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`

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

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

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)::`

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, PureCubicalComplex -> ChainMap 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 $f_*: C_*(X/A) \to C_*(Y/B)$ of cellular chain complexes of pairs. (Typlically 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 $f: X \to Y$ between two regular CW-complexes X, Y and returns an induced chain map $f_*: C_*(X) \to C_*(Y)$ where $C_*(X)$, $C_*(Y)$ are chain homotopic to (but usually smaller than) the cellular chain complexes of X, Y.
Inputs a map $f: X \to Y$ between two simplicial complexes X, Y and returns the induced chain map $f_*: C_*(X) \to C_*(Y)$ 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 $p: X^\Delta \to X$ 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 $p_*: C_*(X^\Delta) \to C_*(X)$ 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 $\iota_*: C_*(X^\Delta) \to C_*(X \times X)$ can be used to compute the cup product.

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

(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 $F{:}R \to S$ of resolutions and returns the induced cochain map $Hom_{\mathbb{Z}G}(S,\mathbb{Z}) \longrightarrow Hom_{\mathbb{Z}G}(R,\mathbb{Z})$.

`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 $F{:}R \to S$ 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 -> EquiChainMap`
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.

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 $F:R \to S$ 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 $A: G \to \mathrm{GL}_n(\mathbb{Z})$. 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 $F:R \to S$ in characteristic 0 and returns the induced chain map $F \otimes_{\mathbb{Z}G} \mathbb{Z}: R \otimes_{\mathbb{Z}G} \mathbb{Z} \longrightarrow S \otimes_{\mathbb{Z}G} \mathbb{Z}$.

`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 $F:R \to S$ in characteristic 0 a prime integer p. It returns the induced chain map $F \otimes_{\mathbb{Z}G} \mathbb{Z}_p: R \otimes_{\mathbb{Z}G} \mathbb{Z}_p \longrightarrow S \otimes_{\mathbb{Z}G} \mathbb{Z}_p$.

Cohomology rings

`AreIsomorphicGradedAlgebras(A,B):: PresentedGradedAlgebra, PresentedGradedAlgebra -> Boolean`
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 $d{:}E \to E$ 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 $d{:}E \to E$ 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 $\alpha{:}S/J \to \ker d/\mathrm{im}\, d$. This isomorphism lifts to the ring homomorphism $h{:}S \to \ker d$. 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

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 $1 \leq k \leq n$. (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.

`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 $H_n(G,\mathbb{Z})$.

$\mathbb{F}_p$-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 $\cdots Rad_2(\mathbb{F}_pG)R \leq Rad_1(\mathbb{F}_pG)R \leq R$.

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 $f: G \times G \to A$ 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 $\alpha: G \to \mathrm{Out}(N)$ and returns the group $N.

`ActingGroup(M):: GOuterGroup -> Group`
Inputs a G-outer group M corresponding to a homomorphism $\alpha: G \to \mathrm{Out}(N)$ 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 $f: G \times G \to N$ 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 process`
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 process`
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 by applying "Tietze like operations" in each dimension. The hope is that S has fewer free generators than R.

`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" , "PSL(4,Z)_d" , "Sp(4,Z)"

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 integers of Q(sqrt(-d)) and subscripts _a, _b , _c , _d denote alternative non-free ZG-resolutions for a given group G.

Data for the first list of resolutions was provided provided by MATHIEU DUTOUR. Data for GL(2,O(-d)) was provided by SEBASTIAN SCHOENENNBECK. Data for SL(2,O(-d)) was provided bySEBASTIAN SCHOENNENBECK for d <= 26 and by ALEXANDER RAHM for d>26 and for the alternative complexes.

`FreeGResolution(P,n)` `FreeGResolution(P,n,p)`
Inputs a non-free ZG-resolution P with finite stabilizer groups, and a positive integer n. It returns a free ZG-resolution of length equal to the minimum of n and the length of P. If one requires only a mod p resolution then the prime p can be entered as an optional third argument.
The free resolution is returned without a contracting homotopy.

`ResolutionGTree(P,n)`
Inputs a non-free ZG-resolution P of dimension 1 (i.e. a G-tree) with finite stabilizer groups, and a positive integer n. It returns a free ZG-resolution of length equal to n.
If P has a contracting homotopy then the free resolution is returned with a contracting homotopy.
This function was written by BUI ANH TUAN.

`ResolutionSL2Z(p,n)`
Inputs positive integers m,n and returns n terms of a ZG-resolution for the group $G = SL(2,Z[1/m])$.
This function is joint work with BUI ANH TUAN.

`ResolutionAbelianGroup(L,n)` `ResolutionAbelianGroup(G,n)`
Inputs a list $L := [m_1, m_2, ..., m_d]$ of nonnegative integers, and a positive integer n. It returns n terms of a $\mathbb{Z}G$-resolution for the abelian group $G = Z_{L[1]} + Z_{L[2]} + \cdots + Z_L[d]$.

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 FG-resolution of the module M (where G is a finite group and F the field of p elements).

Chapter 7

Induced equivariant chain maps

`EquivariantChainMap(R,S,f)`
Inputs a ZG-resolution R, a ZG'-resolution S, and a group homomorphism $f : G \longrightarrow G'$. It outputs a component object M with the following components.

- M!.*source* is the resolution R.
- M!.*target* is the resolution S.
- M!.*mapping*(w,n) is a function which gives the image in S_n, under a chain map induced by f, of a word w in R_n. (Here R_n and S_n are the n-th modules in the resolutions R and S.)
- F!.*properties* is a list of pairs such as ["type", "equivariantChainMap"].

The resolution S must have a contracting homotopy.

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 elements of G. It returns the free ZG-resolution $(R \otimes_{ZH} ZG)$. The returned resolution S has S!.elts:=EltsG. This is a resolution of the ZG-module $(Z \otimes_{ZH} ZG)$. (Here $\otimes_{ZH}$ means tensor over ZH.)

`HomToIntegers(X)`
Inputs either a ZG-resolution $X = R$, or an equivariant chain map $X = (F : R \longrightarrow S)$. It returns the cochain complex or cochain map obtained by applying $HomZG(\_, Z)$ where Z is the trivial module of integers (characteristic 0).

`HomToIntegersModP(R)`
Inputs a ZG-resolution R and returns the cochain complex obtained by applying $HomZG(\_, Z_p)$ where Z_p is the trivial module of integers mod p. (At present this functor does not handle equivariant chain maps.)

`HomToIntegralModule(R,f)`
Inputs a ZG-resolution R and a group homomorphism $f : G \longrightarrow GL_n(Z)$ to the group of $n \times n$ invertible integer matrices. Here Z must have characteristic 0. It returns the cochain complex obtained by applying $HomZG(\_, A)$ where A is the ZG-module Z^n with G action via f. (At present this function does not handle equivariant chain maps.)

`TensorWithIntegralModule(R,f)`
Inputs a ZG-resolution R and a group homomorphism $f : G \longrightarrow GL_n(Z)$ to the group of $n \times n$ invertible integer matrices. Here Z must have characteristic 0. It returns the chain complex obtained by tensoring over ZG with the ZG-module $A = Z^n$ with G action via f. (At present this function does not handle equivariant chain maps.)

`HomToGModule(R,A)`
Inputs a ZG-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.)

`InduceScalars(R,hom)`
Inputs a ZQ-resolution R and a surjective group homomorphism $hom : G \to Q$. It returns the unduced non-free ZG-resolution.

`LowerCentralSeriesLieAlgebra(G) LowerCentralSeriesLieAlgebra(f)`
Inputs a pcp group G. If each quotient G_c/G_{c+1} of the lower central series is free abelian or p-elementary abelian (for fixed prime p) then a Lie algebra $L(G)$ is returned. The abelian group underlying $L(G)$ is the direct sum of the quotients G_c/G_{c+1} . The Lie bracket on $L(G)$ is induced by the commutator in G. (Here $G_1 = G$, $G_{c+1} = [G_c, G]$.)
The function can also be applied to a group homomorphism $f : G \longrightarrow G'$. In this case the induced homomorphism of Lie algebras $L(f) : L(G) \longrightarrow L(G')$ is returned.
If the quotients of the lower central series are not all free or p-elementary abelian then the function returns fail.
This function was written by Pablo Fernandez Ascariz

`TensorWithIntegers(X)`
Inputs either a ZG-resolution $X = R$, or an equivariant chain map $X = (F : R \longrightarrow S)$. It returns the chain complex or chain map obtained by tensoring with the trivial module of integers (characteristic 0).

`FilteredTensorWithIntegers(R)`
Inputs a ZG-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 of integers (characteristic 0).

`TensorWithTwistedIntegers(X,rho)`

Chapter 9

Chain complexes

`ChainComplex(T)`
Inputs a pure cubical complex, or cubical complex, or simplicial complex T and returns the (often very large) cellular chain complex of T.

