Simplicial Homology

A proposed share package for GAP

by

Jean-Guillaume Dumas, Frank Heckenbach, David Saunders & Volkmar Welker

Unité Informatique et Distribution, Grenoble, France
Jean-Guillaume.Dumas@imag.fr, www-id.imag.fr/~jgdumas

Universität Erlangen, Mathematisches Institut, Erlangen, Germany
heckenb@mi.uni-erlangen.de, www.mi.uni-erlangen.de/~heckenb

Computer and Information Sciences Department, University of Delaware, Newark, DE, USA
saunders@udel.edu, www.cis.udel.edu/~saunders

and

Philipps-Universität Marburg, Fachbereich Mathematik und Informatik, Marburg, Germany
welker@mathematik.uni-marburg.de, www.Mathematik.uni-marburg.de/~welker

Preface

1 About the Package

The development of the package was driven by two different targets. The first target is to design efficient algorithms for exact matrix computations (e.g. Smith Normal from) for sparse matrices with entries in the integers. The second target is efficient software to calculate homology of simplicial complexes. Since the crucial step for the second target is the calculation of the Smith form of the ("very sparse") matrices of the differentials in simplicial homology these two targets team up perfectly. The history of the genesis of the package explains the mix of computer algebraists and (geometric) combinatorialists among the authors. In the course of developing the package examples interesting to combinatorialists gave rise to matrices which motivated developments on the algorithmic side. On the other hand computations performed with pre-versions of the package provided data giving rise and/or evidence to conjectures, some of which later on were turned into theorems. It is our expectation that working with the package will allow many application of that kind. The functionality of GAP 4 fits perfectly in this picture. The algebraic and combinatorial objects, their constructors and invariants already implemented facilitate the development of the routines of the package whose average running time and memory consumption is not critical.

In general, we hope the interaction between the algorithmic and geometric/combinatorial side will continue and grow once the package has been released.

Algorithms: Simplicial complex constructions can easily lead to cases where some of the boundary maps of simplicial homology are very large, very sparse matrices. Each of the four algorithms for invariant factors implemented in the package each has strengths and weaknesses. We are involved in continuing research on better methods for this. We would appreciate learning about examples that are hard or impossible to compute by our four algorithms. We may even be able to help get them computed. Such examples or any other communication about the algorithms part of the package are cheerfully received at `saunders@udel.edu`.

The Complexes and their Invariant: We have implemented a number of constructors and a number of particular examples of simplicial complexes. But there is more to do. We are working on expanding the functionality of the package to include the construction and the computation of other invariants of simplicial complexes arising in combinatorics or algebraic topology. Of course our view is lopsided and mostly driven by our own research problems. Thus in order to broaden the applicability of the package we welcome any suggestion, let it be concerned with the existing package or regarding desirable expansions. Also we are interested to hear about successful applications of the package. Concerning those matters, please contact us at `welker@mathmatik.uni-marburg.de`.

Bugs: Finally there are the bugs. Of course we want to hear about them and we want to get rid of them as quickly as possible. Since each of the authors has contributed to different parts of the code, we would like to ask you not to report bugs to a single author but rather to `simphom@cis.udel.edu`.

2 Acknowledgment

Dave Saunders was supported by the NSF which also funded visits of the other authors to Delaware. All three enjoyed the hospitality of Dave and Nancy Saunders which made each visit a very pleasant experience. Volkmar Welker was supported by Deutsche Forschungsgemeinschaft (DFG). Frank Heckenbach was supported by a "Graduiertenstipendium" of Universität Erlangen-Nürnberg. Jean-Guillaume Dumas was supported by the "Eurodoc" program of "Région Rhônes Alpes".

Contents

1 The Homology Package

1.1 What can it do ? What for and Why ?

This share package for GAP 4 provides routines for computing integer homology of simplicial complexes and for computing rank and invariant factors (Smith form) of integer matrices. In particular, the following functionality is provided:

• Generate example complexes, construct new complexes from old.
   → Sample triangulations of particular manifolds.
   → Particular classes of complexes from graph and group theory.
   → Order complexes of posets and sample posets.

• Produce the homology from the invariant factors of the boundary maps.

• We offer 4 algorithms for computing the invariant factors (or Smith form) of the boundary maps. This is done because there is **no single best** choice. Especially if some boundary maps are of large size, it may be that only one of the four methods succeeds.
   → Compute invariant factors by elimination using word size integers
   → Compute invariant factors by elimination using GMP arbitrary precision integers
   → Compute invariant factors by a mix of an iterative method and elimination mod small primes, thus avoiding some expression growth
   → Compute invariant factors by an iterative method in which memory use is linear in the matrix dimension, but which gives incomplete results.

This functionality and more is implemented in the package. When developing this package our "model user" was a researcher who is interested in the homology of a particular series of simplicial complexes, say Δ_n, $n \geq 1$, such that even for small n it is very hard to calculate $\widetilde{H}_\bullet(\Delta_n)$. In this case, calculations of $\widetilde{H}_\bullet(\Delta_n)$ may lead to clues or even conjectures for the general behavior of the homology. This scheme has been applied very successfully in [BBL+99] and [BW99]. On the other hand in [Lut99] predecessors of the package were successfully used in a situation where for a very large number of relatively small simplicial complexes homology had to be computed in order to filter out the "good" complexes. Using the package for teaching algebraic topology or geometric combinatorics is a third possible application which has yet to be explored.

1.2 Installation

This package is written in GAP, Pascal, and C++, and is known to work with compiler versions at least g++ version 2.91.60, gpc version beta-19990118. For C++ we need fairly robust template support. The Pascal code uses some gnu specific features. Both depend on gmp version at least 2.0.2. For convenience we provide a copy of gmp. We do not have sufficient manpower to work on the support of any non-gnu compiler.

The package involves two external binaries, one for homology and one for just matrix computations. No other share package is needed. The Homology package must be installed in the `pkg` subdirectory of the GAP distribution, or you must invoke gap giving it's path (see gap manual).

2 Computing Simplicial Homology

The functions `SimplicialHomology` and `SimplicialBettiNumbers` are the main tools for computing homology. Structure of representations, e.g. the simplicial complex representation, and a few other functions are also described.

2.1 Functions and Data Types

1 ▶ `SimplicialHomology(` *simplicial complex*`,` *index range*`)`

A list, `[`*homology group* `... ]` of homology groups is returned. The groups in the list are $\tilde{H}_i$, reduced homology, for i in the specified *index range*. See the entry for *homology group* in Section `Options` for the available formats to represent a homology group. Also an *algorithm* option may be used to determine the choice of algorithm internally.

2 ▶ *simplicial complex*
▶ `IsSimplicialComplex(` *object* `)`

returns true if *object* can be interpreted as a simplicial complex, i.e. is a dense list of dense lists of objects. Saying it another way, a simplicial complex is represented as a list of its facets which are simplices. In turn a simplex is a list of vertices, and finally a vertex may be denoted by any gap object (number, string, or list).

For example the list `[["a", "b", "c"],["b", "c", "d"],["c","d","a"],["d","a","b"]]` denotes the triangulations of the 2-sphere given by the 2-simplex and

```
[
[ [ 1, 2 ], [ 1, 3 ], [ 2, 1 ] ], [ [ 1, 2 ], [ 1, 3 ], [ 2, 2 ] ],
[ [ 1, 1 ], [ 1, 3 ], [ 2, 1 ] ], [ [ 1, 1 ], [ 1, 3 ], [ 2, 2 ] ],
[ [ 1, 1 ], [ 1, 2 ], [ 2, 1 ] ], [ [ 1, 1 ], [ 1, 2 ], [ 2, 2 ] ]
],
```

denotes the triangulation of the 2-sphere constructed by `SCSuspension(SCSphere(1))` (i.e., suspending the 1-sphere).

3 ▶ *index range*

The *homology index range* is an optional parameter. The default is to compute $\tilde{H}_i$ for i from 0 to the dimension of the complex. It may be

an integer
: denoting the index of one homology group, or the list

`[a,b]`
: denoting the range from a to b inclusive.

4 ▶ `SimplicialBettiNumbers(`*simplicial complex*`,` *index range*`)`

Returns a list of the Betti numbers (the dimension of the torsion free part) of the homology in the specified *index range*. Arguments and options are as for `SimplicialHomology`.

5 ► `SimplicialNTorsionRanks(`*simplicial complex*`,` *n*`,` *index range*`)`

Returns a list of the ranks of the n-torsion of the homology groups for the specified index range. By rank of the n-torsion in group H, we mean $\max\{k|(Z/nZ)^k \text{lies in } H\}$. For example suppose the the group is $Z^7 \times (Z/2Z)^5 \times (Z/12Z) \times (Z/108Z)^2$. represented by list `[7, [2, 5],[12, 1], [108, 2]]`. Then the n-torsion rank is 8, for $n = 2$, the rank is 3 for $n = 3, 4, 6$, or 12, and is 2 for $n =$ any divisor of 108 containing 9.

Other arguments, options, result are as for `SimplicialHomology`.

