Contents

Chapter 1

Cellular complexes

Data $\longrightarrow$ Cellular Complexes

```
RegularCWPolytope(L):: List --> RegCWComplex
```

```
RegularCWPolytope(G,v):: PermGroup, List --> RegCWComplex
```

Inputs a list L of vectors in $\mathbb{R}^n$ and outputs their convex hull as a regular CW-complex.
Inputs a permutation group G of degree d and vector $v \in \mathbb{R}^d$, and outputs the convex hull of the orbit $\{v^g : g \in G\}$ as a regular CW-complex.

```
CubicalComplex(A):: List --> CubicalComplex
```

Inputs a binary array A and returns the cubical complex represented by A. The array A must of course be such that it represents a cubical complex.

```
PureCubicalComplex(A):: List --> PureCubicalComplex
```

Inputs a binary array A and returns the pure cubical complex represented by A.

```
PureCubicalKnot(n,k):: Int, Int --> PureCubicalComplex
```

```
PureCubicalKnot(L):: List --> PureCubicalComplex
```

Inputs integers n,k and returns the k-th prime knot on n crossings as a pure cubical complex (if this prime knot exists).
Inputs a list L describing an arc presentation for a knot or link and returns the knot or link as a pure cubical complex.

```
PurePermutahedralKnot(n,k):: Int, Int --> PurePermutahedralComplex
```

```
PurePermutahedralKnot(L):: List --> PurePermutahedralComplex
```

Inputs integers n,k and returns the k-th prime knot on n crossings as a pure permutahedral complex (if this prime knot exists).
Inputs a list L describing an arc presentation for a knot or link and returns the knot or link as a pure permutahedral complex.

`PurePermutahedralComplex(A):: List --> PurePermComplex`

Inputs a binary array A and returns the pure permutahedral complex represented by A.

`CayleyGraphOfGroup(G,L):: Group, List --> Graph`

Inputs a finite group G and a list L of elements in G.It returns the Cayley graph of the group generated by L.

`EquivariantEuclideanSpace(G,v):: MatrixGroup, List --> EquivariantRegCWComplex`

Inputs a crystallographic group G with left action on $\mathbb{R}^n$ together with a row vector $v \in \mathbb{R}^n$. It returns an equivariant regular CW-space corresponding to the Dirichlet-Voronoi tessellation of $\mathbb{R}^n$ produced from the orbit of v under the action.

`EquivariantOrbitPolytope(G,v):: PermGroup, List --> EquivariantRegCWComplex`

Inputs a permutation group G of degree n together with a row vector $v \in \mathbb{R}^n$. It returns, as an equivariant regular CW-space, the convex hull of the orbit of v under the canonical left action of G on $\mathbb{R}^n$.

`EquivariantTwoComplex(G):: Group --> EquivariantRegCWComplex`

Inputs a suitable group G and returns, as an equivariant regular CW-space, the 2-complex associated to some presentation of G.

`QuillenComplex(G,p):: Group, Int --> SimplicialComplex`

Inputs a finite group G and prime p, and returns the simplicial complex arising as the order complex of the poset of elementary abelian p-subgroups of G.

`RestrictedEquivariantCWComplex(Y,H):: RegCWComplex, Group --> EquivariantRegCWComplex`

Inputs a G-equivariant regular CW-space Y and a subgroup $H \leq G$ for which GAP can find a transversal. It returns the equivariant regular CW-complex obtained by retricting the action to H.

`RandomSimplicialGraph(n,p):: Int, Int --> SimplicialComplex`

Inputs an integer $n \geq 1$ and positive prime p, and returns an Erd\"os-R\'enyi random graph as a 1-dimensional simplicial complex. The graph has n vertices. Each pair of vertices is, with probability p, directly connected by an edge.

`RandomSimplicialTwoComplex(n,p):: Int, Int --> SimplicialComplex`

Inputs an integer $n \geq 1$ and positive prime p, and returns a Linial-Meshulam random simplicial 2-complex. The 1-skeleton of this simplicial complex is the complete graph on n vertices. Each triple of vertices lies, with probability p, in a common 2-simplex of the complex.

`ReadCSVfileAsPureCubicalKnot(str):: String --> PureCubicalComplex`

`ReadCSVfileAsPureCubicalKnot(str,r):: String, Int --> PureCubicalComplex`

`ReadCSVfileAsPureCubicalKnot(L):: List --> PureCubicalComplex`

`ReadCSVfileAsPureCubicalKnot(L,R):: List,List --> PureCubicalComplex`

Reads a CSV file identified by a string str such as "file.pdb" or "path/file.pdb" and returns a 3-dimensional pure cubical complex K. Each line of the file should contain the coordinates of a point in $\mathbb{R}^3$ and the complex K should represent a knot determined by the sequence of points, though the latter is not guaranteed. A useful check in this direction is to test that K has the homotopy type of a circle.
If the test fails then try the function again with an integer $r \geq 2$ entered as the optional second argument. The integer determines the resolution with which the knot is constructed.
The function can also read in a list L of strings identifying CSV files for several knots. In this case a list R of integer resolutions can also be entered. The lists L and R must be of equal length.

`ReadImageAsPureCubicalComplex(str,t):: String, Int --> PureCubicalComplex`

Reads an image file identified by a string str such as "file.bmp", "file.eps", "file.jpg", "path/file.png" etc., together with an integer t between 0 and 765. It returns a 2-dimensional pure cubical complex corresponding to a black/white version of the image determined by the threshold t. The 2-cells of the pure cubical complex correspond to pixels with RGB value $R+G+B \leq t$.

`ReadImageAsFilteredPureCubicalComplex(str,n):: String, Int --> FilteredPureCubicalComplex`

Reads an image file identified by a string str such as "file.bmp", "file.eps", "file.jpg", "path/file.png" etc., together with a positive integer n. It returns a 2-dimensional filtered pure cubical complex of filtration length n. The kth term in the filtration is a pure cubical complex corresponding to a black/white version of the image determined by the threshold $t_k = k \times 765/n$. The 2-cells of the kth term correspond to pixels with RGB value $R+G+B \leq t_k$.

`ReadImageAsWeightFunction(str,t):: String, Int --> RegCWComplex, Function`

Reads an image file identified by a string str such as "file.bmp", "file.eps", "file.jpg", "path/file.png" etc., together with an integer t. It constructs a 2-dimensional regular CW-complex Y from the image, together with a weight function $w\colon Y \to \mathbb{Z}$ corresponding to a filtration on Y of filtration length t. The pair $[Y,w]$ is returned.

`ReadPDBfileAsPureCubicalComplex(str):: String --> PureCubicalComplex`

`ReadPDBfileAsPureCubicalComplex(str,r):: String, Int --> PureCubicalComplex`

Reads a PDB (Protein Database) file identified by a string str such as "file.pdb" or "path/file.pdb" and returns a 3-dimensional pure cubical complex K. The complex K should represent a (protein backbone) knot but this is not guaranteed. A useful check in this direction is to test that K has the homotopy type of a circle.
If the test fails then try the function again with an integer $r \geq 2$ entered as the optional second argument. The integer determines the resolution with which the knot is constructed.

`ReadPDBfileAsPurepermutahedralComplex(str):: String --> PurePermComplex`

`ReadPDBfileAsPurePermutahedralComplex(str,r):: String, Int --> PurePermComplex`

Reads a PDB (Protein Database) file identified by a string str such as "file.pdb" or "path/file.pdb" and returns a 3-dimensional pure permutahedral complex K. The complex K should represent a (protein backbone) knot but this is not guaranteed. A useful check in this direction is to test that K has the homotopy type of a circle.
If the test fails then try the function again with an integer $r \geq 2$ entered as the optional second argument. The integer determines the resolution with which the knot is constructed.

`RegularCWPolytope(L):: List --> RegCWComplex`

`RegularCWPolytope(G,v):: PermGroup, List --> RegCWComplex`

Inputs a list L of vectors in $\mathbb{R}^n$ and outputs their convex hull as a regular CW-complex.
Inputs a permutation group G of degree d and vector $v \in \mathbb{R}^d$, and outputs the convex hull of the orbit $\{v^g : g \in G\}$ as a regular CW-complex.

`SimplicialComplex(L):: List --> SimplicialComplex`

Inputs a list L whose entries are lists of vertices representing the maximal simplices of a simplicial complex, and returns the simplicial complex. Here a "vertex" is a GAP object such as an integer or a subgroup. The list L can also contain non-maximal simplices.

`SymmetricMatrixToFilteredGraph(A,m,s):: Mat, Int, Rat --> FilteredGraph`

`SymmetricMatrixToFilteredGraph(A,m):: Mat, Int --> FilteredGraph`

Inputs an $n \times n$ symmetric matrix A, a positive integer m and a positive rational s. The function returns a filtered graph of filtration length m. The t-th term of the filtration is a graph with n vertices and an edge between the i-th and j-th vertices if the (i, j) entry of A is less than or equal to $t \times s/m$.
If the optional input s is omitted then it is set equal to the largest entry in the matrix A.

`SymmetricMatrixToGraph(A,t):: Mat, Rat --> Graph`

Inputs an $n \times n$ symmetric matrix A over the rationals and a rational number $t \geq 0$, and returns the graph on the vertices $1, 2, \ldots, n$ with an edge between distinct vertices i and j precisely when the (i, j) entry of A is $\leq t$.

Metric Spaces

`CayleyMetric(g,h):: Permutation, Permutation --> Int`

Inputs two permutations g, h and optionally the degree N of a symmetric group containing them. It returns the minimum number of transpositions needed to express $g * h^{-1}$ as a product of transpositions.

`EuclideanMetric(g,h):: List, List --> Rat`

Inputs two vectors $v, w \in \mathbb{R}^n$ and returns a rational number approximating the Euclidean distance between them.

`EuclideanSquaredMetric(g,h):: List, List --> Rat`

Inputs two vectors $v, w \in \mathbb{R}^n$ and returns the square of the Euclidean distance between them.

`HammingMetric(g,h):: Permutation, Permutation --> Int`

Inputs two permutations g, h and optionally the degree N of a symmetric group containing them. It returns the minimum number of integers moved by the permutation $g * h^{-1}$.

`KendallMetric(g,h):: Permutation, Permutation --> Int`

Inputs two permutations g, h and optionally the degree N of a symmetric group containing them. It returns the minimum number of adjacent transpositions needed to express $g * h^{-1}$ as a product of adjacent transpositions. An {\em adjacent} transposition is of the form $(i, i+1)$.

`ManhattanMetric(g,h):: List, List --> Rat`

Inputs two vectors $v, w \in \mathbb{R}^n$ and returns the Manhattan distance between them.

`VectorsToSymmetricMatrix(V):: List --> Matrix`

`VectorsToSymmetricMatrix(V,d):: List, Function --> Matrix`

Inputs a list $V = \{v_1, \ldots, v_k\} \in \mathbb{R}^n$ and returns the $k \times k$ symmetric matrix of Euclidean distances $d(v_i, v_j)$. When these distances are irrational they are approximated by a rational number. As an optional second argument any rational valued function $d(x, y)$ can be entered.

Cellular Complexes $\longrightarrow$ Cellular Complexes

| |

`BoundaryMap(K):: RegCWComplex --> RegCWMap`

Inputs a pure regular CW-complex K and returns the regular CW-inclusion map $\iota : \partial K \hookrightarrow K$ from the boundary ∂K into the complex K.

`CliqueComplex(G,n):: Graph, Int --> SimplicialComplex`

`CliqueComplex(F,n):: FilteredGraph, Int --> FilteredSimplicialComplex`

`CliqueComplex(K,n):: SimplicialComplex, Int --> SimplicialComplex`

Inputs a graph G and integer $n \geq 1$. It returns the n-skeleton of a simplicial complex K with one k-simplex for each complete subgraph of G on $k+1$ vertices.
Inputs a fitered graph F and integer $n \geq 1$. It returns the n-skeleton of a filtered simplicial complex K whose t-term has one k-simplex for each complete subgraph of the t-th term of G on $k+1$ vertices.
Inputs a simplicial complex of dimension $d = 1$ or $d = 2$. If $d = 1$ then the clique complex of a graph returned. If $d = 2$ then the clique complex of a \$2\$-complex is returned.

```
ConcentricFiltration(K,n):: PureCubicalComplex, Int --> FilteredPureCubicalComplex
```

Inputs a pure cubical complex K and integer $n \geq 1$, and returns a filtered pure cubical complex of filtration length n. The t-th term of the filtration is the intersection of K with the ball of radius r_t centred on the centre of gravity of K, where $0 = r_1 \leq r_2 \leq r_3 \leq \cdots \leq r_n$ are equally spaced rational numbers. The complex K is contained in the ball of radius r_n. (At present, this is implemented only for 2- and 3-dimensional complexes.)

```
DirectProduct(M,N):: RegCWComplex, RegCWComplex --> RegCWComplex
```

```
DirectProduct(M,N):: PureCubicalComplex, PureCubicalComplex --> PureCubicalComplex
```

Inputs two or more regular CW-complexes or two or more pure cubical complexes and returns their direct product.

```
FiltrationTerm(K,t):: FilteredPureCubicalComplex, Int --> PureCubicalComplex
```

```
FiltrationTerm(K,t):: FilteredRegCWComplex, Int --> RegCWComplex
```

Inputs a filtered regular CW-complex or a filtered pure cubical complex K together with an integer $t \geq 1$. The t-th term of the filtration is returned.

```
Graph(K):: RegCWComplex --> Graph
```

```
Graph(K):: SimplicialComplex --> Graph
```

Inputs a regular CW-complex or a simplicial complex K and returns its 1-skeleton as a graph.

```
HomotopyGraph(Y):: RegCWComplex --> Graph
```

Inputs a regular CW-complex Y and returns a subgraph $M \subset Y^1$ of the 1-skeleton for which the induced homology homomorphisms $H_1(M,\mathbb{Z}) \to H_1(Y,\mathbb{Z})$ and $H_1(Y^1,\mathbb{Z}) \to H_1(Y,\mathbb{Z})$ have identical images. The construction tries to include as few edges in M as possible, though a minimum is not guaranteed.

```
Nerve(M):: PureCubicalComplex --> SimplicialComplex
```

```
Nerve(M):: PurePermComplex --> SimplicialComplex
```

```
Nerve(M,n):: PureCubicalComplex, Int --> SimplicialComplex
```

```
Nerve(M,n):: PurePermComplex, Int --> SimplicialComplex
```

Inputs a pure cubical complex or pure permutahedral complex M and returns the simplicial complex K obtained by taking the nerve of an open cover of $|M|$, the open sets in the cover being sufficiently small neighbourhoods of the top-dimensional cells of $|M|$. The spaces $|M|$ and $|K|$ are homotopy equivalent by the Nerve Theorem. If an integer $n \geq 0$ is supplied as the second argument then only the n-skeleton of K is returned.

```
RegularCWComplex(K):: SimplicialComplex --> RegCWComplex
```

`RegularCWComplex(K):: PureCubicalComplex --> RegCWComplex`

`RegularCWComplex(K):: CubicalComplex --> RegCWComplex`

`RegularCWComplex(K):: PurePermComplex --> RegCWComplex`

`RegularCWComplex(L):: List --> RegCWComplex`

`RegularCWComplex(L,M):: List,List --> RegCWComplex`

Inputs a simplicial, pure cubical, cubical or pure permutahedral complex K and returns the corresponding regular CW-complex. Inputs a list $L = Y!.boundaries$ of boundary incidences of a regular CW-complex Y and returns Y. Inputs a list $L = Y!.boundaries$ of boundary incidences of a regular CW-complex Y together with a list $M = Y!.orientation$ of incidence numbers and returns a regular CW-complex Y. The availability of precomputed incidence numbers saves recalculating them.

`RegularCWMap(M,A):: PureCubicalComplex, PureCubicalComplex --> RegCWMap`

Inputs a pure cubical complex M and a subcomplex A and returns the inclusion map $A \to M$ as a map of regular CW complexes.

