An Introduction to GAP

Hans FOTSING T.
Joseph ATALAYE

AIMS-Volkswagen Workshop on Computer Algebra

Douala, October 6, 2017

Introduction

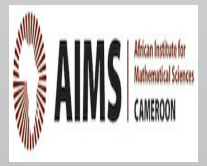
The GAP system developped at Lehrstuhl D für Mathematik, RWTH-Aachen, under the direction of Joachim Neubüser in 1985. In July 2008, GAP was awarded the ACM/SIGSAM Richard Dimick Jenks Memorial Prize for Excellence in Software Engineering applied to Computer Algebra. GAP means Groups, Algorithms and Programming and is a free, open and extensible software package for computation in discrete abstract algebra. The name was chosen to reflect the aim of the system. In this **introduction to GAP**, we will learn about algorithms and programming for algebraic structures such as groups, vector spaces, fields and fields extensions.

1 Generalities on GAP
- About GAP
- Lists
- Functions

1. Generalities on GAP
   - About GAP
   - Lists
   - Functions

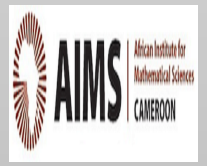
2. Algebraic structures
   - Groups
   - Group Homomorphism
   - Polynomials
   - Rings and Fields

1. Generalities on GAP
   - About GAP
   - Lists
   - Functions
2. Algebraic structures
   - Groups
   - Group Homomorphism
   - Polynomials
   - Rings and Fields
3. Linear Algebra
   - Vector Spaces and Matrices
   - Linear Maps



Outline

1. Generalities on GAP
   - About GAP
   - Lists
   - Functions
2. Algebraic structures
   - Groups
   - Group Homomorphism
   - Polynomials
   - Rings and Fields
3. Linear Algebra
   - Vector Spaces and Matrices
   - Linear Maps



First step in GAP

A simple computation with GAP is as easy as one can imagine.

1. You type the problem just after the promp;
2. terminate it with a semicolon;
3. and then pass the problem to the program with the return key.

For example, to multiply the difference between 9 and 7 by the sum of 5 and 6, that is to calculate $(9-7)*(5+6)$, you type exactly this last sequence of symbols followed by ; and return .

```
gap> (9 - 7) * (5 + 6);
22
```

Besides these arithmetical operators there are comparison operators: $=, <>, <, <=, >$ and $>=$, results are boolean values.

Variables, Objects and Elements

Explanation

- An object denotes something which can be assigned to a variable.

Code

Variables, Objects and Elements

Explanation

- An object denotes something which can be assigned to a variable.
- To define a variable on GAP, we use the assignment operator :=.

Code

- gap> x:=[Factorial(6), 2];y:=[720, 2];;
  [720, 2]

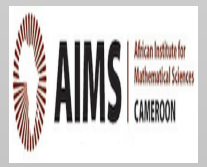
Variables, Objects and Elements

Explanation

- An object denotes something which can be assigned to a variable.
- To define a variable on GAP, we use the assignment operator :=.
- One can check if two variables are equals.

Code

- gap> x:=[Factorial(6), 2];y:=[720, 2];;
  [720, 2]
- gap> x=y;
  true



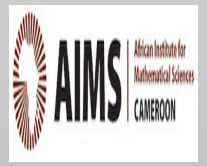
Variables, Objects and Elements

Explanation

- An object denotes something which can be assigned to a variable.
- To define a variable on GAP, we use the assignment operator :=.
- One can check if two variables are equals.
- One can check if two variables are *identical*. This means that, they point on the same object in the memory.

Code

- gap> x:=[Factorial(6), 2];y:=[720, 2];;
  [720, 2]
- gap> x=y;
  true
- gap> IsIdenticalObj(x,y);
  false



Definition of a Lists

Method

- A list is a collection of objects separated by commas and enclosed in brackets.

Code

- gap> primes:= [2, 3, 5, 7, 11, 13, 17, 19, 23, 29];
  [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]

Definition of a Lists

Method

- A list is a collection of objects separated by commas and enclosed in brackets.
- Function *Append* add at the end of a first list a second one.

Code

- gap> primes:= $[2, 3, 5, 7, 11, 13, 17, 19, 23, 29]$;
  $[2, 3, 5, 7, 11, 13, 17, 19, 23, 29]$
- gap> Append(primes, $[31, 37]$);
  gap> primes;
  $[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37]$

Definition of a Lists

Method

- A list is a collection of objects separated by commas and enclosed in brackets.
- Function *Append* add at the end of a first list a second one.
- Function *Add* add an element at the end of a list.

Code

- gap> primes:= [2, 3, 5, 7, 11, 13, 17, 19, 23, 29];
  [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
- gap> Append(primes, [31, 37]);
  gap> primes;
  [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37]
- gap> Add(primes, 41);
  gap> primes;
  [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41]

Operations on Lists

Method

- Ask the length of a list.

Code

- gap> Length(primes);
  13

Operations on Lists

Method

- Ask the length of a list.
- Ask an element from a list by given its position.
- Ask the position of an element.

Code

- gap> Length(primes);
  13
- gap> primes[7]; Position(primes, 17);
  17
  7

Operations on Lists

Method

- Ask the length of a list.
- Ask an element from a list by given its position.
- Ask the position of an element.
- Create an empty list.

Code

- gap> Length(primes);
  13
- gap> primes[7]; Position(primes, 17);
  17
  7
- gap> null:= [];
  [ ]

Operations on Lists

Method

- Ask the length of a list.
- Ask an element from a list by given its position.
- Ask the position of an element.
- Create an empty list.
- Add an element by using assignment.

Code

- gap> Length(primes);
  13
- gap> primes[7]; Position(primes, 17);
  17
  7
- gap> null:= [];
  [ ]
- gap> null[1]:= 14;;null[2]:=14;; null;
  [14, 14]

Operations on Lists

Method

- Ask the length of a list.
- Ask an element from a list by given its position.
- Ask the position of an element.
- Create an empty list.
- Add an element by using assignment.
- Empty positions in a list!