`ChainComplexOfPair(T,S)`
Inputs a pure cubical complex or cubical complex T and contractible subcomplex S. It returns the quotient $C(T)/C(S)$ of cellular chain complexes.

`ChevalleyEilenbergComplex(X,n)`
Inputs either a Lie algebra $X = A$ (over the ring of integers Z or over a field K) or a homomorphism of Lie algebras $X = (f : A \longrightarrow B)$, together with a positive integer n. It returns either the first n terms of the Chevalley-Eilenberg chain complex $C(A)$, or the induced map of Chevalley-Eilenberg complexes $C(f) : C(A) \longrightarrow C(B)$.
(The homology of the Chevalley-Eilenberg complex $C(A)$ is by definition the homology of the Lie algebra A with trivial coefficients in Z or K).
This function was written by PABLO FERNANDEZ ASCARIZ

`LeibnizComplex(X,n)`
Inputs either a Lie or Leibniz algebra $X = A$ (over the ring of integers Z or over a field K) or a homomorphism of Lie or Leibniz algebras $X = (f : A \longrightarrow B)$, together with a positive integer n. It returns either the first n terms of the Leibniz chain complex $C(A)$, or the induced map of Leibniz complexes $C(f) : C(A) \longrightarrow C(B)$.
(The Leibniz complex $C(A)$ was defined by J.-L.Loday. Its homology is by definition the Leibniz homology of the algebra A).
This function was written by PABLO FERNANDEZ ASCARIZ

`SuspendedChainComplex(C)`
Inputs a chain complex C and returns the chain complex S defined by applying the degree shift $S_n = C_{n-1}$ to chain groups and boundary homomorphisms.

`ReducedSuspendedChainComplex(C)`
Inputs a chain complex C and returns the chain complex S defined by applying the degree shift $S_n = C_{n-1}$ to chain groups and boundary homomorphisms for all $n > 0$. The chain complex S has trivial homology in degree 0 and $S_0 = \mathbb{Z}$.

`CoreducedChainComplex(C)` `CoreducedChainComplex(C,2)`
Inputs a chain complex C and returns a quasi-isomorphic chain complex D. In many cases the complex D should be smaller than C. If an optional second input argument is set equal to 2 then an alternative method is used for reducing the size of the chain complex.

`TensorProductOfChainComplexes(C,D)`
Inputs two chain complexes C and D of the same characteristic and returns their tensor product as a chain complex.
This function was written by LE VAN LUYEN.

`LefschetzNumber(F)`
Inputs a chain map $F: C \to C$ with common source and target. It returns the Lefschetz number of the map (that is, the alternating sum of the traces of the homology maps in each degree).

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 modifies A and returns no value.

`SparseRowMult(A,i,k)`
Multiplies the i-th row of a sparse matrix A by k. The sparse matrix A is modified but nothing is returned.

`SparseRowInterchange(A,i,k)`
Interchanges the i-th and j-th rows of a sparse matrix A by k. The sparse matrix A is modified but nothing is returned.

`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 modified but nothing is returned.

`SparseSemiEchelon(A)`
Converts a sparse matrix A to semi-echelon form (which means echelon form up to a permutation of rows). The sparse matrix A is modified but nothing is returned.

`RankMatDestructive(A)`
Returns the rank of a sparse matrix A. The sparse matrix A is modified during the calculation.

`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 homotopy equivalent to the cellular chain complex of Y. The function uses discrete vector fields to calculate a smallish chain complex.

`SparseChainComplexOfRegularCWComplex(Y)`
Inputs a regular CW-complex Y and returns its cellular chain complex as a sparse chain complex. The function SparseChainComplex(Y) will usually return a smaller chain complex.

`SparseBoundaryMatrix(C,n)`
Inputs a sparse chain complex C and integer n. Returns the n-th boundary matrix of the chain complex in sparse format.

`Bettinumbers(C,n)`
Inputs a sparse chain complex C and integer n. Returns the n-th Netti number of the chain complex.

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 characteristic p together with a non-negative intereg n.

- If $X = C$ and $p = 0$ then the torsion coefficients of $H^n(C)$ are retuned. If $X = C$ and p is prime then the dimension of $H^n(C)$ are retuned.
- If $X = (C \longrightarrow D)$ then the induced homomorphism $H^n(C) \longrightarrow H^n(D)$ is returned as a homomorphism of finitely presented groups.

A G-cocomplex C can also be input. The cohomology groups of such a complex may not be abelian. WARNING: in this case Cohomology(C,n) returns the abelian invariants of the n-th cohomology group of C.

`CohomologyModule(C,n)`
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 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 works only for n=1,2,3.)

`CohomologyPrimePart(C,n,p)`
Inputs a cochain complex C in characteristic 0, a positive integer n, and a prime p. It returns a list of those torsion coefficients of $H^n(C)$ that are positive powers of p. The function uses the EDIM package by Frank Luebeck.

`GroupCohomology(X,n)  GroupCohomology(X,n,p)`
Inputs a positive integer n and either

- a finite group $X = G$
- or a nilpotent Pcp-group $X = G$
- or a space group $X = G$
- or a list $X = D$ representing a graph of groups
- or a pair $X = [\text{"}Artin\text{"}, D]$ where D is a Coxeter diagram representing an infinite Artin group G.
- or a pair $X = [\text{"}Coxeter\text{"}, D]$ where D is a Coxeter diagram representing a finite Coxeter group G.

It returns the torsion coefficients of the integral cohomology $H^n(G,Z)$.
There is an optional third argument which, when set equal to a prime p, causes the function to return the the mod p cohomology $H^n(G,Z_p)$ as a list of length equal to its rank.
This function is a composite of more basic functions, and makes choices for a number of parameters. For a particular group you would almost certainly be better using the more basic functions and making the choices yourself!

`GroupHomology(X,n) GroupHomology(X,n,p)`
Inputs a positive integer n and either

- a finite group $X = G$
- or a nilpotent Pcp-group $X = G$
- or a space group $X = G$
- or a list $X = D$ representing a graph of groups
- or a pair $X = [\text{"}Artin\text{"}, D]$ where D is a Coxeter diagram representing an infinite Artin group G.

Chapter 12

Poincare series

`EfficientNormalSubgroups(G)` `EfficientNormalSubgroups(G,k)`
Inputs a prime-power group G and, optionally, a positive integer k. The default is $k = 4$. The function returns a list of normal subgroups N in G such that the Poincare series for G equals the Poincare series for the direct product $(N \times (G/N))$ up to degree k.

`ExpansionOfRationalFunction(f,n)`
Inputs a positive integer n and a rational function $f(x) = p(x)/q(x)$ where the degree of the polynomial $p(x)$ is less than that of $q(x)$. It returns a list $[a_0, a_1, a_2, a_3, \ldots, a_n]$ of the first $n+1$ coefficients of the infinite expansion
$f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \ldots$.

`PoincareSeries(G,n)` `PoincareSeries(R,n)` `PoincareSeries(L,n)` `PoincareSeries(G)`
Inputs a finite p-group G and a positive integer n. It returns a quotient of polynomials $f(x) = P(x)/Q(x)$ whose coefficient of x^k equals the rank of the vector space $H_k(G, Z_p)$ for all k in the range $k = 1$ to $k = n$. (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.

`PoincareSeriesPrimePart(G,p,n)`
Inputs a finite group G, a prime p, and a positive integer n. It returns a quotient of polynomials $f(x) = P(x)/Q(x)$ whose coefficient of x^k equals the rank of the vector space $H_k(G, Z_p)$ for all k in the range $k = 1$ to $k = n$.
The efficiency of this function needs to be improved.
`PoincareSeriesLHS(G)`
Inputs a finite 2-group G and returns a quotient of polynomials $f(x) = P(x)/Q(x)$ whose coefficient of x^k equals the rank of the vector space $H_k(G, Z_2)$ for all k.
This function was written by PAUL SMITH. It use the Singular system for commutative algebra.

`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 construct the cup product are also available.)
Inputs a ZG-resolution R, a vector u representing an element in $H^p(G,Z)$, a vector v representing an element in $H^q(G,Z)$ and the two integers $p,q>0$. It returns a vector w representing the cup product $u \cdot v$ in $H^{p+q}(G,Z)$. This product is associative and $u \cdot v = (-1)pqv \cdot u$. It provides $H^*(G,Z)$ with the structure of an anti-commutative graded ring. This function implements the cup product for characteristic 0 only.
The resolution R needs a contracting homotopy.
To save the function from having to calculate the abelian groups $H^n(G,Z)$ additional input variables can be used in the form $IntegralCupProduct(R,u,v,p,q,P,Q,N)$, where

- P is the output of the command $CR_{C}ocyclesAndCoboundaries(R,p,true)$
- Q is the output of the command $CR_{C}ocyclesAndCoboundaries(R,q,true)$
- N is the output of the command $CR_{C}ocyclesAndCoboundaries(R,p+q,true)$.