`ThickeningFiltration(K,n):: PureCubicalComplex, Int --> FilteredPureCubicalComplex`

`ThickeningFiltration(K,n,s):: PureCubicalComplex, Int, Int --> FilteredPureCubicalComplex`

Inputs a pure cubical complex K and integer $n \geq 1$, and returns a filtered pure cubical complex of filtration length n. The t-th term of the filtration is the t-fold thickening of K. If an integer $s \geq 1$ is entered as the optional third argument then the t-th term of the filtration is the ts-fold thickening of K.

Cellular Complexes $\longrightarrow$ Cellular Complexes (Preserving Data Types)

`ContractedComplex(K):: RegularCWComplex --> RegularCWComplex`

`ContractedComplex(K):: FilteredRegularCWComplex --> FilteredRegularCWComplex`

`ContractedComplex(K):: CubicalComplex --> CubicalComplex`

`ContractedComplex(K):: PureCubicalComplex --> PureCubicalComplex`

`ContractedComplex(K,S):: PureCubicalComplex, PureCubicalComplex --> PureCubicalComplex`

`ContractedComplex(K):: FilteredPureCubicalComplex --> FilteredPureCubicalComplex`

`ContractedComplex(K):: PurePermComplex --> PurePermComplex`

`ContractedComplex(K,S):: PurePermComplex, PurePermComplex --> PurePermComplex`

`ContractedComplex(K):: SimplicialComplex --> SimplicialComplex`

`ContractedComplex(G):: Graph --> Graph`

Inputs a complex (regular CW, Filtered regular CW, pure cubical etc.) and returns a homotopy equivalent subcomplex.
Inputs a pure cubical complex or pure permutahedral complex K and a subcomplex S. It returns a homotopy equivalent subcomplex of K that contains S.
Inputs a graph G and returns a subgraph S such that the clique complexes of G and S are homotopy equivalent.

`ContractibleSubcomplex(K):: PureCubicalComplex --> PureCubicalComplex`

`ContractibleSubcomplex(K):: PurePermComplex --> PurePermComplex`

`ContractibleSubcomplex(K):: SimplicialComplex --> SimplicialComplex`

Inputs a non-empty pure cubical, pure permutahedral or simplicial complex K and returns a contractible subcomplex.

`KnotReflection(K):: PureCubicalComplex --> PureCubicalComplex`

Inputs a pure cubical knot and returns the reflected knot.

`KnotSum(K,L):: PureCubicalComplex, PureCubicalComplex --> PureCubicalComplex`

Inputs two pure cubical knots and returns their sum.

`OrientRegularCWComplex(Y):: RegCWComplex --> Void`

Inputs a regular CW-complex Y and computes and stores incidence numbers for Y. If Y already has incidence numbers then the function does nothing.

`PathComponent(K,n):: SimplicialComplex, Int --> SimplicialComplex`

`PathComponent(K,n):: PureCubicalComplex, Int --> PureCubicalComplex`

`PathComponent(K,n):: PurePermComplex, Int --> PurePermComplex`

Inputs a simplicial, pure cubical or pure permutahedral complex K together with an integer $1 \leq n \leq \beta_0(K)$. The n-th path component of K is returned.

`PureComplexBoundary(M):: PureCubicalComplex --> PureCubicalComplex`

`PureComplexBoundary(M):: PurePermComplex --> PurePermComplex`

Inputs a d-dimensional pure cubical or pure permutahedral complex M and returns a d-dimensional complex consisting of the closure of those d-cells whose boundaries contains some cell with coboundary of size less than the maximal possible size.

`PureComplexComplement(M):: PureCubicalComplex --> PureCubicalComplex`

`PureComplexComplement(M):: PurePermComplex --> PurePermComplex`

Inputs a pure cubical complex or a pure permutahedral complex and returns its complement.

`PureComplexDifference(M,N):: PureCubicalComplex, PureCubicalComplex --> PureCubicalComplex`

`PureComplexDifference(M,N):: PurePermComplex, PurePermComplex --> PurePermComplex`

Inputs two pure cubical complexes or two pure permutahedral complexes and returns the difference $M-N$.

`PureComplexInterstection(M,N):: PureCubicalComplex, PureCubicalComplex --> PureCubicalComp`

`PureComplexIntersection(M,N):: PurePermComplex, PurePermComplex --> PurePermComplex`

Inputs two pure cubical complexes or two pure permutahedral complexes and returns their intersection.

`PureComplexThickened(M):: PureCubicalComplex --> PureCubicalComplex`

`PureComplexThickened(M):: PurePermComplex --> PurePermComplex`

Inputs a pure cubical complex or a pure permutahedral complex and returns the a thickened complex.

`PureComplexUnion(M,N):: PureCubicalComplex, PureCubicalComplex --> PureCubicalComplex`

`PureComplexUnion(M,N):: PurePermComplex, PurePermComplex --> PurePermComplex`

Inputs two pure cubical complexes or two pure permutahedral complexes and returns their union.

`SimplifiedComplex(K):: RegularCWComplex --> RegularCWComplex`

`SimplifiedComplex(K):: PurePermComplex --> PurePermComplex`

`SimplifiedComplex(R):: FreeResolution --> FreeResolution`

`SimplifiedComplex(C):: ChainComplex --> ChainComplex`

Inputs a regular CW-complex or a pure permutahedral complex K and returns a homeomorphic complex with possibly fewer cells and certainly no more cells.
Inputs a free $\mathbb{Z}G$-resolution R of $\mathbb{Z}$ and returns a $\mathbb{Z}G$-resolution S with potentially fewer free generators.
Inputs a chain complex C of free abelian groups and returns a chain homotopic chain complex D with potentially fewer free generators.

`ZigZagContractedComplex(K):: PureCubicalComplex --> PureCubicalComplex`

`ZigZagContractedComplex(K):: FilteredPureCubicalComplex --> FilteredPureCubicalComplex`

`ZigZagContractedComplex(K):: PurePermComplex --> PurePermComplex`

Inputs a pure cubical, filtered pure cubical or pure permutahedral complex and returns a homotopy equivalent complex. In the filtered case, the t-th term of the output is homotopy equivalent to the t-th term of the input for all t.

Cellular Complexes $\longrightarrow$ Homotopy Invariants

| |

`AlexanderPolynomial(K):: PureCubicalComplex --> Polynomial`

`AlexanderPolynomial(K):: PurePermComplex --> Polynomial`

`AlexanderPolynomial(G):: FpGroup --> Polynomial`

Inputs a 3-dimensional pure cubical or pure permutahdral complex K representing a knot and returns the Alexander polynomial of the fundamental group $G = \pi_1(\mathbb{R}^3 \setminus K)$.
Inputs a finitely presented group G with infinite cyclic abelianization and returns its Alexander polynomial.

`BettiNumber(K,n):: SimplicialComplex, Int --> Int`

`BettiNumber(K,n):: PureCubicalComplex, Int --> Int`

`BettiNumber(K,n):: CubicalComplex, Int --> Int`

`BettiNumber(K,n):: PurePermComplex, Int --> Int`

`BettiNumber(K,n):: RegCWComplex, Int --> Int`

`BettiNumber(K,n):: ChainComplex, Int --> Int`

`BettiNumber(K,n):: SparseChainComplex, Int --> Int`

`BettiNumber(K,n,p):: SimplicialComplex, Int, Int --> Int`

`BettiNumber(K,n,p):: PureCubicalComplex, Int, Int --> Int`

`BettiNumber(K,n,p):: CubicalComplex, Int, Int --> Int`

`BettiNumber(K,n,p):: PurePermComplex, Int, Int --> Int`

`BettiNumber(K,n,p):: RegCWComplex, Int, Int --> Int`

Inputs a simplicial, cubical, pure cubical, pure permutahedral, regular CW, chain or sparse chain complex K together with an integer $n \geq 0$ and returns the nth Betti number of K.
Inputs a simplicial, cubical, pure cubical, pure permutahedral or regular CW-complex K together with an integer $n \geq 0$ and a prime $p \geq 0$ or $p = 0$. In this case the nth Betti number of K over a field of characteristic p is returned.

```
EulerCharacteristic(C):: ChainComplex --> Int

EulerCharacteristic(K):: CubicalComplex --> Int

EulerCharacteristic(K):: PureCubicalComplex --> Int

EulerCharacteristic(K):: PurePermComplex --> Int

EulerCharacteristic(K):: RegCWComplex --> Int

EulerCharacteristic(K):: SimplicialComplex --> Int
```

Inputs a chain complex C and returns its Euler characteristic.
Inputs a cubical, or pure cubical, or pure permutahedral or regular CW-, or simplicial complex K and returns its Euler characteristic.

```
EulerIntegral(Y,w):: RegCWComplex, Int --> Int
```

Inputs a regular CW-complex Y and a weight function $w{:}Y \to \mathbb{Z}$, and returns the Euler integral $\int_Y w\,d\chi$.

```
FundamentalGroup(K):: RegCWComplex --> FpGroup

FundamentalGroup(K,n):: RegCWComplex, Int --> FpGroup

FundamentalGroup(K):: SimplicialComplex --> FpGroup

FundamentalGroup(K):: PureCubicalComplex --> FpGroup

FundamentalGroup(K):: PurePermComplex --> FpGroup

FundamentalGroup(F):: RegCWMap --> GroupHomomorphism

FundamentalGroup(F,n):: RegCWMap, Int --> GroupHomomorphism
```

Inputs a regular CW, simplicial, pure cubical or pure permutahedral complex K and returns the fundamental group.
Inputs a regular CW complex K and the number n of some zero cell. It returns the fundamental group of K based at the n-th zero cell.
Inputs a regular CW map F and returns the induced homomorphism of fundamental groups. If the number of some zero cell in the domain of F is entered as an optional second variable then the fundamental group is based at this zero cell.

```
FundamentalGroupOfQuotient(Y):: EquivariantRegCWComplex --> Group
```

Inputs a G-equivariant regular CW complex Y and returns the group G.

```
IsAspherical(F,R):: FreeGroup, List --> Boolean
```

Inputs a free group F and a list R of words in F. The function attempts to test if the quotient group $G=F/\langle R\rangle^F$ is aspherical. If it succeeds it returns $true$. Otherwise the test is inconclusive and $fail$ is returned.

`KnotGroup(K):: PureCubicalComplex --> FpGroup`

`KnotGroup(K):: PureCubicalComplex --> FpGroup`

Inputs a pure cubical or pure permutahedral complex K and returns the fundamental group of its complement. If the complement is path-connected then this fundamental group is unique up to isomorphism. Otherwise it will depend on the path-component in which the randomly chosen base-point lies.

`PiZero(Y):: RegCWComplex --> List`

`PiZero(Y):: Graph --> List`

`PiZero(Y):: SimplicialComplex --> List`

Inputs a regular CW-complex Y, or graph Y, or simplicial complex Y and returns a pair $[cells, r]$ where: $cells$ is a list of vertices of Y representing the distinct path-components; $r(v)$ is a function which, for each vertex v of Y returns the representative vertex $r(v)\in cells$.

`PersistentBettiNumbers(K,n):: FilteredSimplicialComplex, Int --> List`

`PersistentBettiNumbers(K,n):: FilteredPureCubicalComplex, Int --> List`

`PersistentBettiNumbers(K,n):: FilteredRegCWComplex, Int --> List`

`PersistentBettiNumbers(K,n):: FilteredChainComplex, Int --> List`

`PersistentBettiNumbers(K,n):: FilteredSparseChainComplex, Int --> List`

`PersistentBettiNumbers(K,n,p):: FilteredSimplicialComplex, Int, Int --> List`

`PersistentBettiNumbers(K,n,p):: FilteredPureCubicalComplex, Int, Int --> List`

`PersistentBettiNumbers(K,n,p):: FilteredRegCWComplex, Int, Int --> List`

`PersistentBettiNumbers(K,n,p):: FilteredChainComplex, Int, Int --> List`

`PersistentBettiNumbers(K,n,p):: FilteredSparseChainComplex, Int, Int --> List`

Inputs a filtered simplicial, filtered pure cubical, filtered regular CW, filtered chain or filtered sparse chain complex K together with an integer $n\geq 0$ and returns the nth PersistentBetti numbers of K as a list of lists of integers.
Inputs a filtered simplicial, filtered pure cubical, filtered regular CW, filtered chain or filtered sparse chain complex K together with an integer $n\geq 0$ and a prime $p\geq 0$ or $p=0$. In this case the nth PersistentBetti numbers of K over a field of characteristic p are returned.

Data $\longrightarrow$ Homotopy Invariants

| |

`DendrogramMat(A,t,s):: Mat, Rat, Int --> List`

Inputs an $n \times n$ symmetric matrix A over the rationals, a rational $t \geq 0$ and an integer $s \geq 1$. A list $[v_1, \ldots, v_{t+1}]$ is returned with each v_k a list of positive integers. Let $t_k = (k-1)s$. Let $G(A, t_k)$ denote the graph with vertices $1, \ldots, n$ and with distinct vertices i and j connected by an edge when the (i, j) entry of A is $\leq t_k$. The i-th path component of $G(A, t_k)$ is included in the $v_k[i]$-th path component of $G(A, t_{k+1})$. This defines the integer vector v_k. The vector v_k has length equal to the number of path components of $G(A, t_k)$.

Cellular Complexes $\longrightarrow$ Non Homotopy Invariants

| |

`ChainComplex(K):: CubicalComplex --> ChainComplex`

`ChainComplex(K):: PureCubicalComplex --> ChainComplex`

`ChainComplex(K):: PurePermComplex --> ChainComplex`

`ChainComplex(Y):: RegCWComplex --> ChainComplex`

`ChainComplex(K):: SimplicialComplex --> ChainComplex`

Inputs a cubical, or pure cubical, or pure permutahedral or simplicial complex K and returns its chain complex of free abelian groups. In degree n this chain complex has one free generator for each n-dimensional cell of K.
Inputs a regular CW-complex Y and returns a chain complex C which is chain homotopy equivalent to the cellular chain complex of Y. In degree n the free abelian chain group C_n has one free generator for each critical n-dimensional cell of Y with respect to some discrete vector field on Y.

`ChainComplexEquivalence(X):: RegCWComplex --> List`

Inputs a regular CW-complex X and returns a pair $[f_*, g_*]$ of chain maps $f_*: C_*(X) \to D_*(X)$, $g_*: D_*(X) \to C_*(X)$. Here $C_*(X)$ is the standard cellular chain complex of X with one free generator for each cell in X. The chain complex $D_*(X)$ is a typically smaller chain complex arising from a discrete vector field on X. The chain maps f_*, g_* are chain homotopy equivalences.

`ChainComplexOfQuotient(Y):: EquivariantRegCWComplex --> ChainComplex`

Inputs a G-equivariant regular CW-complex Y and returns the cellular chain complex of the quotient space Y/G.

`ChainMap(X,A,Y,B):: PureCubicalComplex, PureCubicalComplex, PureCubicalComplex, PureCubica`

`ChainMap(f):: RegCWMap --> ChainMap`

`ChainMap(f):: SimplicialMap --> ChainComplex`

Inputs a pure cubical complex Y and pure cubical sucomplexes $X \subset Y$, $B \subset Y$,$A \subset B$. It returns the induced chain map $f_*: C_*(X/A) \rightarrow C_*(Y/B)$ of cellular chain complexes of pairs. (Typically one takes A and B to be empty or contractible subspaces, in which case $C_*(X/A) \simeq C_*(X)$, $C_*(Y/B) \simeq C_*(Y)$.)

Inputs a map $f: X \rightarrow Y$ between two regular CW-complexes X, Y and returns an induced chain map $f_*: C_*(X) \rightarrow C_*(Y)$ where $C_*(X)$, $C_*(Y)$ are chain homotopic to (but usually smaller than) the cellular chain complexes of X, Y.

Inputs a map $f: X \rightarrow Y$ between two simplicial complexes X, Y and returns the induced chain map $f_*: C_*(X) \rightarrow C_*(Y)$ of cellular chain complexes.

```
CochainComplex(K):: CubicalComplex --> CochainComplex

CochainComplex(K):: PureCubicalComplex --> CochainComplex

CochainComplex(K):: PurePermComplex --> CochainComplex

CochainComplex(Y):: RegCWComplex --> CochainComplex

CochainComplex(K):: SimplicialComplex --> CohainComplex
```

Inputs a cubical, or pure cubical, or pure permutahedral or simplicial complex K and returns its cochain complex of free abelian groups. In degree n this cochain complex has one free generator for each n-dimensional cell of K.

