

CCTab

Constructing Character Tables

Version 0.4

October 2026

Frank Lübeck

Frank Lübeck Email: Frank.Luebeck@Math.RWTH-Aachen.De
Homepage: <http://www.math.rwth-aachen.de/~Frank.Luebeck>
Address: Lehrstuhl für Algebra und Zahlentheorie RWTH Aachen
Pontdriesch 14/16 52062 Aachen Germany

Copyright

© 2026- by Frank Lübeck

CCTab is free software; you can redistribute it and/or modify it under the terms of the [GNU General Public License](#) as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

Contents

1	The CCTab-Package	4
1.1	Installation of the CCTab package	4
2	The main functionality of CCTab	6
3	Utility functions from the CCTab package	7
3.1	Identifying conjugacy classes of elements	7
3.2	Power maps and rational classes	8
3.3	Induction from elementary subgroups	13
3.4	Some utilities for integer matrices	16
4	Computing character tables with CCTab	18
4.1	The IsCCTable (4.1.1) object	18
4.2	Scalar products	23
4.3	Splitting by the centre	23
4.4	Natural characters	25
	References	26
	Index	27

Chapter 1

The CCTab-Package

(Constructing (ordinary) character tables of finite groups)

1.1 Installation of the CCTab package

To install this package first unpack it inside some GAP root directory in the subdirectory `pkg` (see **(Reference: Installing a GAP Package)**). Then the CCTab package can already be loaded and used. But we strongly recommend to compile a kernel function and some standalone programs as well during installation. This will speed up some computations. The standalone programs assume that the library files and header files of the FLINT library are installed. If this is not yet the case, install them first via your operating systems package manager (package `libflint-dev` or similar), or directly from <https://flintlib.org/>. To install the extra functions go to the directory `pkg/CCTab-...` or `pkg/cctab/` to which the package was extracted and call

```
/bin/sh ./configure [--with-gaproot=path]
```

where `path` is a path to the main GAP root directory (if not given, the default `../..` is assumed).

If the FLINT library is not installed in a place where the compiler finds it by itself, and it is not known to `pkg-config` either, you can specify its location with

```
/bin/sh ./configure --with-gaproot=path --with-flint=prefix
```

where the header files are expected in `prefix/include` and the libraries in `prefix/lib`. Use `--without-flint` to skip the standalone programs altogether; the package then still works, but some computations are slower. Call `./configure --help` for the complete list of options.

The compiler and the compiler flags can be overridden, either in the environment or as arguments of `configure`, for example

```
/bin/sh ./configure CC=clang CFLAGS="-O2 -g"
```

Afterwards call `make` to compile the code. Individual flags can also be overridden at this point, e.g. by `make CFLAGS="-O2"`.

You can run a test of the installation by typing `make test`.

After installation the package can be used with `LoadPackage("CCTab");`.

To see some progress information during longer computations the following `info` class can be set to a non-zero level:

1.1.1 InfoCCTable

▷ InfoCCTable

(info class)

This is an Info class for the CCTab-package. By `SetInfoLevel(InfoCCTable, level)`; with `level` between 1 and 3 you can switch on the printing of some information during the computations of certain CCTab-functions.

The default level is 0.

Chapter 2

The main functionality of CCTab

This package can be useful for users who want to compute ordinary character tables of given finite groups.

By loading this package the default method for `Irr` (**Reference: `Irr`**) is overwritten for those groups for which the `GAP` library would use `IrrDixonSchneider` (**Reference: `IrrDixonSchneider`**).

If this is all you want, you do not need to know any further details about the functions in `CCTab`.

But the functions in `CCTab` can also be used interactively to construct character tables of groups which are not handled easily by the generic algorithm. In this case you may be interested in more details about how this package works.

There are also some functions, e.g., `PositionConjugacyClass` (3.1.1), `PowerMapsOfAllClasses` (3.2.1) or `RationalClassesInfo` (3.2.4) which may be of independent interest for other applications.

Chapter 3

Utility functions from the CCTab package

This package implements various utility functions which may also be of independent interest. For example, there are functions for identifying the conjugacy class of a given element, or for computing all power maps of a finite group, or for computing information about rational conjugacy classes. Furthermore there are utilities for certain integer matrices.

3.1 Identifying conjugacy classes of elements

3.1.1 PositionConjugacyClass

▷ `PositionConjugacyClass( $G$ ,  $x$ [,  $exclude$ ])` (function)

Returns: a positive integer

Let G be a finite group and x an element of G . This function returns the position of the conjugacy class of x in the list `ConjugacyClasses( $G$ )`. It uses `ConjugacyClassInvariants` (3.1.2) to identify the class by computing class invariants of x until the class is uniquely determined.

If the optional argument `exclude` is given, it must be a list of class positions which is guaranteed not to contain the class position of the class of x . This information can sometimes help to speed up the identification.

Example

```
gap> G := AlternatingGroup(5);;
gap> creps := List(ConjugacyClasses(G), Representative);;
gap> List(creps, x-> PositionConjugacyClass(G, x^Random(G)));
[ 1, 2, 3, 4, 5 ]
```

3.1.2 ConjugacyClassInvariants

▷ `ConjugacyClassInvariants( $G$ [,  $funcs$ ])` (function)

Returns: a record

Let G be a finite group. This function computes and returns a record r that is used by `PositionConjugacyClass` (3.1.1) to quickly identify the conjugacy class of an element of G . The result is cached in $G!.ConjugacyClassInvariants$, so it is computed only once.

The record r has components G , $classes$ (the result of `ConjugacyClasses` (**Reference: ConjugacyClasses attribute**) for G), $reps$ (their representatives), and $tree$, a recursively built decision

tree used to distinguish the conjugacy classes of G by successively computed class invariants (such as element order, cycle structure, or characteristic and minimal polynomial for matrix groups), falling back to explicit conjugacy tests if necessary.

The optional argument *funcs*, if given, is a list of functions $fu(r, tree)$ used to (further) build up the tree, overriding the default choices, which depend on whether G is a permutation group, a matrix group, or neither. See `CCInvFuncs.CycStruct` as an example of such a function.

In the following example, the first function compares with the stored class representatives; the second computes the cycle type and the third is only relevant for 5-cycles and does one conjugacy test against the representative of class 4.

