The SCO Package Manual

Simplicial Cohomology of Orbifolds

Version 2015.11.06

August 2011

Simon Görtzen

Simon Görtzen
Email: simon.goertzen@rwth-aachen.de
Homepage: http://wwwb.math.rwth-aachen.de/goertzen/
Address: Lehrstuhl B für Mathematik
RWTH Aachen
Templergraben 64
52062 Aachen
(Germany)

Abstract

This document explains the primary uses of the SCO package. Included in this manual is a documented list of the most important methods and functions you will need. For the theoretical basis of this package please refer to my diploma thesis and the corresponding paper (work in progress; [Gör08a]).

Copyright

Acknowledgements

The SCO package would not have been possible without the theoretical work by I. Moerdijk and D. A. Pronk concerning simplicial cohomology of orbifolds [MP99]. Many thanks to these two, as well as Mohamed Barakat and the Lehrstuhl B für Mathematik at RWTH Aachen University in general. It should be noted that SCO in its current functionality is dependent on the GAP package homalg by M. Barakat [BR08], as it relies on homalg to do the actual computations. This manual was created with the help of the GAPDoc package by M. Neunhöffer and F. Lübeck.

Contents

Chapter 1

Introduction

1.1 Overview over this manual

Chapter 1 is concerned with the technical details of installing and running this package. The following chapter 2 explains how to use SCO to compute simplicial (co-)homology of orbifolds. For the theoretical parts please refer to my diploma thesis and the corresponding paper (work in progress; [Gör08a]). After this chapter you will find some simple examples on using SCO with (finite) groups, manifolds, or some easy orbifolds. Also included in this manual is a documented list of the most important methods and functions you will need to work with SCO's data types OrbifoldTriangulation and SimplicialSet and to create the matrices needed for computations. Anyone interested in source code should just check out the files in the `gap/pkg/SCO/gap/` folder (→ Appendix A).

1.2 Installation of the SCO Package

To install this package just extract the package's archive file to the GAP `pkg/` directory. By default the SCO package is not automatically loaded by GAP when it is installed. You must load the package with `LoadPackage("SCO");` before its functions become available. Please, send me an e-mail if you have any questions, remarks, suggestions, etc. concerning SCO. Also, I would like to hear about applications of this package.
Simon Goertzen

Chapter 2

Usage

There are different ways to use SCO. Please note that for the actual computations the homalg package is required, and you will need both the RingsForHomalg and the GaussForHomalg package to make use of the full computational capabilities. For your information, RingsForHomalg offers support for external computer algebra systems and the rings they support, while GaussForHomalg extends GAP functionality with regards to sparse matrices and computations over fields and $\mathbb{Z}/\langle p^n \rangle$.

2.1 The Examples Script

Regardless of the extend of your installation, you will always be able to call the example script `SCO/examples/examples.g`. This script is not only callable in-GAP by `SCO_Examples` (4.3.6), but also automatically checks which packages you have installed and provides you with the available options. The example script is designed to take you through the ring creation process and then load one of the files of your choice located in the `SCO/examples/orbifolds/` directory. In there you will find a lot of test files with small 0- or 1-dimensional orbifolds, but also the complete triangulations of the 17 orbifolds corresponding to the 2-dimensional wallpaper groups (these should be exactly the uncapitalized files, ranging from `p1.g` to `p6m.g`). Computing the cohomology of these orbifolds was an important part of my diploma thesis [Gör08a] and I have also created a separate document [Gör08b] to present my results.

Please note that the variables *M*, *iso*, and *mu* in the orbifold files have to keep their name for the example script to work correctly. Refer to chapter 3 for concrete examples.

2.2 Working Manually

Once you are familiar with the example script and want to try out your own triangulations, it is best to create your own `.g` file in the `SCO/examples/orbifolds/` directory, then call the script again. If for any reason you do not want to create a file or work with the script, you can always do every step by hand. Check 4 if you need to know more about specific methods and functions. The basic steps are:

- Define a list of maximum simplices
- If applicable, define an isotropy record
- If applicable, define a list encoding the μ-map

- From the above data, create an orbifold triangulation
- Define the simplicial set of the orbifold triangulation
- Create a homalg ring R
- Create boundary or coboundary matrices over R
- Calculate their homology or cohomology

Chapter 3

Examples

Although there are some small examples embedded in chapter 4, we will give some complete examples in this chapter. Most of these could easily be called with the example script mentioned in chapter 2, but we will do them step by step for best reproducability.