6 ► `SimplicialBoundaryMaps(`*simplicial complex*`,` *index range*`,` *filename*`)`

Returns a list of (gap) matrix representations of the boundary maps ∂_i, for i in the specified *index range*. The indexing is such that H_i is the kernel of ∂_{i+1} mod the image of ∂_i. *filename* is an optional parameter. If *filename* is specified, the boundary map ∂_i is written in the sparse matrix format (and not in the gap matrix format) in the file *filename*. When *filename* is specified, the range should be only one index.

7 ► `SimplicialComplexDimension(` *simplicial complex* `)`

returns the largest dimension of any facet of the complex, i.e. one less than the largest number of vertices in any facet.

2.2 Options

1 ► `HomologyInfo(` *n* `)`

sets the homology info level to n. This influences the verbosity of the commentary written by homology's separate executable during the computation. It is equivalent to `SetInfoLevel(InfoHomology,` *n*`)`. You may also set the the info level for standard GAP `InfoTiming` and `InfoWarning` to get the corresponding kind of information from homology code.

2 ► *group formats*
► `SetHomologyGroupFormat(` *a* `)`

sets the format for the representation of a homology group to a.

The variables CyclicPSubgroups, MaximalCyclicSubgroups and GapFormat hold the allowable choices. The default value is that of `CyclicPSubgroups`.

► `HomologyGroupFormats`

a list of the allowed homology group options. Thus HomologyGroupFormats[i] is a valid for, $1 \leq i \leq 3$, for SetHomologyAlgorithm, and also for use as a one-time option. Thus

```
SimplicialHomology(SCMobiusStrip: HomologyGroupFormat:= MaximalCyclicSubgroups);
```

and

```
SimplicialHomology(SCMobiusStrip: HomologyGroupFormat := HomologyGroupFormats[2]);
```

are equivalent. While

```
SetHomologyGroupFormat ("MaximalCyclicSubgroups");
SimplicialHomology(SCMobiusStrip);
```

performs the same computation, but also keeps the option set for future computations.

The format choices are

1 `CyclicPSubgroups`. An (integer) homology group is given in terms of its maximal cyclic p-subgroups. The form is `[`*b*`, [`*p1*`,`*e1*`], [`*p2*`,`*e2*`], ... , [`*pk*`,`*ek*`]]` which denotes $Z^b \times (Z/p_1Z)^{e_1} \times (Z/p_2Z)^{e_2} \times \ldots \times (Z/p_kZ)^{e_k}$. The p_i are distinct and each p_i is a prime power. The torsion is listed starting with

powers of small primes to powers of large primes, within each prime the list is ordered by the size of the groups.

2 `MaximalCyclicSubgroups`. An (integer) homology group is given in terms of its maximal cyclic subgroups. The form is `[`b`, [`$n1$`,`$e1$`], [`$n2$`,`$e2$`], ... , [`nk`,`ek`]]` which denotes $Z^b \times (Z/n_1Z)^{e_1} \times (Z/n_2Z)^{e_2} \times \ldots \times (Z/n_kZ)^{e_k}$. The torsion is listed in increasing size, in fact the n_i are distinct and $n_1|n_2|\ldots|n_k$.

3 `GapFormat`. An (integer) homology group is given in terms of its maximal cyclic subgroups which are listed in a format such that the part specifying torsion can be used as an input to the Gap-function *AbelianGroup*. The form is `[`b`, [`$n1$`, `$n2$`, ... , `nk`]]` which denotes $Z^b \times (Z/n_1Z) \times (Z/n_2Z) \times \ldots \times (Z/n_kZ)$. The torsion is listed in weakly increasing size, in fact $n_1|n_2|\ldots|n_k$.

3 ► *Algorithm Choice*

► `SetHomologyAlgorithm(` a `)`

sets the algorithm option to a. This specifies which algorithm is to be used for computing rank and invariant factors (Smith form) of each boundary map. This influences the working of `SimplicialHomology`, `SimplicialBettiNumbers`, `SimplicialNTorsionRanks` described above and the working of the matrix functions described in chapter 7.

The variables EliminateAlgorithm, EliminateGMPAlgorithm, ValenceElimAlgorithm, and ValenceBBAlgorithm hold the allowable choices. The default value is that of `EliminateAlgorithm`.

► `HomologyAlgorithms`

a list of the allowed algorithm options. Thus HomologyAlgorithms[i] is a valid algorithm, $1 \leq i \leq 4$, for SetHomologyAlgorithm, and also for use as a one-time option. Thus

```
SimplicialHomology(SCMobiusStrip: HomologyAlgorithm := ValenceElimAlgorithm);
```

and

```
SimplicialHomology(SCMobiusStrip: HomologyAlgorithm := HomologyAlgorithms[3]);
```

are equivalent. While

```
SetHomologyAlgorithm("ValenceElimAlgorithm");
```

```
SimplicialHomology(SCMobiusStrip);
```

performs the same computation, but also keeps the option set for future computations.

Briefly, the algorithm choices are

1 `EliminateAlgorithm`. A variant of Gaussian elimination is used. Integers are represented as machine (word size) integers. Overflow is detected and the method returns `fail` if it occurs.

2 `EliminateGMPAlgorithm`. Similar to `EliminateAlgorithm` but switches to use of GMP integers when overflow occurs. It is slower by a factor of about 2 when word size integers would suffice.

3 `ValenceElimAlgorithm`. Uses iterative techniques to determine Betti numbers, and to determine which primes may occur in torsion. Uses elimination mod p to determine the powers of p in the torsion. May work when `EliminateGMPAlgorithm` runs out of memory because there is no growth of the matrix entries with computation mod p. However problems due to matrix fill-in may still occur in the mod p elimination steps. This algorithm is probabilistic. There is a small chance the Betti number may be wrong and a small chance that a torsion prime is missed. The trade-off between certainty and computing time may be driven toward greater certainty by setting the `UncertaintyTolerance` option to a small rational value such as 1/1000000. The algorithm is deterministic with respect to the accuracy of the torsion structure for all primes not missed. See chapter 7 for more comments on probabilistic algorithms.

4 `ValenceBBAlgorithm`. Uses iterative techniques exclusively. Memory usage is bounded above by a constant multiple of n, where n is the largest number of simplices in any single dimension. This is a huge

advantage over the previous methods in some cases. There are cases in which only this method avoids thrashing due to memory demand exceeding machine real memory, and hence only this method succeeds in running to completion. If your boundary matrices have as many as a few million nonzero entries, the algorithm is likely to run to completion in at most a few days (remark based on typical computer, year 2000). However, the method is usually slower than the elimination methods when memory is not the limiting factor. Also the results computed are incomplete. They are incomplete in that only the rank of p-torsion can be computed. When the previous three algorithms fail due to lack of memory, this one may be tried with with SimplicialBettiNumbers() and/or SimplicialNTorsionRanks(). SimplicialHomology() may fail when this algorithm is used because it cannot confirm absence of p^2-torsion for certain primes p. However, if HomologyInfo(n) is set with $n \geq 4$ the commentary will reveal which primes have torsion, their torsion ranks, and other useful information. We are working on improvements to complete the method by computing the rank of p^e torsion for exponent $e > 1$, or at any rate to prove p^2 torsion does not exist when that is the case.

4 ▶ *Certainty control*

▶ `SetUncertaintyTolerance(` ϵ `)`

sets the upper limit for the probability of error (probability of incorrect results) to ϵ, which should be a rational in (0, 1]. This option applies to the valence algorithms: ValenceElimAlgorithm and ValenceBBAlgorithm. The probability of erroneous results will be less than ϵ. The default value is 1, which means the algorithm will choose. Generally this choice will involve relatively low computing time while giving reasonable probability of success. In any case, lower bounds on probability of success are reported by the algorithm when HomologyInfo is set sufficiently high. (e.g. 3). These probabilities will be greater than 1 - ϵ.

▶ `UncertaintyTolerance`

a one-time option. Thus

```
SimplicialHomology( SCMobiusStrip: HomologyAlgorithm := ValenceElimAlgorithm,
UncertaintyTolerance := 1/1000000000 );
```

and

```
SetUncertaintyTolerance( 1/1000000000 );
```

```
SimplicialHomology(SCMobiusStrip: HomologyAlgorithm := ValenceElimAlgorithm);
```

performs the same computation, but the latter keeps the option set for future computations.

3 Example Complexes

3.1 Sample Triangulations of Basic Spaces

These variables and functions provide some example complexes, and complex families triangulating basic topological spaces.

1 ▶ `SCSphere(` n `)`

constructs a simplicial complex triangulating the n-sphere, $n \geq 0$. The triangulation is given as the boundary complex of the n-simplex.

2 ▶ `SCMobiusStrip, SCTorus, SCKleinBottle`

simplicial complexes triangulating the M"obius strip, the 2-torus and the Klein bottle.

3 ▶ `SCGeneralizedKleinBottle(` n `)`

constructs a simplicial complex which triangulates a generalization of the Klein bottle. Where the Klein bottle has 2-torsion in homology, GeneralizedKleinBottle(n) has n-torsion, for $n \geq 2$.

3.2 Graph Complexes

These functions provide some complexes which have been studied in the literature.