Example

```
gap> G := AlternatingGroup(5);
gap> G := AlternatingGroup(5);
gap> ConjugacyClassInvariants(G);
rec( G := Alt( [ 1 .. 5 ] ),
  classes := [ ()^G, (1,2)(3,4)^G, (1,2,3)^G, (1,2,3,4,5)^G,
    (1,2,3,5,4)^G ],
  reps := [ (), (1,2)(3,4), (1,2,3), (1,2,3,4,5), (1,2,3,5,4) ],
  tree := [ [ 1 .. 5 ], function( r, x ) ... end,
    [ 1, 2, 3, 4, 5, fail ],
    [ 1, 2, 3, 4, 5,
      [ [ 1 .. 5 ], function( r, x ) ... end,
        [ [ ], [ ,, 1 ], [ , 1 ], [ 2 ] ],
        [ 1,
          [ [ 4, 5 ], function( r, x ) ... end, [ 4, 5 ],
            [ 4, 5 ] ], 3, 2 ] ] ] ] ] )
```

3.2 Power maps and rational classes

Let $x_1, \dots, x_k$ be conjugacy class representatives of a finite group G . The n -th power map of G is the list of k integers such that the i -th entry is j when x_i^n is conjugate to x_j . If x_i has order m then the power map of class i is a list of length m such that the n -th entry is j if x_i^{n-1} is conjugate to x_j .

Let x be an element of order m in a group G . The rational class of x is the union of the conjugacy classes of all elements which generate the same cyclic subgroup of G as x (that is, of all x^i with $\gcd(i, m) = 1$).

3.2.1 PowerMapsOfAllClasses

- ▷ `PowerMapsOfAllClasses(G)` (attribute)
- ▷ `PowerMapsOfAllClasses(tbl)` (attribute)

Returns: a list of lists of positive integers

Let G be a finite group and number its conjugacy classes as in `ConjugacyClasses(G)`. If x is a representative of class number i and has order m , then the i -th entry of `PowerMapsOfAllClasses(G)` is a list of length m whose j -th entry is the number of the class containing x^{j-1} , for $j = 1, \dots, m$ (so the first entry is always 1, corresponding to x^0).

In this way the result encodes the power maps of *all* conjugacy classes of G at once. The implementation tries to minimize the number of, sometimes expensive, calls to `PositionConjugacyClass` (3.1.1); the algorithm computes the orbit of the class of x under the action of the unit group of the integers modulo m .

In the second form `PowerMapsOfAllClasses(tbl)` the result is with respect to the ordering of conjugacy classes in the character table `tbl`.

Example

```
gap> G := AlternatingGroup(5);
gap> List(ConjugacyClasses(G), c-> Order(Representative(c)));
[ 1, 2, 3, 5, 5 ]
gap> PowerMapsOfAllClasses(G);
[ [ 1 ], [ 1, 2 ], [ 1, 3, 3 ], [ 1, 4, 5, 5, 4 ], [ 1, 5, 4, 4, 5 ] ]
gap> t := CharacterTable("M11");
CharacterTable( "M11" )
gap> PowerMapsOfAllClasses(t);
[ [ 1 ], [ 1, 2 ], [ 1, 3, 3 ], [ 1, 4, 2, 4 ], [ 1, 5, 5, 5, 5 ],
  [ 1, 6, 3, 2, 3, 6 ], [ 1, 7, 4, 7, 2, 8, 4, 8 ],
  [ 1, 8, 4, 8, 2, 7, 4, 7 ], [ 1, 9, 10, 9, 9, 9, 10, 10, 10, 9, 10 ]
  , [ 1, 10, 9, 10, 10, 10, 9, 9, 9, 10, 9 ] ]
```

Several other attributes and functions in this package are implemented using `PowerMapsOfAllClasses`.

3.2.2 RationalClassSets

▷ `RationalClassSets(G)` (attribute)

Returns: a list of lists of positive integers

Returns a list of lists, where each inner list is the set of numbers of the conjugacy classes of G whose representatives generate conjugate cyclic subgroups (i.e., the union of these classes is a single *rational class*). The outer list is sorted by increasing element order and then by increasing class length. The first entry of each inner list can be used as a canonical representative number for that rational class.

Example

```
gap> G := AlternatingGroup(5);
gap> List(ConjugacyClasses(G), c-> Order(Representative(c)));
[ 1, 2, 3, 5, 5 ]
gap> RationalClassSets(G);
[ [ 1 ], [ 2 ], [ 3 ], [ 4, 5 ] ]
```

3.2.3 NrRationalClasses

▷ `NrRationalClasses(G)` (attribute)

Returns: a non-negative integer

Returns the number of rational classes of the finite group G .

Example

```
gap> NrRationalClasses(AlternatingGroup(5));
4
```

3.2.4 RationalClassesInfo

▷ `RationalClassesInfo(G)` (attribute)

Returns: a list of records

Returns a list of records, one for each rational class of G (in the same order as `RationalClassSets(3.2.2)(G)`). This collects detailed information about the classes, used, for example, to encode character values in an `IsCCTable(4.1.1)` object via `EncodeForCCTable(4.1.2)`.

If the rational class consists of a single conjugacy class, say class number i , the record has the components:

classes
[i]

conductor
1

classlen
the length of class i

order
the order of the elements in class i

ind the position of the (rational) character value on this class in a character encoded by EncodeForCCTable (4.1.2)

exponents
[i , i]

Otherwise, that is if the rational class consists of several conjugacy classes, the record has the components:

classes
the list of numbers of the conjugacy classes in this rational class (see RationalClassSets (3.2.2))

classlen
the (common) length of these conjugacy classes

order
the (common) order of their elements

conductor
the smallest $k > 1$ such that the character values on this rational class are contained in $\mathbb{Q}$ extended by k -th roots of unity

ind the list of positions (of length $\text{Phi}(\text{conductor})$) in a character encoded by EncodeForCCTable (4.1.2) which describe the character values on this rational class

cpol
the coefficient list of the conductor-th cyclotomic polynomial

stabilizer
a list of generating invertible residue classes mod order which generate the stabilizer of the class under the unit group mod order

field
the field generated by the character values on this class

dim the degree of field over the rationals

exponents

[i , j , k_1 , ...]: if x is in class i then j is the class of x^{-1} , and x^{k_1} , ... are representatives of the other classes in this rational class (ordered as in `classes`)

ratvec

a sparse encoding of a vector related to the character values on this class, see below

For x in the field generated by $E(k)$ let $f(x)$ be the coefficient at 1 when x is written as a polynomial modulo the k -th cyclotomic polynomial. Let $e = E(k)$, i in $[0.. \text{order}-1]$, and let $x, x^{k_1}, \dots$ be representatives of the classes in the rational class of x . Then let l be the list with $l[i] = f(e^i + e^{i*k_1} + \dots)$. The component `ratvec` is a sparse representation of l : the list $[i_1, l[i_1], i_2, l[i_2], \dots]$, where indices with $l[i] = 0$ are left out. (This data structure is used by a fast function for computing scalar products of generalized characters.)