`IntegralRingGenerators(R,n)`
Inputs at least $n+1$ terms of a ZG-resolution and integer $n>0$. It returns a minimal list of cohomology classes in $H^n(G,Z)$ which, together with all cup products of lower degree classes, generate the group $H^n(G,Z)$.
(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 $u=(n_1,...,n_d)$.)

`ModPCohomologyGenerators(G,n)` `ModPCohomologyGenerators(R)`
Inputs either a p-group G and positive integer n, or else n terms of a minimal Z_pG-resolution R of Z_p. It returns a pair whose first entry is a minimal set of homogeneous generators for the cohomology ring $A=H^*(G,Z_p)$ 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(G,n)` `ModPCohomologyRing(G,n,level)` `ModPCohomologyRing(R)` `ModPCohomologyRing(R,level)`
Inputs either a p-group G and positive integer n, or else n terms of a minimal Z_pG-resolution R of Z_p. It returns the cohomology ring $A=H^*(G,Z_p)$ 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 $A:=ModPCohomologyRing_part_2(A)$.

`ModPRingGenerators(A)`
Inputs a mod p cohomology ring A (created using the preceeding function). It returns a minimal generating set for the ring A. Each generator is homogeneous.
`Mod2CohomologyRingPresentation(G)` `Mod2CohomologyRingPresentation(G,n)` `Mod2CohomologyRingPresentation(A)` `Mod2CohomologyRingPresentation(R)`
When applied to a finite 2-group G this function returns a presentation for the mod 2 cohomology ring $H^*(G,Z_2)$. The Lyndon-Hochschild-Serre spectral sequence is used to prove that the presentation is

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)`
`Mod2CohomologyRingPresentation(A)` `Mod2CohomologyRingPresentation(R)`
When applied to a finite 2-group G this function returns a presentation for the mod 2 cohomology ring $H^*(G,Z_2)$. The Lyndon-Hochschild-Serre spectral sequence is used to prove that the presentation is correct.
When the function is applied to a 2-group G and positive integer n the function first constructs n terms of a free Z_2G-resolution R, then constructs the finite-dimensional graded algebra $A = H^{(}* \leq n)(G,Z_2)$, and finally uses A to approximate a presentation for $H^*(G,Z_2)$. For "sufficiently large" the approximation will be a correct presentation for $H^*(G,Z_2)$.
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.

`PoincareSeriesLHS(G)`
Inputs a finite 2-group G and returns a quotient of polynomials $f(x) = P(x)/Q(x)$ whose coefficient of x^k equals the rank of the vector space $H_k(G,Z_2)$ for all k.
This function was written by PAUL SMITH. It use the Singular system for commutative algebra.

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 follows. For an arbitrary group G let $L^*_{c+1}(G)$ be the $(c+1)$-st term of the upper central series of the group $U = F/[[[R,F],F]...]$ (with c copies of F in the denominator) where F/R is any free presentation of G. This is an invariant of G and we define $M^{(c)}(G)$ to be the kernel of the canonical homomorphism $M^{(c)}(G) \longrightarrow G$. For $c = 1$ the Baer invariant $M^(1)(G)$ is isomorphic to the second integral homology $H_2(G,Z)$.
This function requires the NQ package.

`BogomolovMultiplier(G)`
`BogomolovMultiplier(G, "standard")` `BogomolovMultiplier(G, "homology")` `BogomolovMultiplier(G, "tensor")`
Inputs a finite group G and returns the quotient $H_2(G,Z)/K(G)$ of the second integral homology of G where $K(G)$ is the subgroup of $H_2(G,Z)$ generated by the images of all homomorphisms $H_2(A,Z) \to H_2(G,Z)$ induced from abelian subgroups of G.
Three slight variants of the implementation are available. The defaults "standard" implementation seems to work best on average. But for some groups the "homology" implementation or the "tensor" implementation will be faster. The variants are called by including the appropriate string as the second argument.

`Bogomology(G,n)`
Inputs a finite group G and positive integer n, and returns the quotient $H_n(G,Z)/K(G)$ of the degree n integral homology of G where $K(G)$ is the subgroup of $H_n(G,Z)$ generated by the images of all homomorphisms $H_n(A,Z) \to H_n(G,Z)$ induced from abelian subgroups of G.

`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 $Z^*(G,N)$ of the centre of N. The group $Z^*(G,N)$ is trivial if and only if there is a crossed module $d : E \longrightarrow G$ with $N = Image(d)$ and with $Ker(d)$ equal to the subgroup of E consisting of those elements on which G acts trivially.
If no value for N is entered then it is assumed that $N = G$. In this case the group $Z^*(G,G)$ is trivial if and only if G is isomorphic to a quotient $G = E/Z(E)$ of some group E by the centre of E. (See also the command $UpperEpicentralSeries(G,c)$.)

`NonabelianExteriorProduct(G,N)`
Inputs a finite group G and normal subgroup N. It returns a record E with the following components.

- $E.homomorphism$ is a group homomorphism $\mu : (G \wedge N) \longrightarrow G$ from the nonabelian exterior product $(G \wedge N)$ to G. The kernel of μ is the relative Schur multiplier.
- $E.pairing(x,y)$ is a function which inputs an element x in G and an element y in N and returns $(x \wedge y)$ in the exterior product $(G \wedge N)$.

This function should work for reasonably small nilpotent groups or extremely small non-nilpotent groups.

`NonabelianSymmetricKernel(G)` `NonabelianSymmetricKernel(G,m)`
Inputs a finite or nilpotent infinite group G and returns the abelian invariants of the Fourth homotopy group SG of the double suspension $SSK(G,1)$ of the Eilenberg-Mac Lane space $K(G,1)$.
For non-nilpotent groups the implementation of the function $NonabelianSymmetricKernel(G)$ is far from optimal and will soon be improved. As a temporary solution to this problem, an optional second variable m can be set equal to 0, and then the function efficiently returns the abelian invariants of groups A and B such that there is an exact sequence $0 \longrightarrow B \longrightarrow SG \longrightarrow A \longrightarrow 0$.
Alternatively, the optional second varible m can be set equal to a positive multiple of the order of the symmetric square $(G \tilde{\otimes} G)$. In this case the function returns the abelian invariants of SG. This might help when G is solvable but not nilpotent (especially if the estimated upper bound m is reasonable

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 homomorphism $phi : C \to L$ where:

- the kernel of *phi* lies in both the centre of C and the derived subalgebra of C,
- the kernel of *phi* is a vector space of rank equal to the rank of the second Chevalley-Eilenberg homology of L.

`LeibnizQuasiCoveringHomomorphism(L)`
Inputs a finite dimensional Lie algebra L over a field, and returns a surjective homomorphism *phi* : $C \to L$ of Leibniz algebras where:

- the kernel of *phi* lies in both the centre of C and the derived subalgebra of C,
- the kernel of *phi* is a vector space of rank equal to the rank of the kernel J of the homomorphism $L \otimes L \to L$ from the tensor square to L. (We note that, in general, J is NOT equal to the second Leibniz homology of L.)

`LieEpiCentre(L)`
Inputs a finite dimensional Lie algebra L over a field, and returns an ideal $Z^*(L)$ of the centre of L. The ideal $Z^*(L)$ is trivial if and only if L is isomorphic to a quotient $L = E/Z(E)$ of some Lie algebra E by the centre of E.

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

- $E.homomorphism$ is a Lie homomorphism $\mu : (L \wedge L) \longrightarrow L$ from the nonabelian exterior square $(L \wedge L)$ to L. The kernel of μ is the Lie multiplier.
- $E.pairing(x,y)$ is a function which inputs elements x, y in L and returns $(x \wedge y)$ in the exterior square $(L \wedge L)$.

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

- $T.homomorphism$ is a Lie homomorphism $\mu : (L \otimes L) \longrightarrow L$ from the nonabelian tensor square of L to L.
- $T.pairing(x,y)$ is a function which inputs two elements x, y in L and returns the tensor $(x \otimes y)$ in the tensor square $(L \otimes L)$.

`LieTensorCentre(L)`
Inputs a finite dimensional Lie algebra L over a field and returns the largest ideal N such that the induced homomorphism of nonabelian tensor squares $(L \otimes L) \longrightarrow (L/N \otimes L/N)$ is an isomorphism.