Inputs a regular CW-complex Y and returns a cochain complex C which is chain homotopy equivalent to the cellular cochain complex of Y. In degree n the free abelian cochain group C_n has one free generator for each critical n-dimensional cell of Y with respect to some discrete vector field on Y.

```
CriticalCells(K):: RegCWComplex --> List
```

Inputs a regular CW-complex K and returns its critical cells with respect to some discrete vector field on K. If no discrete vector field on K is available then one will be computed and stored.

```
DiagonalApproximation(X):: RegCWComplex --> RegCWMap, RegCWMap
```

Inputs a regular CW-complex X and outputs a pair $[p, \iota]$ of maps of CW-complexes. The map $p: X^{\Delta} \rightarrow X$ will often be a homotopy equivalence. This is always the case if X is the CW-space of any pure cubical complex. In general, one can test to see if the induced chain map $p_*: C_*(X^{\Delta}) \rightarrow C_*(X)$ is an isomorphism on integral homology. The second map $\iota: X^{\Delta} \hookrightarrow X \times X$ is an inclusion into the direct product. If p_* induces an isomorphism on homology then the chain map $\iota_*: C_*(X^{\Delta}) \rightarrow C_*(X \times X)$ can be used to compute the cup product.

```
Size(Y):: RegCWComplex --> Int

Size(Y):: SimplicialComplex --> Int

Size(K):: PureCubicalComplex --> Int

Size(K):: PurePermComplex --> Int
```

Inputs a regular CW complex or a simplicial complex Y and returns the number of cells in the complex.
Inputs a d-dimensional pure cubical or pure permutahedral complex K and returns the number of d-dimensional cells in the complex.

(Co)chain Complexes $\longrightarrow$ (Co)chain Complexes

`FilteredTensorWithIntegers(R):: FreeResolution, Int --> FilteredChainComplex`

Inputs a free $\mathbb{Z}G$-resolution R for which "*filteredDimension*" lies in NAMESOFCOMPONENTS(R). (Such a resolution can be produced using TWISTERTENSORPRODUCT(), RESOLUTIONNORMALSUBGROUPS() or FREEGRESOLUTION().) It returns the filtered chain complex obtained by tensoring with the trivial module $\mathbb Z$.

`FilteredTensorWithIntegersModP(R,p):: FreeResolution, Int --> FilteredChainComplex`

Inputs a free $\mathbb{Z}G$-resolution R for which "*filteredDimension*" lies in NAMESOFCOMPONENTS(R), together with a prime p. (Such a resolution can be produced using TWISTERTENSORPRODUCT(), RESOLUTIONNORMALSUBGROUPS() or FREEGRESOLUTION().) It returns the filtered chain complex obtained by tensoring with the trivial module $\mathbb F$, the field of p elements.

`HomToIntegers(C):: ChainComplex --> CochainComplex`

`HomToIntegers(R):: FreeResolution --> CochainComplex`

`HomToIntegers(F):: EquiChainMap --> CochainMap`

Inputs a chain complex C of free abelian groups and returns the cochain complex $Hom_{\mathbb{Z}}(C,\mathbb{Z})$.
Inputs a free $\mathbb{Z}G$-resolution R in characteristic 0 and returns the cochain complex $Hom_{\mathbb{Z}G}(R,\mathbb{Z})$.
Inputs an equivariant chain map $F:R \rightarrow S$ of resolutions and returns the induced cochain map $Hom_{\mathbb{Z}G}(S,\mathbb{Z}) \longrightarrow Hom_{\mathbb{Z}G}(R,\mathbb{Z})$.

`TensorWithIntegersModP(C,p):: ChainComplex, Int --> ChainComplex`

`TensorWithIntegersModP(R,p):: FreeResolution, Int --> ChainComplex`

`TensorWithIntegersModP(F,p):: EquiChainMap, Int --> ChainMap`

Inputs a chain complex C of characteristic 0 and a prime integer p. It returns the chain complex $C \otimes_{\mathbb{Z}} \mathbb{Z}_p$ of characteristic p.
Inputs a free $\mathbb{Z}G$-resolution R of characteristic 0 and a prime integer p. It returns the chain complex $R \otimes_{\mathbb{Z}G} \mathbb{Z}_p$ of characteristic p.
Inputs an equivariant chain map $F:R \rightarrow S$ in characteristic 0 a prime integer p. It returns the induced chain map $F \otimes_{\mathbb{Z}G} \mathbb{Z}_p : R \otimes_{\mathbb{Z}G} \mathbb{Z}_p \longrightarrow S \otimes_{\mathbb{Z}G} \mathbb{Z}_p$.

(Co)chain Complexes $\longrightarrow$ Homotopy Invariants

```
Cohomology(C,n):: CochainComplex, Int --> List

Cohomology(F,n):: CochainMap, Int --> GroupHomomorphism

Cohomology(K,n):: CubicalComplex, Int --> List

Cohomology(K,n):: PureCubicalComplex, Int --> List

Cohomology(K,n):: PurePermComplex, Int --> List

Cohomology(K,n):: RegCWComplex, Int --> List

Cohomology(K,n):: SimplicialComplex, Int --> List
```

Inputs a cochain complex C and integer $n \geq 0$ and returns the n-th cohomology group of C as a list of its abelian invariants.

Inputs a chain map F and integer $n \geq 0$. It returns the induced cohomology homomorphism $H_n(F)$ as a homomorphism of finitely presented groups.

Inputs a cubical, or pure cubical, or pure permutahedral or regular CW or simplicial complex K together with an integer $n \geq 0$. It returns the n-th integral cohomology group of K as a list of its abelian invariants.

```
CupProduct(Y):: RegCWComplex --> Function

CupProduct(R,p,q,P,Q):: FreeRes, Int, Int, List, List  --> List
```

Inputs a regular CW-complex Y and returns a function $f(p,q,P,Q)$. This function f inputs two integers $p,q \geq 0$ and two integer lists $P=[p_1,\ldots,p_m]$, $Q=[q_1,\ldots,q_n]$ representing elements $P \in H^p(Y,\mathbb{Z})$ and $Q \in H^q(Y,\mathbb{Z})$. The function f returns a list $P \cup Q$ representing the cup product $P \cup Q \in H^{p+q}(Y,\mathbb{Z})$.

Inputs a free $\mathbb{Z}G$ resolution R of $\mathbb{Z}$ for some group G, together with integers $p,q \geq 0$ and integer lists P,Q representing cohomology classes $P \in H^p(G,\mathbb{Z})$, $Q \in H^q(G,\mathbb{Z})$. An integer list representing the cup product $P \cup Q \in H^{p+q}(G,\mathbb{Z})$ is returned.

```
Homology(C,n):: ChainComplex, Int --> List

Homology(F,n):: ChainMap, Int --> GroupHomomorphism

Homology(K,n):: CubicalComplex, Int --> List

Homology(K,n):: PureCubicalComplex, Int --> List

Homology(K,n):: PurePermComplex, Int --> List

Homology(K,n):: RegCWComplex, Int --> List

Homology(K,n):: SimplicialComplex, Int --> List
```

Inputs a chain complex C and integer $n \geq 0$ and returns the n-th homology group of C as a list of its abelian invariants.
Inputs a chain map F and integer $n \geq 0$. It returns the induced homology homomorphism $H_n(F)$ as a homomorphism of finitely presented groups.
Inputs a cubical, or pure cubical, or pure permutahedral or regular CW or simplicial complex K together with an integer $n \geq 0$. It returns the n-th integral homology group of K as a list of its abelian invariants.

Visualization

| |

`BarCodeDisplay(L) :: List --> void`

Displays a barcode L=PERSITENTBETTINUMBERS(X,N).

`BarCodeCompactDisplay(L) :: List --> void`

Displays a barcode L=PERSITENTBETTINUMBERS(X,N) in compact form.

`CayleyGraphOfGroup(G,L):: Group, List --> Void`

Inputs a finite group G and a list L of elements in G.It displays the Cayley graph of the group generated by L where edge colours correspond to generators.

`Display(G) :: Graph --> void`

`Display(M) :: PureCubicalComplex --> void`

`Display(M) :: PurePermutahedralComplex --> void`

Displays a graph G; a \$2\$- or \$3\$-dimensional pure cubical complex M; a \$3\$-dimensional pure permutahedral complex M.

`DisplayArcPresentation(K) :: PureCubicalComplex --> void`

Displays a 3-dimensional pure cubical knot K=PURECUBICALKNOT(L) in the form of an arc presentation.

`DisplayCSVKnotFile(str) :: String --> void`

Inputs a string str that identifies a csv file containing the points on a piecewise linear knot in $\mathbb{R}^3$. It displays the knot.

`DisplayDendrogram(L):: List --> Void`

Displays the dendrogram L:=DENDROGRAMMAT(A,T,S).

`DisplayDendrogramMat(A,t,s):: Mat, Rat, Int --> Void`

Inputs an $n \times n$ symmetric matrix A over the rationals, a rational $t \geq 0$ and an integer $s \geq 1$. The dendrogram defined by DENDROGRAMMAT(A,T,S) is displayed.

```
DisplayPDBfile(str):: String --> Void
```

Displays the protein backone described in a PDB (Protein Database) file identified by a string str such as "file.pdb" or "path/file.pdb".

```
OrbitPolytope(G,v,L) :: PermGroup, List, List --> void
```

Inputs a permutation group or finite matrix group G of degree d and a rational vector $v \in \mathbb{R}^d$. In both cases there is a natural action of G on $\mathbb{R}^d$. Let $P(G,v)$ be the convex hull of the orbit of v under the action of G. The function also inputs a sublist L of the following list of strings:
["dimension","vertex_degree", "visual_graph", "schlegel", "visual"]
Depending on L, the function displays the following information:\\ the dimension of the orbit polytope $P(G,v)$;\\ the degree of a vertex in the graph of $P(G,v)$;\\ a visualization of the graph of $P(G,v)$;\\ a visualization of the Schlegel diagram of $P(G,v)$;\\ a visualization of the polytope $P(G,v)$ if $d = 2,3$.
The function requires Polymake software.

```
ScatterPlot(L):: List --> Void
```

Inputs a list $L = [[x_1,y_1],\ldots,[x_n,y_n]]$ of pairs of rational numbers and displays a scatter plot of the points in the x-y-plane.

Chapter 2

$\mathbb{Z}G$-Resolutions and Group Cohomology

Resolutions

| |

```
EquivariantChainMap(R,S,f):: FreeResolution, FreeResolution, GroupHomomorphisms --> EquiCh
```

Inputs a free $\mathbb{Z}G$-resolution R of $\mathbb{Z}$, a free $\mathbb{Z}Q$-resolution S of $\mathbb{Z}$, and a group homomorphism $f{:}\,G \to Q$. It returns the induced f-equivariant chain map $F{:}\,R \to S$.

```
FreeGResolution(P,n):: NonFreeResolution, Int --> FreeResolution
```

Inputs a non-free $\mathbb ZG$-resolution $P_\ast$ and a positive integer *n*. It attempts to return *n* terms of a free $\mathbb{Z}G$-resolution of $\mathbb{Z}$. However, the stabilizer groups in the non-free resolution must be such that HAP can construct free resolutions with contracting homotopies for them.
The contracting homotopy on the resolution was implemented by Bui Anh Tuan.

```
ResolutionBieberbachGroup(G):: MatrixGroup --> FreeResolution
```

```
ResolutionBieberbachGroup(G,v):: MatrixGroup, List --> FreeResolution
```

Inputs a torsion free crystallographic group *G*, also known as a Bieberbach group, represented using AFFINECRYSTGROUPONRIGHT as in the GAP package Cryst. It also optionally inputs a choice of vector *v* in the Euclidean space $\mathbb{R}^n$ on which *G* acts freely. The function returns $n+1$ terms of the free ZG-resolution of $\mathbb{Z}$ arising as the cellular chain complex of the tessellation of $\mathbb{R}^n$ by the Dirichlet-Voronoi fundamental domain determined by *v*. No contracting homotopy is returned with the resolution.
This function is part of the HAPcryst package written by Marc Roeder and thus requires the HAPcryst package to be loaded.
The function requires the use of Polymake software.

```
ResolutionCubicalCrustGroup(G,k):: MatrixGroup, Int --> FreeResolution
```

Inputs a crystallographic group *G* represented using AFFINECRYSTGROUPONRIGHT as in the GAP package *Cryst* together with an integer $k \geq 1$. The function tries to find a cubical fundamental domain in the Euclidean space $\mathbb{R}^n$ on which *G* acts. If it succeeds it uses this domain to return $k+1$ terms of a free ZG-resolution of $\mathbb{Z}$.
This function was written by Bui Anh Tuan.

```
ResolutionFiniteGroup(G,k):: Group, Int --> FreeResolution
```

Inputs a finite group G and an integer $k \geq 1$. It returns $k+1$ terms of a free ZG-resolution of $\mathbb{Z}$.

```
ResolutionNilpotentGroup(G,k):: Group, Int --> FreeResolution
```

Inputs a nilpotent group G (which can be infinite) and an integer $k \geq 1$. It returns $k+1$ terms of a free $\mathbb{Z}G$-resolution of $\mathbb{Z}$.

```
ResolutionNormalSeries(L,k):: List, Int --> FreeResolution
```

Inputs a a list L consisting of a chain \$1 $= N_1 \leq N_2 \leq \cdots \leq N_n = G$ of normal subgroups of G, together with an integer $k \geq 1$. It returns $k+1$ terms of a free ZG-resolution of $\mathbb{Z}$.

```
ResolutionPrimePowerGroup(G,k):: Group, Int --> FreeResolution
```

Inputs a finite p-group G and an integer $k \geq 1$. It returns $k+1$ terms of a minimal free $\mathbb{F}G$-resolution of the field $\mathbb{F}$ of p elements.

```
ResolutionSL2Z(m,k):: Int, Int --> FreeResolution
```

Inputs positive integers m,n and returns n terms of a free $\mathbb{Z}G$-resolution of $\mathbb{Z}$ for the group $G = SL_2(\mathbb{Z}[1/m])$.
This function is joint work with Bui Anh Tuan.

```
ResolutionSmallGroup(G,k):: Group, Int --> FreeResolution
```

```
ResolutionSmallGroup(G,k):: FpGroup, Int --> FreeResolution
```

Inputs a small group G and an integer $k \geq 1$. It returns $k+1$ terms of a free ZG-resolution of $\mathbb{Z}$.
If G is a finitely presented group then up to degree \$2\$ the resolution coincides with cellular chain complex of the universal cover of the 2 complex associated to the presentation of G. Thus the boundaries of the generators in degree 3 provide a generating set for the module of identities of the presentation.
This function was written by Irina Kholodna.

```
ResolutionSubgroup(R,H):: FreeResolution, Group --> FreeResolution
```

Inputs a free ZG-resolution of $\mathbb{Z}$ and a finite index subgroup $H \leq G$. It returns a free ZH-resolution of $\mathbb{Z}$.

Algebras $\longrightarrow$ (Co)chain Complexes

| |

```
LeibnizComplex(g,n):: LeibnizAlgebra, Int --> ChainComplex
```

Inputs a Leibniz algebra, or Lie algebra, $\mathfrak{g}$ over a ring $\mathbb{K}$ together with an integer $n \geq 0$. It returns the first n terms of the Leibniz chain complex over $\mathbb{K}$. The complex was implemented by Pablo Fernandez Ascariz.