1 ▶ `SCChessboard(` m`,` n `)`

constructs a simplicial complex representing the m by n chessboard complex, where m and n are positive integers. Each simplex is a set of positions on the m by n board with no two pairs in the same row or in the same column. The complex is isomorphic to the complex of partial matchings of the complete bipartite graph $K_{n,m}$. Thus the facets have dimension min(m, n) - 1, and, for $m \geq n$, the chessboard complex has $m!/(m-n)!$ facets. [BLVŽ94], for example, gives further information and [BBL+99] for open problems concerning this complex.

2 ▶ `SCMatching(` n `)`

constructs a simplicial complex representing the m vertices matching complex, where m is a positive integers. Each simplex is a set of matchings of the complete graph K_n. Thus the facets have dimension [m/2] - 1. [BLVŽ94], for example, gives further information on this complexes.

3 ▶ `SCNot2ConnectedGraphs(` n `)`

constructs a simplicial complex whose facets are in 1-1 correspondence with the set of all not-2-connected graphs, for $n \geq 3$. Each facet corresponds to the set of edges of such a graph. The complex has $n(2^{n-2}-1)$ facets. In [BBL+99] and in parallel work by Turchin (see reference in [BBL+99]) the homology of this complex is shown to be concentrated in dimension $2n-5$ where it is free of rank $(n-2)!$.

4 Constructors for Partially Ordered Sets

These functions provide means to construct and manipulate partially ordered sets, posets for short.

1 ► `OrderRelationToPoset( set , order_relation )`

The functions takes a *set* and an *order_relation* on this set and constructs the representation of the corresponding poset.

2 ► *order_relation*

An *order_relation* on a *set* is a function from *set* × *set* with values *true* and *false* which represents a partial on *set*. It takes the value *true* if the second argument is smaller or equal to the first and *false* otherwise.

3 ► *poset*

A *poset* is represented by its Hasse-diagram. The elements of the poset are numbered by integers where an additional least element with number 1 and an additional largest element numbered *Number* (*poset*) +2 are added. The Hasse-diagram of this extended poset is then given by a list *list* of lists where the entry *list*[i] is the list of elements covering the element i. In particular, the first entry in the *poset* is a list of the minimal elements and the last entry is the empty list. For example the list $[[2,3],[4,5],[5]\cdot[]]$ represents the poset consisting of the 4 elements $2,3,4,5$ such that $2<2,5$, $3<5$ and no other order relations hold. The element 1 is the unique minimal element and the element 6 the unique maximal element.

5 Sample Posets

These variables and functions provide some example posets.

5.1 Examples from Combinatorics

1 ► `Subsets ( set )`

Constructs a list of all subsets of the set *set*.

2 ► `NonTrivialSubsets ( set )`

Constructs a list of all subsets of the set *set* excluding the empty set and the set *set* itself. The poset corresponding to the non-trivial subsets ordered by the inclusion IsSubset(*set1*, *set2*) is isomorphic to the poset of non-trivial faces of the n for n the cardinality of *set* -1. Hence its order complex is a triangulation of a sphere (barycentric subdivision of the simplex). Therefore, homology is concentrated in dimension $n-1$ and is free of rank 1. Open questions arise when studying certain subposet of this poset (see [BW97]).

3 ► `NonTrivialPartitionsSet ( set )`

Constructs a list of all partitions of the set *set* excluding the partitions into singletons and into a single block.

4 ► `IsSetPartitionRefinement ( partition1 , partition2 )`

Takes as arguments two set partitions and returns *true* if *partition2* is a refinement of *partition1* and *false* otherwise. The homology of the order complex of the poset corresponding to NonTrivialPartitionsSet (*set*) with this order relation is known (see for example [Sta82]) to be concentrated in dimension $n-3$ where it is free of rank $(n-1)!$ where n is the cardinality of *set*. For open questions on subposets of this poset see [Bjö94].

5 ► `PartitionsNumber ( n )`

Constructs a list of all partitions of the number n.

6 ► `NonTrivialPartitionsNumber ( n )`

Constructs a list of all partitions of the number n excluding the partitions into 1's and into the single block n.

7 ► `IsNumberPartitionRefinement ( partition1 , partition2 )`

Takes as arguments two number partitions and returns *true* if *partition2* is a refinement of *partition1* and *false* otherwise. The poset corresponding to the non-trivial number partitions ordered by this order relation has $(2,1,\cdots,1)$ is its least element and hence its order complex has vanishing homology since it is a cone. Intervals in this poset still bear one of the big challenges in the area of poset homology. See [Zie96] for the very little that is known here.

8 ► `Compositions ( n )`

Returns a list of all "compositions" of the number n. In GAP compositions are called "ordered partitions". Since composition is the terminology used mostly in combinatorics we use this terminology too.

9 ▶ `NonTrivialCompositions ( ` *n* ` )`

Returns a list of all compositions of the number n excluding the composition into 1's and the composition into the single block n.

10 ▶ `IsCompositionRefinement ( ` *composition1* `, ` *composition2* ` )`

Returns *true* if *composition2* is a refinement of *composition1* and *false* otherwise. The poset corresponding to NonTrivialCompositions (n) ordered by this order relation is isomorphic to the poset of non-trivial subsets of a set of cardinality $n-1$. Hence the homology of its order complex is concentrated in dimension $n-3$ where it is free of rank 1. Open questions on this poset can be found in [SW98].

5.2 Examples from Group Theory

1 ▶ `LatticeOfSubgroups( ` *group* ` )`

Constructs the poset, which is actually a lattice, of subgroups of the group *group*. For solvable groups the homology of the order complex of this poset is known [KT85]. In general, except for particular groups the question is wide open. The function invokes the GAP function *LatticeSubgroups* in order to get hold of the set of subgroups of the group *group*.

2 ▶ `PosetOfCosets ( ` *group* ` )`

Constructs the poset of all cosets of subgroups of the group *group*. For solvable groups the homology of the order complex of this poset is known [Wel92]. In general, except for particular groups the question is wide open. The function invokes the GAP function *LatticeSubgroups* in order to get hold of the set of conjugacy classes of subgroups of the group *group*.

3 ▶ `PosetOfConjugacyClasses ( ` *group* ` )`

Constructs the poset of conjugacy classes of subgroups of the group *group*. Here a conjugacy class `[H]` of the subgroup H is smaller than a conjugacy class `[U]` of a subgroup U if there is a conjugate H^g which is a subgroup of U. For solvable groups the homology of the order complex of this poset is known [Bro99]. In general, except for particular groups the question is wide open. The function invokes the GAP function *LatticeSubgroups* in order to get hold of the set of subgroups of the group *group*.

4 ▶ `QuillenPoset ( ` *group* ` , ` *prime* ` )`

Constructs the poset of nontrivial elementary Abelian subgroups of prime power order for the prime *prime*. Quillen conjectured [Qui78] the the poset id contractible (which implies vanishing homology) if and only if there is non-trivial normal *prime*-subgroup. This conjecture has been settles for "most" cases [AS93]. But the exact homology has been determined in very few cases only. The function invokes the GAP function *LatticeSubgroups* in order to get hold of the set of subgroups of the group *group*.

5 ▶ `IsElementaryAbelianPGroup ( ` *group* ` , ` *prime* ` )`

Returns *true* if *group* is elementary Abelian of prime power order for the prime p.

6 Operators on Complexes

These functions provide means to construct new complexes from existing ones. The mathematical definitions performed by the various functions is given in the Appendix (see also the book by Munkres [Mun84] and the article by Björner [Bjö95]).

6.1 New Complexes from Old

1 ▶ `SCWedge(` [*simplicial complex*, ..] `)`

constructs a simplicial complex representing the wedge product of the complexes in the argument list.

2 ▶ `SCJoin(` *sc1*, *sc2* `)`

constructs a simplicial complex representing the join of simplicial complexes *sc1* and *sc2*.

3 ▶ `SCCone(` *sc* `)`

constructs a simplicial complex representing the cone over *sc*.

4 ▶ `SCSuspension(` *sc* `)`

constructs the suspension over *sc*.

5 ▶ `SCStar (` *sc* , *f* `)`

constructs the star of the face f in the complex *sc*.

6 ▶ `SCLink (` *sc* , *f* `)`

constructs the link of the face f in the complex *sc*.

7 ▶ `SCRestriction (` *sc* , *f* `)`

constructs the restriction of the in the complex *sc* to the set f..

8 ▶ `SCAlexanderDual (` *sc* `)`

constructs the Alexander dual of the complex *sc*.

9 ▶ `SCDeletedJoin (` *sc1* , *sc2* `)`

constructs the deleted join of the complex *sc1* and *sc2*.

6.2 Posets and Complexes

1 ▶ `OrderComplex (` *p* `)`

constructs the order complex of the poset p.

6.3 Modifying the Representation of a Complex

1 ▶ `SCCompactify (` *sc* `)`

deletes redundant faces from the list of faces defining *sc*.