Code

- gap> Length(primes);
  13
- gap> primes[7]; Position(primes, 17);
  17
  7
- gap> null:= [];
  [ ]
- gap> null[1]:= 14;;null[2]:=14;; null;
  [14, 14]
- gap> null[6]:=4;; null[3]:= "string";;
  null;
  [14,14, "string",,,4]

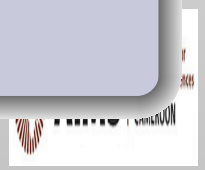
Sets

Method

- A set in GAP is a list that contains no holes (such a list is called dense) and whose elements are strictly sorted w.r.t. $<$; in particular, a set cannot contain duplicates.

Code

- gap> IsSSortedList(null);;
  false



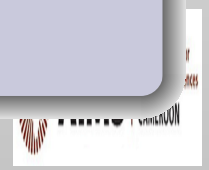
Sets

Method

- A set in GAP is a list that contains no holes (such a list is called dense) and whose elements are strictly sorted w.r.t. $<$; in particular, a set cannot contain duplicates.
- One can construct a set from a list by using function *Set*.

Code

- gap> IsSSortedList(null);;
  false
- gap> set:=Set(null);
  [4,14, "string"]
- gap> IsSSortedList(set);
  true



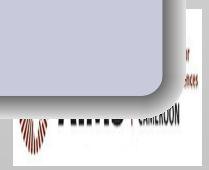
Sets

Method

- A set in GAP is a list that contains no holes (such a list is called dense) and whose elements are strictly sorted w.r.t. $<$; in particular, a set cannot contain duplicates.
- One can construct a set from a list by using function *Set*.
- The operator *in* is used to test whether an object is an element of a set or a list.

Code

- gap> IsSSortedList(null);;
  false
- gap> set:=Set(null);
  [4,14, "string"]
- gap> IsSSortedList(set);
  true
- gap> "plum" in set;
  false



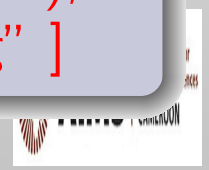
Sets

Method

- A set in GAP is a list that contains no holes (such a list is called dense) and whose elements are strictly sorted w.r.t. $<$; in particular, a set cannot contain duplicates.
- One can construct a set from a list by using function *Set*.
- The operator *in* is used to test whether an object is an element of a set or a list.
- Add an element into a set. This not change the initial set.

Code

- gap> IsSSortedList(null);;
  false
- gap> set:=Set(null);
  [4,14, "string"]
- gap> IsSSortedList(set);
  true
- gap> "plum" in set;
  false
- gap> AddSet(set,"plum");
  [4,14,"plum", "string"]



Gap Algorithms

Functions

- Functions are special GAP objects.

Code

Gap Algorithms

Functions

- Functions are special GAP objects.
- There are many function predefined in GAP.

Code

- gap> Gcd(720, 1254);
  6
- gap> Print(Gcd(720,1254));
  6

Gap Algorithms

Functions

- Functions are special GAP objects.
- There are many function predefined in GAP.
- One can define his function.

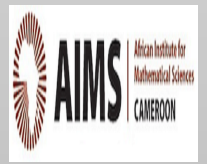
Code

- gap> Gcd(720, 1254);
  6
- gap> Print(Gcd(720,1254));
  6
- gap> cubed:=x− >x^3;
  function(x) ... end
- gap> cubed(2);
  8

if Statements

if statement

```
gap>    fib:=function(n)
   if n=0 then
      return 1;
   elif n=1 then
      return 1;
   else
      return fib(n-1)+fib(n-2);
   fi;
end;
function( n ) ... end
```



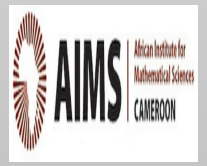
if Statements

if statement

```
gap>    fib:=function(n)
   if n=0 then
      return 1;
   elif n=1 then
      return 1;
   else
      return fib(n-1)+fib(n-2);
   fi;
end;
function( n ) ... end
```

Using his function

- gap> fib(1);
  1
- gap> fib(22);
  28657



while and for loops

while loop

```
gap>   primer:=function(n)
  local i, bool; i:=n-1; bool:=true;
  if n<2 then
    bool:=false;
  else
    while i >= 2 do
      if n mod i=0 then
        bool:=false;
        break;
      fi;
      i=i-1;
    od;
  fi;
  if bool then
    return true;
  else
    return false;
  fi;
end;
```

while and for loops

while loop

```
gap>   primer:=function(n)
  local i, bool; i:=n-1; bool:=true;
  if n<2 then
    bool:=false;
  else
    while i >= 2 do
      if n mod i=0 then
        bool:=false;
        break;
      fi;
      i=i-1;
    od;
  fi;
  if bool then
    return true;
  else
    return false;
  fi;
end;
```

for loop

```
gap>
  primerlessthan:=function(m)
  local n;
  if m<2 then
    Print("there is no primer
    number less than", m);
  else
    for n in [2..m] do
      if primer(n) then
        Print(n, ", ");
      fi;
    od;
  fi;
end;
function( m ) ... end
```

Operations on Lists

Let's set *gap* > L:=[1..9];

Explanation

- Use function *Product* to compute the product of elements of e list.

Code

- *gap* > Product(L);
  362880

Operations on Lists

Let's set *gap* > L:=[1..9];

Explanation

- Use function *Product* to compute the product of elements of e list.
- Use function *List* to apply a function on a List.

Code

- *gap* > Product(L);
  362880
- gap> List(L, fib);
  [1, 2, 3, 5, 8, 13, 21, 34, 55]

Operations on Lists

Let's set *gap* > L:=[1..9];

Explanation

- Use function *Product* to compute the product of elements of e list.
- Use function *List* to apply a function on a List.
- Use function *Filtered* to find specific elements from a list.

Code

- *gap* > Product(L);
  362880
- gap> List(L, fib);
  [1, 2, 3, 5, 8, 13, 21, 34, 55]
- gap> Filtered(L, primer);
  [2, 3, 5, 7]

Operations on Lists

Let's set *gap* > L:=[1..9];

Explanation

- Use function *Product* to compute the product of elements of e list.
- Use function *List* to apply a function on a List.
- Use function *Filtered* to find specific elements from a list.
- Ask to GAP if all the elements of a list satisfying a given condition.