Chapter 17

Generators and relators of groups

`CayleyGraphOfGroupDisplay(G,X)` `CayleyGraphOfGroupDisplay(G,X,"mozilla")`
Inputs a finite group G together with a subset X of G. It displays the corresponding Cayley graph as a .gif file. It uses the Mozilla web browser as a default to view the diagram. An alternative browser can be set using a second argument.
The argument G can also be a finite set of elements in a (possibly infinite) group containing X. The edges of the graph are coloured according to which element of X they are labelled by. The list X corresponds to the list of colours [blue, red, green, yellow, brown, black] in that order.
This function requires Graphviz software.

`IdentityAmongRelatorsDisplay(R,n)` `IdentityAmongRelatorsDisplay(R,n,"mozilla")`
Inputs a free ZG-resolution R and an integer n. It displays the boundary R!.boundary(3,n) as a tessellation of a sphere. It displays the tessellation as a .gif file and uses the Mozilla web browser as a default display mechanism. An alternative browser can be set using a second argument. (The resolution R should be reduced and, preferably, in dimension 1 it should correspond to a Cayley graph for G.)
This function uses GraphViz software.

`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-space K associated to this presentation for the group $G = F/\langle R \rangle^F$.
The function returns "true" if K has trivial second homotopy group. In this case it prints: Presentation is aspherical.
Otherwise it returns "fail" and prints: Presentation is NOT piece-wise Euclidean non-positively curved. (In this case K may or may not have trivial second homotopy group. But it is NOT possible to impose a metric on K which restricts to a Euclidean metric on each 2-cell.)
The function uses Polymake software.

`PresentationOfResolution(R)`
Inputs at least two terms of a reduced ZG-resolution R and returns a record P with components

- $P.freeGroup$ is a free group F,
- $P.relators$ is a list S of words in F,
- $P.gens$ is a list of positive integers such that the i-th generator of the presentation corresponds to the group element R!.elts[P[i]] .

where G is isomorphic to F modulo the normal closure of S. This presentation for G corresponds to the 2-skeleton of the classifying CW-space from which R was constructed. The resolution R requires no contracting homotopy.

`TorsionGeneratorsAbelianGroup(G)`
Inputs an abelian group G and returns a generating set $[x_1, \ldots, x_n]$ where no pair of generators have coprime orders.

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-resolution for the associated Coxeter group W. The non-free resolution is obtained from the permutahedron of type W. A positive integer n can be entered as an optional second variable; just the first n terms of the non-free resolution are then returned.

`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" , "PSL(4,Z)_d" , "Sp(4,Z)"

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. (Subscripts _b , _c , _d denote alternative non-free ZG-resolutions for a given group G.)

Data for the first list of non-free resolutions was provided provided by MATHIEU DUTOUR. Data for the second list was provided by ALEXANDER RAHM.

`QuotientOfContractibleGcomplex(C,D)`
Inputs a non-free ZG-resolution C and a finite subgroup D of G which is a subgroup of each cell stabilizer group for C. Each element of D must preserves the orientation of any cell stabilized by it. It returns the corresponding non-free $Z(G/D)$-resolution. (So, for instance, from the $SL(2,O)$ complex $C = ContractibleGcomplex("SL(2,O-2)")$; we can construct a $PSL(2,O)$-complex using this function.)

`TruncatedGComplex(R,m,n)`
Inputs a non-free ZG-resolution R and two positive integers m and n. It returns the non-free ZG-resolution consisting of those modules in R of degree at least m and at most n.

`FundamentalDomainStandardSpaceGroup(v,G)`
Inputs a crystallographic group G (represented using AffineCrystGroupOnRight as in the GAP package Cryst). It also inputs a choice of vector v in the euclidean space R^n on which G acts. It returns the Dirichlet-Voronoi fundamental cell for the action of G on euclidean space corresponding to the vector v. The fundamental cell is a fundamental domain if G is Bieberbach. The fundamental cell/domain is returned as a "Polymake object". Currently the function only applies to certain crystallographic groups. See the manuals to HAPcryst and HAPpolymake for full details.
This function is part of the HAPcryst package written by MARC ROEDER and is thus only available if HAPcryst is loaded.
The function requires the use of Polymake software.

`OrbitPolytope(G,v,L)`
Inputs a permutation group or matrix group G of degree n and a rational vector v of length n. In both cases there is a natural action of G on v. Let $P(G,v)$ be the convex polytope arising as the convex hull of the Euclidean points in 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 the sublist, the function:

- prints the dimension of the orbit polytope $P(G,v)$;
- prints the degree of a vertex in the graph of $P(G,v)$;

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 returns the extension group determined by the cocycle. The group is returned as a CcGroup.
This is a HAPcocyclic function and thus only works when HAPcocyclic is loaded.

`CocycleCondition(R,n)`
Inputs a resolution R and an integer n>0. It returns an integer matrix M with the following property. Suppose $d = R.dimension(n)$. An integer vector $f = [f_1, \ldots, f_d]$ then represents a ZG-homomorphism $R_n \longrightarrow Z_q$ which sends the ith generator of R_n to the integer f_i in the trivial ZG-module Z_q (where possibly $q = 0$). The homomorphism f is a cocycle if and only if $M^t f = 0 \bmod q$.

`StandardCocycle(R,f,n)`
`StandardCocycle(R,f,n,q)`
Inputs a ZG-resolution R (with contracting homotopy), a positive integer n and an integer vector f representing an n-cocycle $R_n \longrightarrow Z_q$ where G acts trivially on 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 $G_n \longrightarrow Z_q$ corresponding to f. At present the command is implemented only for $n = 2$ or 3.

`Syzygy(R,g)`
Inputs a ZG-resolution R (with contracting homotopy) and a list $g = [g[1], ..., g[n]]$ of elements in G. It returns a word w in R_n. The word w is the image of the n-simplex in the standard bar resolution corresponding to the n-tuple g. This function can be used to construct explicit standard n-cocycles. (Currently implemented only for n<4.)

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 Z is greater than 0 then the next function might be more efficient.

`AddFreeWordsModP(v,w,p)`
Inputs two words v,w in a free ZG-module and the characteristic p of Z. It returns the sum $v+w$. If $p=0$ the previous function might be fractionally quicker.

`AlgebraicReduction(w)`
`AlgebraicReduction(w,p)`
Inputs a word w in a free ZG-module and returns a reduced version of the word in which all pairs of mutually inverse letters have been cancelled. The reduction is performed in a free abelian group unless the characteristic p of Z is entered.

`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 elements of G. The function prints the word w to the screen in the form
$r_1E_1+\ldots+r_nE_n$
where r_i are elements in the group ring ZG, and E_i denotes the i-th free generator of the module.

`TietzeReduction(S,w)`
Inputs a set S of words in a free ZG-module, and a word w in the module. The function returns a word w' such that `{`S,w'`}` generates the same abelian group as `{`S,w`}`. The word w' is possibly shorter (and certainly no longer) than w. This function needs to be improved!

`ResolutionBoundaryOfWord(R,n,w)`
Inputs a resolution R, a positive integer n and a list w representing a word in the free module R_n. It returns the image of w under the n-th boundary homomorphism.

Chapter 21

FpG-modules

`CompositionSeriesOfFpGModules(M)`
Inputs an FpG-module M and returns a list of FpG-modules that constitute a composition series for M.

`DirectSumOfFpGModules(M,N)` `DirectSumOfFpGModules([ M[1], M[2], ..., M[k] ])`
Inputs two FpG-modules M and N with common group and characteristic. It returns the direct sum of M and N as an FpG-Module.
Alternatively, the function can input a list of FpG-modules with common group G. It returns the direct sum of the list.

`FpGModule(A,P)` `FpGModule(A,G,p)`
Inputs a p-group P and a matrix A whose rows have length a multiple of the order of G. It returns the "canonical" FpG-module generated by the rows of A.
A small non-prime-power group G can also be input, provided the characteristic p is entered as a third input variable.

`FpGModuleDualBasis(M)`
Inputs an FpG-module M. It returns a record R with two components:

- $R.freeModule$ is the free module FG of rank one.
- $R.basis$ is a list representing an F-basis for the module $Hom_{FG}(M,FG)$. Each term in the list is a matrix A whose rows are vectors in FG such that $M!.generators[i] \longrightarrow A[i]$ extends to a module homomorphism $M \longrightarrow FG$.

`FpGModuleHomomorphism(M,N,A)` `FpGModuleHomomorphismNC(M,N,A)`
Inputs FpG-modules M and N over a common p-group G. Also inputs a list A of vectors in the vector space spanned by $N!.matrix$. It tests that the function
$M!.generators[i] \longrightarrow A[i]$
extends to a homomorphism of FpG-modules and, if the test is passed, returns the corresponding FpG-module homomorphism. If the test is failed it returns fail.
The "NC" version of the function assumes that the input defines a homomorphism and simply returns the FpG-module homomorphism.