7 Matrix Functions

Functions for direct computation of matrix Smith forms and ranks are described here. These functions give direct access to the matrix computations which underlie the homology tools given earlier. They are intended for application to sparse matrices, hence the prefix `SM`, and they are implemented with respect to the Valence algorithms only. GAP's matrix format is *dense*, that is it stores every entry. This is inappropriate for very large sparse matrices for which it is desirable to store only the nonzero entries. The sparse representation is supported by allowing your matrix to be either a GAP matrix or to be stored in sparse format in a file. Function `gap2SM` converts from the former to the latter.

1 ► `gap2SM(`*matrix*`, `*filename*`)`

The *matrix* in GAP format is written to a file named by string *filename*. The file is created or overwritten. A sparse matrix format is used which consists of a series of triples. The first triple is number of rows, number of columns, and the letter "M". The last triple is "0 0 0" used as an end marker. The intervening triples signify non-zero entries and have the form row-index, col-index, value. These triples must be in increasing lexicographical order. Row column pairs must be distinct. All values must be machine integers.

For example :

32 3200 M
1 1 345
1 2 -23
2 1 77
2 200 31
30 3101 11
0 0 0

denotes a 32 by 3200 matrix with 5 non-zero entries, three of which are clustered at the top left.

7.1 Invariant Factors and Rank Functions

The remaining `SM` functions take the matrix in either form. The InfoHomology, InfoTiming, InfoWarning options are relevant as are UncertaintyTolerance and HomologyAlgorithm options. For the latter only ValenceElimAlgorithm (default) and ValenceBBAlgorithm are appropriate.

1 ► `SMInvariantFactors(`*matrix or file*`:HomologyAlgorithm := `*alg*`)`

returns a list of the form `[ r, c, [f_1,n_1],[f_2, n_2], ... [f_k,n_k]]`, where r and c are the numbers of rows and columns of the matrix respectively, and the f_i are the distinct non-zero invariant factors (Smith form diagonal entries) and n_i are the multiplicity of the f_i. Thus the sum of the n_i is the rank of the matrix.

2 ► `SMSmithForm(`*matrix or file*`:HomologyAlgorithm := `*alg*`)`

returns the same information as SMInvariantFactors, but in expanded form. The format is `[ r, c, [s_1 .. s_min(r,c)]]`, where again r and c denote the number of rows and columns. The third item is the diagonal of the Smith form (it includes the zero entries).

3 ▶ `SMIntegerRank(`*matrix or file*`:HomologyAlgorithm := `*alg*`)`

returns the rank of the matrix over Z. For large matrices, HomologyAlgorithm := ValenceElimAlgorithm is likely to be fastest and be significantly faster than computing the entire Smith form and determining the rank from the result.

4 ▶ `SMPrimeRank(`*matrix or file*`, `p`:HomologyAlgorithm := `*alg*`)`

returns the rank of the matrix mod p, i.e. over Z/pZ. For large matrices, HomologyAlgorithm := ValenceElimAlgorithm is likely to be fastest and be significantly faster than computing the entire Smith form and determining the rank from the result.

5 ▶ `SMPrimePowerRank(`*matrix or file*`, `n`:HomologyAlgorithm := `*alg*`)`

returns the "rank" of the matrix mod n, for n a power of a prime. This is not the McCoy rank (largest i such that the determinantal divisor d_i is nonzero mod n), but is the largest index i such that the i-th invariant factor (smith form entry), $s_i = d_i/d_{i-1}$, is non-zero. For large matrices, HomologyAlgorithm := ValenceElimAlgorithm is likely to be fastest.

Currently ValenceBBAlgorithm cannot compute prime power rank, but when the other algorithms fail due to memory limitations, it may be the only algorithm that can determine which primes occur in the invariant factors and what the prime ranks are.

7.2 On Uncertainty: Why you should trust our probabilistic algorithms

We remark that the valence algorithms, while probabilistic, compute exact, not approximate, results. Typically, random choices are made within the algorithm to *precondition* the input matrix to one for which a fast method will work, or to transform the problem to a simpler one. With some probability this preconditioning or transform may not map correctly to the domain on which the fast method is valid. Upper bounds are known for the worst case probability of error. You may choose a probability of error less than one in a million, one in a billion, etc. as you wish, [DSV]. The algorithms then perform the randomizations in such a way as to assure at most this probability of error for your input matrix.

It is our view that the probabilistic results should be accepted as much as the computational results of deterministic algorithms. By that statement we mean that the correctness of the results is, in all essentials, as certain as that of results from computer runs of implementations of deterministic algorithms.

Part of this view is the belief that no computational result can be taken prima facie as a mathematically proven result. This is because the tools are never fully proven. The mathematical algorithm and even it's implementation, say in C, may be proven rigorously, but always relative to assumptions about the compiler and hardware, unproven for the compiler and hardware actually used.

Given this state of affairs regarding proof and given the fact of life that all code of any complexity has bugs, it is insignificant whether or not the code has a one in a million chance of wrong result due to the particular value of a (pseudo)random variable used. More than one of the last million proofs/algorithms you've examined were wrong, isn't is so? In fact probabilistic code which is significantly simpler in structure than a deterministic alternative is likely to have fewer bugs in its implementation and in its correctness proof. Hence it should be trusted **more** than that alternative.

In a few cases we do not compute in a sufficiently large domain to be able to apply the known bounds on the probability of error. In the cases where this occurs, we consider the preconditioning method to be a heuristic, and the algorithm commentary reports that the probability of correctness is unknown. Experience shows that such heuristic results are usually correct with the Valence algorithms. However. we suggest that greater doubt be applied to such heuristically obtained results than to the probabilistic or deterministic results.

8 Sample Session

In the chapter we demonstrate in a sample session the use of some of the functions of the package described previously.

8.1 Constructing Simplicial Complexes and Computing their Homology

First, start GAP and load the package.

```
welker@merak:~> gap

           #########           ######         ###########           ###
        #############          ######         ############         ####
       ##############         ########        #############       #####
      ###############         ########        #####   ######      #####
     ######         #         #########       #####    #####     ######
    ######                   ##########       #####    #####    #######
    #####                    ##### ####       #####   ######   ########
    ####                    #####  #####      #############   ###  ####
    #####     #######       ####    ####      ###########    ####  ####
    #####     #######      #####    #####     ######        ####   ####
    #####     #######      #####    #####     #####         #############
     #####      #####     ################    #####         #############
     ######     #####     ################    #####         #############
     ################    ##################   #####                ####
      ###############    #####        #####   #####                ####
        #############    #####        #####   #####                ####
         #########      #####          #####  #####                ####

    Information at:  http://www-gap.dcs.st-and.ac.uk/~gap
    ? for help. Copyright and authors list by ?copyright, ?authors

   Loading the library. Please be patient, this may take a while.
GAP4, Version: 4.1 fix 2 of 27-Aug-1999, sparc-sun-solaris2.6-gcc
Components:  small, small2, small3, id2, id3, trans, prim, tbl,
             tom  installed.
gap>  RequirePackage("homology");
true
```

Now the functionality of the package is available to you. Lets reconfirm that the 5-gon has the homology concentrated in dimension 1 where it is free of rank 1.

```
gap> complex:=[[1,2],[2,3],[3,4],[4,5],[1,5]];
[ [ 1, 2 ], [ 2, 3 ], [ 3, 4 ], [ 4, 5 ], [ 1, 5 ] ]
```

Now *complex* is the simplicial complex whose maximal faces are the maximal faces of the 5-gon. We are ready to calculate the homology of the 5-gon.

```
gap> SimplicialHomology(complex);
#I:D Simplicial complex of dimension 1 with 5 facets
#I:D   homology: Computing homology groups
#I:D      faces: Finding faces of dimension 0
#I:D      faces: Found 5 faces of dimension 0
#I:D      faces: Finding faces of dimension 1
#I:D      faces: Found 5 faces of dimension 1
#I:R   homology: H_0 = 0.
#I:R   homology: H_1 = 1.
#I:D   homology: Homology groups computed
[ [ 0 ], [ 1 ] ]
```

Indeed, the the homology is as expected. While computing the package (if in default mode) has told us that we have constructed a complex of dimension 1 which has 5 faces of dimension 1 and 5 of dimension 0.