Resolutions $\longrightarrow$ (Co)chain Complexes

```
HomToIntegers(C):: ChainComplex --> CochainComplex
```

```
HomToIntegers(R):: FreeResolution --> CochainComplex
```

```
HomToIntegers(F):: EquiChainMap --> CochainMap
```

Inputs a chain complex C of free abelian groups and returns the cochain complex $Hom_{\mathbb{Z}}(C,\mathbb{Z})$.
Inputs a free $\mathbb{Z}G$-resolution R in characteristic 0 and returns the cochain complex $Hom_{\mathbb{Z}G}(R,\mathbb{Z})$.
Inputs an equivariant chain map $F\colon R \to S$ of resolutions and returns the induced cochain map $Hom_{\mathbb{Z}G}(S,\mathbb{Z}) \longrightarrow Hom_{\mathbb{Z}G}(R,\mathbb{Z})$.

```
HomToIntegralModule(R,A):: FreeResolution, GroupHomomorphism --> CochainComplex
```

Inputs a free $\mathbb{Z}G$-resolution R in characteristic 0 and a group homomorphism $A\colon G \to \mathrm{GL}_n(\mathbb{Z})$. The homomorphism A can be viewed as the $\mathbb{Z}G$-module with underlying abelian group $\mathbb{Z}^n$ on which G acts via the homomorphism A. It returns the cochain complex $Hom_{\mathbb{Z}G}(R,A)$.

```
TensorWithIntegers(R):: FreeResolution --> ChainComplex
```

```
TensorWithIntegers(F):: EquiChainMap --> ChainMap
```

Inputs a free $\mathbb{Z}G$-resolution R of characteristic 0 and returns the chain complex $R \otimes_{\mathbb{Z}G} \mathbb{Z}$.
Inputs an equivariant chain map $F\colon R \to S$ in characteristic 0 and returns the induced chain map $F \otimes_{\mathbb{Z}G} \mathbb{Z}\colon R \otimes_{\mathbb{Z}G} \mathbb{Z} \longrightarrow S \otimes_{\mathbb{Z}G} \mathbb{Z}$.

```
TensorWithIntegersModP(C,p):: ChainComplex, Int --> ChainComplex
```

```
TensorWithIntegersModP(R,p):: FreeResolution, Int --> ChainComplex
```

```
TensorWithIntegersModP(F,p):: EquiChainMap, Int --> ChainMap
```

Inputs a chain complex C of characteristic 0 and a prime integer p. It returns the chain complex $C \otimes_{\mathbb{Z}} \mathbb{Z}_p$ of characteristic p.
Inputs a free $\mathbb{Z}G$-resolution R of characteristic 0 and a prime integer p. It returns the chain complex $R \otimes_{\mathbb{Z}G} \mathbb{Z}_p$ of characteristic p.
Inputs an equivariant chain map $F\colon R \to S$ in characteristic 0 a prime integer p. It returns the induced chain map $F \otimes_{\mathbb{Z}G} \mathbb{Z}_p\colon R \otimes_{\mathbb{Z}G} \mathbb{Z}_p \longrightarrow S \otimes_{\mathbb{Z}G} \mathbb{Z}_p$.

Cohomology rings

```
AreIsomorphicGradedAlgebras(A,B):: PresentedGradedAlgebra, PresentedGradedAlgebra --> Bool
```

Inputs two freely presented graded algebras $A = \mathbb{F}[x_1,\ldots,x_m]/I$ and $B = \mathbb{F}[y_1,\ldots,y_n]/J$ and returns TRUE if they are isomorphic, and FALSE otherwise. This function was implemented by Paul Smith.

`HAPDerivation(R,I,L):: PolynomialRing, List, List --> Derivation`

Inputs a polynomial ring $R=\mathbb{F}[x_1,\ldots,x_m]$ over a field $\mathbb{F}$ together with a list I of generators for an ideal in R and a list $L=[y_1,\ldots,y_m]\subset R$. It returns the derivation $d\colon E\to E$ for $E=R/I$ defined by $d(x_i)=y_i$. This function was written by Paul Smith. It uses the Singular commutative algebra package.

`HilbertPoincareSeries::PresentedGradedAlgebra --> RationalFunction`

Inputs a presentation $E=\mathbb{F}[x_1,\ldots,x_m]/I$ of a graded algebra and returns its Hilbert-Poincar\'e series. This function was written by Paul Smith and uses the Singular commutative algebra package. It is essentially a wrapper for Singular's Hilbert-Poincare series.

`HomologyOfDerivation(d):: Derivation --> List`

Inputs a derivation $d\colon E\to E$ on a quotient $E=R/I$ of a polynomial ring $R=\mathbb{F}[x_1,\ldots,x_m]$ over a field $\mathbb{F}$. It returns a list $[S,J,h]$ where S is a polynomial ring and J is a list of generators for an ideal in S such that there is an isomorphism $\alpha\colon S/J\to \ker d/\mathrm{im}\, d$. This isomorphism lifts to the ring homomorphism $h\colon S\to \ker d$. This function was written by Paul Smith. It uses the Singular commutative algebra package.

`IntegralCohomologyGenerators(R,n):: FreeResolution, Int --> List`

Inputs at least $n+1$ terms of a free $\mathbb{Z}G$-resolution of $\mathbb{Z}$ and the integer $n\geq 1$. It returns a minimal list of cohomology classes in $H^n(G,\mathbb{Z})$ which, together with all cup products of lower degree classes, generate the group $H^n(G,\mathbb{Z})$. (Let a_i be the i-th canonical generator of the d-generator abelian group $H^n(G,Z)$. The cohomology class $n_1a_1+...+n_da_d$ is represented by the integer vector $u=(n_1,...,n_d)$.)

`LHSSpectralSequence(G,N,r):: Group, Int, Int --> List`

Inputs a finite 2-group G, and normal subgroup N and an integer r. It returns a list of length r whose i-th term is a presentation for the i-th page of the Lyndon-Hochschild-Serre spectral sequence. This function was written by Paul Smith. It uses the Singular commutative algebra package.

`LHSSpectralSequenceLastSheet(G,N):: Group, Int --> List`

Inputs a finite 2-group G and normal subgroup N. It returns presentation for the E_∞ page of the Lyndon-Hochschild-Serre spectral sequence. This function was written by Paul Smith. It uses the Singular commutative algebra package.

`ModPCohomologyGenerators(G,n):: Group, Int --> List`

`ModPCohomologyGenerators(R):: FreeResolution --> List`

Inputs either a p-group G and positive integer n, or else $n+1$ terms of a minimal $\mathbb{F}G$-resolution R of the field $\mathbb{F}$ of p elements. It returns a pair whose first entry is a minimal list of homogeneous generators for the cohomology ring $A=H^*(G,\mathbb{F})$ modulo all elements in degree greater than n. The second entry of the pair is a function DEG which, when applied to a minimal generator, yields its degree. WARNING: the following rule must be applied when multiplying generators x_i together. Only products of the form $x_1*(x_2*(x_3*(x_4*...)))$ with $deg(x_i)\leq deg(x_{i+1})$ should be computed (since the x_i belong to a structure constant algebra with only a partially defined structure constants table).

`ModPCohomologyRing(R):: FreeResolution --> SCAlgebra`

`ModPCohomologyRing(R,level):: FreeResolution, String --> SCAlgebra`

`ModPCohomologyRing(G,n):: Group, Int --> SCAlgebra`

`ModPCohomologyRing(G,n,level):: Group, Int, String --> SCAlgebra`

Inputs either a p-group G and positive integer n, or else n terms of a minimal $\mathbb{F}G$-resolution R of the field $\mathbb{F}$ of p elements. It returns the cohomology ring $A = H^*(G,\mathbb{F})$ modulo all elements in degree greater than n. The ring is returned as a structure constant algebra A. The ring A is graded. It has a component A!.DEGREE(X) which is a function returning the degree of each (homogeneous) element x in GENERATORSOFALGEBRA(A). An optional input variable "*level*" can be set to one of the strings "*medium*" or "*high*". These settings determine parameters in the algorithm. The default setting is "*medium*". When "*level*" is set to "*high*" the ring A is returned with a component A!.NICEBASIS. This component is a pair $[Coeff,Bas]$. Here Bas is a list of integer lists; a "nice" basis for the vector space A can be constructed using the command LIST(BAS,X->PRODUCT(LIST(X,I->BASIS(A)[I]))). The coefficients of the canonical basis element BASIS(A)[I] are stored as COEFF[I]. If the ring A is computed using the setting "*level*" = "*medium*" then the component A!.NICEBASIS can be added to A using the command A:=MODPCOHOMOLOGYRING_PART_2(A).

`Mod2CohomologyRingPresentation(G):: Group --> PresentedGradedAlgebra`

`Mod2CohomologyRingPresentation(G,n):: Group --> PresentedGradedAlgebra`

`Mod2CohomologyRingPresentation(A):: Group --> PresentedGradedAlgebra`

`Mod2CohomologyRingPresentation(R):: Group --> PresentedGradedAlgebra`

When applied to a finite 2-group G this function returns a presentation for the mod-2 cohomology ring $H^*(G,\mathbb{F})$. The Lyndon-Hochschild-Serre spectral sequence is used to prove that the presentation is complete. When the function is applied to a 2-group G and positive integer n the function first constructs $n+1$ terms of a free $\mathbb{F}G$-resolution R, then constructs the finite-dimensional graded algebra $A = H^{(*\leq n)}(G,\mathbb{F})$, and finally uses A to approximate a presentation for $H^*(G,\mathbb{F})$. For "sufficiently large" n the approximation will be a correct presentation for $H^*(G,\mathbb{F})$. Alternatively, the function can be applied directly to either the resolution R or graded algebra A. This function was written by Paul Smith. It uses the Singular commutative algebra package to handle the Lyndon-Hochschild-Serre spectral sequence.

Group Invariants

| |

`GroupCohomology(G,k):: Group, Int --> List`

`GroupCohomology(G,k,p):: Group, Int, Int --> List`

Inputs a group G and integer $k \geq 0$. The group G should either be finite or else lie in one of a range of classes of infinite groups (such as nilpotent, crystallographic, Artin etc.). The function returns the list of abelian invariants of $H^k(G,\mathbb{Z})$.

If a prime p is given as an optional third input variable then the function returns the list of abelian invariants of $H^k(G,\mathbb{Z}_p)$. In this case each abelian invariant will be equal to p and the length of the list will be the dimension of the vector space $H^k(G,\mathbb{Z}_p)$.

```
GroupHomology(G,k):: Group, Int --> List
```

```
GroupHomology(G,k,p):: Group, Int, Int --> List
```

Inputs a group G and integer $k \geq 0$. The group G should either be finite or else lie in one of a range of classes of infinite groups (such as nilpotent, crystallographic, Artin etc.). The function returns the list of abelian invariants of $H_k(G,\mathbb{Z})$.

If a prime p is given as an optional third input variable then the function returns the list of abelian invariants of $H_k(G,\mathbb{Z}_p)$. In this case each abelian invariant will be equal to p and the length of the list will be the dimension of the vector space $H_k(G,\mathbb{Z}_p)$.

```
PrimePartDerivedFunctor(G,R,A,k):: Group, FreeResolution, Function, Int --> List
```

Inputs a group G, an integer $k \geq 0$, at least $k+1$ terms of a free $\mathbb{Z}P$-resolution of $\mathbb{Z}$ for P a Sylow p-subgroup of G. A function such as A=TensorWithIntegers is also entered. The abelian invariants of the p-primary part $H_k(G,A)_{(p)}$ of the homology with coefficients in A is returned.

```
PoincareSeries(G,n):: Group, Int --> RationalFunction
```

```
PoincareSeries(G):: Group --> RationalFunction
```

```
PoincareSeries(R,n):: Group, Int --> RationalFunction
```

```
PoincareSeries(L,n):: Group, Int --> RationalFunction
```

Inputs a finite p-group G and a positive integer n. It returns a quotient of polynomials $f(x) = P(x)/Q(x)$ whose expansion has coefficient of x^k equal to the rank of the vector space $H_k(G,\mathbb{F}_p)$ for all k in the range $1 \leq k \leq n$. (The second input variable can be omitted, in which case the function tries to choose a 'reasonable' value for n. For 2-groups the function PoincareSeriesLHS(G) can be used to produce an $f(x)$ that is correct in all degrees.) In place of the group G the function can also input (at least n terms of) a minimal mod-p resolution R for G. Alternatively, the first input variable can be a list L of integers. In this case the coefficient of x^k in $f(x)$ is equal to the $(k+1)$st term in the list.

```
PoincareSeries(G,n):: Group, Int --> RationalFunction
```

```
PoincareSeries(G):: Group --> RationalFunction
```

```
PoincareSeries(R,n):: Group, Int --> RationalFunction
```

```
PoincareSeries(L,n):: Group, Int --> RationalFunction
```

Inputs a finite p-group G and a positive integer n. It returns a quotient of polynomials $f(x) = P(x)/Q(x)$ whose expansion has coefficient of x^k equal to the rank of the vector space $H_k(G,\mathbb{F}_p)$ for all k in the range $1 \le k \le n$. (The second input variable can be omitted, in which case the function tries to choose a 'reasonable' value for n. For 2-groups the function POINCARESERIESLHS(G) can be used to produce an $f(x)$ that is correct in all degrees.) In place of the group G the function can also input (at least n terms of) a minimal mod-p resolution R for G. Alternatively, the first input variable can be a list L of integers. In this case the coefficient of x^k in $f(x)$ is equal to the $(k+1)$st term in the list.

```
RankHomologyPGroup(G,P,n):: Group, RationalFunction, Int --> Int
```

Inputs a p-group G, a rational function P representing the Poincar\'e series of the mod-p cohomology of G and a positive integer n. It returns the minimum number of generators for the finite abelian p-group $H_nG,\mathbb{Z})$.

$\mathbb{F}_p$-modules

```
GroupAlgebraAsFpGModule:: Group --> FpGModule
```

Inputs a finite p-group G and returns the modular group algebra $\mathbb{F}_pG$ in the form of an $\mathbb{F}_pG$-module.

```
Radical:: FpGModule --> FpGModule
```

Inputs an $\mathbb{F}_pG$-module and returns its radical.

```
RadicalSeries(M):: FpGModule --> List

RadicalSeries(R):: Resolution --> FilteredSparseChainComplex
```

Inputs an $\mathbb{F}_pG$-module M and returns its radical series as a list of $\mathbb{F}_pG$-modules.
Inputs a free $\mathbb{F}_pG$-resolution R and returns the filtered chain complex
$\cdots Rad_2(\mathbb{F}_pG)R \le Rad_1(\mathbb{F}_pG)R \le R$.

Chapter 3

Homological Group Theory

Cocycles

| |

`CcGroup(N,f):: GOuterGroup, StandardCocycle --> CcGroup`

Inputs a G-outer group N with nonabelian cocycle describing some extension $N \rightarrowtail E \twoheadrightarrow G$ together with standard 2-cocycle $f\colon G \times G \to A$ where $A = Z(N)$. It returns the extension group determined by the cocycle f. The group is returned as a cocyclic group.
This function is part of the HAPcocyclic package of functions implemented by Robert F. Morse.

`CocycleCondition(R,n):: FreeRes, Int --> IntMat`

Inputs a free $\mathbb{Z}G$-resolution R of $\mathbb{Z}$ and an integer $n \geq 1$. It returns an integer matrix M with the following property. Let d be the $\mathbb{Z}G$-rank of R_n. An integer vector $f = [f_1, ..., f_d]$ then represents a $\mathbb{Z}G$-homomorphism $R_n \to \mathbb{Z}_q$ which sends the ith generator of R_n to the integer f_i in the trivial $\mathbb{Z}G$-module $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$ (where possibly $q = 0$). The homomorphism f is a cocycle if and only if $M^t f = 0$ mod q.

`StandardCocycle(R,f,n):: FreeRes, List, Int --> Function`

`StandardCocycle(R,f,n,q):: FreeRes, List, Int --> Function`

Inputs a free $\mathbb{Z}G$-resolution R (with contracting homotopy), a positive integer n and an integer vector f representing an n-cocycle $R_n \to \mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$ where G acts trivially on $\mathbb{Z}_q$. It is assumed $q = 0$ unless a value for q is entered. The command returns a function $F(g_1, ..., g_n)$ which is the standard cocycle $G^n \to \mathbb{Z}_q$ corresponding to f. At present the command is implemented only for $n = 2$ or 3.

G-Outer Groups

| |

`ActedGroup(M):: GOuterGroup --> Group`

Inputs a G-outer group M corresponding to a homomorphism $\alpha\colon G \to \mathrm{Out}(N)$ and returns the group N.