Code

- *gap* > Product(L);
  362880
- gap> List(L, fib);
  [1, 2, 3, 5, 8, 13, 21, 34, 55]
- gap> Filtered(L, primer);
  [2, 3, 5, 7]
- gap> ForAll(L, primer);
  false
- gap> ForAll(L, x− > x > 0);
  true

Operations on Lists

Let's set *gap* > L:=[1..9];

Explanation

- Use function *Product* to compute the product of elements of e list.
- Use function *List* to apply a function on a List.
- Use function *Filtered* to find specific elements from a list.
- Ask to GAP if all the elements of a list satisfying a given condition.
- However, there is a function *Primes* in gap which gives the $n-$th primer number.

Code

- *gap* > Product(L);
  362880
- gap> List(L, fib);
  [1, 2, 3, 5, 8, 13, 21, 34, 55]
- gap> Filtered(L, primer);
  [2, 3, 5, 7]
- gap> ForAll(L, primer);
  false
- gap> ForAll(L, x− > x > 0);
  true
- gap> Primes[100];;

Operations on Lists

Let's set *gap* > L:=[1..9];

Explanation

- Use function *Product* to compute the product of elements of e list.
- Use function *List* to apply a function on a List.
- Use function *Filtered* to find specific elements from a list.
- Ask to GAP if all the elements of a list satisfying a given condition.
- However, there is a function *Primes* in gap which gives the $n-$th primer number.

Code

- *gap* > Product(L);
  362880
- gap> List(L, fib);
  [1, 2, 3, 5, 8, 13, 21, 34, 55]
- gap> Filtered(L, primer);
  [2, 3, 5, 7]
- gap> ForAll(L, primer);
  false
- gap> ForAll(L, x− > x > 0);
  true
- gap> Primes[100];;
- gap> Filtered(L, IsPrimeInt);
  [2, 3, 5, 7]

Outline

1. Generalities on GAP
   - About GAP
   - Lists
   - Functions
2. Algebraic structures
   - Groups
   - Group Homomorphism
   - Polynomials
   - Rings and Fields
3. Linear Algebra
   - Vector Spaces and Matrices
   - Linear Maps

Permutation groups

- *gap* > S3:=Group((1,2),(1,2,3));;
  S4:=Group((1,2), (1,2,3,4));
  Group([(1,2), (1,2,3,4)])
- gap>Size(S4)=Order(S4);
  true

Permutation groups

- *gap* > S3:=Group((1,2),(1,2,3));;
  S4:=Group((1,2), (1,2,3,4));
  Group([(1,2), (1,2,3,4)])
- gap>Size(S4)=Order(S4);
  true
- gap> IsAbelian(S4);
  false

Permutation groups

- *gap* > S3:=Group((1,2),(1,2,3));;
  S4:=Group((1,2), (1,2,3,4));
  Group([(1,2), (1,2,3,4)])
- gap>Size(S4)=Order(S4);
  true
- gap> IsAbelian(S4);
  false
- Center(S4);
  Group(())

Permutation groups

- *gap* > S3:=Group((1,2),(1,2,3));;
  S4:=Group((1,2), (1,2,3,4));
  Group([(1,2), (1,2,3,4)])
- gap>Size(S4)=Order(S4);
  true
- gap> IsAbelian(S4);
  false
- Center(S4);
  Group(())
- gap> Elements(S3);
  [(), (2,3), (1,2), (1,2,3), (1,3,2), (1,3)]

Permutation groups

- *gap* > S3:=Group((1,2),(1,2,3));;
  S4:=Group((1,2), (1,2,3,4));
  Group([(1,2), (1,2,3,4)])
- gap>Size(S4)=Order(S4);
  true
- gap> IsAbelian(S4);
  false
- Center(S4);
  Group(())
- gap> Elements(S3);
  [(), (2,3), (1,2), (1,2,3), (1,3,2), (1,3)]
- gap>(1,2,3)^-1;
  (1,3,2)

Permutation groups

- *gap* > S3:=Group((1,2),(1,2,3));;
  S4:=Group((1,2), (1,2,3,4));
  Group([(1,2), (1,2,3,4)])
- gap>Size(S4)=Order(S4);
  true
- gap> IsAbelian(S4);
  false
- Center(S4);
  Group(())
- gap> Elements(S3);
  [(), (2,3), (1,2), (1,2,3), (1,3,2), (1,3)]
- gap>(1,2,3)^-1;
  (1,3,2)
- gap> Centralizer(S4,(1,2));
  Group([(1,2), (3,4), (1,2)(3,4)])

Permutation groups

- *gap* > S3:=Group((1,2),(1,2,3));;
  S4:=Group((1,2), (1,2,3,4));
  Group([(1,2), (1,2,3,4)])
- gap>Size(S4)=Order(S4);
  true
- gap> IsAbelian(S4);
  false
- Center(S4);
  Group(())
- gap> Elements(S3);
  [(), (2,3), (1,2), (1,2,3), (1,3,2), (1,3)]
- gap>(1,2,3)^-1;
  (1,3,2)
- gap> Centralizer(S4,(1,2));
  Group([(1,2), (3,4), (1,2)(3,4)])
- gap>Normalizer(S4,Group((2,3,4)));
  Group([(2,3,4), (3,4)])
- gap> DerivedSubgroup(S4);
  Group([(1,2,3), (2,3,4)])

Permutation groups

- *gap* > S3:=Group((1,2),(1,2,3));;
  S4:=Group((1,2), (1,2,3,4));
  Group([(1,2), (1,2,3,4)])
- gap>Size(S4)=Order(S4);
  true
- gap> IsAbelian(S4);
  false
- Center(S4);
  Group(())
- gap> Elements(S3);
  [(), (2,3), (1,2), (1,2,3), (1,3,2), (1,3)]
- gap>(1,2,3)^-1;
  (1,3,2)
- gap> Centralizer(S4,(1,2));
  Group([(1,2), (3,4), (1,2)(3,4)])
- gap>Normalizer(S4,Group((2,3,4)));
  Group([(2,3,4), (3,4)])
- gap> DerivedSubgroup(S4);
  Group([(1,2,3), (2,3,4)])
- gap>
  IsSimple(DerivedSubgroup(S4));;
  IsSimple(DerivedSubgroup(S5));