`DesuspensionFpGModule(M,n)` `DesuspensionFpGModule(R,n)`
Inputs a positive integer n and and FpG-module M. It returns an FpG-module D^nM which is mathematically related to M via an exact sequence $0 \longrightarrow D^nM \longrightarrow R_n \longrightarrow \ldots \longrightarrow R_0 \longrightarrow M \longrightarrow 0$ where R_* is a free resolution. (If $G = Group(M)$ is of prime-power order then the resolution is minimal.)
Alternatively, the function can input a positive integer n and at least n terms of a free resolution R of M.

`RadicalOfFpGModule(M)`
Inputs an FpG-module M with G a p-group, and returns the Radical of M as an FpG-module. (Ig G is not a p-group then a submodule of the radical is returned.

`RadicalSeriesOfFpGModule(M)`
Inputs an FpG-module M and returns a list of FpG-modules that constitute the radical series for M.

`GeneratorsOfFpGModule(M)`
Inputs an FpG-module M and returns a matrix whose rows correspond to a minimal generating set for M.

`ImageOfFpGModuleHomomorphism(f)`
Inputs an FpG-module homomorphism $f : M \longrightarrow N$ and returns its image $f(M)$ as an FpG-module.

`GroupAlgebraAsFpGModule(G)`
Inputs a p-group G and returns its mod p group algebra as an FpG-module.

`IntersectionOfFpGModules(M,N)`

Chapter 22

Meataxe modules

`DesuspensionMtxModule(M)`
Inputs a meataxe module M over the field of p elements and returns an FpG-module DM. The two modules are related mathematically by the existence of a short exact sequence $DM \longrightarrow FM \longrightarrow M$ with FM a free module. Thus the homological properties of DM are equal to those of M with a dimension shift.
(If $G := Group(M.generators)$ is a p-group then FM is a projective cover of M in the sense that the homomorphism $FM \longrightarrow M$ does not factor as $FM \longrightarrow P \longrightarrow M$ for any projective module P.)

`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 minimal list of row vectors in V which generate V as a G-module (where G=Group(M.generators)).

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 where $G = E/N$.
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).

`GOuterGroupHomomorphismNC(A,B,phi)` `GOuterGroupHomomorphismNC()`
Inputs G-outer groups A and B with common acting group, and a group homomorphism phi:ActedGroup(A) –> ActedGroup(B). It returns the corresponding G-outer homomorphism PHI:A–> B. No check is made to verify that phi is actually a group homomorphism which preserves the G-action.
The function can be used without an argument. In this case an empty outer group homomorphism *PHI* is returned. The components must then be set.

`GOuterHomomorphismTester(A,B,phi)`
Inputs G-outer groups A and B with common acting group, and a group homomorphism phi:ActedGroup(A) –> ActedGroup(B). It tests whether phi is a group homomorphism which preserves the G-action.
The function can be used without an argument. In this case an empty outer group homomorphism *PHI* is returned. The components must then be set.

`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, and returns their group-theoretic direct product as a G-outer group. The outer action on the direct product is the diagonal one.
The function also applies to a list Lst of G-outer groups with common acting group.
For a direct product D constructed using this function, the embeddings and projections can be obtained (as G-outer group homomorphisms) using the functions Embedding(D,i) and Projection(D,i).

Chapter 24

Cat-1-groups

`AutomorphismGroupAsCatOneGroup(G)`
Inputs a group G and returns the Cat-1-group C corresponding to the crossed module $G \to Aut(G)$.

`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-module (i.e. abelian G-outer group) where G is the fundamental group of C.

`QuasiIsomorph(C)`
Inputs a cat-1-group C and returns a cat-1-group D for which there exists some homomorphism $C \to D$ that induces isomorphisms on homotopy groups.
This function was implemented by LE VAN LUYEN.

`ModuleAsCatOneGroup(G,alpha,M)`
Inputs a group G, an abelian group M and a homomorphism $\alpha: G \to Aut(M)$. It returns the Cat-1-group C corresponding th the zero crossed module $0: M \to G$.

`MooreComplex(C)`
Inputs a cat-1-group C and returns its Moore complex as a G-complex (i.e. as a complex of groups considered as 1-outer groups).

`NormalSubgroupAsCatOneGroup(G,N)`
Inputs a group G with normal subgroup N. It returns the Cat-1-group C corresponding th the inclusion crossed module $N \to G$.

`XmodToHAP(C)`
Inputs a cat-1-group C obtained from the Xmod package and returns a cat-1-group D for which IsHapCatOneGroup(D) returns true.
It returns "fail" id C has not been produced by the Xmod package.

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 the nerve of G as a simplicial group of length n.

This function applies both to cat-1-groups for which IsHapCatOneGroup(G) is true, and to cat-1-groups produced using the Xmod package.

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. The function returns the first $1+dim$ terms of a simplicial group with $n-1$st homotopy group equal to G and all other homotopy groups equal to zero.

This function was implemented by VAN LUYEN LE.

`EilenbergMacLaneSimplicialGroupMap(f,n,dim)`
Inputs a group homomorphism $f : G \to Q$, a positive integer n, and a positive integer dim. The function returns the first $1+dim$ terms of a simplicial group homomorphism $f : K(G,n) \to K(Q,n)$ of Eilenberg-MacLane simplicial groups.

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 CW-space X represented by the homotopy type of the simplicial group. Thus the homology groups of C are the integral homology groups of X.

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 map $f : C(G) \to C(Q)$ of chain complexes whose homology is the integral homology of the simplicial group G and Q respectively.

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 of G. It returns, as a group, the (n)-th homology group of its Moore complex. Thus HomotopyGroup(G,0) returns the "fundamental group" of G.

`Representation of elements in the bar resolution`
For a group G we denote by $B_n(G)$ the free $\mathbb{Z}G$-module with basis the lists $[g_1|g_2|...|g_n]$ where the g_i range over G.

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:

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 connected subgraphs D_i.

`CoxeterDiagramDegree(D,v)`
Inputs a Coxeter diagram D and vertex v. It returns the degree of v (i.e. the number of edges incident with v).

`CoxeterDiagramDisplay(D)  CoxeterDiagramDisplay(D,"web browser")`
Inputs a Coxeter diagram D and displays it as a .gif file. It uses the Mozilla web browser as a default to view the diagram. An alternative browser can be set using a second argument.
This function requires Graphviz software.

`CoxeterDiagramFpArtinGroup(D)`
Inputs a Coxeter diagram D and returns the corresponding finitely presented Artin group.

`CoxeterDiagramFpCoxeterGroup(D)`
Inputs a Coxeter diagram D and returns the corresponding finitely presented Coxeter group.

`CoxeterDiagramIsSpherical(D)`
Inputs a Coxeter diagram D and returns "true" if the associated Coxeter groups is finite, and returns "false" otherwise.

`CoxeterDiagramMatrix(D)`
Inputs a Coxeter diagram D and returns a matrix representation of it. The matrix is given as a function $DiagramMatrix(D)(i,j)$ where i,j can range over the vertices.

`CoxeterSubDiagram(D,V)`
Inputs a Coxeter diagram D and a subset V of its vertices. It returns the full sub-diagram of D with vertex set V.

`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 where x and y range over the generating set for G stored by GAP. The subgroup is probably only meaningful when G is an Artin or Coxeter group.

`GraphOfGroupsDisplay(D)  GraphOfGroupsDisplay(D,"web browser")`
Inputs a graph of groups D and displays it as a .gif file. It uses the Mozilla web browser as a default to view the diagram. An alternative browser can be set using a second argument.
This function requires Graphviz software.

`GraphOfResolutions(D,n)`
Inputs a graph of groups D and a positive integer n. It returns a graph of resolutions, each resolution being of length n. It uses the function ResolutionGenericGroup() to produce the 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 web browser as a default to view the diagram.
This function requires Graphviz software.

`GraphOfGroupsTest(D)`
Inputs an object D and itries to test whether it is a Graph of Groups. However, it DOES NOT test the injectivity of any homomorphisms. It returns true if D passes the test, and false otherwise.
Note that there is no function $IsHapGraphOfGroups()$ because no special data type has been created for these graphs.

#

Chapter 27

Torsion Subcomplexes

The Torsion Subcomplex subpackage has been conceived and implemented by BUI ANH TUAN and ALEXANDER D. RAHM

`RigidFacetsSubdivision( X )`
It inputs an n-dimensional G-equivariant CW-complex X on which all the cell stabilizer subgroups in G are finite. It returns an n-dimensional G-equivariant CW-complex Y which is topologically the same as X, but equipped with a G-CW-structure which is rigid.

`IsPNormal( G, p)`
Inputs a finite group G and a prime p. Checks if the group G is p-normal for the prime p. Zassenhaus defines a finite group to be p-normal if the center of one of its Sylow p-groups is the center of every Sylow p-group in which it is contained.

