Mini-course on GAP – Lecture 1

Leandro Vendramin

Vrije Universiteit Brussel

March 2023

WIDS
WISKUNDE &
DATA SCIENCE

GAP is a system for computational discrete algebra. It is freely available here: http://www.gap-system.org/

What are we going to do here?
Outline:

- Arithmetics
- Basic programming
- Linear algebra
- Elementary group theory
- Advanced group theory, representation theory

Immediately after running GAP we will see some information related to the distribution we have installed. We will also see that GAP is ready:

```
gap>
```

To close GAP one uses `quit`:

```
gap> quit;
```

Every command should end with the symbol `;` (semicolon). The symbol `;;` (double semicolon) also is used to end a command but it means that no screen output will be produced.

```
gap> 2+5;;
gap> 2+5;
7
```

To see information related commands or functions, tutorials and manuals one uses the symbol `?` (question mark). Here we have some examples:

```
gap> ?tutorial
gap> ?sets
gap> ?help
gap> ?permutations
gap> ?Eigenvalues
gap> ?CyclicGroup
gap> ?FreeGroup
gap> ?SylowSubgroup
```

To make the command line more readable one could use the symbol \ (backslash):

```
gap> # Let us compute 1+2+3
gap> 1\
> +2\
> +3;
6
```

The function `LogTo` saves the subsequent interaction to be logged to a file. It means that everything you see on your terminal will also appear in this file.

```
gap> # Save the output to the file mylog
gap> LogTo("mylog");
```

This is extremely useful! When the function `LogTo` is called with no parameters GAP will stop writing a log file.

```
gap> # Stop saving the output
gap> LogTo();
```

One can do basic arithmetic operations with rational numbers:

```
gap> 1+1;
2
gap> 2*3;
6
gap> 8/2;
4
gap> (1/3)+(2/5);
11/15
gap> 2*(-6)+4;
-8
gap> NumeratorRat(3/5);
3
gap> DenominatorRat(3/5);
5
```

One uses `mod` to obtain the remainder after division of a by m, where a is the dividend and m is the divisor.

```
gap> 6 mod 4;
2
gap> -6 mod 5;
4
```

There are several functions that one can use for specific purposes. For example `Factors` returns the factorization of an integer and `IsPrime` detects whether an integer is prime or not.

```
gap> Factors(10);
[ 2, 5 ]
gap> Factors(18);
[ 2, 3, 3 ]
gap> IsPrime(1800);
false
gap> Factors(37);
[ 37 ]
gap> IsPrime(37);
true
```

Other useful functions: `Sqrt` computes square roots, `Factorial` computes the factorial of a positive integer, `Gcd` computes the greatest common divisor of a finite list of integers, `Lcm` computes the least commom multiple.

```
gap> Sqrt(25);
5
gap> Factorial(15);
1307674368000
gap> Gcd(10,4);
2
gap> Lcm(10,4,2,6);
60
```

We can also work in cyclotomic fields. `CF` creates a cyclotomic field. To create primitive roots of 1 one uses the function `E`. More precisely: `E(n)` returns $e^{2\pi i/n}$.

```
gap> E(6) in Rationals;
false
gap> E(6) in Cyclotomics;
true
gap> E(3) in CF(3);
true
gap> E(3) in CF(4);
false
gap> E(3)^2+E(3);
-1
gap> E(6);
-E(3)^2
```

Tipically, cyclotomic numbers will be represented as rational linear combinations of primitive roots of 1.

`Inverse` (resp. `AdditiveInverse`) returns the multiplicative (resp. additive) inverse of an element.

```
gap> AdditiveInverse(2/3);
-2/3
gap> Inverse(2/3);
3/2
gap> AdditiveInverse(E(7));
-E(7)
gap> Inverse(E(7));
E(7)^6
```

Exercise: Conway FRACTRAN language

FRACTRAN is a programming language invented by J. Conway. A FRACTRAN program is simply an ordered list of positive rationals together with an initial positive integer input n. The program is run by updating the integer n as follows:

- For the first rational f in the list for which $nf \in \mathbb{Z}$, replace n by nf.
- Repeat this rule until no rational in the list produces an integer when multiplied by n, then stop.

Write an implementation of the FRACTRAN language.

Starting with $n = 2$, the program

$$\frac{17}{91}, \frac{78}{85}, \frac{19}{51}, \frac{23}{38}, \frac{29}{33}, \frac{77}{29}, \frac{95}{23}, \frac{77}{19}, \frac{1}{17}, \frac{11}{13}, \frac{13}{11}, \frac{15}{2}, \frac{1}{7}, 55$$

produces the sequence

$$2, 15, 825, 725, 1925, 2275, 425, 390, 330, 290, 770 \ldots$$

In 1987, J. Conway proved that this sequence contains the set

$$\{2^p : p \text{ prime}\}.$$

See `https://oeis.org/A007542` for more information.