Permutation groups

- *gap* > S3:=Group((1,2),(1,2,3));;
  S4:=Group((1,2), (1,2,3,4));
  Group([(1,2), (1,2,3,4)])
- gap>Size(S4)=Order(S4);
  true
- gap> IsAbelian(S4);
  false
- Center(S4);
  Group(())
- gap> Elements(S3);
  [(), (2,3), (1,2), (1,2,3), (1,3,2), (1,3)]
- gap>(1,2,3)^-1;
  (1,3,2)
- gap> Centralizer(S4,(1,2));
  Group([(1,2), (3,4), (1,2)(3,4)])
- gap>Normalizer(S4,Group((2,3,4)));
  Group([(2,3,4), (3,4)])
- gap> DerivedSubgroup(S4);
  Group([(1,2,3), (2,3,4)])
- gap>
  IsSimple(DerivedSubgroup(S4));;
  IsSimple(DerivedSubgroup(S5));
- gap> CommutatorSubgroup(S4,S3);
  Group([(1,3,2), (1,2,4)])

Permutation groups

- *gap* > S3:=Group((1,2),(1,2,3));;
  S4:=Group((1,2), (1,2,3,4));
  Group([(1,2), (1,2,3,4)])
- gap>Size(S4)=Order(S4);
  true
- gap> IsAbelian(S4);
  false
- Center(S4);
  Group(())
- gap> Elements(S3);
  [(), (2,3), (1,2), (1,2,3), (1,3,2), (1,3)]
- gap>(1,2,3)^-1;
  (1,3,2)
- gap> Centralizer(S4,(1,2));
  Group([(1,2), (3,4), (1,2)(3,4)])
- gap>Normalizer(S4,Group((2,3,4)));
  Group([(2,3,4), (3,4)])
- gap> DerivedSubgroup(S4);
  Group([(1,2,3), (2,3,4)])
- gap>
  IsSimple(DerivedSubgroup(S4));;
  IsSimple(DerivedSubgroup(S5));
- gap> CommutatorSubgroup(S4,S3);
  Group([(1,3,2), (1,2,4)])
- gap> SylowSubgroup(S4,2);
  Group([(3,4), (1,2)(3,4), (1,4)(2,3)])

Permutation groups

- *gap* > S3:=Group((1,2),(1,2,3));;
  S4:=Group((1,2), (1,2,3,4));
  Group([(1,2), (1,2,3,4)])
- gap>Size(S4)=Order(S4);
  true
- gap> IsAbelian(S4);
  false
- Center(S4);
  Group(())
- gap> Elements(S3);
  [(), (2,3), (1,2), (1,2,3), (1,3,2), (1,3)]
- gap>(1,2,3)^-1;
  (1,3,2)
- gap> Centralizer(S4,(1,2));
  Group([(1,2), (3,4), (1,2)(3,4)])
- gap>Normalizer(S4,Group((2,3,4)));
  Group([(2,3,4), (3,4)])
- gap> DerivedSubgroup(S4);
  Group([(1,2,3), (2,3,4)])
- gap>
  IsSimple(DerivedSubgroup(S4));;
  IsSimple(DerivedSubgroup(S5));
- gap> CommutatorSubgroup(S4,S3);
  Group([(1,3,2), (1,2,4)])
- gap> SylowSubgroup(S4,2);
  Group([(3,4), (1,2)(3,4),
  (1,4)(2,3)])
- gap> S41:=Subgroup(S4, [(1,2), (3,4)]);
  Group([(1,2), (3,4)])

Permutation groups

- *gap* > S3:=Group((1,2),(1,2,3));;
  S4:=Group((1,2), (1,2,3,4));
  Group([(1,2), (1,2,3,4)])
- gap>Size(S4)=Order(S4);
  true
- gap> IsAbelian(S4);
  false
- Center(S4);
  Group(())
- gap> Elements(S3);
  [(), (2,3), (1,2), (1,2,3), (1,3,2), (1,3)]
- gap>(1,2,3)^-1;
  (1,3,2)
- gap> Centralizer(S4,(1,2));
  Group([(1,2), (3,4), (1,2)(3,4)])
- gap>Normalizer(S4,Group((2,3,4)));
  Group([(2,3,4), (3,4)])
- gap> DerivedSubgroup(S4);
  Group([(1,2,3), (2,3,4)])
- gap>
  IsSimple(DerivedSubgroup(S4));;
  IsSimple(DerivedSubgroup(S5));
- gap> CommutatorSubgroup(S4,S3);
  Group([(1,3,2), (1,2,4)])
- gap> SylowSubgroup(S4,2);
  Group([(3,4), (1,2)(3,4),
  (1,4)(2,3)])
- gap> S41:=Subgroup(S4, [(1,2),
  (3,4)]);
  Group([(1,2), (3,4)])
- gap> S41=S4;
  false

Group Homomorphism

Even subgroup of S_4

- *gap* >
  hom:=GroupHomomorphismByImages(S4,S3,[(1,2),(1,2,3,4)],[(1,3),(1,2)]);
  [(1,2), (1,2,3,4)] − > [(1,3), (1,2)]
- gap> Kernel(hom);
  Group([(1,2)(3,4), (1,3)(2,4)])

Group Homomorphism

Even subgroup of S_4

- *gap* >
  hom:=GroupHomomorphismByImages(S4,S3,[(1,2),(1,2,3,4)],[(1,3),(1,2)]);
  [(1,2), (1,2,3,4)] − > [(1,3), (1,2)]
- gap> Kernel(hom);
  Group([(1,2)(3,4), (1,3)(2,4)])
- gap> Image(hom);
  Group([(1,3), (1,2)])

Group Homomorphism

Even subgroup of S_4

- *gap* >
  hom:=GroupHomomorphismByImages(S4,S3,[(1,2),(1,2,3,4)],[(1,3),(1,2)]);
  [(1,2), (1,2,3,4)] − > [(1,3), (1,2)]
- gap> Kernel(hom);
  Group([(1,2)(3,4), (1,3)(2,4)])
- gap> Image(hom);
  Group([(1,3), (1,2)])
- h:=GroupHomomorphismByFunction(S4,S2,
  >function(x) if SignPerm(x)=-1 then return (1,2); else return (); fi; end);;

