B.1,B.2)
  -------result-------
> true 2 B.2 * B.1
> <Artin, -2, [
```

```
> (1, 3, 2),
> (2, 3),
> (2, 3),
> (1, 2, 3),
> (1, 3, 2)
> ], 0>
```

*Exercise* 9G.2. doing? List the main commands of `MyIsConjugate(x)` and what they do. ◇

## REFERENCES


[BC06] W. Bosma and J. Cannon, editors. *Discovering mathematics with Magma*, volume 19 of *Algorithms and Computation in Mathematics*. Springer-Verlag, Berlin, 2006. Reducing the abstract to the concrete. doi:10.1007/978-3-540-37634-7.

[BC23] W. Bosma and J. Cannon, editors. *Handbook of Magma Functions*. Magma is distributed by the Computational Algebra Group at the University of Sydney. 2023. URL: https://magma.maths.usyd.edu.au/magma/handbook/.

[CP01] J. Cannon and C. Playoust. *An Introduction to Algebraic Programming with Magma*. 2001. URL: https://magma.maths.usyd.edu.au/magma/pdf/intro.pdf.

[Cra08] D. Craven. *Computing with Magma*. 2008. URL: https://web.mat.bham.ac.uk/D.A.Craven/magma.html.

[Tub23] D. Tubbenhauer. *Magma in a nutshell on YouTube*. 2023. URL: https://www.youtube.com/playlist?list=PLuFcVFHMIfhJ07kMJWR1SsoyD2MxprDdL.



D.T.: THE UNIVERSITY OF SYDNEY, SCHOOL OF MATHEMATICS AND STATISTICS F07, OFFICE CARSLAW 609, NSW 2006, AUSTRALIA, WWW.MATHS.USYD.EDU.AU/U/DON/

*Email address*: donald.taylor@sydney.edu.au

D.T.#2: THE UNIVERSITY OF SYDNEY, SCHOOL OF MATHEMATICS AND STATISTICS F07, OFFICE CARSLAW 827, NSW 2006, AUSTRALIA, WWW.DTUBBENHAUER.COM, HTTPS://ORCID.ORG/0000-0001-7265-5047

*Email address*: daniel.tubbenhauer@sydney.edu.au