matrixss
A GAP4 Package

Version 0.9
by

Henrik Bäärnhielm

Department of Mathematics

Imperial College of Science, Technology and Medicine

email: henrik.baarnhielm@imperial.ac.uk

December 2004

This package was originally made as part of a student project at the Department of Mathematics, Imperial College of Science, Technology and Medicine in London, United Kingdom. It was made in partial fulfillment of the requirements for the degree of Master of Science in Pure Mathematics.

I would like to thank my supervisors Prof. Alexander Ivanov at Imperial College and Dr. Leonard Soicher at Queen Mary, University of London, for their encouragement and help.

I am also particularly grateful to Alexander Hulpke at Colorado State University in Fort Collins, USA, for suggesting this project to me, and for his constant help and advice during my work.

Contents

1 Introduction

This is a manual for the matrixss package, which is a package for the GAP system for computational group theory. It contains an implementation of Schreier-Sims algorithm for matrix groups, including both the standard deterministic and the standard probabilistic approach. There is also an implementation of the so-called Schreier-Todd-Coxeter-Sims algorithm, which uses coset enumeration to possibly speed up the process, but this algorithm is mainly used for verifying the output of the probabilistic algorithm. An implementation of the *Verify* routine by Sims can also be used for verification purposes, and finally, an implementation of the nearly linear time Schreier-Sims algorithm is also included in the package.

No theory about the Schreier-Sims algorithm will be covered in this manual, since the theory is well-known and can easily be found elsewhere. For example, the author of the package has written a report about it, see [Bää04]. Other references are [But91], [CSS99] and [Ser03]. In this manual, we are instead only concerned with the actual implementation, and how to use the package in GAP.

The package can be downloaded from its homepage, which is

`http://matrixss.sourceforge.net`

The package author can be reached at `redstar_@sourceforge.net`.

1.1 Usage

Since the Schreier-Sims algorithm usually is just an initial step for other algorithms, there is little actual user interaction. When the package is loaded, a method for `Size` is installed for finite matrix groups, which uses the algorithms in this package to compute the order of the group.

At a lower level, the package installs an attribute `StabChainMatrixGroup` for finite matrix groups, see 2.1.1, where the base and strong generating set are stored. They can thus easily be used by anyone.

Currently, this is the only interaction between the package and the rest of GAP, but it should be sufficient.

2 Implementation

Here is the documentation from the package source code. As the package works with matrix groups acting on vector spaces, references to *group elements* means *matrices*, ie a list of row vectors, each of which is list of field elements. References to *points* means *row vectors*, ie elements of the vector space on which the group acts.

The code tries to avoid the computation of inverse matrices as much as possible, and to accomplish this, the inverse of a group element is stored together with the element in a list of length 2. Each time some computation is made with the element, a similar computation is made with the inverse, so that they are kept consistent. Therefore, in many cases in the code, *group element* means an immutable list of 2 matrices that are inverses to each other.

2.1 General code

These are the general declarations used by the package. Most notably the attribute `StabChainMatrixGroup` which is the core of the package functionality.

1 ► `StabChainMatrixGroup( G )` A

Declare new attribute for storing base, SGS and Schreier trees. The attribute is computed using the Schreier-Sims algorithm for finite matrix groups, which is the main content of the package.

The attribute is a record with two components:

`SchreierStructure`
: the main information structure, see 2.1.2.

`SGS`
: a list of the strong generators

The corresponding attribute operations are aware of a few Options.

`SimpleSchreierTree`
: calculate coset representatives at the moment of creation of the Schreier trees, thus making them have height 1. This should make the algorithm significantly faster.

`ExtendSchreierTree`
: Do not recompute Schreier trees at each run of a given level, but extend the Schreier trees from the last run at that level.

`AlternatingActions`
: Always prepend a base point with the line that contains it, using the projective action on the line.

`CleverBasePoints`
: Choose an initial list of base points using `BasisVectorsForMatrixAction`, which is made by O'Brien and Murray.

`ShallowSchreierTree`
: Create the Schreier trees using the method described by Babai et al (1991) which guarantees logarithmic depth. This Option is only used when the option `SimpleSchreierTree` is *not* defined.