`ActingGroup(M):: GOuterGroup --> Group`

Inputs a G-outer group M corresponding to a homomorphism $\alpha\colon G \to \mathrm{Out}(N)$ and returns the group G.

```
Centre(M):: GOuterGroup --> GOuterGroup
```

Inputs a G-outer group M and returns its group-theoretic centre as a G-outer group.

```
GOuterGroup(E,N):: Group, Subgroup --> GOuterGroup
```

```
GOuterGroup():: Group, Subgroup --> GOuterGroup
```

Inputs a group E and normal subgroup N. It returns N as a G-outer group where $G = E/N$. A nonabelian cocycle $f\colon G \times G \to N$ is attached as a component of the G-Outer group.
The function can be used without an argument. In this case an empty outer group C is returned. The components must be set using SETACTINGGROUP(C,G), SETACTEDGROUP(C,N) and SETOUTERACTION(C,ALPHA).

G-cocomplexes

| |

```
CohomologyModule(C,n):: GCocomplex, Int --> GOuterGroup
```

Inputs a G-cocomplex C together with a non-negative integer n. It returns the cohomology $H^n(C)$ as a G-outer group. If C was constructed from a $\mathbb{Z}G$-resolution R by homing to an abelian G-outer group A then, for each x in $H := CohomologyModule(C,n)$, there is a function $f := H!.representativeCocycle(x)$ which is a standard n-cocycle corresponding to the cohomology class x. (At present this is implemented only for $n = 1,2,3$.)

```
HomToGModule(R,A):: FreeRes, GOuterGroup --> GCocomplex
```

Inputs a $\mathbb{Z}G$-resolution R and an abelian G-outer group A. It returns the G-cocomplex obtained by applying $HomZG(\_,A)$. (At present this function does not handle equivariant chain maps.)

Chapter 4

Parallel Computation

Six Core Functions

```
ChildCreate():: Void --> Child process

ChildProcess("computer.address.ie"):: String --> Child process

ChildProcess(["-m", "100000M", "-T"]):: List --> Child process

ChildProcess("computer.ac.wales", ["-m", "100000M", "-T"]):: String, List --> Child proces
```

Starts a GAP session as a child process and returns a stream to the child process. If no argument is given then the child process is created on the local machine; otherwise the argument should be: (1) the address of a remote computer for which ssh has been configured to require no password from the user; (2) or a list of GAP command line options; (3) or the address of a computer followed by a list of command line options.


```
ChildCreate():: Void --> Child process

ChildProcess("computer.address.ie"):: String --> Child process

ChildProcess(["-m", "100000M", "-T"]):: List --> Child process

ChildProcess("computer.ac.wales", ["-m", "100000M", "-T"]):: String, List --> Child proces
```

Starts a GAP session as a child process and returns a stream to the child process. If no argument is given then the child process is created on the local machine; otherwise the argument should be: (1) the address of a remote computer for which ssh has been configured to require no password from the user; (2) or a list of GAP command line options; (3) or the address of a computer followed by a list of command line options.

Chapter 5

Resolutions of the ground ring

`TietzeReducedResolution(R)` Inputs a $\mathbb{Z}G$-resolution R and returns a $\mathbb{Z}G$-resolution S which is obtained from R b
`ResolutionArithmeticGroup("PSL(4,Z)",n)` Inputs a positive integer n and one of the following strings:

"SL(2,Z)" , "SL(3,Z)" , "PGL(3,Z[i])" , "PGL(3,Eisenstein_Integers)" , "PSL(4,Z)" , "PSL(4,Z)_b" , "PSL(4,Z)_c" , "

or the string

"GL(2,O(-d))"

for d=1, 2, 3, 5, 6, 7, 10, 11, 13, 14, 15, 17, 19, 21, 22, 23, 26, 43

or the string

"SL(2,O(-d))"

for d=2, 3, 5, 7, 10, 11, 13, 14, 15, 17, 19, 21, 22, 23, 26, 43, 67, 163

or the string

"SL(2,O(-d))_a"

for d=2, 7, 11, 19.

It returns n terms of a free ZG-resolution for the group G described by the string. Here O(-d) denotes the ring of integ

Data for the first list of resolutions was provided provided by MATHIEU DUTOUR. Data for GL(2,O(-d)) was provide
`FreeGResolution(P,n)` `FreeGResolution(P,n,p)` Inputs a non-free ZG-resolution P with finite stabilizer groups
`ResolutionGTree(P,n)` Inputs a non-free ZG-resolution P of dimension 1 (i.e. a G-tree) with finite stabilizer group
`ResolutionSL2Z(p,n)` Inputs positive integers m,n and returns n terms of a ZG-resolution for the group $G = SL(2,$
`ResolutionAbelianGroup(L,n)` `ResolutionAbelianGroup(G,n)` Inputs a list $L := [m_1, m_2, ..., m_d]$ of nonnegativ
`ResolutionAlmostCrystalGroup(G,n)` Inputs a positive integer n and an almost crystallographic pcp group G. It r
`ResolutionAlmostCrystalQuotient(G,n,c)` `ResolutionAlmostCrystalQuotient(G,n,c,false)` An almost
`ResolutionArtinGroup(D,n)` Inputs a Coxeter diagram D and an integer $n > 1$. It returns n terms of a free ZG-reso
`ResolutionAsphericalPresentation(F,R,n)` Inputs a free group F, a set R of words in F which constitute an as
`ResolutionBieberbachGroup( G )` `ResolutionBieberbachGroup( G, v )` Inputs a torsion free crystallogra
`ResolutionCoxeterGroup(D,n)` Inputs a Coxeter diagram D and an integer $n > 1$. It returns k terms of a free ZG-r
`ResolutionDirectProduct(R,S)` `ResolutionDirectProduct(R,S,"internal")` Inputs a ZG-resolution R and
`ResolutionExtension(g,R,S)` `ResolutionExtension(g,R, S,"TestFiniteness")` `ResolutionExtension`
`ResolutionFiniteDirectProduct(R,S)` `ResolutionFiniteDirectProduct(R,S, "internal")` Inputs a ZG
`ResolutionFiniteExtension(gensE,gensG,R,n)` `ResolutionFiniteExtension(gensE,gensG,R,n,true)`
`ResolutionFiniteGroup(gens,n)` `ResolutionFiniteGroup(gens,n,true)` `ResolutionFiniteGroup(gens,`
`ResolutionFiniteSubgroup(R,K)` `ResolutionFiniteSubgroup(R,gensG,gensK)` Inputs a ZG-resolution for a
`ResolutionGraphOfGroups(D,n)` `ResolutionGraphOfGroups(D,n,L)` Inputs a graph of groups D and a positi
`ResolutionNilpotentGroup(G,n)` `ResolutionNilpotentGroup(G,n,"TestFiniteness")` Inputs a nilpotent
`ResolutionNormalSeries(L,n)` `ResolutionNormalSeries(L,n,true)` `ResolutionNormalSeries(L,n,fal`
`ResolutionPrimePowerGroup(P,n)` `ResolutionPrimePowerGroup(G,n,p)` Inputs a p-group P and integer n>0
`ResolutionSmallFpGroup(G,n)` `ResolutionSmallFpGroup(G,n,p)` Inputs a small finitely presented group G
`ResolutionSubgroup(R,K)` Inputs a ZG-resolution for an (infinite) group G and a subgroup K of finite index $|G:K$
`ResolutionSubnormalSeries(L,n)` Inputs a positive integer n and a list $L = [L_1, \ldots, L_k]$ of subgroups L_i of a finit
`TwistedTensorProduct(R,S,EhomG,GmapE,NhomE,NEhomN,EltsE,Mult,InvE)` Inputs a ZG-resolution R, a ZN-
`ConjugatedResolution(R,x)` Inputs a ZG-resoluton R and an element x from some group containing G. It returns a
`RecalculateIncidenceNumbers(R)` Inputs a ZG-resoluton R which arises as the cellular chain complex of a regula

Chapter 6

Resolutions of modules

`ResolutionFpGModule(M,n)` Inputs an FpG-module M and a positive integer n. It returns n terms of a minimal free

Chapter 7

Induced equivariant chain maps

`EquivariantChainMap(R,S,f)` Inputs a ZG-resolution R, a ZG'-resolution S, and a group homomorphism $f : G \longrightarrow$

Chapter 8

Functors

•

`ExtendScalars(R,G,EltsG)` Inputs a ZH-resolution R, a group G containing H as a subgroup, and a list $EltsG$ of e
`HomToIntegers(X)` Inputs either a ZG-resolution $X = R$, or an equivariant chain map $X = (F : R \longrightarrow S)$. It returns
`HomToIntegersModP(R)` Inputs a ZG-resolution R and returns the cochain complex obtained by applying $HomZG($
`HomToIntegralModule(R,f)` Inputs a ZG-resolution R and a group homomorphism $f : G \longrightarrow GL_n(Z)$ to the group
`TensorWithIntegralModule(R,f)` Inputs a ZG-resolution R and a group homomorphism $f : G \longrightarrow GL_n(Z)$ to the
`HomToGModule(R,A)` Inputs a ZG-resolution R and an abelian G-outer group A. It returns the G-cocomplex obtaine
`InduceScalars(R,hom)` Inputs a ZQ-resolution R and a surjective group homomorphism $hom : G \to Q$. It returns t
`LowerCentralSeriesLieAlgebra(G)` `LowerCentralSeriesLieAlgebra(f)` Inputs a pcp group G. If each quo
`TensorWithIntegers(X)` Inputs either a ZG-resolution $X = R$, or an equivariant chain map $X = (F : R \longrightarrow S)$. It r
`FilteredTensorWithIntegers(R)` Inputs a ZG-resolution R for which "filteredDimension" lies in NamesOfComp
`TensorWithTwistedIntegers(X,rho)` Inputs either a ZG-resolution $X = R$, or an equivariant chain map $X = (F :$
`TensorWithIntegersModP(X,p)` Inputs either a ZG-resolution $X = R$, or a characteristics 0 chain complex, or an e
`TensorWithTwistedIntegersModP(X,p,rho)` Inputs either a ZG-resolution $X = R$, or an equivariant chain map X
`TensorWithRationals(R)` Inputs a ZG-resolution R and returns the chain complex obtained by tensoring with the tr

Chapter 9

Chain complexes

`ChainComplex(T)` Inputs a pure cubical complex, or cubical complex, or simplicial complex T and returns the (often
`ChainComplexOfPair(T,S)` Inputs a pure cubical complex or cubical complex T and contractible subcomplex S. It r
`ChevalleyEilenbergComplex(X,n)` Inputs either a Lie algebra $X = A$ (over the ring of integers Z or over a field
`LeibnizComplex(X,n)` Inputs either a Lie or Leibniz algebra $X = A$ (over the ring of integers Z or over a field K)
`SuspendedChainComplex(C)` Inputs a chain complex C and returns the chain complex S defined by applying the deg
`ReducedSuspendedChainComplex(C)` Inputs a chain complex C and returns the chain complex S defined by applyin
`CoreducedChainComplex(C)` `CoreducedChainComplex(C,2)` Inputs a chain complex C and returns a quasi-isomo
`TensorProductOfChainComplexes(C,D)` Inputs two chain complexes C and D of the same characteristic and return
`LefschetzNumber(F)` Inputs a chain map $F: C \to C$ with common source and target. It returns the Lefschetz number

Chapter 10

Sparse Chain complexes

`SparseMat(A)` Inputs a matrix A and returns the matrix in sparse format.
`TransposeOfSparseMat(A)` Inputs a sparse matrix A and returns its transpose sparse format.
`ReverseSparseMat(A)` Inputs a sparse matrix A and modifies it by reversing the order of the columns. This function
`SparseRowMult(A,i,k)` Multiplies the i-th row of a sparse matrix A by k. The sparse matrix A is modified but nothi
`SparseRowInterchange(A,i,k)` Interchanges the i-th and j-th rows of a sparse matrix A by k. The sparse matrix A
`SparseRowAdd(A,i,j,k)` Adds k times the j-th row to the i-th row of a sparse matrix A. The sparse matrix A is mod
`SparseSemiEchelon(A)` Converts a sparse matrix A to semi-echelon form (which means echelon form up to a permu
`RankMatDestructive(A)` Returns the rank of a sparse matrix A. The sparse matrix A is modified during the calculat
`RankMat(A)` Returns the rank of a sparse matrix A.
`SparseChainComplex(Y)` Inputs a regular CW-complex Y and returns a sparse chain complex which is chain homot
`SparseChainComplexOfRegularCWComplex(Y)` Inputs a regular CW-complex Y and returns its cellular chain comp
`SparseBoundaryMatrix(C,n)` Inputs a sparse chain complex C and integer n. Returns the n-th boundary matrix of t
`Bettinumbers(C,n)` Inputs a sparse chain complex C and integer n. Returns the n-th Netti number of the chain com

Chapter 11

Homology and cohomology groups

`Cohomology(X,n)` Inputs either a cochain complex $X = C$ (or G-cocomplex C) or a cochain map $X = (C \longrightarrow D)$ in
`CohomologyModule(C,n)` Inputs a G-cocomplex C together with a non-negative integer n. It returns the cohomolog
`CohomologyPrimePart(C,n,p)` Inputs a cochain complex C in characteristic 0, a positive integer n, and a prime p. I
`GroupCohomology(X,n)` `GroupCohomology(X,n,p)` Inputs a positive integer n and eithera finite group $X = G$ or a
`GroupHomology(X,n)` `GroupHomology(X,n,p)` Inputs a positive integer n and eithera finite group $X = G$ or a nilpo
`PersistentHomologyOfQuotientGroupSeries(S,n)` `PersistentHomologyOfQuotientGroupSeries(S,n,p,R`
`PersistentCohomologyOfQuotientGroupSeries(S,n)` `PersistentCohomologyOfQuotientGroupSeries(S,n`
`UniversalBarCode("UpperCentralSeries",n,d)` `UniversalBarCode("UpperCentralSeries",n,d,k)` Input
`PersistentHomologyOfSubGroupSeries(S,n)` `PersistentHomologyOfSubGroupSeries(S,n,p,Resolution_`
`PersistentHomologyOfFilteredChainComplex(C,n,p)` Inputs a filtered chain complex C (of characteristic 0 or
`PersistentHomologyOfCommutativeDiagramOfPGroups(D,n)` Inputs a commutative diagram D of finite p-grou
`PersistentHomologyOfFilteredPureCubicalComplex(M,n)` Inputs a filtered pure cubical complex M and a non
`PersistentHomologyOfPureCubicalComplex(L,n,p)` Inputs a positive integer n, a prime p and an increasing cha
`ZZPersistentHomologyOfPureCubicalComplex(L,n,p)` Inputs a positive integer n, a prime p and any sequence L
`RipsHomology(G,n)` `RipsHomology(G,n,p)` Inputs a graph G, a non-negative integer n (and optionally a prime nu
`BarCode(P)` Inputs an integer persistence matrix P and returns the same information in the form of a binary matrix (c
`BarCodeDisplay(P)` `BarCodeDisplay(P,"mozilla")` `BarCodeCompactDisplay(P)` `BarCodeCompactDisplay`
`Homology(X,n)` Inputs either a chain complex $X = C$ or a chain map $X = (C \longrightarrow D)$.If $X = C$ then the torsion coeffic
`HomologyPb(C,n)` This is a back-up function which might work in some instances where $Homology(C,n)$ fails. It is
`HomologyVectorSpace(X,n)` Inputs either a chain complex $X = C$ or a chain map $X = (C \longrightarrow D)$ in prime characte
`HomologyPrimePart(C,n,p)` Inputs a chain complex C in characteristic 0, a positive integer n, and a prime p. It retu
`LeibnizAlgebraHomology(A,n)` Inputs a Lie or Leibniz algebra $X = A$ (over the ring of integers Z or over a field K
`LieAlgebraHomology(A,n)` Inputs a Lie algebra A (over the integers or a field) and a positive integer n. It returns th
`PrimePartDerivedFunctor(G,R,F,n)` Inputs a finite group G, a positive integer n, at least $n+1$ terms of a ZP-reso
`RankHomologyPGroup(G,n)` `RankHomologyPGroup(R,n)` `RankHomologyPGroup(G,n,"empirical")` Inputs a (sm
`RankPrimeHomology(G,n)` Inputs a (smallish) p-group G together with a positive integer n. It returns a function dim

Chapter 12

Poincare series

•

`EfficientNormalSubgroups(G)` `EfficientNormalSubgroups(G,k)` Inputs a prime-power group G and, optional

`ExpansionOfRationalFunction(f,n)` Inputs a positive integer n and a rational function $f(x) = p(x)/q(x)$ where t

`PoincareSeries(G,n)` `PoincareSeries(R,n)` `PoincareSeries(L,n)` `PoincareSeries(G)` Inputs a fi

`PoincareSeriesPrimePart(G,p,n)` Inputs a finite group G, a prime p, and a positive integer n. It returns a quotie

`PoincareSeriesLHS(G)` Inputs a finite 2-group G and returns a quotient of polynomials $f(x) = P(x)/Q(x)$ whose c

`Prank(G)` Inputs a p-group G and returns the rank of the largest elementary abelian subgroup.