Group Homomorphism

Even subgroup of S_4

- *gap* >
  hom:=GroupHomomorphismByImages(S4,S3,[(1,2),(1,2,3,4)],[(1,3),(1,2)]);
  [(1,2), (1,2,3,4)] − > [(1,3), (1,2)]
- gap> Kernel(hom);
  Group([(1,2)(3,4), (1,3)(2,4)])
- gap> Image(hom);
  Group([(1,3), (1,2)])
- h:=GroupHomomorphismByFunction(S4,S2,
  >function(x) if SignPerm(x)=-1 then return (1,2); else return (); fi; end);;
- gap> Kernel(h);
  Group([(2,3,4), (1,2)(3,4)])

Group Homomorphism

Even subgroup of S_4

- *gap* >
  hom:=GroupHomomorphismByImages(S4,S3,[(1,2),(1,2,3,4)],[(1,3),(1,2)]);
  [(1,2), (1,2,3,4)] − > [(1,3), (1,2)]
- gap> Kernel(hom);
  Group([(1,2)(3,4), (1,3)(2,4)])
- gap> Image(hom);
  Group([(1,3), (1,2)])
- h:=GroupHomomorphismByFunction(S4,S2,
  >function(x) if SignPerm(x)=-1 then return (1,2); else return (); fi; end);;
- gap> Kernel(h);
  Group([(2,3,4), (1,2)(3,4)])
- Aut:=InnerAutomorphism(S4,(1,2));;

Group Homomorphism

Even subgroup of S_4

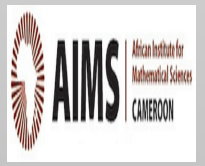
- *gap* >
  hom:=GroupHomomorphismByImages(S4,S3,[(1,2),(1,2,3,4)],[(1,3),(1,2)]);
  [(1,2), (1,2,3,4)] − > [(1,3), (1,2)]
- gap> Kernel(hom);
  Group([(1,2)(3,4), (1,3)(2,4)])
- gap> Image(hom);
  Group([(1,3), (1,2)])
- h:=GroupHomomorphismByFunction(S4,S2,
  >function(x) if SignPerm(x)=-1 then return (1,2); else return (); fi; end);;
- gap> Kernel(h);
  Group([(2,3,4), (1,2)(3,4)])
- Aut:=InnerAutomorphism(S4,(1,2));;
- gap> IsGroupHomomorphism(h);
  true

Polynomial

- *gap* > x:=X(Rationals, "x");;
- gap> IsPolynomial((x^2-1)/(x-2));;
  IsPolynomial((x^2-1)/(x-1));;

Polynomial

- *gap* > x:=X(Rationals, "x");;
- gap> IsPolynomial((x^2-1)/(x-2));;
  IsPolynomial((x^2-1)/(x-1));;
- gap> Value(x^2-1,2);
  3



Polynomial

- *gap* > x:=X(Rationals, "x");;
- gap> IsPolynomial((x^2-1)/(x-2));;
  IsPolynomial((x^2-1)/(x-1));;
- gap> Value(x^2-1,2);
  3
- Gcd(x^10-x,x^15-x);
  x^2-x
- gap> ShowGcd(x^10-x,x^15-x);;

Polynomial

- *gap* > x:=X(Rationals, "x");;
- gap> IsPolynomial((x^2-1)/(x-2));;
  IsPolynomial((x^2-1)/(x-1));;
- gap> Value(x^2-1,2);
  3
- Gcd(x^10-x,x^15-x);
  x^2-x
- gap> ShowGcd(x^10-x,x^15-x);;
- gap> Factors(x^2-1);
  [x-1, x+1]

- gap> RootsOfPolynomial(x^10-x);
  [1, 0]
- gap> RootsOfPolynomial(CF(3),x^10-x);
  [1, 0, E(3), E(3)^2]

Polynomial

- *gap* > x:=X(Rationals, "x");;
- gap> IsPolynomial((x^2-1)/(x-2));;
  IsPolynomial((x^2-1)/(x-1));;
- gap> Value(x^2-1,2);
  3
- Gcd(x^10-x,x^15-x);
  x^2-x
- gap> ShowGcd(x^10-x,x^15-x);;
- gap> Factors(x^2-1);
  [x-1, x+1]

- gap> RootsOfPolynomial(x^10-x);
  [1, 0]
- gap> RootsOfPolynomial(CF(3),x^10-x);
  [1, 0, E(3), E(3)^2]
- gap> Resultant(x^10-x,x+1,x);
  2

Rings and Fields

- gap> NumberSmallRings(8);
  52

Rings and Fields

- gap> NumberSmallRings(8);
  52
- gap> R:=SmallRing(8,40);;

Rings and Fields

- gap> NumberSmallRings(8);
  52
- gap> R:=SmallRing(8,40);;
- gap> Elements(R);
  [0*a, c, b, b+c, a, a+c, a+b, a+b+c]

Rings and Fields

- gap> NumberSmallRings(8);
  52
- gap> R:=SmallRing(8,40);;
- gap> Elements(R);
  [0*a, c, b, b+c, a, a+c, a+b, a+b+c]
- gap> AssignGeneratorVariables(R);;

Rings and Fields

- gap> NumberSmallRings(8);
  52
- gap> R:=SmallRing(8,40);;
- gap> Elements(R);
  [0*a, c, b, b+c, a, a+c, a+b, a+b+c]
- gap> AssignGeneratorVariables(R);;
- gap> ShowAdditionTable(R);;

Rings and Fields

- gap> NumberSmallRings(8);
  52
- gap> R:=SmallRing(8,40);;
- gap> Elements(R);
  [0*a, c, b, b+c, a, a+c, a+b, a+b+c]
- gap> AssignGeneratorVariables(R);;
- gap> ShowAdditionTable(R);;
- gap> S:=Subrings(R);;

Rings and Fields

- gap> NumberSmallRings(8);
  52
- gap> R:=SmallRing(8,40);;
- gap> Elements(R);
  [0*a, c, b, b+c, a, a+c, a+b, a+b+c]
- gap> AssignGeneratorVariables(R);;
- gap> ShowAdditionTable(R);;
- gap> S:=Subrings(R);;
- gap> List(S,GeneratorsOfRing);
  [[0*a], [a], [b], [a+b], [c], [a, b], [b, c], [a, c],[a+b, c], [a, b, c]]