`Random`
: Use probabilistic algorithm.

`Linear`
: Use nearly linear-time algorithm. This takes precedence over `Random`, if it is present.

2 ► `ssInfo` V

Main structure holding information for the algorithm. This is not a global variable, but the same structure is used in all the variants of the algorithm, but all members are not necessarily used.

The structure `ssInfo` is a list of records, with a record for each level in the algorithm, ie one record for each base point. New base points may of course be added to the base during the execution of the algorithm, and then a new record is added to the end of the list.

The members of the record are:

`partialSGS`
: the elements in the current partial SGS that fixes all points at lower levels, or the whole partial SGS for the first level

`partialBase`
: the base point for this level

`action`
: the action (function) at this level

`points`
: the field where the base point `partialBase` comes from

`hash`
: the hash function for the Schreier tree at this level

`schreierTree`
: the Schreier tree for this level, representing the basic orbit at this level, ie the orbit of `partial-Base` under the action of `partialSGS` at the previous (lower) level. Thus, the root of the tree is `partialBase`.

`oldSGS`
: the whole partial SGS at the last call of SchreierSims at this level

`IsIdentity`
: the function to check if a point is the identity at this level

3 ► `MatrixSchreierSimsInfo` V

The GAP InfoClass used by the package, for debugging purposes.

4 ► `MATRIXSS_DEBUGLEVEL` V

The internal debugging level. This is really obsolete and the above info class should be used instead.

5 ► `MATRIXSS_BasePointStore` V

A list of hopefully good base points, ie base points with small orbits. They are fetched with `BasisVectors-ForMatrixAction` which is due to O'Brien and Murray, and as long as the list is non-empty, new base points will be shifted from it.

6 ► `MatrixGroupOrderStabChain( ssInfo )` F

Computes the order of the group defined by the given Schreier trees, see 2.1.2.

These are the common functions used by all the variants of the Schreier-Sims algorithm implemented in the package.

These are the core functions of the package.

7 ▶ `Size( G )` A

A method for `Size` for finite matrix groups, that uses the implementation of Schreier-Sims algorithm in this package, ie it uses the `StabChainMatrixGroup` attribute to compute the order of `G`.

This method is only installed if `MATRIXSS_TEST` is not defined when the package is loaded.

8 ▶ `MATRIXSS_GetPartialBaseSGS( generators, identity, field )` F

Constructs a partial base and a partial SGS given a set of generators for a group. Returns the partial SGS and the `ssInfo` structure, see 2.1.2.

`generators`
: given set of generators

`identity`
: group identity element (the identity matrix)

`field`
: the vector space on which the group acts

9 ▶ `MATRIXSS_Membership( ssInfo, element, identity )` F

Check if an element belongs to a group, using sifting

`ssInfo`
: Main information structure about our stabiliser chain. The Schreier trees is used during the sifting.

`element`
: the element to check for membership

`identity`
: group identity

10 ▶ `MATRIXSS_NewBasePoint( element, identity, field )` F

Find a point not in base that is moved by the given element (which fixes the base)

`element`
: the bad element that fixes the whole base

`identity`
: the group identity (the identity matrix)

`field`
: the vector space on which the group acts

11 ▶ `MATRIXSS_GetSchreierGenerator( schreierTree, generator, point, action, identity, IsIdentity )` F

Creates a Schreier generator for the stabiliser in the group which has `generator` as one of its generators. The stabiliser fixes `point` under `action`.

12 ▶ `MATRIXSS_ExtendBase( ssInfo, badElement, identity )` F

Add a new base point to the base, so that the given element is not in the stabiliser of the point

`ssInfo`
: main information structure for the current Schreier-Sims run

`badElement`
: the element that fixes all current base points

`identity`
: the group identity

13 ► `MATRIXSS_AugmentBase( ssInfo, newPoint, action, hash, identity )` F

Add a new base point to the base, so that the given element is not in the stabiliser of the point

`ssInfo`
: main information structure for the current Schreier-Sims run

`newPoint`
: the point to add to the base

`action`
: the action for the new point

`hash`
: the dictionary info for the new point

`identity`
: the group identity

14 ► `MATRIXSS_OrbitElement( schreierTree, point, action, identity, IsIdentity )` F

Compute the group element that connects the root of the Schreier tree to a given point. This function assumes that the point actually is in the orbit described by the given Schreier tree.