Let us turn to a more complicated example. Historically this is actually the first example where predecessors of the program were applied successfully. Let us calculate the homology of the complex of not 2-connected graphs on 7 vertices. By [BBL+99] it is known that the homology is concentrated in dimension $2 \cdot 7 - 5 = 9$ and is free of dimension $(7-2)! = 120$. So let us reconfirm the homology in dimension 9 only.

```
gap> SimplicialHomology(SCNot2ConnectedGraphs(7),9);
#I:D Simplicial complex of dimension 15 with 217 facets
#I:D   homology: Computing homology groups
#I:D      faces: Finding faces of dimension 8
#I:D      faces: Found 247100 faces of dimension 8
#I:D      faces: Finding faces of dimension 9
#I:D      faces: Found 219135 faces of dimension 9
#I:D   homology: Finding rank of boundary map d9
#I:D      elimination: Triangulating matrix by modified Gauss elimination
#I:D      elimination: current torsion-free rank: 0, current rank: 0, max. rank: 247100
#I:D      elimination: current torsion-free rank: 715, current rank: 715, max. rank: 247100
```

The last line tells us that rows of the matrix of the 9th differential processed so far have a span of rank 715. Using general implications from the fact that the sequence of differential is a complex (i.e., $\partial_{i-1} \circ \partial_i = 0$) the maximal possible rank this matrix can have is 247100. Let us look at the output a few minutes later.

```
#I:D      elimination: current torsion-free rank: 128311, current rank: 128311, max. rank: 247100
#I:D      elimination: current torsion-free rank: 128330, current rank: 128330, max. rank: 247100
#I:D      elimination: Matrix triangulated
#I:R   homology: Rank d9 = 128330
#I:D      faces: Finding faces of dimension 10
#I:D      faces: Found 135765 faces of dimension 10
#I:D   homology: Finding rank of boundary map d10
#I:D      elimination: Triangulating matrix by modified Gauss elimination
#I:D      elimination: current torsion-free rank: 0, current rank: 0, max. rank: 90805
```

The 9th differential has been brought into Smith Normal form. But for calculation of the homology group in homological dimension 9 the program has already started to analyze the matrix of the 10th differential. Another few minutes waiting bring us to:

```
#I:D      elimination: current torsion-free rank: 90344, current rank: 90344, max. rank: 90805
#I:D      elimination: current torsion-free rank: 90685, current rank: 90685, max. rank: 90805
#I:D      elimination: Matrix triangulated
#I:R   homology: Rank d10 = 90685
#I:R   homology: H_9 = 120.
#I:D   homology: Homology groups computed
[ [ 120 ] ]
```

After Smith Normal forms of the 9th and 10th differential are calculated the program determines the homology group we have asked for and indeed it is as expected. Note that in the process of calculating this group we have brought within roughly 10 minutes (on a Ultra 10 with 64 MB) two matrixes into Smith Normal form whose row and column sizes are in the 5 or 6 digit numbers. Even though less than 0,1 percent of row entries are different from 0 this is surprising since it is not for granted that in the elimination process the matrices will not eventually become dense.

Simplicial complexes that are of interest in combinatorics often come up as order complexes of partially ordered sets. The next part of the session shows how complexes of that kind can be constructed.

```
gap> Pi4:=NonTrivialPartitionsSet([1..4]);
[ [ [ 1 ], [ 2 ], [ 3, 4 ] ], [ [ 1 ], [ 2, 3 ], [ 4 ] ], [ [ 1 ], [ 2, 3, 4 ] ],
  [ [ 1 ], [ 2, 4 ], [ 3 ] ], [ [ 1, 2 ], [ 3 ], [ 4 ] ], [ [ 1, 2 ], [ 3, 4 ] ],
  [ [ 1, 2, 3 ], [ 4 ] ], [ [ 1, 2, 4 ], [ 3 ] ], [ [ 1, 3 ], [ 2 ], [ 4 ] ],
  [ [ 1, 3 ], [ 2, 4 ] ], [ [ 1, 3, 4 ], [ 2 ] ], [ [ 1, 4 ], [ 2 ], [ 3 ] ],
  [ [ 1, 4 ], [ 2, 3 ] ] ]
```

We have defined a set *Pi4* whose elements are the non-trivial partitions of the set $\{1, \ldots, 4\}$. In order to impose an order relation on that set we need a function that takes two elements of the set as arguments and returns *true* or *false* as the answer to the questions whether the second argument is smaller or equal to the first. For set-partitions and the refinement as an order relation this function is implemented and called *IsSetPartitionRefinement*. Thus we are in position to convert the set *Pi4* into a partially ordered set as represented in our package.

```
gap> PosetPi4:=OrderRelationToPoset(Pi4,IsSetPartitionRefinement);
[ [ 2, 3, 5, 6, 10, 13 ], [ 4, 7, 12 ], [ 4, 8, 14 ], [ 15 ], [ 4, 9, 11 ], [ 7, 8, 9 ], [ 15 ],
  [ 15 ], [ 15 ], [ 8, 11, 12 ], [ 15 ], [ 15 ], [ 9, 12, 14 ], [ 15 ], [  ] ]
gap> OCPi4:=OrderComplex(PosetPi4);
[ [ 2, 4 ], [ 2, 7 ], [ 2, 12 ], [ 3, 4 ], [ 3, 8 ], [ 3, 14 ], [ 5, 4 ], [ 5, 9 ], [ 5, 11 ],
  [ 6, 7 ], [ 6, 8 ], [ 6, 9 ], [ 10, 8 ], [ 10, 11 ], [ 10, 12 ], [ 13, 9 ], [ 13, 12 ],
  [ 13, 14 ] ]
```

After converting the set *Pi4* into a poset *PosetPi4* the elements are no longer set-partitions but rather numbers. So far we have not yet implemented functions that convert single numbers into poset elements and vice versa. Such a function is projected for the next release. After we have the poset in hand the function *OrderComplex* constructs the order complex of the poset. Now we can calculate homology.

```
gap> SimplicialHomology(OCPi4);
#I:D Simplicial complex of dimension 1 with 18 facets
#I:D   homology: Computing homology groups
#I:D      faces: Finding faces of dimension 0
#I:D      faces: Found 13 faces of dimension 0
#I:D      faces: Finding faces of dimension 1
#I:D      faces: Found 18 faces of dimension 1
#I:D   homology: Finding rank of boundary map d1
#I:D      elimination: Triangulating matrix by modified Gauss elimination
#I:D      elimination: current torsion-free rank: 0, current rank: 0, max. rank: 12
```

```
#I:D      elimination: current torsion-free rank: 12, current rank: 12, max. rank: 12
#I:D      elimination: Matrix triangulated
#I:R   homology: Rank d1 = 12
#I:R   homology: H_0 = 0.
#I:R   homology: H_1 = 6.
#I:D   homology: Homology groups computed
[ [ 0 ], [ 6 ] ]
```

For the classes of posets which are implemented in the package and come from group theory the order relation is clear and the step from the set with order relation to the poset is hard coded. For example this is done for the poset of p-subgroups of a finite group G.

```
S2S4:=QuillenPoset(SymmetricGroup(4),2);
[ [ 2, 3, 4, 5, 6, 7, 8, 9, 10 ], [ 15, 16 ], [ 15, 17 ], [ 15, 18 ], [ 16 ], [ 17 ], [ 18 ],
  [ 17 ], [ 18 ], [ 16 ], [ 30 ], [ 30 ], [ 30 ], [ 30 ], [ 30 ], [ 30 ], [ 30 ], [ 30 ], [ 30 ],
  [ 30 ], [ 30 ], [ 30 ], [ 30 ], [ 30 ], [ 30 ], [ 30 ], [ 30 ], [ 30 ], [ 30 ], [  ] ]
gap> OC:=OrderComplex(S2S4);
[ [ 2, 15 ], [ 2, 16 ], [ 3, 15 ], [ 3, 17 ], [ 4, 15 ], [ 4, 18 ], [ 5, 16 ], [ 6, 17 ],
  [ 7, 18 ], [ 8, 17 ], [ 9, 18 ], [ 10, 16 ] ]
```

We have constructed the order complex of all non-trivial p-subgroups of the symmetric group on 4 letters for $p = 2$.

```
gap> SimplicialHomology(OC);
#I:D Simplicial complex of dimension 1 with 12 facets
#I:D   homology: Computing homology groups
#I:D      faces: Finding faces of dimension 0
#I:D      faces: Found 13 faces of dimension 0
#I:D      faces: Finding faces of dimension 1
#I:D      faces: Found 12 faces of dimension 1
#I:D   homology: Finding rank of boundary map d1
#I:D      elimination: Triangulating matrix by modified Gauss elimination
#I:D      elimination: current torsion-free rank: 0, current rank: 0, max. rank: 12
#I:D      elimination: current torsion-free rank: 12, current rank: 12, max. rank: 12
#I:D      elimination: Matrix triangulated
#I:R   homology: Rank d1 = 12
#I:R   homology: H_0 = 0.
#I:R   homology: H_1 = 0.
#I:D   homology: Homology groups computed
[ [ 0 ], [ 0 ] ]
```

This is not surprising. By simple arguments one can show that whenever there is a non-trivial normal p-subgroup then the order complex of the poset of non-trivial p-subgroups is contractible and hence has trivial homology. Clearly, the Kleinian 4-group contained in the alternating group on 4 letters is a normal 2-subgroup of the symmetric group.