Chapter 13

Cohomology ring structure

`IntegralCupProduct(R,u,v,p,q)` `IntegralCupProduct(R,u,v,p,q,P,Q,N)` (Various functions used to co
`IntegralRingGenerators(R,n)` Inputs at least $n+1$ terms of a ZG-resolution and integer $n>0$. It returns a minim
`ModPCohomologyGenerators(G,n)` `ModPCohomologyGenerators(R)` Inputs either a p-group G and positive int
`ModPCohomologyRing(G,n)` `ModPCohomologyRing(G,n,level)` `ModPCohomologyRing(R)` `ModPCohomolog`
`ModPRingGenerators(A)` Inputs a mod p cohomology ring A (created using the preceeding function). It returns a
`Mod2CohomologyRingPresentation(G)` `Mod2CohomologyRingPresentation(G,n)` `Mod2CohomologyRingPr`

Chapter 14

Cohomology rings of p-groups (mainly $p = 2$)

The functions on this page were written by PAUL SMITH. (They are included in HAP but they are also independently included in Paul Smiths HAPprime package.)

- `Mod2CohomologyRingPresentation(G)` `Mod2CohomologyRingPresentation(G,n)` `Mod2CohomologyRingPr`
`PoincareSeriesLHS(G)` Inputs a finite 2-group G and returns a quotient of polynomials $f(x) = P(x)/Q(x)$ whose c

Chapter 15

Commutator and nonabelian tensor computations

`BaerInvariant(G,c)` Inputs a nilpotent group G and integer c>0. It returns the Baer invariant $M^(c)(G)$ defined as
`BogomolovMultiplier(G)`
`BogomolovMultiplier(G, "standard")` `BogomolovMultiplier(G, "homology")` `BogomolovMultiplier(`
`Bogomology(G,n)` Inputs a finite group G and positive integer n, and returns the quotient $H_n(G,Z)/K(G)$ of the deg
`Coclass(G)` Inputs a group G of prime-power order p^n and nilpotency class c say. It returns the integer $r = n - c$.
`EpiCentre(G,N)` `EpiCentre(G)` Inputs a finite group G and normal subgroup N and returns a subgroup $Z^*(G,N)$ o
`NonabelianExteriorProduct(G,N)` Inputs a finite group G and normal subgroup N. It returns a record E with the
`NonabelianSymmetricKernel(G)` `NonabelianSymmetricKernel(G,m)` Inputs a finite or nilpotent infinite grou
`NonabelianSymmetricSquare(G)` `NonabelianSymmetricSquare(G,m)` Inputs a finite or nilpotent infinite grou
`NonabelianTensorProduct(G,N)` Inputs a finite group G and normal subgroup N. It returns a record E with the fo
`NonabelianTensorSquare(G)` `NonabelianTensorSquare(G,m)` Inputs a finite or nilpotent infinite group G and
`RelativeSchurMultiplier(G,N)` Inputs a finite group G and normal subgroup N. It returns the homology group
`TensorCentre(G)` Inputs a group G and returns the largest central subgroup N such that the induced homomorphism
`ThirdHomotopyGroupOfSuspensionB(G)` `ThirdHomotopyGroupOfSuspensionB(G,m)` Inputs a finite or nilpote
`UpperEpicentralSeries(G,c)` Inputs a nilpotent group G and an integer c. It returns the c-th term of the upper ep

Chapter 16

Lie commutators and nonabelian Lie tensors

•

Functions on this page are joint work with HAMID MOHAMMADZADEH, and implemented by him.

`LieCoveringHomomorphism(L)` Inputs a finite dimensional Lie algebra L over a field, and returns a surjective Lie ho

`LeibnizQuasiCoveringHomomorphism(L)` Inputs a finite dimensional Lie algebra L over a field, and returns a surje

`LieEpiCentre(L)` Inputs a finite dimensional Lie algebra L over a field, and returns an ideal $Z^*(L)$ of the centre of L

`LieExteriorSquare(L)` Inputs a finite dimensional Lie algebra L over a field. It returns a record E with the follow

`LieTensorSquare(L)` Inputs a finite dimensional Lie algebra L over a field and returns a record T with the followir

`LieTensorCentre(L)` Inputs a finite dimensional Lie algebra L over a field and returns the largest ideal N such that

Chapter 17

Generators and relators of groups

•

`CayleyGraphOfGroupDisplay(G,X)` `CayleyGraphOfGroupDisplay(G,X,"mozilla")` Inputs a finite group G
`IdentityAmongRelatorsDisplay(R,n)` `IdentityAmongRelatorsDisplay(R,n,"mozilla")` Inputs a free ZG
`IsAspherical(F,R)` Inputs a free group F and a set R of words in F. It performs a test on the 2-dimensional CW-s
`PresentationOfResolution(R)` Inputs at least two terms of a reduced ZG-resolution R and returns a record P wit
`TorsionGeneratorsAbelianGroup(G)` Inputs an abelian group G and returns a generating set $[x_1,\ldots,x_n]$ where no

Chapter 18

Orbit polytopes and fundamental domains

•

`CoxeterComplex(D)` `CoxeterComplex(D,n)` Inputs a Coxeter diagram D of finite type. It returns a non-free ZW-re
`ContractibleGcomplex("PSL(4,Z)")` Inputs one of the following strings:

"SL(2,Z)" , "SL(3,Z)" , "PGL(3,Z[i])" , "PGL(3,Eisenstein_Integers)" , "PSL(4,Z)" , "PSL(4,Z)_b" , "PSL(4,Z)_c" , "

or one of the following strings

"SL(2,O-2)" , "SL(2,O-7)" , "SL(2,O-11)" , "SL(2,O-19)" , "SL(2,O-43)" , "SL(2,O-67)" , "SL(2,O-163)"

It returns a non-free ZG-resolution for the group G described by the string. The stabilizer groups of cells are finite. (S

Data for the first list of non-free resolutions was provided provided by MATHIEU DUTOUR. Data for the second list w
`QuotientOfContractibleGcomplex(C,D)` Inputs a non-free ZG-resolution C and a finite subgroup D of G which is
`TruncatedGComplex(R,m,n)` Inputs a non-free ZG-resolution R and two positive integers m and n. It returns the no
`FundamentalDomainStandardSpaceGroup(v,G)` Inputs a crystallographic group G (represented using AffineCryst
`OrbitPolytope(G,v,L)` Inputs a permutation group or matrix group G of degree n and a rational vector v of length
`PolytopalComplex(G,v)` `PolytopalComplex(G,v,n)` Inputs a permutation group or matrix group G of degree
`PolytopalGenerators(G,v)` Inputs a permutation group or matrix group G of degree n and a rational vector v of l
`VectorStabilizer(G,v)` Inputs a permutation group or matrix group G of degree n and a rational vector of degree

Chapter 19

Cocycles

•

`CcGroup(A,f)` Inputs a G-module A (i.e. an abelian G-outer group) and a standard 2-cocycle f $GxG--->A$. It r
`CocycleCondition(R,n)` Inputs a resolution R and an integer n>0. It returns an integer matrix M with the followin
`StandardCocycle(R,f,n)`
`StandardCocycle(R,f,n,q)` Inputs a ZG-resolution R (with contracting homotopy), a positive integer n and an in
`Syzygy(R,g)` Inputs a ZG-resolution R (with contracting homotopy) and a list $g=[g[1],...,g[n]]$ of elements in G. I

Chapter 20

Words in free ZG-modules

`AddFreeWords(v,w)` Inputs two words v, w in a free ZG-module and returns their sum $v+w$. If the characteristic of
`AddFreeWordsModP(v,w,p)` Inputs two words v, w in a free ZG-module and the characteristic p of Z. It returns the
`AlgebraicReduction(w)`
`AlgebraicReduction(w,p)` Inputs a word w in a free ZG-module and returns a reduced version of the word in whi
`Multiply Word(n,w)` Inputs a word w and integer n. It returns the scalar multiple $n \cdot w$.
`Negate([i,j])` Inputs a pair $[i, j]$ of integers and returns $[-i, j]$.
`NegateWord(w)` Inputs a word w in a free ZG-module and returns the negated word $-w$.
`PrintZGword(w,elts)` Inputs a word w in a free ZG-module and a (possibly partial but sufficient) listing elts of the
`TietzeReduction(S,w)` Inputs a set S of words in a free ZG-module, and a word w in the module. The function re
`ResolutionBoundaryOfWord(R,n,w)` Inputs a resolution R, a positive integer n and a list w representing a word in t

Chapter 21

FpG-modules

`CompositionSeriesOfFpGModules(M)` Inputs an FpG-module M and returns a list of FpG-modules that constitut
`DirectSumOfFpGModules(M,N)` `DirectSumOfFpGModules([ M[1], M[2], ..., M[k] ])` Inputs two FpG-
`FpGModule(A,P)` `FpGModule(A,G,p)` Inputs a p-group P and a matrix A whose rows have length a multiple of the
`FpGModuleDualBasis(M)` Inputs an FpG-module M. It returns a record R with two components:$R.freeModule$ is t
`FpGModuleHomomorphism(M,N,A)` `FpGModuleHomomorphismNC(M,N,A)` Inputs FpG-modules M and N over a c
`DesuspensionFpGModule(M,n)` `DesuspensionFpGModule(R,n)` Inputs a positive integer n and and FpG-modul
`RadicalOfFpGModule(M)` Inputs an FpG-module M with G a p-group, and returns the Radical of M as an FpG-mo
`RadicalSeriesOfFpGModule(M)` Inputs an FpG-module M and returns a list of FpG-modules that constitute the r
`GeneratorsOfFpGModule(M)` Inputs an FpG-module M and returns a matrix whose rows correspond to a minimal
`ImageOfFpGModuleHomomorphism(f)` Inputs an FpG-module homomorphism $f : M \longrightarrow N$ and returns its image f
`GroupAlgebraAsFpGModule(G)` Inputs a p-group G and returns its mod p group algebra as an FpG-module.
`IntersectionOfFpGModules(M,N)` Inputs two FpG-modules M,N arising as submodules in a common free modu
`IsFpGModuleHomomorphismData(M,N,A)` Inputs FpG-modules M and N over a common p-group G. Also inputs a
`MaximalSubmoduleOfFpGModule(M)` Inputs an FpG-module M and returns one maximal FpG-submodule of M.
`MaximalSubmodulesOfFpGModule(M)` Inputs an FpG-module M and returns the list of maximal FpG-submodules
`MultipleOfFpGModule(w,M)` Inputs an FpG-module M and a list $w := [g_1,...,g_t]$ of elements in the group $G = M!$
`ProjectedFpGModule(M,k)` Inputs an FpG-module M of ambient dimension $n|G|$, and an integer k between 1 and
`RandomHomomorphismOfFpGModules(M,N)` Inputs two FpG-modules M and N over a common group G. It returns
`Rank(f)` Inputs an FpG-module homomorphism $f : M \longrightarrow N$ and returns the dimension of the image of f as a vect
`SumOfFpGModules(M,N)` Inputs two FpG-modules M,N arising as submodules in a common free module $(FG)^n$ w
`SumOp(f,g)` Inputs two FpG-module homomorphisms $f,g : M \longrightarrow N$ with common sorce and common target. It re
`VectorsToFpGModuleWords(M,L)` Inputs an FpG-module M and a list $L = [v_1,\ldots,v_k]$ of vectors in M. It returns a

Chapter 22

Meataxe modules

•

`DesuspensionMtxModule(M)` Inputs a meataxe module M over the field of p elements and returns an FpG-module D
`FpG_to_MtxModule(M)` Inputs an FpG-module M and returns an isomorphic meataxe module.
`GeneratorsOfMtxModule(M)` Inputs a meataxe module M acting on, say, the vector space V. The function returns a

Chapter 23

G-Outer Groups

`GOuterGroup(E,N)` `GOuterGroup()` Inputs a group E and normal subgroup N. It returns N as a G-outer group wher
`GOuterGroupHomomorphismNC(A,B,phi)` `GOuterGroupHomomorphismNC()` Inputs G-outer groups A and B with c
`GOuterHomomorphismTester(A,B,phi)` Inputs G-outer groups A and B with common acting group, and a group ho
`Centre(A)` Inputs G-outer group A and returns the group theoretic centre of ActedGroup(A) as a G-outer group.
`DirectProductGog(A,B)` `DirectProductGog(Lst)` Inputs G-outer groups A and B with common acting group, an

Chapter 24

Cat-1-groups

`AutomorphismGroupAsCatOneGroup(G)` Inputs a group G and returns the Cat-1-group C corresponding to the cross
`HomotopyGroup(C,n)` Inputs a cat-1-group C and an integer n. It returns the nth homotopy group of C.
`HomotopyModule(C,2)` Inputs a cat-1-group C and an integer n=2. It returns the second homotopy group of C as a G
`QuasiIsomorph(C)` Inputs a cat-1-group C and returns a cat-1-group D for which there exists some homomorphism
`ModuleAsCatOneGroup(G,alpha,M)` Inputs a group G, an abelian group M and a homomorphism $\alpha: G \to Aut(M)$.
`MooreComplex(C)` Inputs a cat-1-group C and returns its Moore complex as a G-complex (i.e. as a complex of group
`NormalSubgroupAsCatOneGroup(G,N)` Inputs a group G with normal subgroup N. It returns the Cat-1-group C corr
`XmodToHAP(C)` Inputs a cat-1-group C obtained from the Xmod package and returns a cat-1-group D for which IsHap

Chapter 25

Simplicial groups

`NerveOfCatOneGroup(G,n)` Inputs a cat-1-group G and a positive integer n. It returns the low-dimensional part of th

This function applies both to cat-1-groups for which IsHapCatOneGroup(G) is true, and to cat-1-groups produced usi

This function was implemented by VAN LUYEN LE.
`EilenbergMacLaneSimplicialGroup(G,n,dim)` Inputs a group G, a positive integer n, and a positive integer dim.

This function was implemented by VAN LUYEN LE.
`EilenbergMacLaneSimplicialGroupMap(f,n,dim)` Inputs a group homomorphism $f: G \to Q$, a positive integer n

This function was implemented by VAN LUYEN LE.
`MooreComplex(G)` Inputs a simplicial group G and returns its Moore complex as a G-complex.