`schreierTree`
: Schreier tree for the orbit to use

`point`
: the point to check if it is in the orbit

`action`
: the action that was used to create the Schreier tree

`identity`
: the group identity (the identity matrix)

`IsIdentity`
: function to use when checking if a group element is equal to the identity

15 ► `MATRIXSS_ComputeSchreierTree( tree, generators, action, root, hash, identity )` F

Fill a Schreier tree that contains only the root.

`tree`
: The Schreier tree to fill.

`generators`
: The generators for the group that gives rise to the orbit represented by the Schreier tree.

`action`
: The action of the group on the point set.

`root`
: The root point of the tree.

`hash`
: The dictionary info for the tree, used to create hash function.

`identity`
: the group identity (the identity matrix)

16 ▶ `MATRIXSS_ExtendSchreierTree( oldTree, generators, oldGenerators, action, dictinfo )` F

Extends an existing Schreier tree by a given set of generators

`oldTree`
: The Schreier tree to extend, ie a Dictionary.

`generators`
: The generators for the group that gives rise to the orbit represented by the Schreier tree.

`oldGenerators`
: The current generators (edge-labels) of `oldTree`.

`action`
: The action of the group on the point set.

`dictinfo`
: The Dictionary info used when `oldTree` was created.

17 ▶ `MATRIXSS_OrbitElement_ToddCoxeter( schreierTree, point, action, identity, IsIdentity, free-Group, genMap )` F

Special version of `MATRIXSS_OrbitElement`, see 2.1.14, that also calculates the Word in the generators of the group element it returns.

More specifically, it computes the Word of the generators of the corresponding free group.

`schreierTree`
: Schreier tree for the orbit to use

`point`
: the point to check if it is in the orbit

`action`
: the action that was used to create the Schreier tree

`identity`
: the group identity (the identity matrix)

`IsIdentity`
: function to use when checking if a group element is equal to the identity

`freeGroup`
: corresponding free group to the group whose generators form the set of edge labels of `schreierTree`

`genMap`
: list of 2 lists of the same length, the first being the edge labels of `schreierTree` (the generators of the corresponding group), and the second being the corresponding generators of `freeGroup`

18 ▶ `MATRIXSS_Membership_ToddCoxeter( ssInfo, element, identity, freeGroup )` F

Special version of `MATRIXSS_Membership`, see 2.1.9, that also expresses the sifted group element as a word in the generators of a given free group.

`ssInfo`
: Main information structure about our stabiliser chain. The Schreier trees is used during the sifting.

`element`
: the element to check for membership

`identity`
: group identity

`freeGroup`
: the free group in which the sifted element will be expressed

19 ► `MATRIXSS_GetSchreierGenerator_ToddCoxeter( schreierTree, generator, point, action, identity, IsIdentity, freeGroup, genMap )` F

Special version of `MATRIXSS_GetSchreierGenerator`, see 2.1.11, that also returns the Schreier generator as a Word in the generators of `freeGroup`, using `genMap` to map the generators to the free group. See 2.1.17.

20 ► `MATRIXSS_CreateShallowSchreierTree( orbitTree, root, generators, labels, action, identity, hash )` F

Create a shallow Schreier tree, ie with at most logarithmic height.

`orbitTree`
: Given tree representing the same orbit as the shallow Schreier to be computed.

`root`
: The root point of the tree.

`generators`
: From this set will any new edge labels be taken.

`labels`
: The elements that, together with its inverses, will form the edge labels of the tree.

`action`
: The action of the group on the point set.

`identity`
: the group identity (the identity matrix)

`hash`
: The dictionary info for the tree, used to create hash function.

21 ► `MATRIXSS_GetSchreierTree( oldTree, root, generators, oldGenerators, action, hash, identity )` F

Returns a Schreier tree. This routine encapsulates the other Schreier tree functions.

`oldTree`
: The Schreier tree to extend, in case there should be an extension.