`TorsionSubcomplex( C, p)`
Inputs either a cell complex with action of a group as a variable or a group name. In HAP, presently the following cell complexes with stabilisers fixing their cells pointwise are available, specified by the following "groupName" 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)",

where the symbol O[-m] stands for the ring of integers in the imaginary quadratic number field Q(sqrt(-m)), the latter being the extension of the field of rational numbers by the square root of minus the square-free positive integer m. The additive structure of this ring O[-m] is given as the module Z[omega] over the natural integers Z with basis {1, omega}, and omega being the square root of minus m if m is congruent to 1 or 2 modulo four; else, in the case m congruent 3 modulo 4, the element omega is the arithmetic mean with 1, namely $(1+sqrt(-m))/2$.

The function TorsionSubcomplex prints the cells with p-torsion in their stabilizer on the screen and returns the incidence matrix of the 1-skeleton of this cellular subcomplex, as well as a Boolean value on whether the cell complex has its cell stabilisers fixing their cells pointwise.

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" , "PSL(4,Z)_d" , "Sp(4,Z)"

provided by MATHIEU DUTOUR.

`DisplayAvailableCellComplexes();`
Displays the cell complexes that are available in HAP.

`VisualizeTorsionSkeleton( groupName, p)`
Executes the function TorsionSubcomplex(groupName, p) and visualizes its output, namely the incidence matrix of the 1-skeleton of the p-torsion subcomplex, as a graph.

`ReduceTorsionSubcomplex( C, p)`
This function start with the same operations as the function TorsionSubcomplex(C, p), and if the cell stabilisers are fixing their cells pointwise, it continues as follows.

It prints on the screen which cells to merge and which edges to cut off in order to reduce the p-torsion subcomplex without changing the equivariant Farrell cohomology. Finally, it prints the representative cells, their stabilizers and the Abelianization of the latter.

`EquivariantEulerCharacteristic( X )`
It inputs an n-dimensional Γ-equivariant CW-complex X all the cell stabilizer subgroups in Γ are finite. It returns the equivariant euler characteristic obtained by using mass formula $\sum_\sigma (-1)^{dim\sigma} \frac{1}{card(\Gamma_\sigma)}$

Chapter 28

Simplicial Complexes

`Homology(T,n)` `Homology(T)`
Inputs a pure cubical complex, or cubical complex, or simplicial complex T and a non-negative integer n. It returns the n-th integral homology of T as a list of torsion integers. If no value of n is input then the list of all homologies of T in dimensions 0 to Dimension(T) is returned .
`RipsHomology(G,n)` `RipsHomology(G,n,p)`
Inputs a graph G, a non-negative integer n (and optionally a prime number p). It returns the integral homology (or mod p homology) in degree n of the Rips complex of G.

`Bettinumbers(T,n)` `Bettinumbers(T)`
Inputs a pure cubical complex, or cubical complex, simplicial complex or chain complex T and a non-negative integer n. The rank of the n-th rational homology group $H_n(T,\mathbb{Q})$ is returned. If no value for n is input then the list of Betti numbers in dimensions 0 to Dimension(T) is returned .
`ChainComplex(T)`
Inputs a pure cubical complex, or cubical complex, or simplicial complex T and returns the (often very large) cellular chain complex of T.

`CechComplexOfPureCubicalComplex(T)`
Inputs a d-dimensional pure cubical complex T and returns a simplicial complex S. The simplicial complex S has one vertex for each d-cube in T, and an n-simplex for each collection of n+1 d-cubes with non-trivial common intersection. The homotopy types of T and S are equal.

`PureComplexToSimplicialComplex(T,k)`
Inputs either a d-dimensional pure cubical complex T or a d-dimensional pure permutahedral complex T together with a non-negative integer k. It returns the first k dimensions of a simplicial complex S. The simplicial complex S has one vertex for each d-cell in T, and an n-simplex for each collection of n+1 d-cells with non-trivial common intersection. The homotopy types of T and S are equal.
For a pure cubical complex T this uses a slightly different algorithm to the function CechComplexOfPureCubicalComplex(T) but constructs the same simplicial complex.

`RipsChainComplex(G,n)`
Inputs a graph G and a non-negative integer n. It returns $n+1$ terms of a chain complex whose homology is that of the nerve (or Rips complex) of the graph in degrees up to n.

`VectorsToSymmetricMatrix(M)` `VectorsToSymmetricMatrix(M,distance)`
Inputs a matrix M of rational numbers and returns a symmetric matrix S whose (i,j) entry is the distance between the i-th row and j-th rows of M where distance is given by the sum of the absolute values of the coordinate differences.
Optionally, a function distance(v,w) can be entered as a second argument. This function has to return a rational number for each pair of rational vectors v,w of length Length(M[1]).

`EulerCharacteristic(T)`
Inputs a pure cubical complex, or cubical complex, or simplicial complex T and returns its Euler characteristic.

`MaximalSimplicesToSimplicialComplex(L)`
Inputs a list L whose entries are lists of vertices representing the maximal simplices of a simplicial complex. The simplicial complex is returned. Here a "vertex" is a GAP object such as an integer or a subgroup.

`SkeletonOfSimplicialComplex(S,k)`
Inputs a simplicial complex S and a positive integer k less than or equal to the dimension of S. It returns the truncated k-dimensional simplicial complex S^k (and leaves S unchanged).

`GraphOfSimplicialComplex(S)`
Inputs a simplicial complex S and returns the graph of S.

`ContractibleSubcomplexOfSimplicialComplex(S)`
Inputs a simplicial complex S and returns a (probably maximal) contractible subcomplex of S.

Chapter 29

Cubical Complexes

`ArrayToPureCubicalComplexA,n)`
Inputs an integer array A of dimension d and an integer n. It returns a d-dimensional pure cubical complex corresponding to the black/white "image" determined by the threshold n and the values of the entries in A. (Integers below the threshold correspond to a black pixel, and higher integers correspond to a white pixel.)

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

`FramedPureCubicalComplex(M)`
Inputs a pure cubical complex M and returns the pure cubical complex with a border of zeros attached the each face of the boundary array M!.boundaryArray. This function just adds a bit of space for performing operations such as thickenings to M.

`RandomCubeOfPureCubicalComplex(M)`
Inputs a pure cubical complex M and returns a pure cubical complex R with precisely the same dimensions as M. The complex R consist of one cube selected at random from M.

`PureCubicalComplexIntersection(S,T)`
Inputs two pure cubical complexes with common dimension and array size. It returns the intersection of the two complexes. (An entry in the binary array of the intersection has value 1 if and only if the corresponding entries in the binary arrays of S and T both have value 1.)

`PureCubicalComplexUnion(S,T)`
Inputs two pure cubical complexes with common dimension and array size. It returns the union of the two complexes. (An entry in the binary array of the union has value 1 if and only if at least one of the corresponding entries in the binary arrays of S and T has value 1.)

`PureCubicalComplexDifference(S,T)`
Inputs two pure cubical complexes with common dimension and array size. It returns the difference S-T. (An entry in the binary array of the difference has value 1 if and only if the corresponding entry in the binary array of S is 1 and the corresponding entry in the binary array of T is 0.)

`ReadImageAsPureCubicalComplex("file.png",n)`
Reads an image file ("file.png", "file.eps", "file.bmp" etc) and an integer n between 0 and 765. It returns a 2-dimensional pure cubical complex based on the black/white version of the image determined by the threshold n.

`ReadLinkImageAsPureCubicalComplex("file.png")` `ReadLinkImageAsPureCubicalComplex("file.png"`
Reads an image file ("file.png", "file.eps", "file.bmp" etc) containing a knot or link diagram, and optionally a positive integer n. The integer n should be a little larger than the line thickness in the link diagram, and if not provided then n is set equal to 10. The function tries to output the corresponding knot or link as a 3-dimensional pure cubical complex. Ideally the link diagram should be produced with line thickness 6 in Xfig, and the under-crossing spaces should not be too large or too small or too near one another. The function does not always succeed: it applies several checks, and if one of these checks fails then the function returns "fail".

`ReadImageSequenceAsPureCubicalComplex("directory",n)`
Reads the name of a directory containing a sequence of image files (ordered alphanumerically), and an integer n between 0 and 765. It returns a 3-dimensional pure cubical complex based on the black/white version of the images determined by the threshold n.
`Size(T)`
This returns the number of non-zero entries in the binary array of the cubical complex, or pure cubical complex T.
`Dimension(T)`
This returns the dimension of the cubical complex, or pure cubical complex T.

Chapter 30

Regular CW-Complexes

`SimplicialComplexToRegularCWComplex(K)`
Inputs a simplicial complex K and returns the corresponding regular CW-complex.