8.2 Using the Valence algorithm

We turn to another example: let us calculate the homology of the complex of matchings of the complete graph with 11 vertices.

```
gap> mk11 := SCMatching(11);
[ [[1,2],[3,4],[5,6],[7,8],[9,10]],
[[1,2],[3,4],[5,6],[7,8],[9,11]],
[[1,2],[3,4],[5,6],[7,8],[10,11]],
[[1,2],[3,4],[5,6],[7,9],[8,10]],
[[1,2],[3,4],[5,6],[7,9],[8,11]],
[[1,2],[3,4],[5,6],[7,9],[10,11]],
[[1,2],[3,4],[5,6],[7,10],[8,9]],
```

This is a pretty big complex so we have truncated the output. We have now defined the complex, let us compute its homology.

```
gap> SimplicialHomology(mk11);
#I:D Simplicial complex of dimension 4 with 10395 facets
#I:D   homology: Computing homology groups
#I:D      faces: Finding faces of dimension 0
#I:D      faces: Found 55 faces of dimension 0
#I:D      faces: Finding faces of dimension 1
#I:D      faces: Found 990 faces of dimension 1
#I:D   homology: Finding rank of boundary map d1
#I:R   homology: Rank d1 = 54
#I:R   homology: H_0 = 0.
#I:D      faces: Finding faces of dimension 2
#I:D      faces: Found 6930 faces of dimension 2
#I:D   homology: Finding rank of boundary map d2
#I:R   homology: Rank d2 = 936
#I:R   homology: H_1 = 0.
#I:D      faces: Finding faces of dimension 3
#I:D      faces: Found 17325 faces of dimension 3
#I:D   homology: Finding rank of boundary map d3
#I:R   homology: Rank d3 = 5994
#I:R   homology: H_2 = 0.
#I:D      faces: Finding faces of dimension 4
#I:D      faces: Found 10395 faces of dimension 4
#I:D   homology: Finding rank of boundary map d4
#I:D      elimination: Triangulating matrix by modified Gauss elimination
#I:D      elimination: current torsion-free rank: 0, current rank: 0, max. rank: 11331
#I:D      elimination: current torsion-free rank: 2998, current rank: 2998, max. rank: 11331
#I:D      elimination: current torsion-free rank: 6983, current rank: 6983, max. rank: 11331
#I:D      elimination: current torsion-free rank: 8966, current rank: 8966, max. rank: 11331
#I:E      elimination: Coefficient overflow! Please use the GMP algorithm.
fail
```

After a few seconds the algorithm outputs that it failed. This is due to coefficient growth in the Smith form computation, we should switch to one of the other algorithms. As we realize now that we may have some difficulties, we set higher info levels for homology and timings, and try the algorithm using Gnu MultiPrecision integers (GMP).

```
gap> SetInfoLevel(InfoTiming,15);
gap> SetInfoLevel(InfoHomology,15);
gap> SimplicialHomology(mk11:HomologyAlgorithm:=EliminateGMPAlgorithm);
#I:D Simplicial complex of dimension 4 with 10395 facets
#I:R   homology: Rank d1 = 54
#I:R   homology: H_0 = 0.
```

```
#I:R   homology: Rank d2 = 936
#I:R   homology: H_1 = 0.
#I:R   homology: Rank d3 = 5994
#I:R   homology: H_2 = 0.
#I:D      faces: Finding faces of dimension 4
#I:D      faces: Found 10395 faces of dimension 4
#I:D   homology: Finding rank of boundary map d4
#I:D      elimination: Triangulating matrix by modified Gauss elimination
#I:D      elimination: current torsion-free rank: 0, current rank: 0, max. rank: 11331
#I:D      elimination: current torsion-free rank: 2998, current rank: 2998, max. rank: 11331
#I:S      elimination: 7000/10395 rows done; Elapsed:0.958s; 0.194123s to expected completion.
#I:D      elimination: current torsion-free rank: 6983, current rank: 6983, max. rank: 11331
#I:S      elimination: 9000/10395 rows done; Elapsed:1.562s; 0.0269255s to expected completion.
#I:D      elimination: current torsion-free rank: 8966, current rank: 8966, max. rank: 11331
#I:S      elimination: 10000/10395 rows done; Elapsed:87.58s; 85.561s to expected completion.
#I:D      elimination: current torsion-free rank: 9851, current rank:9926, max. rank: 11331
^C
```

Then, the output shows that the last rows are indeed quite hard to compute, the algorithm even seems to be stuck on the last 300 rows, so we stopped the computation (the actual computation would have completed but in more than an hour an a half); We want to try the Valence algorithm which is designed to deal with coefficient growth.

```
gap> SimplicialHomology(mk11:HomologyAlgorithm:=ValenceElimAlgorithm);
#I:D Simplicial complex of dimension 4 with 10395 facets
#I:D   homology: Computing homology groups
#I:D      faces: Finding faces of dimension 0
#I:D      faces: Found 55 faces of dimension 0
#I:D      faces: Finding faces of dimension 1
#I:D      faces: Found 990 faces of dimension 1
#I:D      SNFV: Smith Normal Form of Integer Matrix using Valence algorithm
#I:D      SNFV: Loading the matrix ...
#I:D         Smith: Smith Form Computation
#I:D            Int Vale: Integer Valence
#I:D               Gersh: Cassini Ovals method, to create a bound on the coefficients of A^t . A
#I:R               Gersh: Cassini Ovals eigen value bound : 72
#I:T               Gersh: cpu  time : 0.004 seconds
#I:R            Int Vale: Min Poly Degree : 2
#I:D            Int Vale: Valence is 1540, correct with probability at least 1 - 7.457251e-04
#I:T            Int Vale: cpu  time : 0.025 seconds
#I:R         Smith: rank mod 2 : 54
#I:R         Smith: rank mod 5 : 54
#I:R         Smith: rank mod 7 : 54
#I:R         Smith: rank mod 11 : 54
#I:R         Smith: Integer rank (mod 5) : 54, correct with probability at least 1 - 7.457251e-04
#I:R      SNFV: Smith form correct with probability at least 1 - 7.457251e-04
#I:T      SNFV: cpu  time : 0.09 seconds
#I:R   homology: H_0 = 0.
#I:D      faces: Finding faces of dimension 2
#I:D      faces: Found 6930 faces of dimension 2
#I:D      SNFV: Smith Normal Form of Integer Matrix using Valence algorithm
#I:D      SNFV: Loading the matrix ...
#I:D         Smith: Smith Form Computation
```

```
#I:D           Int Vale: Integer Valence
#I:D              Gersh: Cassini Ovals method, to create a bound on the coefficients of A^t . A
#I:R              Gersh: Cassini Ovals eigen value bound : 63
#I:R           Int Vale: Min Poly Degree : 4
#I:D           Int Vale: Valence is 438900, correct with probability at least 1 - 4.027169e-06
#I:R        Smith: rank mod 19 : 936
#I:R        Smith: rank mod 2 : 936
#I:R        Smith: rank mod 3 : 936
#I:R        Smith: rank mod 5 : 936
#I:R        Smith: rank mod 7 : 936
#I:R        Smith: rank mod 11 : 936
#I:R        Smith: Integer rank (mod 5) : 936, correct with probability at least 1 - [redacted]027169e-06
#I:R     SNFV: Smith form correct with probability at least 1 - 4.027169e-06
#I:T     SNFV: cpu  time : 15.936 seconds
#I:R  homology: H_1 = 0.
#I:D     faces: Found 17325 faces of dimension 3
#I:D     SNFV: Smith Normal Form of Integer Matrix using Valence algorithm
#I:D     SNFV: Loading the matrix ...
#I:D        Smith: Smith Form Computation
#I:D           Int Vale: Integer Valence
#I:D              Gersh: Cassini Ovals method, to create a bound on the coefficients of A^t . A
#I:R              Gersh: Cassini Ovals eigen value bound : 40
#I:R           Int Vale: Min Poly Degree : 6
#I:D           Int Vale: Valence is 5266800, correct with probability at least 1 - 4.295011e-08
#I:R        Smith: rank mod 19 : 5994
#I:R        Smith: rank mod 2 : 5994
#I:R        Smith: rank mod 3 : 5994
#I:R        Smith: rank mod 5 : 5994
#I:R        Smith: rank mod 7 : 5994
#I:R        Smith: rank mod 11 : 5994
#I:R        Smith: Integer rank (mod 5) : 5994, correct with probability at least 1 - [redacted]295011e-08
#I:R     SNFV: Smith form correct with probability at least 1 - 4.295011e-08
#I:T     SNFV: cpu  time : 415.322 seconds
#I:R  homology: H_2 = 0.
#I:D     faces: Finding faces of dimension 4
#I:D     faces: Found 10395 faces of dimension 4
#I:T     faces: cpu  time : 0.123 seconds
#I:D     SNFV: Smith Normal Form of Integer Matrix using Valence algorithm
#I:D     SNFV: Loading the matrix ...
#I:D        Smith: Smith Form Computation
#I:D           Int Vale: Integer Valence
#I:D              Gersh: Cassini Ovals method, to create a bound on the coefficients of A^t . A
#I:R              Gersh: Cassini Ovals eigen value bound : 15
#I:R           Int Vale: Min Poly Degree : 8
#I:D           Int Vale: Valence is 277200, correct with probability at least 1 - 1.806114e-05
#I:D           Gauss: Elimination of 10395 rows mod 2.
#I:S           Gauss: 1000/10394 rows done;
#I:S           Gauss: 2000/10394 rows done;
#I:S           Gauss: 3000/10394 rows done; Elapsed:10.708s; 24.733s to expected completion.
#I:S           Gauss: 4000/10394 rows done; Elapsed:13.187s; 19.9329s to expected completion.
#I:S           Gauss: 5000/10394 rows done; Elapsed:16.086s; 17.0339s to expected completion.
#I:S           Gauss: 6000/10394 rows done; Elapsed:19.371s; 13.7489s to expected completion.
```

```
#I:S           Gauss: 7000/10394 rows done; Elapsed:22.278s; 8.86765s to expected completion.
#I:S           Gauss: 8000/10394 rows done; Elapsed:24.922s; 3.89436s to expected completion.
#I:S           Gauss: 9000/10394 rows done; Elapsed:27.757s; 3.10843s to expected completion.
#I:S           Gauss: 10000/10394 rows done; Elapsed:47.765s; 13.8515s to expected completion.
#I:R        Smith: rank mod 2 : 10143
#I:R        Smith: rank mod 3 : 10098
#I:R        Smith: rank mod 5 : 10143
#I:R        Smith: rank mod 7 : 10143
#I:R        Smith: rank mod 11 : 10143
#I:R        Smith: Integer rank (mod 5) : 10143,
#I:R               correct with probability at least 1 - 1.806114e-05
#I:R        Smith: rank mod 9 : 10143
#I:R     SNFV: Smith form correct with probability at least 1 - 1.806114e-05
#I:T     SNFV: cpu  time : 639.243 seconds
#I:R  homology: H_3 = 1188 + Z_3^45.
#I:R  homology: H_4 = 252.
#I:D  homology: Homology groups computed
#I:T  homology: cpu  time : 1073.95 seconds
[ [ 0 ], [ 0 ], [ 0 ], [ 1188, [ 3, 45 ] ], [ 252 ] ]
```