`root`
: The root point of the tree.

`generators`
: The generators for the group that gives rise to the orbit represented by the Schreier tree.

`oldGenerators`
: The current generators (edge-labels) of `oldTree`.

`action`
: The action of the group on the point set.

`hash`
: The dictionary info for the tree, used to create hash function.

`identity`
: the group identity (the identity matrix)

22 ► `MATRIXSS_MonotoneTree( root, elements, action, identity, dictinfo )` F

Create a monotone Schreier tree with given root and edge labels.

`root`
: The root point of the tree.

`elements`
: The elements that, together with its inverses, will form the edge labels of the tree.

`action`
: The action of the group on the point set.

`identity`
: the group identity (the identity matrix)

`dictinfo`
: The dictionary info for the tree, used to create hash function.

23 ► `MATRIXSS_RandomCosetRepresentative( schreierTree, action, identity )` F

Return a random coset representative from the transversal defined by `schreierTree`.

24 ► `MATRIXSS_RandomOrbitPoint( schreierTree )` F

Returns a random point in the orbit given by `schreierTree`.

25 ► `MATRIXSS_RandomSchreierGenerator( schreierTree, elements, action, identity )` F

Return a random Schreier generator constructed from the points in `schreierTree` and the generators in `elements`.

26 ► `MATRIXSS_RandomSubproduct( elements, identity )` F

Return a random subproduct of `elements`.

These are also core function, but of slightly less importance, or mainly of technical nature.

27 ► `MATRIXSS_SubProdGroups` V

A Dictionary of SymmetricGroups, used when permuting random subproducts.

28 ► `MATRIXSS_CopySchreierTree( tree, dictinfo )` F

Creates a copy of a whole Schreier tree, ie of makes a copy of the Dictionary.

`tree`
: the Dictionary to copy

`dictinfo`
: the dictinfo that was used when creating `tree`

29 ► `MATRIXSS_GetOrbitSize( schreierTree )` F

Get size of orbit defined by the given Schreier tree.

30 ► `MATRIXSS_GetOrbit( schreierTree )` F

Return all points (as a list) in the orbit of the point which is root of the Schreier tree, ie return all keys in the Dictionary. The list is not necessarily sorted, and it is mutable.

31 ► `MATRIXSS_IsPointInOrbit( schreierTree, point )` F

Check if the given point is in the orbit defined by the given Schreier tree.

32 ► `MATRIXSS_CreateInitialSchreierTree( root, dictinfo, identity )` F

Create a Schreier tree containing only the root.

`root`
: The base point that is to be the root of the Schreier tree.

`dictinfo`
: Used when creating the Dictionary that is the Schreier tree

`identity`
: the group identity element

33 ▶ `MATRIXSS_GetSchreierTreeEdge( schreierTree, point )` F

Get the label of the edge originating at the given point, and directed towards the root of the given Schreier tree.

34 ▶ `MATRIXSS_ProjectiveIsIdentity( element, identity )` F

Identity check when using projective action (all scalar matrices are considered equal to the identity)

`element`
: the group element to check if it is equal to identity

`identity`
: the group identity (the identity matrix)

35 ▶ `MATRIXSS_IsIdentity( element, identity )` F

Identity check when using normal point action

`element`
: the group element to check if it is equal to identity

`identity`
: the group identity (the identity matrix)

36 ▶ `MATRIXSS_PointAction( point, element )` F

The action of a group element (a matrix) on a point (a row vector). The action is from the right

`point`
: The point (row vector) to act on.

`element`
: The group element (matrix) that acts.

37 ▶ `MATRIXSS_ProjectiveAction( point, element )` F

The projective action of a matrix on a row vector. The one-dimensional subspace corresponding to the point is represented by the corresponding normed row vector

`point`
: The point to act on. Must be a **normed** row vector.

`element`
: The group element (matrix) that acts.

38 ▶ `MATRIXSS_DebugPrint( level, message )` F

Internal function for printing debug messages. Uses the internal variable `MATRIXSS_DEBUGLEVEL`, see 2.1.4, to determine if the message should be printed.