Example

```
gap> G := AlternatingGroup(5);;
gap> G := AlternatingGroup(5);;
gap> rci := RationalClassesInfo(G);;
gap> rci[2];
rec( classes := [ 2 ], classlen := 15, conductor := 1,
      exponents := [ 2, 2 ], ind := 2, order := 2 )
gap> rci[4];
rec( classes := [ 4, 5 ], classlen := 12, conductor := 5,
      cpol := [ 1, 1, 1, 1, 1 ], dim := 4, exponents := [ 4, 4, 2 ],
      field := NF(5,[ 1, 4 ]), ind := [ 4 .. 7 ], order := 5,
      ratvec := [ 0, 2, 2, -1, 4, -1 ], stabilizer := [ 1, 4 ] )
```

3.2.5 MaximalCyclics

▷ `MaximalCyclics( $G$ )`

(attribute)

Returns: a list of positive integers

Returns a list l of class numbers such that the representatives of the classes $i \in l$ generate representatives of the maximal cyclic subgroups (that is cyclic subgroups not contained in larger cyclic subgroups) of G , up to conjugacy. The returned numbers are always first entries of the lists in `RationalClassSets` (3.2.2).

Example

```
gap> MaximalCyclics(AlternatingGroup(5));
[ 4, 3, 2 ]
```

3.2.6 InduceAllFromCyclicSubgroup

▷ `InduceAllFromCyclicSubgroup( $G$ ,  $i$ )`

(function)

▷ `InduceAllFromCyclicSubgroup( $CCT$ ,  $i$ )`

(function)

Returns: a list of row vectors of cyclotomic numbers

In the first form G is a group. In the second form CCT is an `IsCCTable` (4.1.1) object of a group. Let i be the number of one of its conjugacy classes. This function returns the pairwise distinct characters of the group induced from the linear characters of the cyclic subgroup generated by a representative of class i (as lists of values, not as class function objects).

In the first form each value is the list of character values in the order of `ConjugacyClasses( $G$ )`.

In the second form each vector instead has one entry for every *rational* class of the group, in the order given by RationalClassSets (3.2.2) (i.e., conjugacy classes belonging to the same rational class share one entry).

Example

```
gap> G := AlternatingGroup(5);
gap> G := AlternatingGroup(5);
gap> T := CCTable(G);
gap> InduceAllFromCyclicSubgroup(G, 4);
[ [ 12, 0, 0, 2, 2 ], [ 12, 0, 0, E(5)+E(5)^4, E(5)^2+E(5)^3 ],
  [ 12, 0, 0, E(5)^2+E(5)^3, E(5)+E(5)^4 ] ]
gap> InduceAllFromCyclicSubgroup(T, 4);
[ [ 12, 0, 0, 2 ], [ 12, 0, 0, E(5)+E(5)^4 ],
  [ 12, 0, 0, E(5)^2+E(5)^3 ] ]
```

3.2.7 InducedFromAllMaximalCyclicSubgroups

▷ InducedFromAllMaximalCyclicSubgroups(*G*) (function)

Returns: a list of lists of cyclotomic numbers

Returns the set of all characters of *G* induced from irreducible characters of the maximal cyclic subgroups of *G*, computed via MaximalCyclics (3.2.5) and InduceAllFromCyclicSubgroup (3.2.6). These characters always span the vector space of class functions on *G*.

Example

```
gap> G := AlternatingGroup(5);
gap> G := AlternatingGroup(5);
gap> indcyc := InducedFromAllMaximalCyclicSubgroups(G);
[ [ 12, 0, 0, 2, 2 ], [ 12, 0, 0, E(5)^2+E(5)^3, E(5)+E(5)^4 ],
  [ 12, 0, 0, E(5)+E(5)^4, E(5)^2+E(5)^3 ], [ 20, 0, -1, 0, 0 ],
  [ 20, 0, 2, 0, 0 ], [ 30, -2, 0, 0, 0 ], [ 30, 2, 0, 0, 0 ] ]
gap> Rank(indcyc);
5
```

3.2.8 PowerMapCharacters

▷ PowerMapCharacters(*G*, *n*) (function)

Returns: a list of lists of integers

This function returns a list of class functions on the finite group *G* (as lists of values in the order of the conjugacy classes in *G*), associated to the map $g \mapsto g^n$ on *G*. For each rational class *C* of *G*, the class function has value $|G|/\gcd(|G|, n)$ on elements *x* with $x^n \in C$, and 0 everywhere else. According to a theorem of Frobenius (see [Ser77, Section 11.2]) these class functions are generalized characters (we sum the functions in the cited reference over all conjugacy classes in a rational class to get rational integer coefficients).

Only the non-zero class functions (for rational classes in the image of $g \mapsto g^n$) are returned.

Example

```
gap> G := AlternatingGroup(5);
gap> pmchars := PowerMapCharacters(G, 2);
[ [ 30, 30, 0, 0, 0 ], [ 0, 0, 30, 0, 0 ], [ 0, 0, 0, 30, 30 ] ]
gap> MatScalarProducts(Irr(G), pmchars);
[ [ 8, 2, 10, -6, -6 ], [ 10, 10, -10, 0, 0 ], [ 12, -12, 0, 6, 6 ] ]
```

3.2.9 SmallPowerMapCharacters

▷ `SmallPowerMapCharacters( $G$ )` (function)

Returns: a list of lists of integers

Applies `PowerMapCharacters` (3.2.8) for a selection of divisors n of $|G|$ that are large (so that the value $|G|/\gcd(|G|, n)$ occurring in `PowerMapCharacters` (3.2.8) is small but non-zero), one n associated to each element order occurring in G . Class functions with the same pattern of zero entries are reduced to the one with (componentwise) minimal non-zero values.

Example

```
gap> G := AlternatingGroup(5);
gap> G := AlternatingGroup(5);
gap> SmallPowerMapCharacters(G);
[ [ 0, 0, 0, 5, 5 ], [ 0, 0, 3, 0, 0 ], [ 0, 4, 0, 0, 0 ],
  [ 4, 0, 4, 4, 4 ], [ 3, 3, 0, 3, 3 ], [ 5, 5, 5, 0, 0 ] ]
```

And here is a utility function that makes use of `PowerMapsOfAllClasses` (3.2.1).

3.2.10 ActionAutomorphismsOnConjugacyClasses

▷ `ActionAutomorphismsOnConjugacyClasses( $G, A$ )` (function)

▷ `ActionAutomorphismsOnRationalClasses( $G, A$ )` (function)

Returns: a permutation group

Let A be a group of automorphisms of G (for example `AutomorphismGroup( $G$ )`). The first function returns the permutation group describing the induced action of A on the conjugacy classes of G (identified by their position in `ConjugacyClasses( $G$ )`). The generators of the result correspond to `GeneratorsOfGroup( $A$ )`.

The second function returns the permutation group for the induced action on the rational classes of G .

The first function finds the classes of the images of generators of maximal cyclic subgroups and deduces the rest from `PowerMapsOfAllClasses` (3.2.1) for G .