This function was implemented by VAN LUYEN LE.
`ChainComplexOfSimplicialGroup(G)` Inputs a simplicial group G and returns the cellular chain complex C of a C

This function was implemented by VAN LUYEN LE.
`SimplicialGroupMap(f)` Inputs a homomorphism $f: G \to Q$ of simplicial groups. The function returns an induced

This function was implemented by VAN LUYEN LE.
`HomotopyGroup(G,n)` Inputs a simplicial group G and a positive integer n. The integer n must be less than the length
`Representation of elements in the bar resolution` For a group G we denote by $B_n(G)$ the free $\mathbb{Z}G$-module

We represent a word

$w = h_1.[g_{11}|g_{12}|...|g_{1n}] - h_2.[g_{21}|g_{22}|...|g_{2n}] + ... + h_k.[g_{k1}|g_{k2}|...|g_{kn}]$

in $B_n(G)$ as a list of lists:

$[[+1, h_1, g_{11}, g_{12}, ..., g_{1n}], [-1, h_2, g_{21}, g_{22}, ...|g_{2n}] + ... + [+1, h_k, g_{k1}, g_{k2}, ..., g_{kn}]$.
`BarResolutionBoundary(w)` This function inputs a word w in the bar resolution module $B_n(G)$ and returns its imag

This function was implemented by VAN LUYEN LE.
`BarResolutionHomotopy(w)` This function inputs a word w in the bar resolution module $B_n(G)$ and returns its imag

This function is currently being implemented by VAN LUYEN LE.
`Representation of elements in the bar complex` For a group G we denote by $BC_n(G)$ the free abelian group

We represent a word

$w = [g_{11}|g_{12}|...|g_{1n}] - [g_{21}|g_{22}|...|g_{2n}] + ... + [g_{k1}|g_{k2}|...|g_{kn}]$

in $BC_n(G)$ as a list of lists:

$[[+1, g_{11}, g_{12}, ..., g_{1n}], [-1, g_{21}, g_{22}, ...|g_{2n}] + ... + [+1, g_{k1}, g_{k2}, ..., g_{kn}]$.
`BarComplexBoundary(w)` This function inputs a word w in the n-th term of the bar complex $BC_n(G)$ and returns its i

This function was implemented by VAN LUYEN LE.
`BarResolutionEquivalence(R)` This function inputs a free ZG-resolution R. It returns a component object HE wit

$equiv(n,-): B_n(G) \to B_{n+1}(G)$

satisfying w - $\psi(\phi(w)) = d(n+1, equiv(n,w)) + equiv(n-1, d(n,w)).where$d(n,-)$: B_n(G) \to B_{n-1}(G)$ is the boundar

This function was implemented by VAN LUYEN LE.

`BarComplexEquivalence(R)`

This function inputs a free ZG-resolution R. It first constructs the chain complex $T = TensorWithIntegerts(R)$. The function returns a component object HE with components

- HE!.phi(n,w) is a function which inputs a non-negative integer n and a word w in $BC_n(G)$. It returns the image of w in T_n under a chain equivalence $\phi\colon BC_n(G) \to T_n$.
- HE!.psi(n,w) is a function which inputs a non-negative integer n and an element w in T_n. It returns the image of w in $BC_n(G)$ under a chain equivalence $\psi\colon T_n \to BC_n(G)$.
- HE!.equiv(n,w) is a function which inputs a non-negative integer n and a word w in $BC_n(G)$. It returns the image of w in $BC_{n+1}(G)$ under a homomorphism $equiv(n,-)\colon BC_n(G) \to BC_{n+1}(G)$ satisfying
$$w - \psi(\phi(w)) = d(n+1, equiv(n,w)) + equiv(n-1, d(n,w)).$$
where $d(n,-)\colon BC_n(G) \to BC_{n-1}(G)$ is the boundary homomorphism in the bar complex.

This function was implemented by VAN LUYEN LE.

`Representation of elements in the bar cocomplex`

For a group G we denote by $BC^n(G)$ the free abelian group with basis the lists $[g_1|g_2|...|g_n]$ where the g_i range over G.

We represent a word
$$w = [g_{11}|g_{12}|...|g_{1n}] - [g_{21}|g_{22}|...|g_{2n}] + ... + [g_{k1}|g_{k2}|...|g_{kn}]$$
in $BC^n(G)$ as a list of lists:
$$[[+1, g_{11}, g_{12}, ..., g_{1n}], [-1, g_{21}, g_{22}, ...|g_{2n}] + ... + [+1, g_{k1}, g_{k2}, ..., g_{kn}].$$

`BarCocomplexCoboundary(w)`

This function inputs a word w in the n-th term of the bar cocomplex $BC^n(G)$ and returns its image under the coboundary homomorphism $d^n\colon BC^n(G) \to BC^{n+1}(G)$ in the bar cocomplex.

This function was implemented by VAN LUYEN LE.

Chapter 26

Coxeter diagrams and graphs of groups

`CoxeterDiagramComponents(D)` Inputs a Coxeter diagram D and returns a list $[D_1, ..., D_d]$ of the maximal connect
`CoxeterDiagramDegree(D,v)` Inputs a Coxeter diagram D and vertex v. It returns the degree of v (i.e. the number
`CoxeterDiagramDisplay(D)` `CoxeterDiagramDisplay(D,"web browser")` Inputs a Coxeter diagram D and d
`CoxeterDiagramFpArtinGroup(D)` Inputs a Coxeter diagram D and returns the corresponding finitely presented A
`CoxeterDiagramFpCoxeterGroup(D)` Inputs a Coxeter diagram D and returns the corresponding finitely presentec
`CoxeterDiagramIsSpherical(D)` Inputs a Coxeter diagram D and returns "true" if the associated Coxeter groups
`CoxeterDiagramMatrix(D)` Inputs a Coxeter diagram D and returns a matrix representation of it. The matrix is giv
`CoxeterSubDiagram(D,V)` Inputs a Coxeter diagram D and a subset V of its vertices. It returns the full sub-diagrar
`CoxeterDiagramVertices(D)` Inputs a Coxeter diagram D and returns its set of vertices.
`EvenSubgroup(G)` Inputs a group G and returns a subgroup G^+. The subgroup is that generated by all products xy v
`GraphOfGroupsDisplay(D)` `GraphOfGroupsDisplay(D,"web browser")` Inputs a graph of groups D and dis
`GraphOfResolutions(D,n)` Inputs a graph of groups D and a positive integer n. It returns a graph of resolutions,
`GraphOfGroups(D)` Inputs a graph of resolutions D and returns the corresponding graph of groups.
`GraphOfResolutionsDisplay(D)` Inputs a graph of resolutions D and displays it as a .gif file. It uses the Mozilla
`GraphOfGroupsTest(D)` Inputs an object D and itries to test whether it is a Graph of Groups. However, it DOES N
`TreeOfGroupsToContractibleGcomplex(D,G)` Inputs a graph of groups D which is a tree, and also inputs the fur
`TreeOfResolutionsToContractibleGcomplex(D,G)` Inputs a graph of resolutions D which is a tree, and also in

#

Chapter 27

Torsion Subcomplexes

The Torsion Subcomplex subpackage has been conceived and implemented by BUI ANH TUAN and ALEXANDER D
`RigidFacetsSubdivision( X )` It inputs an n-dimensional G-equivariant CW-complex X on which all the cell stab
`IsPNormal( G, p)` Inputs a finite group G and a prime p. Checks if the group G is p-normal for the prime p. Zasse
`TorsionSubcomplex( C, p)` Inputs either a cell complex with action of a group as a variable or a group name. In H

"SL(2,O-2)" , "SL(2,O-7)" , "SL(2,O-11)" , "SL(2,O-19)" , "SL(2,O-43)" , "SL(2,O-67)" , "SL(2,O-163)",

where the symbol O[-m] stands for the ring of integers in the imaginary quadratic number field Q(sqrt(-m)), the latter

The function TorsionSubcomplex prints the cells with p-torsion in their stabilizer on the screen and returns the incide

It is also possible to input the cell complexes

"SL(2,Z)" , "SL(3,Z)" , "PGL(3,Z[i])" , "PGL(3,Eisenstein_Integers)" , "PSL(4,Z)" , "PSL(4,Z)_b" , "PSL(4,Z)_c" , "

provided by MATHIEU DUTOUR.
`DisplayAvailableCellComplexes();` Displays the cell complexes that are available in HAP.
`VisualizeTorsionSkeleton( groupName, p)` Executes the function TorsionSubcomplex(groupName, p) and vis
`ReduceTorsionSubcomplex( C, p)` This function start with the same operations as the function TorsionSubcomple

It prints on the screen which cells to merge and which edges to cut off in order to reduce the p-torsion subcomplex wi
`EquivariantEulerCharacteristic( X )` It inputs an n-dimensional Γ-equivariant CW-complex X all the cell stab
`CountingCellsOfACellComplex( X )` It inputs an n-dimensional Γ-equivariant CW-complex X on which all the ce
`CountingControlledSubdividedCells( X )` It inputs an n-dimensional Γ-equivariant CW-complex X on which a
`CountingBaryCentricSubdividedCells( X )` It inputs an n-dimensional Γ-equivariant CW-complex X on which
`EquivariantSpectralSequencePage( C, m, n)` It inputs a triple (C,m,n) where C is either a groupName explain
`ExportHapCellcomplexToDisk( C, groupName)` It inputs a cell complex C which is stored as a variable in the me

Chapter 28

Simplicial Complexes

`Homology(T,n)` `Homology(T)` Inputs a pure cubical complex, or cubical complex, or simplicial complex T and a nor
`RipsHomology(G,n)` `RipsHomology(G,n,p)` Inputs a graph G, a non-negative integer n (and optionally a prime nur
`Bettinumbers(T,n)` `Bettinumbers(T)` Inputs a pure cubical complex, or cubical complex, simplicial complex c
`ChainComplex(T)` Inputs a pure cubical complex, or cubical complex, or simplicial complex T and returns the (often
`CechComplexOfPureCubicalComplex(T)` Inputs a d-dimensional pure cubical complex T and returns a simplicial c
`PureComplexToSimplicialComplex(T,k)` Inputs either a d-dimensional pure cubical complex T or a d-dimensiona
`RipsChainComplex(G,n)` Inputs a graph G and a non-negative integer n. It returns $n+1$ terms of a chain complex w
`VectorsToSymmetricMatrix(M)` `VectorsToSymmetricMatrix(M,distance)` Inputs a matrix M of rational numb
`EulerCharacteristic(T)` Inputs a pure cubical complex, or cubical complex, or simplicial complex T and returns
`MaximalSimplicesToSimplicialComplex(L)` Inputs a list L whose entries are lists of vertices representing the ma
`SkeletonOfSimplicialComplex(S,k)` Inputs a simplicial complex S and a positive integer k less than or equal to th
`GraphOfSimplicialComplex(S)` Inputs a simplicial complex S and returns the graph of S.
`ContractibleSubcomplexOfSimplicialComplex(S)` Inputs a simplicial complex S and returns a (probably maxim
`PathComponentsOfSimplicialComplex(S,n)` Inputs a simplicial complex S and a nonnegative integer n. If $n=0$ t
`QuillenComplex(G)` Inputs a finite group G and returns, as a simplicial complex, the order complex of the poset of r
`SymmetricMatrixToIncidenceMatrix(S,t)` `SymmetricMatrixToIncidenceMatrix(S,t,d)` Inputs a symmetri
`IncidenceMatrixToGraph(M)` Inputs a symmetric 0/1 matrix M. It returns the graph with one vertex for each row o
`CayleyGraphOfGroup(G,A)` Inputs a group G and a set A of generators. It returns the Cayley graph.
`PathComponentsOfGraph(G,n)` Inputs a graph G and a nonnegative integer n. If $n=0$ the number of path compone
`ContractGraph(G)` Inputs a graph G and tries to remove vertices and edges to produce a smaller graph G' such that t
`GraphDisplay(G)` This function uses GraphViz software to display a graph G.
`SimplicialMap(K,L,f)` `SimplicialMapNC(K,L,f)` Inputs simplicial complexes K , L and a function $f\colon K!.\mathit{vertice}$
`ChainMapOfSimplicialMap(f)` Inputs a simplicial map $f\colon K \to L$ and returns the corresponding chain map $C_*(f)\colon C$
`SimplicialNerveOfGraph(G,d)` Inputs a graph G and returns a d-dimensional simplicial complex K whose 1-skele

Chapter 29

Cubical Complexes

ArrayToPureCubicalComplexA,n) Inputs an integer array A of dimension d and an integer n. It returns a d-dimens

PureCubicalComplexA,n) Inputs a binary array A of dimension d. It returns the corresponding d-dimensional pure

FramedPureCubicalComplex(M) Inputs a pure cubical complex M and returns the pure cubical complex with a bord

RandomCubeOfPureCubicalComplex(M) Inputs a pure cubical complex M and returns a pure cubical complex R wit

PureCubicalComplexIntersection(S,T) Inputs two pure cubical complexes with common dimension and array s

PureCubicalComplexUnion(S,T) Inputs two pure cubical complexes with common dimension and array size. It ret

PureCubicalComplexDifference(S,T) Inputs two pure cubical complexes with common dimension and array size

ReadImageAsPureCubicalComplex("file.png",n) Reads an image file ("file.png", "file.eps", "file.bmp" etc) an

ReadLinkImageAsPureCubicalComplex("file.png") ReadLinkImageAsPureCubicalComplex("file.png"

ReadImageSequenceAsPureCubicalComplex("directory",n) Reads the name of a directory containing a sequen

Size(T) This returns the number of non-zero entries in the binary array of the cubical complex, or pure cubical comp

Dimension(T) This returns the dimension of the cubical complex, or pure cubical complex T.

WritePureCubicalComplexAsImage(T,"filename","ext") Inputs a 2-dimensional pure cubical complex T, and

ViewPureCubicalComplex(T) ViewPureCubicalComplex(T,"mozilla") Inputs a 2-dimensional pure cubical co

Homology(T,n) Homology(T) Inputs a pure cubical complex, or cubical complex, or simplicial complex T and a nor

Bettinumbers(T,n) Bettinumbers(T) Inputs a pure cubical complex, or cubical complex, simplicial complex or c

DirectProductOfPureCubicalComplexes(M,N) Inputs two pure cubical complexes M,N and returns their direct p

SuspensionOfPureCubicalComplex(M) Inputs a pure cubical complex M and returns a pure cubical complex with

EulerCharacteristic(T) Inputs a pure cubical complex, or cubical complex, or simplicial complex T and returns

PathComponentOfPureCubicalComplex(T,n) Inputs a pure cubical complex T and an integer n in the rane 1, ..., B

ChainComplex(T) Inputs a pure cubical complex, or cubical complex, or simplicial complex T and returns the (often

ChainComplexOfPair(T,S) Inputs a pure cubical complex or cubical complex T and subcomplex S. It returns the q

ExcisedPureCubicalPair(T,S) Inputs a pure cubical complex T and subcomplex S. It returns the pair $[T \setminus intS, S$

ChainInclusionOfPureCubicalPair(S,T) Inputs a pure cubical complex T and subcomplex S. It returns the chai

ChainMapOfPureCubicalPairs(M,S,N,T) Inputs a pure cubical complex N and subcomplexes M, T and S in T. It

ContractPureCubicalComplex(T) Inputs a pure cubical complex T of dimension d and removes d-dimensional ce

ContractedComplex(T) Inputs a pure cubical complex T and returns a structural copy of the complex obtained from

ZigZagContractedPureCubicalComplex(T) Inputs a pure cubical complex T and returns a homotopy equivalent p

ContractCubicalComplex(T) Inputs a cubical complex T and removes cells without changing the homotopy type o

DVFReducedCubicalComplex(T) Inputs a cubical complex T and returns a non-regular cubical complex R by constr

SkeletonOfCubicalComplex(T,n) Inputs a cubical complex, or pure cubical complex T and positive integer n. It r

ContractibleSubomplexOfPureCubicalComplex(T) Inputs a pure cubical complex T and returns a maximal cont

AcyclicSubomplexOfPureCubicalComplex(T) Inputs a pure cubical complex T and returns a (not necessarily con

HomotopyEquivalentMaximalPureCubicalSubcomplex(T,S) Inputs a pure cubical complex T together with a pu

HomotopyEquivalentMinimalPureCubicalSubcomplex(T,S) Inputs a pure cubical complex T together with a pu

BoundaryOfPureCubicalComplex(T) Inputs a pure cubical complex T and returns its boundary as a pure cubical co

SingularitiesOfPureCubicalComplex(T,radius,tolerance) Inputs a pure cubical complex T together with a

ThickenedPureCubicalComplex(T) Inputs a pure cubical complex T and returns a pure cubical complex S. If a euc

CropPureCubicalComplex(T) Inputs a pure cubical complex T and returns a pure cubical complex S obtained from

BoundingPureCubicalComplex(T) Inputs a pure cubical complex T and returns a contractible pure cubical comple

MorseFiltration(M,i,t,bool) MorseFiltration(M,i,t) Inputs a pure cubical complex M of dimension d, an

ComplementOfPureCubicalComplex(T) Inputs a pure cubical complex T and returns a pure cubical complex S. A c