Exercise: Conway "look and say" sequence

The first terms of Conway's "look and say" sequence are

```
1
11
21
1211
111221
312211
```

After guessing how each term is computed, write a script to create the first terms of the sequence.

To solve these two exercises we need some basic programming: conditionals, functions, strings, lists, ranges, sets, records, loops...
We will see these things later!

Let us review some basic mathematical objects in GAP :

- Permutations.
- Finite fields.
- Matrices.

Permutations

A permutation in n letters is a bijective map

$$\sigma\colon \{1,\dots,n\} \to \{1,\dots,n\}.$$

For example, the permutation $\binom{1234}{3124}$ is the bijective map such that $1 \mapsto 3$, $2 \mapsto 1$, $3 \mapsto 2$ and $4 \mapsto 4$.

Usually one writes a permutation as a product of disjoint cycles. For example:

$$\begin{pmatrix}1234\\2413\end{pmatrix} = (1243), \qquad \begin{pmatrix}12345\\21435\end{pmatrix} = (12)(34)(5) = (12)(34).$$

The permutation $\binom{12345}{21435} = (12)(34)$ in GAP is `(1,2)(3,4)`.

Permutations

The function `IsPerm` checks whether some object is a permutation. Let us see some examples:

```
gap> IsPerm((1,2)(3,4));
true
gap> (1,2)(3,4)(5)=(1,2)(3,4);
true
gap> (1,2)(3,4)=(3,4)(2,1);
true
gap> IsPerm(25);
false
gap> IsPerm([1,2,3,4]);
false
```

The image of an element `i` under the natural right action of a permutation `p` is `i^p`. The preimage of the element `i` under `p` can be obtained with `i/p`. In the following example we compute the image of 1 and the preimage of 3 by the permutation (123):

```
gap> 2^(1,2,3);
3
gap> 2/(1,2,3);
1
```

Permutations

Composition of permutations will be performed from left to right. For example

$$(123)(234) = (13)(24)$$

as the following code shows:

```
gap> (1,2,3) * (2,3,4);
(1,3)(2,4)
```

To obtain the inverse of a permutation one uses `Inverse`:

```
gap> Inverse((1,2,3));
(1,3,2)
gap> (1,2,3)^(-1);
(1,3,2)
```

Permutations

Let σ be a permutation written as a product of disjoint cycles. The function `ListPerm` returns a list containing $\sigma(i)$ at position i.

```
gap> # The permutation (12) in two letters
gap> ListPerm((1,2));
[ 2, 1 ]
gap> # The permutation (12) in four letters
gap> ListPerm((1,2), 4);
[ 2, 1, 3, 4 ]
gap> ListPerm((1,2,3)(4,5));
[ 2, 3, 1, 5, 4 ]
gap> ListPerm((1,3));
[ 3, 2, 1 ]
```

Conversely, any list of this type can be transformed into a permutation with the function `PermList`.

```
gap> PermList([1,2,3]);
()
gap> PermList([2,1]);
(1,2)
```

The sign of a permutation σ is the number $(-1)^k$, where one writes $\sigma = \tau_1 \cdots \tau_k$ as a product of transpositions. To compute the sign of a permutation one uses the function `SignPerm`.

```
gap> SignPerm(());
1
gap> SignPerm((1,2));
-1
gap> SignPerm((1,2,3,4,5));
1
gap> SignPerm((1,2)(3,4,5));
-1
gap> SignPerm((1,2)(3,4));
1
```

An exercise on permutations

For a given positive integer n construct the permutation $\sigma \in \mathrm{Sym}_n$ given by $\sigma(j) = n - j + 1$, Write σ as a product of disjoint cycles and compute its sign.

Finite fields

To create the finite field of p^n elements (here p is a prime number) we use the function `GF`. The characteristic of a field can be obtained with `Characteristic`.

```
gap> GF(2);
GF(2)
gap> GF(9);
GF(3^2)
gap> Characteristic(Rationals);
0
gap> Characteristic(CF(3));
0
gap> Characteristic(CF(4));
0
gap> Characteristic(GF(2));
2
gap> Characteristic(GF(9));
3
```

Finite fields

Let p be a prime number and let F denote the field with $q = p^n$ elements, for some $n \in \mathbb{N}$. The subset

$$\{x \in F : x \neq 0\}$$

is a cyclic group of size $q - 1$; say generated by ζ. Then

$$F = \{0, \zeta, \zeta^2, \ldots, \zeta^{q-1}\},$$

so each non-zero element of F is then a power of ζ.

```
gap> Size(GF(4));
4
gap> Elements(GF(4));
[ 0*Z(2), Z(2)^0, Z(2^2), Z(2^2)^2 ]
gap> Z(4);
Z(2^2)
gap> Inverse(Z(4));
Z(2^2)^2
```