2.2 Deterministic algorithm

These are the special routines for the deterministic version of Schreier-Sims algorithm.

1 ► `StabChainMatrixGroup( G )` A

An implementation of the Schreier-Sims algorithm, for matrix groups, probabilistic version. See 2.1.1 for general information about the attribute.

2 ► `SchreierSims( ssInfo, partialSGS, level, identity )` F

The main Schreier-Sims function, which is called for each level.

`ssInfo`
: main information structure for the current Schreier-Sims run

`partialSGS`
: given partial strong generating set

`level`
: the level of the call to Schreier-Sims

`identity`
: the group identity

2.3 Probabilistic algorithm

These are the special routines for the probabilistic implementation of Schreier-Sims algorithm.

1 ► `StabChainMatrixGroup( G )` A

An implementation of the Schreier-Sims algorithm, for matrix groups, probabilistic version. See 2.1.1 for general information about the attribute.

In addition to the general Options of the attribute `StabChainMatrixGroup`, the probabilistic algorithm is aware of the following:

`Probability`
: (lower bound for) probability of correct solution, which defaults to 3/4

`Verify`
: Boolean parameter which signifies if the base and SGS computed using the random Schreir-Sims algorithm should be verified using the Schreier-Todd-Coxeter-Sims algorithm. Defaults to `false`.

`OrderLowerBound`
: Lower bound for the order of `G`, must be ≥ 1. Defaults to 1.

`OrderUpperBound`
: Upper bound for the order of `G`, or 0 if unknown. Defaults to 0.

Note that if the order of `G` is known, so that `OrderLowerBound` = `OrderUpperBound` = `Size(G)` then the randomized algorithm always produces a correct base and SGS, so there is no need of verification. Also, the verification will extend the given base and SGS to a complete base and SGS if needed.

2 ► `RandomSchreierSims( ssInfo, partialSGS, maxIdentitySifts, identity, low_order, high_order )`
F

The main random Schreier-Sims function.

`ssInfo`
: main information structure for the Schreier-Sims

`partialSGS`
: given partial strong generating set

`maxIdentitySifts`
: maximum number of consecutive elements that sifts to identity before the algorithm terminates

`identity`
: the group identity

`lowOrder`
: lower bound on the group order (must be ≥ 1)

`highOrder`
: upper bound on the group order, or 0 if not available

2.4 STCS algorithm

These are the Schreier-Todd-Coxeter-Sims routines, ie Schreier-Sims algorithm with additional calls to Todd-Coxeter coset enumeration to possibly speed up the process. It is known to be fast when the input is already a base and SGS, and therefore it is good for verifying a proposed base and SGS, for example the output of a probabilistic algorithm.

1 ▸ `MATRIXSS_SchreierToddCoxeterSims( ssInfo, partialSGS, level, identity, cosetFactor )` F

The main function for the Schreier-Todd-Coxeter-Sims algorithm. It is very similar to ordinary Schreier-Sims algorithm and has a similar interface.

`ssInfo`
: main information structure for the current Schreier-Sims run

`partialSGS`
: given partial strong generating set

`level`
: the level of the call to Schreier-Sims

`identity`
: the group identity

`cosetFactor`
: the quotient of the maximum number of cosets generated during coset enumeration and the corresponding orbit size

2.5 Nearly linear time algorithm

These are the special routines for the nearly linear-time version, described in Babai et al, 1991.

1 ▸ `StabChainMatrixGroup( G )` A

An implementation of the Schreier-Sims algorithm, for matrix groups. This version is inspired by the nearly linear time algorithm, described in [Ser03]. See 2.1.1 for general information about the attribute.

2 ▸ `ConstructSGS( ssInfo, partialSGS, identity )` F

The main Schreier-Sims function for the nearly linear-time algorithm.

`ssInfo`
: main information structure for the current Schreier-Sims run

`partialSGS`
: given partial strong generating set

`identity`
: the group identity

3 ▶ `CompletePointStabiliserSubgroup( ssInfo, element, level, identity, maxIdentitySifts )` F

The work-horse of the nearly linear-time algorithm, called for each level.