And with this algorithm we were able to succeed after only 18 minutes. On the other hand, the answer is probabilistic, that is to say it has a chance of being false, but the algorithm tells us that there is less that one chance in a thousand that this is the case! Anyway, we would like to be more confident about this result, therefore we set the `UncertaintyTolerance` to a lower level, and try again.

```
gap> SetUncertaintyTolerance(10^-20);
gap> SimplicialHomology(mk11:HomologyAlgorithm:=ValenceElimAlgorithm);
#I:D Simplicial complex of dimension 4 with 10395 facets
#I:D     faces: Found 55 faces of dimension 0
#I:D     faces: Found 990 faces of dimension 1
#I:R     SNFV: Smith form correct with probability at least 1 - 1.282482e-22
#I:T     SNFV: cpu  time : 0.176 seconds
#I:R  homology: H_0 = 0.
#I:D     faces: Found 6930 faces of dimension 2
#I:R     SNFV: Smith form correct with probability at least 1 - 2.630265e-22
#I:T     SNFV: cpu  time : 16.873 seconds
#I:R  homology: H_1 = 0.
#I:D     faces: Found 17325 faces of dimension 3
#I:R     SNFV: Smith form correct with probability at least 1 - 7.923057e-23
#I:T     SNFV: cpu  time : 418.794 seconds
#I:R  homology: H_2 = 0.
#I:D     faces: Found 10395 faces of dimension 4
#I:R     SNFV: Smith form correct with probability at least 1 - 4.522239e-22
#I:T     SNFV: cpu  time : 643.755 seconds
#I:R  homology: H_3 = 1188 + Z_3^45.
#I:R  homology: H_4 = 252.
#I:T  homology: cpu  time : 1083.02 seconds
[ [ 0 ], [ 0 ], [ 0 ], [ 1188, [ 3, 45 ] ], [ 252 ] ]
```

Now we are confident that there is less than one chance in a hundred billion billions that this answer is false, and this took only slightly more time!

8.3 Using the Sparse Matrix Functions

Let us now use the sparse matrix functions to Smith forms. We first can try on a sample 4×5 matrix.

```
gap> a:=[[0,3,-3,0,0],[-1,0,-2,2,1],[3,6,0,-6,3],[0,6,-6,0,6]];
[ [ 0, 3, -3, 0, 0 ], [ -1, 0, -2, 2, 1 ], [ 3, 6, 0, -6, 3 ], [ 0, 6, -6, 0, 6 ] ]
gap> NormalFormIntMat(a,1);
rec( normal := [ [ 1, 0, 0, 0, 0 ], [ 0, 3, 0, 0, 0 ],
[ 0, 0, 6, 0, 0 ], [ 0, 0, 0, 0, 0 ] ], rank := 3 )
gap> SMSmithForm(a);
[ 4, 5, [ 1, 3, 6, 0 ] ]
```

We could also have used the file format possibilities for a and for the matrix example of chapter 7 (supposing this matrix is in the file “sparse_mat_32x3200.ex”)

```
gap> gap2SM(a,"/tmp/matrix_4x5.sms");
gap> SMSmithForm("/tmp/matrix_4x5.sms");
[ 4, 5, [ 1, 3, 6, 0 ] ]
gap> SMSmithForm("sparse_mat_32x3200.ex");
[ 32, 3200, [ 1, 1, 253, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ] ]
```

Now we would like to compute the Smith form of the third boundary map of the matching complex with 9 vertices. As this matrix is quite large we can use the sparse format and files to deal with it in the following way:

```
gap> SimplicialBoundaryMaps(SCMatching(9),3,"/tmp/mk9.b3.sms");
gap> SMInvariantFactors("/tmp/mk9.b3.sms");
[ 945, 1260, [ 1, 867 ], [ 3, 8 ] ]

gap> quit;
welker@merak:~>
```

Done !

A Simplicial Complexes and Simplicial Homology

The introduction to some parts of algebraic topology given in the following sections follows mostly the book by Munkres [Mun84]. The combinatorial aspects and a very good general survey on topological methods in combinatorics can be found in the article by Björner [Bjö95].

A.1 Basic Concepts

A (abstract) *simplicial complex* Δ over a (finite) *ground set* Ω is a collection of subsets of Ω such that whenever $A \subseteq B \in \Omega$ and $B \in \Delta$ then $A \in \Delta$. In particular, $\emptyset \in \Delta$ for all non-empty simplicial complexes Δ. An element $A \in \Delta$ is called *face* of Δ and its *dimension* $\dim(\Delta)$ is given by $\#A - 1$. The dimension $\mathbf{dim}(\Delta)$ of the simplicial complex Δ is the maximum of the dimension of its faces. The shift by one when passing from dimension to cardinality of a face A is due to the fact that dimension is the geometric dimension of the simplex given as the convex hull spanned by $\#A$ linearly independent unit vectors in real n=space (for sufficiently large n). The *f-vector* of Δ is the vector $f(\Delta) = (f_-1, \cdots, f_{\dim(\mathbf{\Delta})})$ where f_i denotes the number of faces of dimension i in Δ. The (reduced) *Euler-characteristic* $\widetilde{\chi}(\Delta)$ of Δ is given by the alternating sum $f_{-1} + f_0 + \cdots + (-1)^{\dim(\Delta)} f_{\mathbf{dim}(\Delta)}$. For each $i \geq 0$ we denote by $C_i(\Delta)$ the free $\mathbf{Z}$-module spanned by the faces of Δ of dimension i. In particular, $C_i(\Delta) \cong \mathbf{Z}^{f_i}$ and there is a basis of $C_i(\Delta)$ given by unit-vectors e_A indexed by the faces A of dimension i. The module $C_i(\Delta)$ is called the i-th *chain group* of Δ. For the subsequent definitions we have to fix a linear ordering on the ground set Ω. For simplicity we will assume that $\Omega = \{1, \cdot\cdot, n\}$ and the order is given by the usual order on natural numbers. The $\mathbf{Z}$-module homomorphism $\partial_i : C_i(\Delta) \to C_{i-1}(\Delta)$ is defined by sending the base vector e_A, for $A = \{j_0 < \cdots < j_i\}$, to

$$\sum_{l=0}^{i} (-1)^l e_{A \setminus \{j_l\}}.$$

It is easily checked that $\partial_{i-1} \circ \partial_i = 0$ for all $i \geq 0$. This implies that $\mathbf{Im}\partial_{i+1} \subseteq \mathbf{Ker}\partial_i$. Now the i-th (reduced) *simplicial homology group* $\tilde{H}_i(\Delta)$ is defined as the $\mathbf{Z}$-module $\mathbf{Ker}(\partial_i)/\mathbf{Im}(\partial_{i+1})$. It is easily verified that $\widetilde{H}_{-1} = 0$ unless $\Delta = \{\emptyset\}$ for which it is $\mathbf{Z}$ and that $\widetilde{H}_i(\emptyset) = 0$ for all i. When dealing with the empty face, $\Delta = \emptyset$ or $\Delta = \{\emptyset\}$ there are different conventions. For example many algebraic topologists define abstract simplicial complexes analogous to our definition except that they forbid the empty face, on the other hand other introductions to algebraic topology forbid the empty complex. Our approach is motivated by the applications of calculations of f-vectors and homology groups in combinatorics. Also we do not require – as some texts do – that for all $\omega \in \Omega$ the singleton $\{\omega\}$ is a face of Δ.

A.2 Constructions

If A is a face of Δ then $\mathrm{star}_\Delta(A)$ denotes the *star* $\{B \subseteq \Omega \mid A \cup B \in \Omega\}$ of A in Δ. The *link* $\mathrm{link}_\Delta(A)$ of A in Δ is the simplicial complex $\{B \in \mathrm{star}_\Delta(A) \mid A \cap B = \emptyset\}$. If A is an arbitrary subset of the ground set Ω then Δ_A denotes the *restriction* $\{B \in \Delta \mid B \subseteq A\}$ of Δ to A.