Rings and Fields

- gap> NumberSmallRings(8);
  52
- gap> R:=SmallRing(8,40);;
- gap> Elements(R);
  [0*a, c, b, b+c, a, a+c, a+b, a+b+c]
- gap> AssignGeneratorVariables(R);;
- gap> ShowAdditionTable(R);;
- gap> S:=Subrings(R);;
- gap> List(S,GeneratorsOfRing);
  [[0*a], [a], [b], [a+b], [c], [a, b], [b, c], [a, c],[a+b, c], [a, b, c]]
- gap> I:=Ideals(R);

Rings and Fields

- gap> NumberSmallRings(8);
  52
- gap> R:=SmallRing(8,40);;
- gap> Elements(R);
  [0*a, c, b, b+c, a, a+c, a+b, a+b+c]
- gap> AssignGeneratorVariables(R);;
- gap> ShowAdditionTable(R);;
- gap> S:=Subrings(R);;
- gap> List(S,GeneratorsOfRing);
  [[0*a], [a], [b], [a+b], [c], [a, b], [b, c], [a, c],[a+b, c], [a, b, c]]
- gap> I:=Ideals(R);
- gap> Q:=R/I[4]
  <ring with 2 generators>

Rings and Fields

- gap> NumberSmallRings(8);
  52
- gap> R:=SmallRing(8,40);;
- gap> Elements(R);
  [0*a, c, b, b+c, a, a+c, a+b, a+b+c]
- gap> AssignGeneratorVariables(R);;
- gap> ShowAdditionTable(R);;
- gap> S:=Subrings(R);;
- gap> List(S,GeneratorsOfRing);
  [[0*a], [a], [b], [a+b], [c], [a, b], [b, c], [a, c],[a+b, c], [a, b, c]]
- gap> I:=Ideals(R);
- gap> Q:=R/I[4]
  <ring with 2 generators>

- gap> ShowMultiplicationTable(Q);;

Rings and Fields

- gap> NumberSmallRings(8);
  52
- gap> R:=SmallRing(8,40);;
- gap> Elements(R);
  [0*a, c, b, b+c, a, a+c, a+b, a+b+c]
- gap> AssignGeneratorVariables(R);;
- gap> ShowAdditionTable(R);;
- gap> S:=Subrings(R);;
- gap> List(S,GeneratorsOfRing);
  [[0*a], [a], [b], [a+b], [c], [a, b], [b, c], [a, c],[a+b, c], [a, b, c]]
- gap> I:=Ideals(R);
- gap> Q:=R/I[4]
  <ring with 2 generators>
- gap> ShowMultiplicationTable(Q);;

```
for i in [1..11] do
  ShowAdditionTable(SmallRing(4,i));
od;
```

Residue classes

Residue Class Rings

- *gap* > Z6:=ZmodnZ(6);
  (Integers mod 6)
- gap> fam:=ElementsFamily(FamilyObj(Z6));;

Residue classes

Residue Class Rings

- *gap* > Z6:=ZmodnZ(6);
  (Integers mod 6)
- gap> fam:=ElementsFamily(FamilyObj(Z6));;
- gap> ;;
- a;
  ZmodnZObj(5, 6)

Residue classes

Residue Class Rings

- *gap* > Z6:=ZmodnZ(6);
  (Integers mod 6)
- gap> fam:=ElementsFamily(FamilyObj(Z6));;
- gap> ;;
- a;
  ZmodnZObj(5, 6)
- gap> Int(a+a);;
- gap>Z13:=ZmodnZ(13);
  GF(13)

Residue classes

Residue Class Rings

- *gap* > Z6:=ZmodnZ(6);
  (Integers mod 6)
- gap> fam:=ElementsFamily(FamilyObj(Z6));;
- gap> ;;
- a;
  ZmodnZObj(5, 6)
- gap> Int(a+a);;
- gap>Z13:=ZmodnZ(13);
  GF(13)
- gap> ShowAdditionTable(ZmodnZ(5));
  ShowMultiplicationTable(ZmodnZ(5));

Finite Fields and Galois Theory

Finite Fields and Galois Theory

- gap> GF25:=GF(5,2);; One(GF25)
  Z(5)^0

Finite Fields and Galois Theory

Finite Fields and Galois Theory

- gap> GF25:=GF(5,2);; One(GF25)
  Z(5)^0
- gap> a:=3*One(GF25); b:=4*One(GF25);
  Z(5)^3
  Z(5)^2

Finite Fields and Galois Theory

Finite Fields and Galois Theory

- gap> GF25:=GF(5,2);; One(GF25)
  Z(5)^0
- gap> a:=3*One(GF25); b:=4*One(GF25);
  Z(5)^3
  Z(5)^2
- gap> Int(a); a+b; a*b;
  3
  Z(5)
  Z(5)

Finite Fields and Galois Theory

Finite Fields and Galois Theory

- gap> GF25:=GF(5,2);; One(GF25)
  Z(5)^0
- gap> a:=3*One(GF25); b:=4*One(GF25);
  Z(5)^3
  Z(5)^2
- gap> Int(a); a+b; a*b;
  3
  Z(5)
  Z(5)
- gap> f:=x^2-5;; Q5:=AlgebraicExtension(Rationals,f);
  <algebraic extension over the Rationals of degree 2>

Finite Fields and Galois Theory

Finite Fields and Galois Theory

- gap> GF25:=GF(5,2);; One(GF25)
  Z(5)^0
- gap> a:=3*One(GF25); b:=4*One(GF25);
  Z(5)^3
  Z(5)^2
- gap> Int(a); a+b; a*b;
  3
  Z(5)
  Z(5)
- gap> f:=x^2-5;; Q5:=AlgebraicExtension(Rationals,f);
  <algebraic extension over the Rationals of degree 2>
- gap>a:=RootOfDefiningPolynomial(Q5);; 2*a+a^3+a^2+5;
  7*a+10

Finite Fields and Galois Theory

Finite Fields and Galois Theory

- gap> GF25:=GF(5,2);; One(GF25)
  Z(5)^0
- gap> a:=3*One(GF25); b:=4*One(GF25);
  Z(5)^3
  Z(5)^2
- gap> Int(a); a+b; a*b;
  3
  Z(5)
  Z(5)
- gap> f:=x^2-5;; Q5:=AlgebraicExtension(Rationals,f);
  <algebraic extension over the Rationals of degree 2>
- gap>a:=RootOfDefiningPolynomial(Q5);; 2*a+a^3+a^2+5;
  7*a+10
- gap> GaloisGroup(GF25);
  <group with 1 generators>