Finite fields

In GAP each non-zero element of the finite field `GF(q)` will be a power of the generator `Z(q)`. The zero of `GF(q)` will be `0*Z(q)` or equivalently `Zero(GF(q))`. `One(GF(q))` will be the multiplicative neutral element of `GF(q)`.

```
gap> Zero(GF(4));
0*Z(2)
gap> 0 in GF(4);
false
gap> Zero(Rationals);
0
gap> One(GF(4));
Z(2)^0
gap> 1 in GF(4);
false
gap> One(Rationals);
1
```

Finite fields

To recognize elements in finite fields with a prime number of elements one uses the function `Int`.

```
gap> Elements(GF(5));
[ 0*Z(5), Z(5)^0, Z(5), Z(5)^2, Z(5)^3 ]
gap> Int(Z(5)^0);
1
gap> Int(Z(5)^1);
2
gap> Int(Z(5)^2);
4
gap> Int(Z(5)^3);
3
```

An exercise on permutation polynomials

Prove that the map

$$f : \mathbb{Z}/8 \to \mathbb{Z}/8, \quad f(x) = 2x^2 + x,$$

defines a permutation on the ring $\mathbb{Z}/8$. Can you write this permutation as a product of disjoint cycles?

Matrices

For us a matrix will be just a rectangular array of numbers. The size of a matrix can be obtained with `DimensionsMat`. Sometimes (for example if one has an integer matrix) the function `Display` shows matrices in a nice way.

```
gap> m := [[1,2,3],[4,5,6]];;
gap> Display(m);
[ [  1,  2,  3 ],
  [  4,  5,  6 ] ]
gap> m[1][1];
1
gap> m[1][2];
2
gap> m[2][1];
4
gap> DimensionsMat(m);
[ 2, 3 ]
```

Matrices

Let $v = (1, 2, 3)$ and $w = (0, 5, -7)$ be row vectors of $\mathbb{Q}^3$. Let us check that $-5v = (-5, -10, -15)$ and $2v - w = (2, -1, 13)$.

```
gap> v := [1,2,3];;
gap> w := [0,5,-7];;
gap> IsRowVector(v);
true
gap> IsRowVector(w);
true
gap> -5*v;
[ -5, -10, -15 ]
gap> 2*v-w;
[ 2, -1, 13 ]
```

We also check that the inner product between v and w is -11:

```
gap> v*w;
-11
```

A very simple exercise with matrices

Let

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}, \quad B = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \end{pmatrix}, \quad C = \begin{pmatrix} 1 & 2 \\ 6 & 1 \\ 0 & 2 \end{pmatrix}.$$

Compute A^3, BC, CB, $A + CB$ and $2A - 5CB$.

Matrices

To construct a null matrix one uses the function `NullMat`. The identity is constructed with the function `IdentityMat`. To construct diagonal matrices one uses `DiagonalMat`.

```
gap> Display(NullMat(2,3));
[ [  0,  0,  0 ],
  [  0,  0,  0 ] ]
gap> Display(IdentityMat(3));
[ [  1,  0,  0 ],
  [  0,  1,  0 ],
  [  0,  0,  1 ] ]
gap> Display(DiagonalMat([1,2]));
[ [  1,  0 ],
  [  0,  2 ] ]
```

Matrices

We know that `matrix[i][j]` returns the (i, j)-element of our matrix. To extract submatrices from a matrix one uses

```
matrix{rows}{columns}
```

like in the following example:

```
gap> m := [\
> [1, 2, 3, 4, 5],\
> [6, 7, 8, 9, 3],\
> [3, 2, 1, 2, 4],\
> [7, 5, 3, 0, 0],\
> [0, 0, 0, 0, 1]];
gap> m{[1..3]}{[1..3]};
[ [ 1, 2, 3 ], [ 6, 7, 8 ], [ 3, 2, 1 ] ]
gap> m{[2,4,5]}{[1,3]};
[ [ 6, 8 ], [ 7, 3 ], [ 0, 0 ] ]
```

Matrices

It is possible to work with matrices with coefficients in arbitrary rings. Let us start working with matrices over the finite field of five elements:

```
gap> m := [[1,2,3],[3,2,1],[0,0,2]]*One(GF(5));
[ [ Z(5)^0, Z(5), Z(5)^3 ],
  [ Z(5)^3, Z(5), Z(5)^0 ],
  [ 0*Z(5), 0*Z(5), Z(5) ] ]
gap> Display(m);
 1 2 3
 3 2 1
 . . 2
```