`ssInfo`
: main information structure for the current Schreier-Sims run

`element`
: the element which

`partialSGS`
: given partial strong generating set

`identity`
: the group identity

2.6 Verify routine

1 ▶ `MatrixSchreierSimsVerify( ssInfo, SGS, identity )` F

The *Verify* routine by Sims. Checks whether the given `ssInfo` and `SGS` encodes a base and strong generating set, and returns a record with components `Residue` and `Level`. In case the verification succeeds, the level is 0 and the residue is the identity. Otherwise the residue is an element that is in the stabiliser of the group at the indicated level, but is not in the group at the next higher level.

`ssInfo`
: proposed structure to check

`SGS`
: proposed SGS to check

`identity`
: the group identity

2 ▶ `MATRIXSS_VerifyLevel( ssInfo, partialSGS, level, identity )` F

Checks that the stabiliser of the group at the given level is the same as the group at the next higher level.

`ssInfo`
: proposed structure to check

`partialSGS`
: proposed SGS to check

`level`
: level to check

`identity`
: the group identity

3 ▶ `MATRIXSS_VerifyMultipleGenerators( generators, schreierTree, point, action, hash, subGenerators, ssInfo, SGS, identity, points, IsIdentity, field )` F

Verifies that the stabiliser of the group generated by `generators`, at the point `point` is the same group as the group generated by `generators` minus `subGenerators`. If so, the identity is returned, and otherwise an element that is in the difference is returned.

`ssInfo` and `SGS` should be a base and strong generating set for the smaller group.

`generators`
: generators of the bigger group

`schreierTree`
: Schreier tree for the orbit of `point` under `action` of the group generated by `generators`

`point`
: the point to get the stabiliser of

`action`
: the action to use when calculating the stabiliser

`hash`
: the dictionary info of `schreierTree`

`subGenerators`
: the additional generators of the bigger group

`ssInfo`
: main structure for the base and sgs of the smaller group

`SGS`
: strong generating set for the smaller group

`identity`
: the group identity

`points`
: the point set to which `point` belong

`IsIdentity`
: the identity check function for the larger group

`field`
: the finite field of the larger group

4 ► `MATRIXSS_VerifySingleGenerator( generators, schreierTree, point, action, hash, subGenerator, ssInfo, SGS, identity )` F

Verifies that the stabiliser of the group generated by `generators`, at the point `point` is the same group as the group generated by `generators` minus `subGenerator`. If so, the identity is returned, and otherwise an element that is in the difference is returned.

`ssInfo` and `SGS` should be a base and strong generating set for the smaller group.

`generators`
: generators of the bigger group

`schreierTree`
: Schreier tree for the orbit of `point` under `action` of the group generated by `generators`

`point`
: the point to get the stabiliser of

`action`
: the action to use when calculating the stabiliser

`hash`
: the dictionary info of `schreierTree`

`subGenerator`
: the additional generator of the bigger group

`ssInfo`
: main structure for the base and sgs of the smaller group

`SGS`
: strong generating set for the smaller group

`identity`
: the group identity

5 ► `MATRIXSS_StabiliserGens( ssInfo, partialSGS, point, action, dictinfo, identity )` F

Return generators of the stabiliser of the group generated by the strong generators `partialSGS` at `point` under `action`. The `ssInfo` structure should be a base for the group.

`ssInfo`
: main structure for the base

`partialSGS`
: strong generating set for the given group

`point`
: the point to get stabiliser at

`action`
: the action to use when computing stabiliser

`dictinfo`
: the dictionary info of `schreierTree`

`identity`
: the group identity

6 ► `MATRIXSS_IsBlockOfImprimitivity( schreierTree, generators, block, action, identity )` F

Checks whether `block` is a block of imprimitivity for `action` of the group given by `generators` on the set of points given by `schreierTree`.

`schreierTree`
: Schreier tree for the point set

`generators`
: generators of the acting group

`block`
: the block to check for imprimitivity

`action`
: the action to use

`identity`
: the group identity

7 ► `MATRIXSS_DecomposeOrbit( schreierTree, root, generators, action, hash, identity )` F

Decompose the orbit given by `schreierTree`, with root point `root`, into orbits of the group generated by `generators`, under `action`.