Outline

1. Generalities on GAP
   - About GAP
   - Lists
   - Functions
2. Algebraic structures
   - Groups
   - Group Homomorphism
   - Polynomials
   - Rings and Fields
3. Linear Algebra
   - Vector Spaces and Matrices
   - Linear Maps

Vector Spaces and Matrices

Vector Spaces

- *gap* > F:=Rationals;;
- gap> V:= VectorSpace(F, [[-1, 1, 1], [1, 0, 2], [1,-1,0]]);
  <vector space over Rationals, with 3 generators>

Vector Spaces and Matrices

Vector Spaces

- *gap* > F:=Rationals;;
- gap> V:= VectorSpace(F, [[-1, 1, 1], [1, 0, 2], [1,-1,0]]);
  <vector space over Rationals, with 3 generators>
- gap> [1,0,3] in V;
  true

Vector Spaces and Matrices

Vector Spaces

- *gap* > F:=Rationals;;
- gap> V:= VectorSpace(F, [[-1, 1, 1], [1, 0, 2], [1,-1,0]]);
  <vector space over Rationals, with 3 generators>
- gap> [1,0,3] in V;
  true
- gap> B:=Basis(V,[[-1,1,1],[1,0,2],[1,-1,0]]);;
- gap> Coefficients(B, [1, 0, 3]);
  [1, 1, 1]

Vector Spaces and Matrices

Vector Spaces

- *gap* > F:=Rationals;;
- gap> V:= VectorSpace(F, [[-1, 1, 1], [1, 0, 2], [1,-1,0]]);
  <vector space over Rationals, with 3 generators>
- gap> [1,0,3] in V;
  true
- gap> B:=Basis(V,[[-1,1,1],[1,0,2],[1,-1,0]]);;
- gap> Coefficients(B, [1, 0, 3]);
  [1, 1, 1]
- gap> F^3;
  (Rationals^3)

Vector Spaces and Matrices

Vector Spaces

- *gap* > F:=Rationals;;
- gap> V:= VectorSpace(F, [[-1, 1, 1], [1, 0, 2], [1,-1,0]]);
  <vector space over Rationals, with 3 generators>
- gap> [1,0,3] in V;
  true
- gap> B:=Basis(V,[[-1,1,1],[1,0,2],[1,-1,0]]);;
- gap> Coefficients(B, [1, 0, 3]);
  [1, 1, 1]
- gap> F^3;
  (Rationals^3)

Vectors and Matrices

- gap> Vec:=[-2,1,3];;
  Mat:=[[1,-1,3], [2,0,-1], [2,1,3]];;

Vector Spaces and Matrices

Vector Spaces

- *gap* > F:=Rationals;;
- gap> V:= VectorSpace(F, [[-1, 1, 1], [1, 0, 2], [1,-1,0]]);
  <vector space over Rationals, with 3 generators>
- gap> [1,0,3] in V;
  true
- gap> B:=Basis(V,[[-1,1,1],[1,0,2],[1,-1,0]]);;
- gap> Coefficients(B, [1, 0, 3]);
  [1, 1, 1]
- gap> F^3;
  (Rationals^3)

Vectors and Matrices

- gap> Vec:=[-2,1,3];;
  Mat:=[[1,-1,3], [2,0,-1], [2,1,3]];;
- gap> Vec*Mat; Mat*Vec;
  [6, 5, 2]
  [6, -7, 6]

Vector Spaces and Matrices

Vector Spaces

- *gap* > F:=Rationals;;
- gap> V:= VectorSpace(F, [[-1, 1, 1], [1, 0, 2], [1,-1,0]]);
  <vector space over Rationals, with 3 generators>
- gap> [1,0,3] in V;
  true
- gap> B:=Basis(V,[[-1,1,1],[1,0,2],[1,-1,0]]);;
- gap> Coefficients(B, [1, 0, 3]);
  [1, 1, 1]
- gap> F^3;
  (Rationals^3)

Vectors and Matrices

- gap> Vec:=[-2,1,3];;
  Mat:=[[1,-1,3], [2,0,-1], [2,1,3]];;
- gap> Vec*Mat; Mat*Vec;
  [6, 5, 2]
  [6, -7, 6]
- gap> Vec*Vec;
  DeterminantMat(Mat)
  14
  15

Vector Spaces and Matrices

Vector Spaces

- *gap* > F:=Rationals;;
- gap> V:= VectorSpace(F, [[-1, 1, 1], [1, 0, 2], [1,-1,0]]);
  <vector space over Rationals, with 3 generators>
- gap> [1,0,3] in V;
  true
- gap> B:=Basis(V,[[-1,1,1],[1,0,2],[1,-1,0]]);;
- gap> Coefficients(B, [1, 0, 3]);
  [1, 1, 1]
- gap> F^3;
  (Rationals^3)

Vectors and Matrices

- gap> Vec:=[-2,1,3];;
  Mat:=[[1,-1,3], [2,0,-1], [2,1,3]];;
- gap> Vec*Mat; Mat*Vec;
  [6, 5, 2]
  [6, -7, 6]
- gap> Vec*Vec;
  DeterminantMat(Mat)
  14
  15
- gap> Display(Mat);
  [[1, -1, 3],
  [2, 0, -1],
  [2, 1, 3]]

Vector Spaces and Matrices

Vector Spaces

- *gap* > F:=Rationals;;
- gap> V:= VectorSpace(F, [[-1, 1, 1], [1, 0, 2], [1,-1,0]]);
  <vector space over Rationals, with 3 generators>
- gap> [1,0,3] in V;
  true
- gap> B:=Basis(V,[[-1,1,1],[1,0,2],[1,-1,0]]);;
- gap> Coefficients(B, [1, 0, 3]);
  [1, 1, 1]
- gap> F^3;
  (Rationals^3)

Vectors and Matrices

- gap> Vec:=[-2,1,3];;
  Mat:=[[1,-1,3], [2,0,-1], [2,1,3]];;
- gap> Vec*Mat; Mat*Vec;
  [6, 5, 2]
  [6, -7, 6]
- gap> Vec*Vec;
  DeterminantMat(Mat)
  14
  15
- gap> Display(Mat);
  [[1, -1, 3],
  [2, 0, -1],
  [2, 1, 3]]
- gap> Mat2:=[[-2,1,2],[0,2,1]];;
  Mat2*Mat; Mat*Mat2;
  [[4, 4, -1], [6, 1, 1]]

Using a Matrice in an algorithm

- Be careful when using a matrix in an algorithm.