Example

```
gap> G := AlternatingGroup(5);
gap> A := AutomorphismGroup(G);
<group of size 120 with 3 generators>
gap> ActionAutomorphismsOnConjugacyClasses(G, A);
Group([ (4,5), (), () ])
gap> ActionAutomorphismsOnRationalClasses(G, A);
Group(() )
```

3.3 Induction from elementary subgroups

To compute the irreducible characters of a finite group G we are interested in the characters of G induced from p -elementary subgroups $\langle x \rangle \times P$, where p is a prime, $x \in G$ is of order not divisible by p , and P is a p -subgroup of the centralizer of x in G . An elementary subgroup of G is a p -elementary subgroup for some prime p . We are mainly interested in maximal elementary subgroups (those not contained in larger elementary subgroups).

3.3.1 MaximalNonCyclicElementarySubgroups

▷ `MaximalNonCyclicElementarySubgroups(G)` (attribute)

Returns: a list of pairs of positive integers

This function returns a list of pairs $[i, p]$, where i is the number of a conjugacy class in G and p is a prime. To each such pair we consider a subgroup $\langle x \rangle \times P$, where x is a representative of class i and P is a Sylow- p -subgroup of the centralizer of x . These subgroups form, up to conjugacy in G , the non-cyclic maximal elementary subgroups of G .

Brauer's theorem on the characterization of characters of G says that the full lattice of generalized characters of G (that is all integer linear combinations of irreducible characters) is spanned by the characters induced from the maximal elementary subgroups of G , that is the maximal cyclic subgroups (see `MaximalCyclics` (3.2.5)) and the subgroups described by this function. See [Isa76, Chapter 8] for more details.

Example

```
gap> G := SL(4,2);;
gap> MaximalNonCyclicElementarySubgroups(G);
[ [ 1, 3 ], [ 1, 2 ], [ 6, 2 ] ]
```

3.3.2 InducedFromElementary

▷ `InducedFromElementary(G, i, p[, opts...])` (function)

▷ `InducedFromElementary(CCT, i, p[, opts...])` (function)

Returns: a list of lists of cyclotomic numbers, or an integer

In the first form G is a group, in the second form CCT is an `IsCCTable` (4.1.1) object of a group G . Let i be an integer and p be a prime, such that x contained the i -th conjugacy class of G has order not divisible by p . We consider the p -elementary subgroup $E = \langle x \rangle \times S$ where S is a Sylow- p -subgroup of the centralizer of x .

This function returns the pairwise distinct characters of G induced from the irreducible characters of E (as lists of values, in the order of `ConjugacyClasses(G)`).

The following further optional arguments may be given, in any order:

"linear"

only the characters induced from the linear characters of E are computed

"nonlinear"

only the characters induced from the non-linear irreducible characters of E are computed

a positive integer *max*

if the number of conjugacy classes of E , say k , is larger than *max*, no characters are computed and k is returned instead (this can be used to skip elementary subgroups that would be too expensive to handle)

Example

```
gap> G := AlternatingGroup(5);;
gap> InducedFromElementary(G, 1, 2);
[ [ 15, -1, 0, 0, 0 ], [ 15, 3, 0, 0, 0 ] ]
```

The following two functions try to find the fusion with fewer class identifications, respectively provide an iterator for the irreducible characters of an elementary subgroup. This can be useful in cases where elementary subgroups have a huge number of conjugacy classes.

3.3.3 FusionElementaryCCTable

▷ FusionElementaryCCTable(CCT, i, p) (function)

Returns: a list of length 2

Let CCT be an IsCCTable (4.1.1) object of a group G . The arguments i and p describe an elementary subgroup $E = \langle x \rangle \times S$, as explained in InducedFromElementary (3.3.2).

This function computes the fusion of the conjugacy classes of E into the classes of G . The implementation tries to avoid class identifications and uses the power maps. The result is a list [fus, needed]. Here fus is a list with one entry of form

[j, [e1, k1, k2, ...], [e2, ...], ...]

for every class j of G representing a rational class (see RationalClassSets (3.2.2)) that occurs in the image of the fusion. Here $e1, e2, \dots$ are exponents and they are followed by numbers k such that the classes of $x^{e_i}y$ with y in class k of S fuse into class j of G .

The second entry needed is the set of numbers of classes of S occurring in fus.

Example

```
gap> G := AlternatingGroup(5);;
gap> FusionElementaryCCTable(CCTable(G), 1,2);
[[ [ 1, [ 0, 1 ] ], [ 2, [ 0, 2, 3, 4 ] ] ], [ 1, 2, 3, 4 ] ]
```

3.3.4 InductionDataFromElementaryCCTable

▷ InductionDataFromElementaryCCTable(CCT, i, p) (function)

Returns: a record

Let CCT, i and p be as for FusionElementaryCCTable (3.3.3), describing a p -elementary subgroup $E = \langle x \rangle \times S$ of the group G underlying CCT. This function provides an alternative, more memory efficient way to compute the characters of G induced from the irreducible characters of E than InducedFromElementary (3.3.2). This is useful in particular when S has a large number of conjugacy classes.

The irreducible characters of the p -group S are described via BaumClausenInfo (see IrrBaumClausen (**Reference:** IrrBaumClausen)), and the record returned by this function contains a component next, a function without arguments. Each call of .next() returns $|x|$ induced characters from E ; or fail when there are no more such characters.

In contrast to InducedFromElementary (3.3.2), this function does not detect and remove induced characters occurring several times.

Example

```
gap> G := SymmetricGroup(5);;
gap> ind := InductionDataFromElementaryCCTable(CCTable(G), 1,2);;
gap> ind.next();
[[ 15, 3, 3, 0, 1, 0, 0 ] ]
gap> ind.next();
[[ 15, 3, -1, 0, -1, 0, 0 ] ]
gap> ind.next();
[[ 15, -3, 3, 0, -1, 0, 0 ] ]
gap> ind.next();
[[ 15, -3, -1, 0, 1, 0, 0 ] ]
gap> ind.next();
[[ 30, 0, -2, 0, 0, 0, 0 ] ]
gap> ind.next();
fail
```

3.4 Some utilities for integer matrices

This package uses a variant of the LLL algorithm to find the irreducible characters from another integral basis of the lattice of generalized characters. And there are functions to compute Hermite normal forms and inverses of unimodular matrices.

There are variants of these functions written in **GAP**. But we also provide variants which use standalone programs which need to be compiled during installation. These depend on the **FLINT** library, see [FLI]. These standalone programs use several threads. The number of these threads is configured in the variable `NRFlintThreads`, it defaults to 8 (but you can change it, depending on the number of cores of your machine).

3.4.1 HermiteIntMat

▷ `HermiteIntMat(mat)` (function)

Returns: an integer matrix

Returns the Hermite normal form of the integer matrix *mat*. The **GAP** variant is just `HermiteNormalFormIntegerMat` (**Reference:** `HermiteNormalFormIntegerMat`), the standalone variant uses a translation of DoNFIM to FLINT.