PureCubicalComplexToTextFile(file,M) Inputs a pure cubical complex M and a string containing the address of

ThickeningFiltration(M,n) ThickeningFiltration(M,n,k) Inputs a pure cubical complex M and a positive in

Dendrogram(M) Inputs a filtered pure cubical complex M and returns data that specifies the dendrogram (or phyloger

DendrogramDisplay(M) Inputs a filtered pure cubical complex M, or alternatively inputs the out from the command

DendrogramToPersistenceMat(D) Inputs the output of the function Dendrogram(M) and returns the corresponding

ReadImageAsFilteredPureCubicalComplex(file,n) Inputs a string containing the path to an image file, togethe

ComplementOfFilteredPureCubicalComplex(M) Inputs a filtered pure cubical complex M and returns the comple

PersistentHomologyOfFilteredPureCubicalComplex(M,n) Inputs a filtered pure cubical complex M and a non

Chapter 30

Regular CW-Complexes

`SimplicialComplexToRegularCWComplex(K)` Inputs a simplicial complex *K* and returns the corresponding regular
`CubicalComplexToRegularCWComplex(K)` `CubicalComplexToRegularCWComplex(K,n)` Inputs a pure cubical co
`CriticalCellsOfRegularCWComplex(Y)` `CriticalCellsOfRegularCWComplex(Y,n)` Inputs a regular CW-comp
`ChainComplex(Y)` Inputs a regular CW-complex *Y* and returns the cellular chain complex of a CW-complex W whos
`ChainComplexOfRegularCWComplex(Y)` Inputs a regular CW-complex *Y* and returns the cellular chain complex of
`FundamentalGroup(Y)` `FundamentalGroup(Y,n)` Inputs a regular CW-complex *Y* and, optionally, the number of s

Chapter 31

Knots and Links

`PureCubicalKnot(L)` `PureCubicalKnot(n,i)` Inputs a list $L = [[m1,n1],[m2,n2],...,[mk,nk]]$ of pairs of integers c
`ViewPureCubicalKnot(L)` Inputs a pure cubical link L and displays it.
`KnotSum(K,L)` Inputs two pure cubical knots K, L and returns their sum as a pure cubical knot. This function is not d
`KnotGroup(K)` Inputs a pure cubical link K and returns the fundamental group of its complement. The group is retur
`AlexanderMatrix(G)` Inputs a finitely presented group G whose abelianization is infinite cyclic. It returns the Alexa
`AlexanderPolynomial(K)` `AlexanderPolynomial(G)` Inputs either a pure cubical knot K or a finitely presented gr
`ProjectionOfPureCubicalComplex(K)` Inputs an \$n\$-dimensional pure cubical complex K and returns an n-1-dim
`ReadPDBfileAsPureCubicalComplex(file)` `ReadPDBfileAsPureCubicalComplex(file,m ,c)` Inputs a protei

Chapter 32

Knots and Quandles

Knots
`PresentationKnotQuandle(gaussCode)` Inputs a Gauss Code of a knot (with the orientations; see *GaussCodeOf*
`PD2GC(PD)` Inputs a Planar Diagram of a knot; outputs the Gauss Code associated (with the orientations).
`PlanarDiagramKnot(n,k)` Returns a Planar Diagram for the *k*-th knot with *n* crossings (*n*UNKNOWNEntity(le)12
`GaussCodeKnot(n,k)` Returns a Gauss Code (with orientations) for the *k*-th knot with *n* crossings (*n*UNKNOWNE
`PresentationKnotQuandleKnot(n,k)` Returns generators and relators (in the form of a record) for the *k*-th knot v
`NumberOfHomomorphisms(genRelQ,finiteQ)` Inputs generators and relators *genRelQ* of a knot quandle (in the fo
`PartitionedNumberOfHomomorphisms(genRelQ,finiteQ)` Inputs generators and relators *genRelQ* of a knot qua
Quandles
`ConjugationQuandle(G,n)` Inputs a finite group *G* and an integer *n*; outputs the associated *n*-fold conjugation qua
`FirstQuandleAxiomIsSatisfied(M)`
`SecondQuandleAxiomIsSatisfied(M)`
`ThirdQuandleAxiomIsSatisfied(M)` Inputs a finite magma *M*; returns true if *M* satisfy the first/second/third axio
`IsQuandle(M)` Inputs a finite magma *M*; returns true if *M* is a quandle, false otherwise.
`Quandles(n)` Returns a list of all quandles of size *n*, *n*UNKNOWNEntity(le)6. If *n*UNKNOWNEntity(ge)7, it retur
`Quandle(n,k)` Returns the *k*-th quandle of size *n* (*n*UNKNOWNEntity(le)6) if such a quandle exists, fail otherwis
`IdQuandle(Q)` Inputs a quandle *Q*; and outputs a list of integers [*n*,*k*] such that *Q* is isomorphic to `Quandle(n,k)`.
`IsLatin(Q)` Inputs a finite quandle *Q*; returns true if *Q* is latin, false otherwise.
`IsConnectedQuandle(Q)` Inputs a finite quandle *Q*; returns true if *Q* is connected, false otherwise.
`ConnectedQuandles(n)` Returns a list of all connected quandles of size *n*.
`ConnectedQuandle(n,k)` Returns the *k*-th quandle of size *n* if such a quandle exists, fail otherwise.
`IdConnectedQuandle(Q)` Inputs a connected quandle *Q*; and outputs a list of integers [*n*,*k*] such that *Q* is isomorph
`IsQuandleEnvelope(Q,G,e,stigma)` Inputs a set *Q*, a permutation group *G*, an element *e* UNKNOWNEntity(isin
`QuandleQuandleEnveloppe(Q,G,e,stigma)` Inputs a set *Q*, a permutation group *G*, an element *e* UNKNOWNE
`KnotInvariantCedric(genRelQ,n,m)` Inputs generators and relators of a knot quandle (in the form of a record,
`RightMultiplicationGroupAsPerm(Q)` Inputs a connected quandle *Q*; output its right multiplication group whose
`RightMultiplicationGroup(Q)` Inputs a connected quandle *Q*; output its right multiplication group whose elemen
`AutomorphismGroupQuandleAsPerm(Q)` Inputs a connected quandle *Q*; outputs its automorphism group whose ele
`AutomorphismGroupQuandle(Q)` Inputs a connected quandle *Q*; outputs its automorphism group whose elements ar

Chapter 33

Finite metric spaces and their filtered complexes

`CayleyMetric(g,h,N)` `CayleyMetric(g,h)` Inputs two permutations g,h and optionally the degree N of a symm
`HammingMetric(g,h,N)` `HammingMetric(g,h)` Inputs two permutations g,h and optionally the degree N of a sy
`KendallMetric(g,h,N)` `KendallMetric(g,h)` Inputs two permutations g,h and optionally the degree N of a sy
`EuclideanSquaredMetric(v,w)` Inputs two vectors v,w of equal length and returns the sum of the squares of the c
`EuclideanApproximatedMetric(v,w)` Inputs two vectors v,w of equal length and returns a rational approximatio
`ManhattanMetric(v,w)` Inputs two vectors v,w of equal length and returns the sum of the absolute values of the co
`VectorsToSymmetricMatrix(L)` `VectorsToSymmetricMatrix(L,D)` Inputs a list L of vectors and optionally a
`SymmetricMatDisplay(S)` `SymmetricMatDisplay(L,V)` Inputs an $n \times n$ symmetric matrix S of non-negative in
`SymmetricMatrixToFilteredGraph(S,t,m)` Inputs an integer symmetric matrix S, a positive integer t and a posi
`PermGroupToFilteredGraph(S,D)` Inputs a permutation group G and a metric D defined on permutations. The fu

Chapter 34

Commutative diagrams and abstract categories

COMMUTATIVE DIAGRAMS

`HomomorphismChainToCommutativeDiagram(H)` Inputs a list $H = [h_1, h_2, ..., h_n]$ of mappings such that the compo
`NormalSeriesToQuotientDiagram(L)` `NormalSeriesToQuotientDiagram(L,M)` Inputs an increasing (or decre
`NerveOfCommutativeDiagram(D)` Inputs a commutative diagram D and returns the commutative diagram ND con
`GroupHomologyOfCommutativeDiagram(D,n)` `GroupHomologyOfCommutativeDiagram(D,n,prime)` `GroupH`
`PersistentHomologyOfCommutativeDiagramOfPGroups(D,n)` Inputs a commutative diagram D of finite p-grou

ABSTRACT CATEGORIES

`CategoricalEnrichment(X,Name)` Inputs a structure X such as a group or group homomorphism, together with th
`IdentityArrow(X)` Inputs an object X in some category, and returns the identity arrow on the object X.
`InitialArrow(X)` Inputs an object X in some category, and returns the arrow from the initial object in the category
`TerminalArrow(X)` Inputs an object X in some category, and returns the arrow from X to the terminal object in the
`HasInitialObject(Name)` Inputs the name of a category and returns true or false depending on whether the catego
`HasTerminalObject(Name)` Inputs the name of a category and returns true or false depending on whether the categ
`Source(f)` Inputs an arrow f in some category, and returns its source.
`Target(f)` Inputs an arrow f in some category, and returns its target.
`CategoryName(X)` Inputs an object or arrow X in some category, and returns the name of the category.
`"*", "=", "+", "-"` Composition of suitable arrows f, g is given by $f * g$ when the source of f equals the target o
`Object(X)` Inputs an object X in some category, and returns the GAP structure Y such that $X = CategoricalEnrichm$
`Mapping(X)` Inputs an arrow f in some category, and returns the GAP structure Y such that $f = CategoricalEnrichm$
`IsCategoryObject(X)` Inputs X and returns true if X is an object in some category.
`IsCategoryArrow(X)` Inputs X and returns true if X is an arrow in some category.

Chapter 35

Arrays and Pseudo lists

`Array(A,f)` Inputs an array A and a function f. It returns the the array obtained by applying f to each entry of A (an
`PermuteArray(A,f)` Inputs an array A of dimension d and a permutation f of degree at most d. It returns the array B
`ArrayDimension(A)` Inputs an array A and returns its dimension.
`ArrayDimensions(A)` Inputs an array A and returns its dimensions.
`ArraySum(A)` Inputs an array A and returns the sum of its entries.
`ArrayValue(A,x)` Inputs an array A and a coordinate vector x. It returns the value of the entry in A with coordinate x
`ArrayValueFunctions(d)` Inputs a positive integer d and returns an efficient version of the function ArrayValue for
`ArrayAssign(A,x,n)` Inputs an array A and a coordinate vector x and an integer n. It sets the entry of A with coordi
`ArrayAssignFunctions(d)` Inputs a positive integer d and returns an efficient version of the function ArrayAssign f
`ArrayIterate(d)` Inputs a positive integer d and returns a function ArrayIt(Dimensions,f). This function inputs a lis
`BinaryArrayToTextFile(file,A)` Inputs a string containing the address of a file, and an array A of 0s and 1s. The
`FrameArray(A)` Inputs an array A and returns the array obtained by appending a 0 to the beginning and end of each "
`UnframeArray(A)` Inputs an array A and returns the array obtained by removing the first and last entry in each "row"
`Add(L,x)` Let L be a pseudo list of length n, and x an object compatible with the entries in L. If x is not in L then this
`Append(L,K)` Let L be a pseudo list and K a list whose objects are compatible with those in L. This operation applies
`ListToPseudoList(L)` Inputs a list L and returns the pseudo list representation of L.

Chapter 36

Parallel Computation - Core Functions

`ChildProcess()` `ChildProcess("computer.ac.wales")` `ChildProcess(["-m", "100000M", "-T"])` `ChildP`

- open a shell on thishost
- cd .ssh
- ls
-> if id_dsa, id_rsa etc exists, skip the next two steps!
- ssh-keygen -t rsa
- ssh-keygen -t dsa
- scp *.pub user@remotehost:~/
- ssh remotehost -l user
- cat id_rsa.pub >> .ssh/authorized_keys
- cat id_dsa.pub >> .ssh/authorized_keys
- rm id_rsa.pub id_dsa.pub
- exit

You should now be able to connect from "thishost" to "remotehost" without a password prompt.)
`ChildClose(s)` This closes the stream s to a child GAP process.
`ChildCommand("cmd;",s)` This runs a GAP command "cmd;" on the child process accessed by the stream s. Here "
`NextAvailableChild(L)` Inputs a list *L* of child processes and returns a child in *L* which is ready for computation (a
`IsAvailableChild(s)` Inputs a child process *s* and returns true if s is currently available for computations, and false
`ChildPut(A,"B",s)` This copies a GAP object A on the parent process to an object B on the child process s. (The co
`ChildGet("A",s)` This functions copies a GAP object A on the child process s and returns it on the parent process. (
`HAPPrintTo("file",R)` Inputs a name "file" of a new text file and a HAP object R. It writes the object R to "file". C
`HAPRead("file",R)` Inputs a name "file" containing a HAP object R and returns the object. Currently this is only im

Chapter 37

Parallel Computation - Extra Functions

`ChildFunction("function(arg);",s)` This runs the GAP function "function(arg);" on a child process accessed b
`ChildRead(s)` This returns, as a string, the output of the last application of $ChildFunction("function(arg);",s)$.
`ChildReadEval(s)` This returns, as an evaluated string, the output of the last application of $ChildFunction("functio$
`ParallelList(I,fn,L)` Inputs a list I, a function fn such that $fn(x)$ is defined for all x in I, and a list of children L.

Chapter 38

Some functions for accessing basic data

`BoundaryMap(C)` Inputs a resolution, chain complex or cochain complex C and returns the function $C!.boundary$.
`BoundaryMatrix(C,n)` Inputs a chain or cochain complex C and integer n>0. It returns the n-th boundary map of C
`Dimension(C)`
`Dimension(M)` Inputs a resolution, chain complex or cochain complex C and returns the function $C!.dimension$. Al
`EvaluateProperty(X,"name")` Inputs a component object X (such as a ZG-resolution or chain map) and a string "
`GroupOfResolution(R)` Inputs a ZG-resolution R and returns the group G.
`Length(R)` Inputs a resolution R and returns its length (i.e. the number of terms of R that HAP has computed).
`Map(f)` Inputs a chain map, or cochain map or equivariant chain map f and returns the mapping function (as opposed
`Source(f)` Inputs a chain map, or cochain map, or equivariant chain map, or FpG-module homomorphism f and retu
`Target(f)` Inputs a chain map, or cochain map, or equivariant chain map, or FpG-module homomorphism f and retu

Chapter 39

Miscellaneous

`SL2Z(p)` `SL2Z(1/m)` Inputs a prime p or the reciprocal $1/m$ of a square free integer m. In the first case the functio
`BigStepLCS(G,n)` Inputs a group G and a positive integer n. It returns a subseries $G = L_1 > L_2 > \ldots L_k = 1$ of the low
`Classify(L,Inv)` Inputs a list of objects L and a function Inv which computes an invariant of each object. It return
`RefineClassification(C,Inv)` Inputs a list $C := Classify(L, OldInv)$ and returns a refined classification accordi
`Compose(f,g)` Inputs two FpG-module homomorphisms $f : M \longrightarrow N$ and $g : L \longrightarrow M$ with $Source(f) = Target(g$
`HAPcopyright()` This function provides details of HAP'S GNU public copyright licence.
`IsLieAlgebraHomomorphism(f)` Inputs an object f and returns true if f is a homomorphism $f : A \longrightarrow B$ of Lie al
`IsSuperperfect(G)` Inputs a group G and returns "true" if both the first and second integral homology of G is trivi
`MakeHAPManual()` This function creates the manual for HAP from an XML file.
`PermToMatrixGroup(G,n)` Inputs a permutation group G and its degree n. Returns a bijective homomorphism $f :$
`SolutionsMatDestructive(M,B)` Inputs an $m \times n$ matrix M and a $k \times n$ matrix B over a field. It returns a k×m mat
`LinearHomomorphismsPersistenceMat(L)` Inputs a composable sequence L of vector space homomorphisms. It
`NormalSeriesToQuotientHomomorphisms(L)` Inputs an (increasing or decreasing) chain L of normal subgroups i
`TestHap()` This runs a representative sample of HAP functions and checks to see that they produce the correct outp

Index