`schreierTree`
: Schreier tree for the orbit to decompose

`root`
: root point of `schreierTree`

`generators`
: generators of the decomposing group

`action`
: the action of the decomposing group

`hash`
: the dictionary info of `schreierTree`

`identity`
: the group identity

8 ► `MATRIXSS_BaseChange( ssInfo, partialSGS, level, identity )` F

Flips the base point at `level` with the next higher base point and update the `ssInfo` structure.

`ssInfo`
: main structure for the base

`partialSGS`
: strong generating set corresponding to `ssInfo`

`level`
: the level for the base change

`identity`
: the group identity

2.7 Test and benchmark routines

These are the main routines for testing and benchmarking the package.

1 ► `MatrixSchreierSimsTest( maxDegree, maxFieldSize )` F

Compares results of the above function with the built-in GAP Size method for a bunch of classical matrix groups. (`GL`, `SL`, etc)

`maxDegree`
: maximum matrix size for classical matrix groups to be used for testing

`maxFieldSize`
: maximum finite field size for classical matrix groups to be used for testing

2 ► `MatrixSchreierSimsBenchmark( maxDegree, maxFieldSize, maxReeSize, maxSuzukiSize )` F

Check speed of package routines against classical matrix groups and the matrix representations of Ree and Suzuki sporadic groups.

`maxDegree`
: maximum matrix size for classical matrix groups to be used for testing

`maxFieldSize`
: maximum finite field size for classical matrix groups to be used for testing

`maxReeSize`
: maximum `ReeGroup` size, see 48.1.11 in the reference manual.

`maxSuzukiSize`
: maximum `SuzukiGroup` size, see 48.1.10 in the reference manual.

These are auxiliary functions for test and benchmark.

3 ► `MATRIXSS_GetTestGroups( maxDegree, maxFieldSize )` F

Creates a list of classical matrix groups to use when testing the package. The groups are `GL`, `SL`, `GO`, `SO`, `GU` and `SU`.

`maxDegree`
: maximum matrix size for classical matrix groups to be used for testing

`maxFieldSize`
: maximum finite field size for classical matrix groups to be used for testing

4 ► `MATRIXSS_GetBenchmarkGroups( maxDegree, maxFieldSize )` F

Creates a list of classical matrix groups and sporadic groups to use when benchmarking the package. The classical groups are `GL`, `SL`, `GO` and `SO`. The sporadic groups are `Ree` and `Sz`.

`maxDegree`
: maximum matrix size for classical matrix groups to be used for testing

`maxFieldSize`
: maximum finite field size for classical matrix groups to be used for testing

`maxReeSize`
: maximum `ReeGroup` size, see 48.1.11 in the reference manual.

`maxSuzukiSize`
: maximum `SuzukiGroup` size, see 48.1.10 in the reference manual.

5 ► `MATRIXSS_TimedCall( call, args )` F

Runs the specified function with arguments and return the running time in milliseconds, as given by Runtime, see 7.6.2 in the reference manual.

`call`
: function to call

`args`
: list of arguments to `call`

Bibliography

[Bää04] Henrik Bäärnhielm. The Schreier-Sims algorithm for matrix groups. Master's thesis, Imperial College of Science, Technology and Medicine, 2004.

[But91] Gregory Butler. *Fundamental algorithms for permutation groups*, volume 559 of *Lecture Notes in Computer Science*. Springer, 1991.

[CSS99] Hans Cuypers, Leonard H. Soicher, and Hans Sterk. Working with finite groups. In Arjeh M. Cohen, Hans Cuypers, and Hans Sterk, editors, *Some Tapas of Computer Algebra*, volume 4 of *Algorithms and Computation in Mathematics*, chapter 8, pages 184–207. Springer, 1999.

[Ser03] Ákos Seress. *Permutation group algorithms*, volume 152 of *Cambridge Tracts in Mathematics*. Cambridge University Press, 2003.

Index

This index covers only this manual. A page number in *italics* refers to a whole section which is devoted to the indexed subject. Keywords are sorted with case and spaces ignored, e.g., "`PermutationCharacter`" comes before "permutation group".