`CubicalComplexToRegularCWComplex(K)` `CubicalComplexToRegularCWComplex(K,n)`
Inputs a pure cubical complex (or cubical complex) K and returns the corresponding regular CW-complex. If a positive integer n is entered as an optional second argument, then just the n-skeleton of K is returned.

`CriticalCellsOfRegularCWComplex(Y)` `CriticalCellsOfRegularCWComplex(Y,n)`
Inputs a regular CW-complex Y and returns the critical cells of Y with respect to some discrete vector field. If Y does not initially have a discrete vector field then one is constructed.
If a positive integer n is given as a second optional input, then just the critical cells in dimensions up to and including n are returned.
The function $CriticalCellsOfRegularCWComplex(Y)$ works by homotopy reducing cells starting at the top dimension. The function $CriticalCellsOfRegularCWComplex(Y,n)$ works by homotopy coreducing cells starting at dimension 0. The two methods may well return different numbers of cells.

`ChainComplex(Y)`
Inputs a regular CW-complex Y and returns the cellular chain complex of a CW-complex W whose cells correspond to the critical cells of Y with respect to some discrete vector field. If Y does not initially have a discrete vector field then one is constructed.

`ChainComplexOfRegularCWComplex(Y)`
Inputs a regular CW-complex Y and returns the cellular chain complex of Y.

`FundamentalGroup(Y)` `FundamentalGroup(Y,n)`
Inputs a regular CW-complex Y and, optionally, the number of some 0-cell. It returns the fundamental group of Y based at the 0-cell n. The group is returned as a finitely presented group. If n is not specified then it is set $n = 1$. The algorithm requires a discrete vector field on Y. If Y does not initially have a discrete vector field then one is constructed.

Chapter 31

Knots and Links

`PureCubicalKnot(L)` `PureCubicalKnot(n,i)`
Inputs a list $L = [[m1,n1],[m2,n2],...,[mk,nk]]$ of pairs of integers describing a cubical arc presentation of a link with all vertical lines at the front and all horizontal lines at the back. The bottom horizontal line extends from the m1-th column to the n1-th column. The second to bottom horizontal line extends from the m2-th column to the n2-th column. And so on. The link is returned as a 3-dimensional pure cubical complex.
Alternatively the function inputs two integers n, i and returns the i-th prime knot on n crossings.

`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 defined for links with more than one component.

`KnotGroup(K)`
Inputs a pure cubical link K and returns the fundamental group of its complement. The group is returned as a finitely presented group.

`AlexanderMatrix(G)`
Inputs a finitely presented group G whose abelianization is infinite cyclic. It returns the Alexander matrix of the presentation.

`AlexanderPolynomial(K)` `AlexanderPolynomial(G)`
Inputs either a pure cubical knot K or a finitely presented group G whose abelianization is infinite cyclic. The Alexander Polynomial is returned.

`ProjectionOfPureCubicalComplex(K)`
Inputs an n-dimensional pure cubical complex K and returns an n-1-dimensional pure cubical complex K'. The returned complex is obtained by projecting Euclidean n-space onto Euclidean n-1-space.

`ReadPDBfileAsPureCubicalComplex(file)` `ReadPDBfileAsPureCubicalComplex(file,m,c)`
Inputs a protein database file describing a protein, and optionally inputs a positive integer m and character string c. The default values for the optional inputs are m=5 and c="A". It loads the chain of amino acids labelled by c in the file as a 3-dimensional pure cubical complex of the homotopy type of a circle.
It might happen that the function fails to construct a pure cubical complex of the homotopy type of a circle. In this case retry with a larger integer m.

Chapter 32

Knots and Quandles

Knots

`PresentationKnotQuandle(gaussCode)`
Inputs a Gauss Code of a knot (with the orientations; see *GaussCodeOfPureCubicalKnot* in HAP package) and outputs the generators and relators of the knot quandle associated (in the form of a record).

`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 ($n \leq 12$) if it exists; fail otherwise.

`GaussCodeKnot(n,k)`
Returns a Gauss Code (with orientations) for the k-th knot with n crossings ($n \leq 12$) if it exists; fail otherwise.

`PresentationKnotQuandleKnot(n,k)`
Returns generators and relators (in the form of a record) for the k-th knot with n crossings ($n \leq 12$) if it exists; fail otherwise.

`NumberOfHomomorphisms(genRelQ,finiteQ)`
Inputs generators and relators *genRelQ* of a knot quandle (in the form of a record, see above) and a finite quandle $finiteQ$; outputs the number of homomorphisms from the former to the latter.

`PartitionedNumberOfHomomorphisms(genRelQ,finiteQ)`
Inputs generators and relators *genRelQ* of a knot quandle (in the form of a record, see above) and a finite connected quandle $finiteQ$; outputs a partition of the number of homomorphisms from the former to the latter.
Quandles

`ConjugationQuandle(G,n)`
Inputs a finite group G and an integer n; outputs the associated n-fold conjugation quandle.

`FirstQuandleAxiomIsSatisfied(M)`
`SecondQuandleAxiomIsSatisfied(M)`
`ThirdQuandleAxiomIsSatisfied(M)`
Inputs a finite magma M; returns true if M satisfy the first/second/third axiom of a quandle, false otherwise.

`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, $n \leq 6$. If $n \geq 7$, it returns fail.

`Quandle(n,k)`
Returns the k-th quandle of size n ($n \leq 6$) if such a quandle exists, fail otherwise.

`IdQuandle(Q)`
Inputs a quandle Q; and outputs a list of integers $[n,k]$ such that Q is isomorphic to `Quandle(n,k)`. If $n \geq 7$, then it returns [n,fail] (where n is the size of Q).

`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.

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 symmetric group containing them. It returns the minimum number of transpositions needed to express $g*h^-1$ as a product of transpositions.

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

`KendallMetric(g,h,N)` `KendallMetric(g,h)`
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 adjacent transposition has the for $(i,i+1)$.

`EuclideanSquaredMetric(v,w)`
Inputs two vectors v,w of equal length and returns the sum of the squares of the components of $v-w$. In other words, it returns the square of the Euclidean distance between v and w.

`EuclideanApproximatedMetric(v,w)`
Inputs two vectors v,w of equal length and returns a rational approximation to the square root of the sum of the squares of the components of $v-w$. In other words, it returns an approximation to the Euclidean distance between v and w.

`ManhattanMetric(v,w)`
Inputs two vectors v,w of equal length and returns the sum of the absolute values of the components of $v-w$. This is often referred to as the taxi-cab distance between v and w.

`VectorsToSymmetricMatrix(L)` `VectorsToSymmetricMatrix(L,D)`
Inputs a list L of vectors and optionally a metric D. The default is $D = ManhattanMetric$. It returns the symmetric matrix whose i-j-entry is $S[i][j] = D(L[i],L[j])$.

`SymmetricMatDisplay(S)` `SymmetricMatDisplay(L,V)`
Inputs an $n \times n$ symmetric matrix S of non-negative integers and an integer t in $[0..100]$. Optionally it inputs a list $V = [V_1,...,V_k]$ of disjoint subsets of $[1..n]$. It displays the graph with vertex set $[1..n]$ and with an edge between i and j if $S[i][j] < t$. If the optional list V is input then the vertices in V_i will be given a common colour distinct from other vertices.

`SymmetricMatrixToFilteredGraph(S,t,m)`
Inputs an integer symmetric matrix S, a positive integer t and a positive integer m. The function returns a filtered graph of filtration length t. The k-th term of the filtration is a graph with one vertex for each row of S. There is an edge in this graph between the i-th and j-th vertices if the entry $S[i][j]$ is less than or equal to $k*m/t$.

`PermGroupToFilteredGraph(S,D)`
Inputs a permutation group G and a metric D defined on permutations. The function returns a filtered graph. The k-th term of the filtration is a graph with one vertex for each element of the group G. There is an edge in this graph between vertices g and h if $D(g,h)$ is less than some integer threshold t_k. The thresholds $t_1 < t_2 < ... < t_N$ are chosen to form as long a sequence as possible subject to each term of the filtration being a distinct graph.

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 composite $h_1 h_2 ... h_n$ is defined. It returns the list of composable homomorphism as a commutative diagram.

`NormalSeriesToQuotientDiagram(L)` `NormalSeriesToQuotientDiagram(L,M)`
Inputs an increasing (or decreasing) list $L = [L_1, L_2, ..., L_n]$ of normal subgroups of a group G with $G = L_n$. It returns the chain of quotient homomorphisms $G/L_i \to G/L_{i+1}$ as a commutative diagram. Optionally a subseries M of L can be entered as a second variable. Then the resulting diagram of quotient groups has two rows of horizontal arrows and one row of vertical arrows.