Example

```
gap> HermiteIntMat(RandomUnimodularMat(4));
[ [ 1, 0, 0, 0 ], [ 0, 1, 0, 0 ], [ 0, 0, 1, 0 ], [ 0, 0, 0, 1 ] ]
```

3.4.2 PermutedFractionFreeIntegerGaussPositiveDefinite

▷ `PermutedFractionFreeIntegerGaussPositiveDefinite(A)` (function)

Returns: a list [B, perm]

Here *A* must be a square, symmetric, positive definite integer matrix. This function computes a fraction free Gauss elimination of *A*, together with a permutation *perm* of the rows and columns of *A*, such that the diagonal entries of the resulting matrix *B* are minimal (the *i*-th diagonal entry of *B* is the determinant of the upper left $i \times i$ submatrix of the permuted *A*, that is, its *i*-th principal minor).

Remark: The matrix *B* is the transpose of the μ -matrix in classical LLL, with its *i*-th row multiplied by the *i*-th principal minor d_i .

Example

```
gap> A := [ [ 9, -3, 1, -2 ], [ -3, 3, 1, 1 ], [ 1, 1, 13, -2 ],
> [ -2, 1, -2, 5 ] ];;
gap> PermutedFractionFreeIntegerGaussPositiveDefinite(A);
[ [ [ 3, 1, -3, 1 ], [ 0, 14, -3, -7 ], [ 0, 0, 81, 21 ],
[ 0, 0, 0, 900 ] ], (1,3,4,2) ]
```

3.4.3 InverseUnimodularMat

▷ `InverseUnimodularMat(A)` (function)

Returns: an integer matrix

Here *A* must be a unimodular integer matrix (that is, a square integer matrix with determinant ± 1). This function returns the inverse of *A*, which again is an integer matrix.

This is a simplified variant of `InverseRatMat` (**EDIM:** `InverseRatMat`), based on a *p*-adic lifting approach.

Example

```
gap> A := RandomUnimodularMat(5);;
gap> A := RandomUnimodularMat(5);;
gap> A * InverseUnimodularMat(A);
[ [ 1, 0, 0, 0, 0 ], [ 0, 1, 0, 0, 0 ], [ 0, 0, 1, 0, 0 ],
  [ 0, 0, 0, 1, 0 ], [ 0, 0, 0, 0, 1 ] ]
```

3.4.4 LLLTransformUnimodularGram

▷ LLLTransformUnimodularGram(*gr*) (function)

Returns: a unimodular integer matrix

Here *gr* must be a symmetric, positive definite integer matrix (a Gram matrix). This function returns a unimodular integer matrix *H* such that $H * gr * \text{TransposedMat}(H)$ is the identity matrix.

The algorithm is similar to the modular variant of the LLL-algorithm described in [Sto96], but here the main loop is parallelized.

Example

```
gap> A := RandomUnimodularMat(5);;
gap> A := RandomUnimodularMat(5);;
gap> gr := A * TransposedMat(A);;
gap> h := LLLTransformUnimodularGram(gr);;
gap> h * gr * TransposedMat(h);
[ [ 1, 0, 0, 0, 0 ], [ 0, 1, 0, 0, 0 ], [ 0, 0, 1, 0, 0 ],
  [ 0, 0, 0, 1, 0 ], [ 0, 0, 0, 0, 1 ] ]
```

Chapter 4

Computing character tables with CCTab

An `IsCCTable` (4.1.1) object is the central data structure used by this package to incrementally collect information about the generalized and irreducible characters of a finite group, using an induce-reduce algorithm.

4.1 The `IsCCTable` (4.1.1) object

4.1.1 `CCTable`

- ▷ `CCTable( $G$ )` (attribute)
 Returns: an `IsCCTable` (4.1.1) object
- ▷ `IsCCTable( $CCT$ )` (filter)

For a finite group G `CCTable` returns an (initially empty) *partial character table container* object, in the filter `IsCCTable`. Such an object stores incrementally growing information about the (still unknown) generalized and irreducible characters of G , and is used as the central data structure for computing the irreducible characters of G with the tools in the `CCTab` package. The group G is stored as `UnderlyingGroup( $CCT$ )` in the result CCT . Many attributes of G (for example `RationalClassesInfo` (3.2.4)) can also be called with CCT (and delegate to G).

Generalized characters are stored in an `IsCCTable` object as integer vectors (for each rational class only the value on the representing conjugacy class is stored by `Phi(c)` integers where c is the conductor of that rational class). See `EncodeForCCTable` (4.1.2) below for a more precise description and for computing this representation of a generalized character from a class function object.

Generalized characters can be added to an `IsCCTable` object with `ImportToCCTable` (4.1.3). The lattice spanned by the known generalized characters is stored via the Hermite normal form of the characters, encoded as integer vectors (in a component `!.hnfs`).

If the group G has a non-trivial center it can be useful to call `SplitByCentre` (4.3.1) with CCT . This is not done by default because this may cause some non-negligible computations with conjugacy classes.

The default method for `Irr` (**Reference:** `Irr`) for an `IsCCTable` object computes the full set of irreducible characters of G . If the `CCTab` package is loaded then for many groups G its `CharacterTable( $G$ )` will also have an attribute `CCTable` and delegate the computation of the irreducible characters to this `CCTable`.

Example

```

gap> G := AlternatingGroup(5);;
gap> G := AlternatingGroup(5);;
gap> CCT := CCTable(G);
CCTable( Alt( [ 1 .. 5 ] ) )
gap> IsCCTable(CCT);
true
gap> Irr(CCT);
[ [ 1, 1, 1, 1, 0, 0, 0 ], [ 3, -1, 0, 0, 0, -1, -1 ],
  [ 3, -1, 0, 1, 0, 1, 1 ], [ 4, 0, 1, -1, 0, 0, 0 ],
  [ 5, 1, -1, 0, 0, 0, 0 ] ]
gap> ExpandFromCCTable(CCT, Irr(CCT));
[ [ 1, 1, 1, 1, 1 ], [ 3, -1, 0, -E(5)^2-E(5)^3, -E(5)-E(5)^4 ],
  [ 3, -1, 0, -E(5)-E(5)^4, -E(5)^2-E(5)^3 ], [ 4, 0, 1, -1, -1 ],
  [ 5, 1, -1, 0, 0 ] ]

```

4.1.2 EncodeForCCTable

▷ `EncodeForCCTable(CCT, l)` (function)

Returns: an integer vector, or a list of integer vectors

▷ `ExpandFromCCTable(CCT[, enc])` (function)

Returns: a character, or a list of characters

Let *CCT* be an `IsCCTable` (4.1.1) object. The argument *l* is either a single character, or a list of characters, of the underlying group of *CCT*, each given either as a list of values (in the order of the `ConjugacyClasses` (**Reference: ConjugacyClasses attribute**) of the underlying group) or as a class function object (with values in the order of `IdentificationOfConjugacyClasses` (**Reference: IdentificationOfConjugacyClasses**)) of its character table.