Matrices

Now let us work with 3×3 matrices with coefficients in the ring $\mathbb{Z}/4$. Let us compute the identity of $M_3(\mathbb{Z}/4)$:

```
gap> m := IdentityMat(3, ZmodnZ(4));;
gap> Display(m);
matrix over Integers mod 4:
[ [  1,  0,  0 ],
  [  0,  1,  0 ],
  [  0,  0,  1 ] ]
```

One uses the function `Inverse` to compute the inverse of an invertible (square) matrix. This function returns `fail` if the matrix is not invertible.

```
gap> m := [[1, -2, -1], [0, 1, 0], [1, -1, 0]];;
gap> Display(Inverse(m));
[ [   0,   1,   1 ],
  [   0,   1,   0 ],
  [  -1,  -1,   1 ] ]
gap> Inverse([[1,0],[2,0]]);
fail
```

`IsIdentityMat` returns either **`true`** if the argument is the identity matrix or **`false`** otherwise.

```
gap> IsIdentityMat(m*Inverse(m));
true
```

We use `TransposedMat` to compute the transpose of a matrix:

```
gap> m := [[1, -2, -1], [0, 1, 0], [1, -1, 0]];;
gap> Display(TransposedMat(m)*m);
[ [   2,  -3,  -1 ],
  [  -3,   6,   2 ],
  [  -1,   2,   1 ] ]
```

A tricky exercise with matrices

Compute the eigenvalues and eigenvectors of the matrix

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 6 & 7 & 8 \end{pmatrix} \in \mathbb{Q}^{3\times 3}.$$

WARNING:

The function `Eigenvectors` returns generators of the eigenspaces, where $v \neq 0$ is an eigenvector of A with eigenvalue λ if and only if $vA = \lambda v$.

Let us start with basic GAP programming.
Outline:

- Objects and variables
- Conditionals
- Functions
- Strings
- Lists
- Ranges
- Sets
- Loops

Objects and variables

An object is something that we can assign to a variable. So an object could be either a number, a string, a group, a field, an element of a group, a group homomorphism, a ring, a matrix, a vector space...

Objects and variables

To assign an object to a variable one uses the operator := as the following example shows:

```
gap> p := 32;;
gap> p;
32
gap> p = 32;
true
gap> p := p+1;;
gap> p;
33
gap> p = 32;
false
```

WARNING:

Don't forget that the symbols = (conditional) and := (assignment operator) are different!

Objects and variables

What if I forgot to assign the result of a calculation for further use? We can do the following:

```
gap> 2*(5+1)-6;
6
gap> n := last;
6
```

One also has `last2` and `last3`.

Conditionals

There are three very important operators: **not**, **and**, **or**. We also have comparison operators; for example the expression `x<>y` returns **true** if `x` and `y` are different, and **false** otherwise.

```
gap> x := 20;; y := 10;;
gap> x <> y;
true
gap> x > y;
true
gap> (x > 0) or (x < y);
true
gap> (x > 0) and (x < y);
false
gap> (2*y < x);
false
gap> (2*y <= x);
true
gap> not (x < y);
true
```

Conditionals

The `if` statement allows one to execute statements depending on the value of some boolean expression.

```
gap> n := 10;;
gap> if n mod 2 = 0 then
> n := n/2;
> else
> n := (n+1)/2;
> fi;
gap> n;
5
```

Better examples will appear soon, we need to use functions!

Functions

There are two ways of constructing functions. For example, to construct the map $x \mapsto x^2$ either we use the one-line definition

```
gap> square := x->x^2;
function( x ) ... end
```

or the classical

```
gap> square := function(x)
> return x^2;
> end;
function( x ) ... end
```

In both cases we will obtain the same result!

Functions

One can also define functions with no arguments.

```
gap> hi := function()
> Display("Hello world");
> end;
function(  ) ... end
gap> hi();
Hello world
```

Functions

Let us write a function to compute the map

$$f\colon n \mapsto \begin{cases} n^3 & \text{si } n \equiv 0 \bmod 3, \\ n^5 & \text{si } n \equiv 1 \bmod 3, \\ 0 & \text{otherwise} . \end{cases}$$

Functions

Here is the code and some experiments:

```
gap> f := function(n)
> if n mod 3 = 0 then
> return n^3;
> elif n mod 3 = 1 then
> return n^5;
> else
> return 0;
> fi;
> end;
function( n ) ... end
gap> f(10);
100000
gap> f(5);
0
gap> f(4);
1024
```

Functions

The Fibonacci sequence f_n is defined as $f_1 = f_2 = 1$ and

$$f_{n+1} = f_n + f_{n-1}$$

for $n \geq 2$. The following function computes Fibonacci numbers:

```
gap> fibonacci := function(n)
> if n = 1 or n = 2 then
> return 1;
> else
> return fibonacci(n-1)+fibonacci(n-2);
> fi;
> end;
function( n ) ... end
gap> fibonacci(10);
55
```

Question: Can you compute f_{100} with this method?