If Δ and Γ are two simplicial complexes over disjoint ground sets Ω_Δ and Ω_Γ then $\Delta * \Gamma$ denotes the *join* $\{A \cup B | A \in \Delta, B \in \Gamma\}$ of Δ and Γ as a simplicial complex over $\Omega_\Delta \cup \Omega_\Gamma$. It is easily seen that the join operation is commutative and associative up to an isomorphism of simplicial complexes (i.e. a bijective map which preserves inclusion of faces). The homology of the join is given in homological dimension i by the degree $i-1$ part of the graded tensor product of the homology groups of Δ and Γ.

If Δ and Γ are two simplicial complexes over not necessarily disjoint ground sets then the *deleted join* of Δ and Γ is constructed as follows. We take copies Δ' and Γ' of Δ and Γ over disjoint ground sets. Then the deleted join of Δ and Γ is the simplicial complex whose faces are the unions of faces of Δ' and Γ' such that the respective faces of Δ and Γ have empty intersection.

For two simplicial complex Δ and Γ over disjoint ground sets Ω_Δ and Ω_Γ the *wedge* $\Delta \vee \Gamma$ is the simplicial complex obtained by identifying one singleton in Δ with another in Γ. Thus we obtain a simplicial complex on the ground set $(\Omega_\Delta \cup \Omega_\Gamma)/\equiv$ where $\equiv$ is the equivalence relation whose only non-singleton class identifies the two elements of the crucial singletons. Note that the wedge is up to homotopy well defined only if Δ and Γ are connected. Otherwise the homotopy my depend on the point chosen. By a simple Meyer-Vietoris type argument the homology is in any case independent of the chosen wedge point and given by $\widetilde{H}_i(\Delta \vee \Gamma) \cong \widetilde{H}_i(\Delta) \oplus \widetilde{H}_i(\Delta)$.

For a simplicial complex Δ over Ω its *Alexander-dual* Δ^* is the simplicial complex over Ω whose face A satisfy $\Omega \setminus A \notin \Delta$. One checks $\Delta^{**} = \Delta$. Since Δ^* can be geometrically realized as a deformation retract of the complement of Δ on the boundary of the $(n-1)$-simplex, for $n = \#\Omega$, it follows by Alexander-duality (see [Mun84]) that

$$\widetilde{H}_i(\Delta) \cong \widetilde{H}^{n-i-1}(\Delta^*).$$

The *minimal non-faces* of a simplicial complex Δ are the subsets of the ground set of Δ with the property that every proper subset is already in Δ. It is clear that the complements of the minimal non-faces are the maximal faces of the Alexander-dual.

B Remarks on the Algorithms

The algorithms EliminateAlgorithm and EliminateGMPAlgorithm use the same procedure, differ only in the integer representation. The strategy used is to triangulate using partial pivoting to find unit pivot elements whenever possible. Failing that, a gcd step is used. One then knows the smith form entries for the zero rows and for the rows whose pivot element is a unit.

In a second phase, the classical smith form algorithm is applied to the remaining rows. It involves row and column operations determined by gcd commutations. It is described for instance in [New72].

The algorithms ValenceElimAlgorithm and ValenceBBAlgorithm, [DSV00], are designed for speed and reduced memory usage, particularly for sparse matrices.

The fundamental tool in these methods is a probabilistic algorithm of Wiedemann, [Wie86], for computing the minimal polynomial of an integer matrix mod a prime. This method requires only O(n) space beyond that used for storage of the matrix. the matrix is not modified during the computation. The polynomial returned is certainly a factor of the true minimal polynomial and with high probability it is the minimal polynomial. We have *never* known it to give an incorrect result. For an $m \times n$ matrix A, $m > n$, the Wiedemann algorithm takes time $\mathrm{O}(n\omega)$, and the time is actually proportional to $k\omega$, where k is the degree of the minimal polynomial and where ω is the cost of matrix vector product. Note that ω is $\mathrm{O}(dn)$ for the boundary matrix at dimension d of a simplicial complex.

The term valence refers to the coefficient of the least degree term of the minimal polynomial. Any prime which occurs in the Smith form necessarily is a divisor of the valence. We use this fact to restrict attention to only those primes for determining the Smith form entries. Fortunately, we have had the experience that for some classes of simplicial complex, the minimal polynomials we work with are of very low degree.

For computing the rank of matrices (mod a prime) we use the fact that a matrix can be constructed, [EK97] which has the same rank as A and with high probability has the property that its characteristic polynomial is a power of x times its minimal polynomial. In this case the rank may be determined from the minimal polynomial.

Thus after computing the valence we may compute the integer rank, and hence get Betti numbers, by computing the rank mod a prime not dividing the valence.

After this point the two algorithms diverge. ValenceElimAlgorithm uses an elimination mod p for each prime p dividing the valence to find the contribution of p and its powers to the invariant factors. That elimination is subject to fill-in as is experienced in integer elimination, but not to the growth in entry size. ValenceBBAlgorithm, to maintain low memory use, restricts itself to computation of the rank mod p for each prime in the valence. Thus it determines the first index at which p appears in an invariant factor (Smith form diagonal entry). We do not yet have a low memory use algorithm to determine the exact power of these primes in the invariant factors.

C Bibliography

[AS93] M. Aschbacher and S.D. Smith. On Quillen's conjecture for the p-subgroup complex. *Ann. Math., II.*, 137:473–529, 1993.

[BBL+99] E. Babson, A. Björner, S. Linusson, J. Shareshian, and V. Welker. Complexes of not i-connected graphs. *Topology*, 38:271–299, 1999.

[Bjö94] A. Björner. Subspace arrangements. In *Proceedings of the First European congress of mathematics*, volume I, invited lectures, pages 321–370. Birkhäuser, 1994.

[Bjö95] A. Björner. Topological methods. In *Handbook of Combinatorics*, volume II, pages 1819–1872. North-Holland, 1995.

[BLVŽ94] A. Björner, L. Lovász, S.T. Vrecica, and R.T. Živaljevic. Chessboard complexes and matching complexes. *J. Lond. Math. Soc., II. Ser. 49, No.1, 25-39*, 49:25–39, 1994.

[Bro99] K. Brown. The coset poset and probabilistic Zeta-function of a finite group. Preprint, 1999.

[BW97] A. Björner and M. Wachs. Shellable nonpure complexes and posets. *Transactions of the American Mathematical Society*, 349:3945–3975, 1997.

[BW99] A. Björner and V. Welker. Complexes of directed graphs. *SIAM Journal on Discrete Mathematics*, 12(4):413–424, 1999.

[DSV] J-G. Dumas, B. D. Saunders, and G. Villard. On efficient sparse integer matrix Smith normal form computations. preprint.

[DSV00] J-G. Dumas, B. D. Saunders, and G. Villard. Integer Smith form via the Valence: experience with large sparse matrices from Homology. In *ISSAC'2000: Proceedings of the 2000 International Symposium on Symbolic and Algebraic Computation, Saint Andrews, Scotland*, pages 95–105, August 2000.

[EK97] W. Eberly and E. Kaltofen. On randomized Lanczos algorithms. In Wolfgang W. Küchlin, editor, *ISSAC '97. Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation, July 21–23, 1997, Maui, Hawaii*, pages 176–183, 1997.

[KT85] C. Kratzer and J. Thévenaz. Type d'homotopie des treillis et treillis des sous-groupes d'un groupe fini. *Comment. Math. Helv.*, 60:85–106, 1985.

[Lut99] F. Lutz. *Triangulated Manifolds with Few Vertices and Vertex-Transitive Group Actions*. Shaker Verlag, Aachen, 1999. Author's disseration at Technische Universität Berlin.

[Mun84] J.R. Munkres. *Elements of Algebraic Topology*. Addison-Wesley, 1984.

[New72] M. Newman. *Integral Matrices*, volume 45 of *Pure and Applied Mathematics, a Series of Monographs and Textbooks*. Academic Press, 1972.

[Qui78] D. Quillen. Homotopy properties of the poset of nontrivial p-subgroups of a group. *Adv. Math.*, 28:101–128, 1978.

[Sta82] R.P. Stanley. Some aspects of groups acting on finite posets. *J. Comb. Theory, Ser. A*, 32:132–161, 1982.

[SW98] B. Shapiro and V. Welker. Combinatorics & topology of stratifications of the space of monic polynomials with real coefficients. *Result. Math.*, 33:338–355, 1998.

[Wel92] V. Welker. The poset of conjugacy classes of subgroups in a finite solvable group. *J. Algebra*, 148:203–218, 1992.

[Wie86] D. H. Wiedemann. Solving sparse linear equations over finite fields. *IEEE Transactions on Information Theory*, 32(1):54–62, January 1986.

[Zie96] G.M. Ziegler. On the poset of partitions of an integer. *J. Comb. Theory, Ser. A*, 42:215–222, 1996.

D 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".