`EncodeForCCTable` returns the encoded form in which characters are stored in the `IsCCTable` (4.1.1) object: for each rational class (see `RationalClassesInfo` (3.2.4)) with conductor 1 this is simply the integer value on that class; for a rational class with conductor $c > 1$ the value on that class is a cyclotomic integer in the c -th cyclotomic field, and it is encoded by the (integer) coefficients of its representation as a polynomial in $E(c)$ modulo the c -th cyclotomic polynomial. The entries of `RationalClassesInfo(CCT)` have a component `.ind` telling the positions of these integer coefficients. If *l* is a single character, a single integer vector is returned, otherwise a list of vectors, one for each character in *l*.

`ExpandFromCCTable` is the converse function. If the optional argument *enc* is not given, all irreducible characters of *CCT* found so far are expanded.

Example

```

gap> G := AlternatingGroup(5);;
gap> G := AlternatingGroup(5);;
gap> CCT := CCTable(G);;
gap> ind := InducedFromAllMaximalCyclicSubgroups(G);;
gap> enc := EncodeForCCTable(CCT, ind);
[ [ 12, 0, 0, 2, 0, 0, 0 ], [ 12, 0, 0, 0, 0, 1, 1 ],
  [ 12, 0, 0, -1, 0, -1, -1 ], [ 20, 0, -1, 0, 0, 0, 0 ],
  [ 20, 0, 2, 0, 0, 0, 0 ], [ 30, -2, 0, 0, 0, 0, 0 ],
  [ 30, 2, 0, 0, 0, 0, 0 ] ]
gap> ExpandFromCCTable(CCT);
[ ]
gap> ExpandFromCCTable(CCT, enc);

```

```
[ [ 12, 0, 0, 2, 2 ], [ 12, 0, 0, E(5)^2+E(5)^3, E(5)+E(5)^4 ],
  [ 12, 0, 0, E(5)+E(5)^4, E(5)^2+E(5)^3 ], [ 20, 0, -1, 0, 0 ],
  [ 20, 0, 2, 0, 0 ], [ 30, -2, 0, 0, 0 ], [ 30, 2, 0, 0, 0 ] ]
```

4.1.3 ImportToCCTable

- ▷ ImportToCCTable(*CCT*, *chs*) (function)
- ▷ ImportEncodedToCCTable(*CCT*, *enc*) (function)

Let *CCT* be an IsCCTable (4.1.1) object, and let *chs* be a generalized character, or list of generalized characters, of the underlying group, given as for EncodeForCCTable (4.1.2) (that is, *not* yet encoded). ImportToCCTable encodes *chs* via EncodeForCCTable (4.1.2) and imports the result into *CCT* via ImportEncodedToCCTable.

ImportEncodedToCCTable imports *enc* into *CCT*: the (encoded) projections of already known irreducible characters of *CCT* are subtracted, and the remaining vectors are added to the list(s) of newly found generalized characters stored in *CCT* (split according to the splitting centre of *CCT*, see SplitByCentre (4.3.1), if that has been set). This does not yet update the Hermite normal forms describing the lattice(s) of found characters, see UpdateHNFCCTable (4.1.4).

Example

```
gap> G := AlternatingGroup(5);;
gap> ind := InducedFromAllMaximalCyclicSubgroups(G);;
gap> CCT := CCTable(G);;
gap> ImportToCCTable(CCT, ind);
gap> UpdateHNFCCTable(CCT);
0
gap> HasFullRankCCTable(CCT);
true
```

4.1.4 UpdateHNFCCTable

- ▷ UpdateHNFCCTable(*CCT*) (function)
- Returns:** a non-negative integer

Let *CCT* be an IsCCTable (4.1.1) object. This function incorporates the generalized characters recently imported via ImportEncodedToCCTable (4.1.3) (or ImportToCCTable (4.1.3)) into the Hermite normal form basis (or bases, in case of a splitting centre, see SplitByCentre (4.3.1)) of the lattice of generalized characters of *CCT* found so far, using AddVectorToHNF (4.1.10).

If HasFullRankCCTable (4.1.5) is true for *CCT*, the return value is the index by which the lattice found so far has shrunk (that is, the product of all indices returned by the individual calls to AddVectorToHNF (4.1.10)); otherwise the Hermite normal forms are simply recomputed from scratch and 0 is returned.

4.1.5 HasFullRankCCTable

- ▷ HasFullRankCCTable(*CCT*) (function)
- Returns:** true or false

Let *CCT* be an IsCCTable (4.1.1) object. This function returns true if the total number of characters currently collected in the Hermite normal form data of *CCT* (see UpdateHNFCCTable (4.1.4)), together with the irreducible characters already found, equals the number of conjugacy classes of the

underlying group, that is if the lattice of characters found so far already has full rank (the index in the full lattice of generalized characters is finite).

For example, the induced characters from all (maximal) cyclic subgroups always span a lattice of full rank, see the example for `ImportToCCTable` (4.1.3).

4.1.6 FindIndexCCTable

▷ `FindIndexCCTable(CCT)` (function)

Returns: a positive integer, or fail

Let `CCT` be an `IsCCTable` (4.1.1) object. If `HasFullRankCCTable(CCT) = true` this function returns the index of the lattice spanned by the class functions stored in `CCT` in the full lattice spanned by the irreducible characters of the underlying group. Otherwise, fail is returned.

Example

```
gap> G := AlternatingGroup(5);;
gap> G := AlternatingGroup(5);;
gap> CCT := CCTable(G);;
gap> ImportInducedFromAllMaximalCyclicToCCTable(CCT);
gap> FindIndexCCTable(CCT);
2
gap> ImportToCCTable(CCT, SmallPowerMapCharacters(G));
gap> UpdateHNFCCTable(CCT);
2
gap> FindIrreduciblesInFullLattices(CCT);
gap> CCT!.irr;
[ [ 1, 1, 1, 1, 0, 0, 0 ], [ 3, -1, 0, 0, 0, -1, -1 ],
  [ 3, -1, 0, 1, 0, 1, 1 ], [ 4, 0, 1, -1, 0, 0, 0 ],
  [ 5, 1, -1, 0, 0, 0, 0 ] ]
```

4.1.7 ImportInducedFromAllMaximalCyclicToCCTable

▷ `ImportInducedFromAllMaximalCyclicToCCTable(CCT)` (function)

Let `CCT` be an `IsCCTable` (4.1.1) object of a group G . Unless this has already been done (recorded in a component of `CCT`), this function computes the characters of G induced from all maximal cyclic subgroups of G (via `InducedFromAllMaximalCyclicSubgroups` (3.2.7)), imports them into `CCT`, and updates its Hermite normal form data. It also computes and stores, for each part of the (possibly split) lattice, the index of the resulting lattice of generalized characters in the full lattice of generalized characters, using that this index is only divisible by primes p for which the Sylow p -subgroups of G are not cyclic. The algorithm makes use of `ElementaryDivisorsPPartRk` (**EDIM: ElementaryDivisorsPPartRk**).