Immutable matrix

- gap> row:=[1,0];; mat1:=[row,[0,1]];; mat2:=[row,[0,1]];;
- gap> mat1[2][1]:=3;; mat1; mat2;
  [[1, 0], [3, 1]]
  [[1, 0], [0, 1]]

Using a Matrice in an algorithm

- Be careful when using a matrix in an algorithm.

Immutable matrix

- gap> row:=[1,0];; mat1:=[row,[0,1]];; mat2:=[row,[0,1]];;
- gap> mat1[2][1]:=3;; mat1; mat2;
  [[1, 0], [3, 1]]
  [[1, 0], [0, 1]]
- gap> mat1[1][2]:=5;;mat1;mat2;
  [[1, 5], [3, 1]]
  [[1, 5], [0, 1]]

Using a Matrice in an algorithm

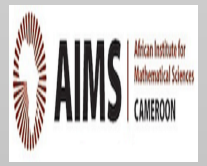
- Be careful when using a matrix in an algorithm.
- One can use *ShallowCopy*(*vector*) (Resp. *MutableCopyMat*(*matrix*)) to have a duplicated *mutable vector* (Resp. *matrix*).

Immutable matrix

- gap> row:=[1,0];; mat1:=[row,[0,1]];; mat2:=[row,[0,1]];;
- gap> mat1[2][1]:=3;; mat1; mat2;
  [[1, 0], [3, 1]]
  [[1, 0], [0, 1]]
- gap> mat1[1][2]:=5;;mat1;mat2;
  [[1, 5], [3, 1]]
  [[1, 5], [0, 1]]
- gap> mat3:=MutableCopyMat(mat1);; mat3[1][2]:=3;; mat3;mat2;
  [[1, 3], [3, 1]]
  [[1, 5], [0, 1]]

Linear Maps

- *gap* > V:=Rationals^2;;
  W:=Rationals^3;;



Linear Maps

- *gap* > V:=Rationals^2;;
  W:=Rationals^3;;
- *gap* >
  f:=LeftModuleHomomorphismByImages(V,W,
  [[1,0], [0,1]], [[1,1,1],[1,0,-1]]);;

Linear Maps

- *gap* > V:=Rationals^2;;
  W:=Rationals^3;;
- *gap* >
  f:=LeftModuleHomomorphismByImages(V,W,
  [[1,0], [0,1]], [[1,1,1],[1,0,-1]]);;
- *gap* > Image(f, [1,1]);
  [2, 1, 0]

Linear Maps

- *gap* > V:=Rationals^2;;
  W:=Rationals^3;;
- *gap* >
  f:=LeftModuleHomomorphismByImages(V,W,
  [[1,0], [0,1]], [[1,1,1],[1,0,-1]]);;
- *gap* > Image(f, [1,1]);
  [2, 1, 0]
- *gap* >
  h:=LeftModuleHomomorphismByMatrix(Basis(V),
  [[1,0,-1], [2,1,-1],[1,2,1]], Basis(V));
  <linear mapping by matrix, (
  Rationals^3) − > (Rationals^3)>
- *gap* > Image(h, [1,0,0]);
  [1, 0, -1]

Linear Maps

- *gap* > V:=Rationals^2;;
  W:=Rationals^3;;
- *gap* >
  f:=LeftModuleHomomorphismByImag
  [[1,0], [0,1]], [[1,1,1],[1,0,-1]]);;
- *gap* > Image(f, [1,1]);
  [2, 1, 0]
- *gap* >
  h:=LeftModuleHomomorphismByMat
  [[1,0,-1], [2,1,-1],[1,2,1]], Basis(V));
  <linear mapping by matrix, (
  Rationals^3) − > (Rationals^3)>
- *gap* > Image(h, [1,0,0]);
  [1, 0, -1]

- gap> Kernel(h); Image(h);
  <vector space over Rationals, with 1 generators>
  <vector space over Rationals, with 3 generators>

Linear Maps

- *gap* > V:=Rationals^2;;
  W:=Rationals^3;;
- *gap* >
  f:=LeftModuleHomomorphismByImag
  [[1,0], [0,1]], [[1,1,1],[1,0,-1]]);;
- *gap* > Image(f, [1,1]);
  [2, 1, 0]
- *gap* >
  h:=LeftModuleHomomorphismByMat
  [[1,0,-1], [2,1,-1],[1,2,1]], Basis(V));
  <linear mapping by matrix, (
  Rationals^3) − > (Rationals^3)>
- *gap* > Image(h, [1,0,0]);
  [1, 0, -1]

- gap> Kernel(h); Image(h);
  <vector space over Rationals, with 1 generators>
  <vector space over Rationals, with 3 generators>
- gap> W:=VectorSpace(Rationals, [[1,1,1]]);;
- gap>
  g:=NaturalHomomorphismBySubspace
  <linear mapping by matrix, (
  Rationals^3) − > (Rationals^2)>

Linear Maps

- *gap* > V:=Rationals^2;;
  W:=Rationals^3;;
- *gap* >
  f:=LeftModuleHomomorphismByImag
  [[1,0], [0,1]], [[1,1,1],[1,0,-1]]);;
- *gap* > Image(f, [1,1]);
  [2, 1, 0]
- *gap* >
  h:=LeftModuleHomomorphismByMat
  [[1,0,-1], [2,1,-1],[1,2,1]], Basis(V));
  <linear mapping by matrix, (
  Rationals^3) − > (Rationals^3)>
- *gap* > Image(h, [1,0,0]);
  [1, 0, -1]

- gap> Kernel(h); Image(h);
  <vector space over Rationals, with 1 generators>
  <vector space over Rationals, with 3 generators>
- gap> W:=VectorSpace(Rationals, [[1,1,1]]);;
- gap>
  g:=NaturalHomomorphismBySubspace
  <linear mapping by matrix, (
  Rationals^3) − > (Rationals^2)>
- gap> Image(g,[1,1,1]);
  [0,0]

Thanks for your kindness attention.