`NerveOfCommutativeDiagram(D)`
Inputs a commutative diagram D and returns the commutative diagram ND consisting of all possible composites of the arrows in D.

`GroupHomologyOfCommutativeDiagram(D,n)` `GroupHomologyOfCommutativeDiagram(D,n,prime)`
`GroupHomologyOfCommutativeDiagram(D,n,prime,Resolution_Algorithm)`
Inputs a commutative diagram D of p-groups and positive integer n. It returns the commutative diagram of vector spaces obtained by applying mod p homology.
Non-prime power groups can also be handled if a prime p is entered as the third argument. Integral homology can be obtained by setting $p = 0$. For $p = 0$ the result is a diagram of groups.
A particular resolution algorithm, such as ResolutionNilpotentGroup, can be entered as a fourth argument. For positive p the default is ResolutionPrimePowerGroup. For $p = 0$ the default is ResolutionFiniteGroup.

`PersistentHomologyOfCommutativeDiagramOfPGroups(D,n)`
Inputs a commutative diagram D of finite p-groups and a positive integer n. It returns a list containing, for each homomorphism in the nerve of D, a triple $[k,l,m]$ where k is the dimension of the source of the induced mod p homology map in degree n, l is the dimension of the image, and m is the dimension of the cokernel.

ABSTRACT CATEGORIES

`CategoricalEnrichment(X,Name)`
Inputs a structure X such as a group or group homomorphism, together with the name of some existing category such as Name:=Category_of_Groups or Category_of_Abelian_Groups. It returns, as appropriate, an object or arrow in the named category.

`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 to X.

`TerminalArrow(X)`
Inputs an object X in some category, and returns the arrow from X to the terminal object in the category.

`HasInitialObject(Name)`
Inputs the name of a category and returns true or false depending on whether the category has an initial object.

`HasTerminalObject(Name)`
Inputs the name of a category and returns true or false depending on whether the category has a terminal object.

`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 of g. (Warning: this differes to the standard GAP convention.)
Equality is tested using $f=g$.
In an additive category the sum and difference of suitable arrows is given by $f+g$ and $f-g$.

`Object(X)`
Inputs an object X in some category, and returns the GAP structure Y such that $X = CategoricalEnrichment(Y, CategoryName(X))$.

`Mapping(X)`
Inputs an arrow f in some category, and returns the GAP structure Y such that $f = CategoricalEnrichment(Y, CategoryName(X))$.

`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 (and leaves A unchanged).

`PermuteArray(A,f)`
Inputs an array A of dimension d and a permutation f of degree at most d. It returns the array B defined by $B[i1][i2]...[id] = A[f(i1)][f(i2)]...A[f(id)]$ (and leaves A unchanged).

`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 arrays of dimension d.

`ArrayAssign(A,x,n)`
Inputs an array A and a coordinate vector x and an integer n. It sets the entry of A with coordinate x equal to n.

`ArrayAssignFunctions(d)`
Inputs a positive integer d and returns an efficient version of the function ArrayAssign for arrays of dimension d.

`ArrayIterate(d)`
Inputs a positive integer d and returns a function ArrayIt(Dimensions,f). This function inputs a list Dimensions of d positive integers and also a function $f(x)$. It applies the function $f(x)$ to each integer list x of length d with entries $x[i]$ in the range [1..Dimension[i]].

`BinaryArrayToTextFile(file,A)`
Inputs a string containing the address of a file, and an array A of 0s and 1s. The array represents a pure cubical complex. A representation of this complex is written to the file in a format that can be read by the CAPD (Computer Assisted Proofs in Dynamics) software developed by Marian Mrozek and others.
The second input A can also be a pure cubical complex.

`FrameArray(A)`
Inputs an array A and returns the array obtained by appending a 0 to the beginning and end of each "row" of the array.

`UnframeArray(A)`
Inputs an array A and returns the array obtained by removing the first and last entry in each "row" of the array.

`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 operation converts L into a pseudo list of length n+1 by adding x as the final entry. If x is in L the operation has no effect on L.

`Append(L,K)`
Let L be a pseudo list and K a list whose objects are compatible with those in L. This operation applies Add(L,x) for each x in K.

Chapter 36

Parallel Computation - Core Functions

`ChildProcess()` `ChildProcess("computer.ac.wales")` `ChildProcess(["-m", "100000M", "-T"])` `ChildProcess("computer.ac.wales", ["-m", "100000M", "-T"])`
This 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.
(To configure ssh so that the user can login without a password prompt from "thishost" to "remotehost" either consult "man ssh" or

- 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 "cmd;" is a string representing the command.

`NextAvailableChild(L)`
Inputs a list *L* of child processes and returns a child in *L* which is ready for computation (as soon as such a child is available).

`IsAvailableChild(s)`
Inputs a child process *s* and returns true if s is currently available for computations, and false otherwise.

`ChildPut(A,"B",s)`
This copies a GAP object A on the parent process to an object B on the child process s. (The copying relies on the function PrintObj(A);)

`ChildGet("A",s)`
This functions copies a GAP object A on the child process s and returns it on the parent process. (The copying relies on the function PrintObj(A);)

`HAPPrintTo("file",R)`
Inputs a name "file" of a new text file and a HAP object R. It writes the object R to "file". Currently this is only implemented for R equal to a resolution.

`HAPRead("file",R)`
Inputs a name "file" containing a HAP object R and returns the object. Currently this is only implemented for R equal to a resolution.

Chapter 37

Parallel Computation - Extra Functions

`ChildFunction("function(arg);",s)`
This runs the GAP function "function(arg);" on a child process accessed by the stream s. The output from "func;" can be accessed via the stream.

`ChildRead(s)`
This returns, as a string, the output of the last application of $ChildFunction(\text{"}function(arg);\text{"},s)$.

`ChildReadEval(s)`
This returns, as an evaluated string, the output of the last application of $ChildFunction(\text{"}function(arg);\text{"},s)$.

`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. It uses the children in L to compute $List(I,x->fn(x))$. (Obviously the function fn must be defined on all child processes in 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 as a matrix.

`Dimension(C)`
`Dimension(M)`
Inputs a resolution, chain complex or cochain complex C and returns the function $C!.dimension$.
Alternatively, inputs an FpG-module M and returns its dimension as a vector space over the field of p elements.

`EvaluateProperty(X,"name")`
Inputs a component object X (such as a ZG-resolution or chain map) and a string "name" (such as "characteristic" or "type"). It searches $X.property$ for the pair ["name",value] and returns value. If $X.property$ does not exist, or if ["name",value] does not exist, it returns fail.

`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 to the target or the source of f) .

`Source(f)`
Inputs a chain map, or cochain map, or equivariant chain map, or FpG-module homomorphism f and returns it source.

`Target(f)`
Inputs a chain map, or cochain map, or equivariant chain map, or FpG-module homomorphism f and returns its target.

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 function returns the conjugate $SL(2,Z)^P$ of the special linear group $SL(2,Z)$ by the matrix $P = [[1,0],[0,p]]$. In the second case it returns the group $SL(2,Z[1/m])$.

`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 lower central series of G such that L_i/L_{i+1} has order greater than n.

`Classify(L,Inv)`
Inputs a list of objects L and a function Inv which computes an invariant of each object. It returns a list of lists which classifies the objects of L according to the invariant..

`RefineClassification(C,Inv)`
Inputs a list $C := Classify(L, OldInv)$ and returns a refined classification according to the invariant Inv.

`Compose(f,g)`
Inputs two FpG-module homomorphisms $f : M \longrightarrow N$ and $g : L \longrightarrow M$ with $Source(f) = Target(g)$. It returns the composite homomorphism $fg : L \longrightarrow N$.
This also applies to group homomorphisms f, 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 algebras (preserving the Lie bracket).

`IsSuperperfect(G)`
Inputs a group G and returns "true" if both the first and second integral homology of G is trivial. Otherwise, it returns "false".

`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 $f : G \longrightarrow M$ where M is a group of permutation matrices.

`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 matrix S satisfying $SM = B$. The function will leave matrix M unchanged but will probably change matrix B.
(This is a trivial rewrite of the standard GAP function $SolutionMatDestructive$(<mat>,<vec>) .)

`LinearHomomorphismsPersistenceMat(L)`
Inputs a composable sequence L of vector space homomorphisms. It returns an integer matrix containing the dimensions of the images of the various composites. The sequence L is determined up to isomorphism by this matrix.

`NormalSeriesToQuotientHomomorphisms(L)`
Inputs an (increasing or decreasing) chain L of normal subgroups in some group G. This G is the largest group in the chain. It returns the sequence of composable group homomorphisms $G/L[i] \rightarrow G/L[i+/-1]$.

`TestHap()`
This runs a representative sample of HAP functions and checks to see that they produce the correct output.

Index