Example

```
gap> G := MathieuGroup(24);;
gap> CCT := CCTable(G);;
gap> ImportInducedFromAllMaximalCyclicToCCTable(CCT);
gap> FindIndexCCTable(CCT);
4718592
gap> StringCollectedFactors(FindIndexCCTable(CCT));
"2^19 3^2"
```

4.1.8 FindIrreduciblesInFullLattices

▷ FindIrreduciblesInFullLattices(*CCT*) (function)

Let *CCT* be an IsCCTable (4.1.1) object. For every orthogonal subspace in the HNF data of *CCT* which contains an integer basis of the corresponding sublattice this function computes the irreducible characters in that sublattice using a variant of the LLL algorithm (see LLLTransformUnimodularGram (3.4.4)). The irreducible characters are then moved into the list of known irreducibles (*CCT*!.irr) and this part of the HNF data is cleared. If SplitByCentre (4.3.1) was used, the function also adds Galois conjugates of the found irreducibles.

See FindIndexCCTable (4.1.6) for an example.

4.1.9 IrrCCTableHNF

▷ IrrCCTableHNF(*CCT*) (function)

Returns: a list of irreducible characters, or fail

Let *CCT* be an IsCCTable (4.1.1) object of a group *G*. This function implements an induce-reduce algorithm that computes and returns the list of irreducible characters of *G*, and is installed as the default method for Irr (**Reference:** Irr) for IsCCTable (4.1.1) objects.

It successively imports into *CCT* (via ImportToCCTable (4.1.3) and UpdateHNFCTable (4.1.4)) the characters induced from all maximal cyclic subgroups (via ImportInducedFromAllMaximalCyclicToCCTable (4.1.7)), the trivial and NaturalCharacters (4.4.1) of *G*, cheap generalized characters obtained from power maps (via SmallPowerMapCharacters (3.2.9)), and, if still needed, the characters induced from the maximal non-cyclic elementary subgroups of *G* (via MaximalNonCyclicElementarySubgroups (3.3.1) and InductionDataFromElementaryCCTable (3.3.4)), stopping as soon as FindIndexCCTable (4.1.6) returns 1. It then calls FindIrreduciblesInFullLattices (4.1.8) to extract the irreducible characters from the completed lattice.

Example

```
gap> G := AlternatingGroup(5);;
gap> G := AlternatingGroup(5);;
gap> CCT := CCTable(G);;
gap> IrrCCTableHNF(CCT);
[ [ 1, 1, 1, 1, 0, 0, 0 ], [ 3, -1, 0, 0, 0, -1, -1 ],
  [ 3, -1, 0, 1, 0, 1, 1 ], [ 4, 0, 1, -1, 0, 0, 0 ],
  [ 5, 1, -1, 0, 0, 0, 0 ] ]
```

Finally, we document a utility function for adjusting the Hermite normal form of an integer by adding a further vector to the lattices spanned by its rows.

4.1.10 AddVectorToHNF

▷ AddVectorToHNF(*hnf*, *piv*, *v*) (function)

Returns: a positive integer

Here *hnf* is an integer matrix in Hermite normal form, already of the (full) rank of the ambient lattice, *piv* is the list of its pivot column positions, and *v* is a further integer vector in the rational span of the rows of *hnf*. This function changes *hnf* in place to a Hermite normal form of the lattice generated by the old rows of *hnf* together with *v* (which, since the rank does not change, still has the

same number of rows), and returns the index of the old lattice of *hnf* in this new lattice (which is 1 if *v* was already contained in the old lattice). This is used by `UpdateHNFCCTable` (4.1.4).

Example

```
gap> hnf := [ [ 1, 0, 1, 478, -2, -649 ], [ 0, 1, 1, 362, 2, -492 ],
> [ 0, 0, 0, 546, 0, -742 ] ];
gap> piv := [1,2,4];
gap> v := [3, 7, 1, 6, 0, -1];
gap> AddVectorToHNF(hnf, piv, v);
39
gap> hnf;
[ [ 1, 0, 1, 2, -1090, 779 ], [ 0, 1, 1, 12, -798, 558 ],
[ 0, 0, 0, 14, 32, -42 ] ]
```

4.2 Scalar products

4.2.1 CCTScalarProduct

▷ `CCTScalarProduct(CCT, c, d)` (function)

Returns: an integer

Let *CCT* be an `IsCCTable` (4.1.1) object, and let *c* and *d* be two generalized characters of *CCT*, encoded as integer vectors as produced by `EncodeForCCTable` (4.1.2). This function returns the scalar product of *c* and *d*.

This is used as the method for `ScalarProduct` (**Reference: `ScalarProduct` for two vector objects**) for characters of an `IsCCTable` (4.1.1) object. The `CCTab` also provides faster implementations of this function as kernel extension. (It can be checked if this is installed with `IsBound(CCTScalarProductInternal2)`).

Note that this function cannot be used for arbitrary class functions, it only works for class functions which are rational linear combinations of irreducible characters.

Example

```
gap> G := AlternatingGroup(5);
gap> G := AlternatingGroup(5);
gap> CCT := CCTable(G);
gap> irr := Irr(CCT);
[ [ 1, 1, 1, 1, 0, 0, 0 ], [ 3, -1, 0, 0, 0, -1, -1 ],
[ 3, -1, 0, 1, 0, 1, 1 ], [ 4, 0, 1, -1, 0, 0, 0 ],
[ 5, 1, -1, 0, 0, 0, 0 ] ]
gap> List(irr, c->List(irr, d->ScalarProduct(CCT, c, d)));
[ [ 1, 0, 0, 0, 0 ], [ 0, 1, 0, 0, 0 ], [ 0, 0, 1, 0, 0 ],
[ 0, 0, 0, 1, 0 ], [ 0, 0, 0, 0, 1 ] ]
```

4.3 Splitting by the centre

4.3.1 SplitByCentre

▷ `SplitByCentre(CCT)` (operation)

▷ `SplitByCentre(CCT, zind)` (method)

▷ `HasSplittingCentre(CCT)` (filter)

▷ `SplittingCentre(CCT)` (attribute)

Let *CCT* be an `IsCCTable` (4.1.1) object of a group *G*. In the second form, *zind* must be a list of numbers of conjugacy classes of *G* forming a subgroup *Z* of the center of *G* (with the class of the identity first). In the first form, *zind* is computed as the set of classes of length 1, that is the full centre of *G*.

Each irreducible character of *G* restricted to *Z* is a multiple of an irreducible character of *Z* (Schur's Lemma). We get an orthogonal decomposition of the lattice of generalized characters into sublattices spanned by irreducible characters whose restrictions to *Z* are multiples of the same irreducible character. `SplitByCentre` computes a data structure which allows to split any generalized character according to this decomposition. We only need to store class functions for one character of *Z* per Galois orbit (the others can be computed by `GaloisCyc` (**Reference: GaloisCyc for a cyclotomic**)).

This splitting is applied to all generalized characters already stored in *CCT* and to all generalized characters which are imported afterwards.

After a successful call with non-trivial *Z* the filter `HasSplittingCentre` is set in *CCT* and the attribute `SplittingCentre` of *CCT* is set to *zind*.

Example

```
gap> G := SL(4,5);;
gap> CCT := CCTable(G);;
gap> SplitByCentre(CCT);
gap> HasSplittingCentre(CCT);
true
gap> SplittingCentre(CCT);
[ 1, 10, 19, 28 ]
```

4.3.2 SplitCharacterByCentre

- ▷ `SplitCharacterByCentre(CCT, ch)` (operation)
- ▷ `SplitEncodedCharacterByCentre(CCT, ech)` (operation)

Returns: a list of (encoded) characters

Let *CCT* be an `IsCCTable` (4.1.1) object with `HasSplittingCentre(CCT) = true`. In the first form let *ch* be a generalized character of the underlying group, given by a list of values on *all* conjugacy classes (not encoded as by `EncodeForCCTable` (4.1.2)). In the second form *ech* is a character encoded for *CCT*. This function decomposes the character according to the splitting centre (see `SplitByCentre` (4.3.1)).

Example

```
gap> G := SmallGroup(96, 14);;
gap> G := SmallGroup(96, 14);;
gap> CCT := CCTable(G);;
gap> SplitByCentre(CCT);
gap> U := TrivialSubgroup(G);;
gap> reg := InducedClassFunction(TrivialCharacter(U), G);;
gap> SplitCharacterByCentre(CCT, reg);
[ [ 24, 0, 0, 24, 0, 24, 0, 0, 0, 0, 0, 0, 0, 0, 24, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0 ],
  [ 24, 0, 0, -24, 0, 24, 0, 0, 0, 0, 0, 0, 0, 0, -24, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0 ],
  [ 24, 0, 0, 24, 0, -24, 0, 0, 0, 0, 0, 0, 0, 0, -24, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0 ],
  [ 24, 0, 0, -24, 0, -24, 0, 0, 0, 0, 0, 0, 0, 0, 24, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0 ] ]
```

4.4 Natural characters

4.4.1 NaturalCharacters

▷ `NaturalCharacters( $G$ )` (attribute)

Returns: a list of characters

Returns a list of characters of the finite group G derived from the way G is represented, these are usually cheap to compute. The following cases are handled:

permutation groups

the permutation characters on the orbits of moved points, minus the trivial character

matrix groups in characteristic zero

the character of traces of the elements

matrix groups over finite fields

the permutation characters on the non-zero vectors of the natural module (over the field of definition and over its quadratic extension), and on the one-dimensional subspaces of the natural module

otherwise

the regular character of G

Example

```
gap> NaturalCharacters(AlternatingGroup(5));
[ Character( CharacterTable( Alt( [ 1 .. 5 ] ) ),
  [ 4, 0, 1, -1, -1 ] ) ]
gap> NaturalCharacters(SL(3,5));
[ Character( CharacterTable( SL(3,5) ),
  [ 124, 4, 4, 0, 0, 4, 24, 4, 4, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0 ] ),
  Character( CharacterTable( SL(3,5) ),
  [ 15624, 24, 24, 0, 0, 24, 624, 24, 24, 0, 0, 24, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0 ] ),
  Character( CharacterTable( SL(3,5) ),
  [ 30, 6, 0, 6, 6, 2, 5, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, -1,
    -1, -1, -1, -1, -1, -1, -1, -1, -1 ] ) ]
gap> NaturalCharacters(SmallGroup(24,4));
[ Character( CharacterTable( <pc group of size 24 with
  4 generators> ), [ 24, 0, 0, 0, 0, 0, 0, 0, 0, 0 ] ) ]
```

References

- [FLI] The FLINT team. *FLINT: Fast Library for Number Theory*. <https://flintlib.org>. 16
- [Isa76] I. M. Isaacs. *Character Theory of Finite Groups*. Academic Press, 1976. 14
- [Ser77] J.-P. Serre. *Linear Representations of Finite Groups*, volume 42 of *Graduate Texts in Mathematics*. Springer-Verlag, 1977. 12
- [Sto96] A. Storjohann. Faster algorithms for integer lattice basis reduction. Technical Report 249, ETH Zürich, Department of Computer Science, 1996. 17

Index

CCTab, [4](#)

ActionAutomorphismsOnConjugacyClasses, [13](#)

ActionAutomorphismsOnRationalClasses, [13](#)

AddVectorToHNF, [22](#)

CCTable, [18](#)

CCTScalarProduct, [23](#)

ConjugacyClassInvariants, [7](#)

EncodeForCCTable, [19](#)

ExpandFromCCTable, [19](#)

FindIndexCCTable, [21](#)

FindIrreduciblesInFullLattices, [22](#)

FusionElementaryCCTable, [15](#)

HasFullRankCCTable, [20](#)

HasSplittingCentre, [23](#)

HermiteIntMat, [16](#)

ImportEncodedToCCTable, [20](#)

ImportInducedFromAllMaximalCyclicToCCTable, [21](#)

ImportToCCTable, [20](#)

InduceAllFromCyclicSubgroup, [11](#)
forCCTable, [11](#)

InducedFromAllMaximalCyclicSubgroups, [12](#)

InducedFromElementary, [14](#)
forCCTable, [14](#)

InductionDataFromElementaryCCTable, [15](#)

InfoCCTable, [5](#)

InverseUnimodularMat, [16](#)

IrrCCTableHNF, [22](#)

IsCCTable, [18](#)

License, [2](#)

LLLTransformUnimodularGram, [17](#)

MaximalCyclics, [11](#)

MaximalNonCyclicElementarySubgroups, [14](#)

NaturalCharacters, [25](#)

NrRationalClasses, [9](#)

PermutedFractionFreeIntegerGauss-PositiveDefinite, [16](#)

PositionConjugacyClass, [7](#)

PowerMapCharacters, [12](#)

PowerMapsOfAllClasses, [8](#)
for character table, [8](#)

RationalClassesInfo, [9](#)

RationalClassSets, [9](#)

SmallPowerMapCharacters, [13](#)

SplitByCentre, [23](#)
withZ, [23](#)

SplitCharacterByCentre, [24](#)

SplitEncodedCharacterByCentre, [24](#)

SplittingCentre, [23](#)

UpdateHNFCCTable, [20](#)