3.1 Example 1: Klein Bottle

Suppose we want to calculate the cohomology of the Klein Bottle. First, we need a triangulation. It could look like this:

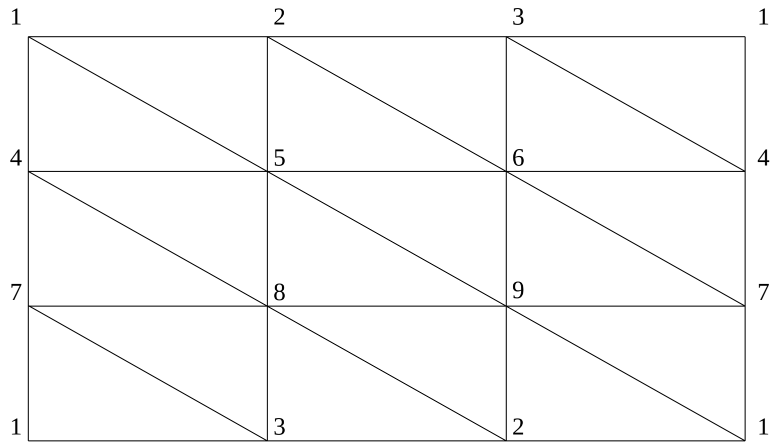

Figure 3.1: triangulation

This results in the following list of maximum simplices:

Example

```
gap> M := [ [1,2,4], [1,2,7], [1,3,6], [1,3,8], [1,4,6], [1,7,8],
> [2,3,5], [2,3,9], [2,4,5], [2,7,9], [3,5,6], [3,8,9],
> [4,5,7], [4,6,9], [4,7,9], [5,6,8], [5,7,8], [6,8,9] ];;
```

As there is no isotropy and therefore no μ-map, we can continue with the orbifold triangulation and simplicial set:

Example
```
gap> ot := OrbifoldTriangulation( M, "Klein Bottle" );
<OrbifoldTriangulation "Klein Bottle" of dimension 2. 18 simplices on 9 vertic\
es without Isotropy>
gap> ss := SimplicialSet( ot );
<The simplicial set of the orbifold triangulation "Klein Bottle", computed up \
to dimension 0 with Length vector [ 18 ]>
```

Now we will need a homalg ring. As this is a small example we can compute directly over $\mathbb{Z}$, so we can use GAP. In case you have RingsForHomalg installed you might want to try computing in another computer algebra system with the command `HomalgRingOfIntegersInCAS()`, replacing "CAS" with the corresponding system.

Example
```
gap> R := HomalgRingOfIntegers();
Z
```

We are almost there. Let us create some coboundary matrices and compute their cohomology:

Example
```
gap> c := CreateCoboundaryMatrices( ss, 4, R );;
gap> C := Cohomology( c, R );
------------------------------------------------->>>>  Z^(1 x 1)
------------------------------------------------->>>>  Z^(1 x 1)
------------------------------------------------->>>>  Z/< 2 >
------------------------------------------------->>>>  0
------------------------------------------------->>>>  0
<A graded cohomology object consisting of 5 left modules at degrees
[ 0 .. 4 ]>
```

This is the cohomology of the Klein Bottle.

3.2 Example 2: V_4

SCO can also be used to compute group cohomology, as every group can be represented as an orbifold with just a single point. For V_4, it would look like this:

Example
```
gap> M := [ [1] ];;
gap> V4 := Group( (1,2), (3,4) );;
gap> iso := rec( 1 := V4 );;
gap> ot := OrbifoldTriangulation( M, iso, "V4" );
<OrbifoldTriangulation "V4" of dimension 0. 1 simplex on 1 vertex with Isotrop\
y on 1 vertex>
gap> ss := SimplicialSet( ot );
<The simplicial set of the orbifold triangulation "V4", computed up to dimensi\
on 0 with Length vector [ 1 ]>
gap> R := HomalgRingOfIntegers();
Z
gap> c := CreateCoboundaryMatrices( ss, 4, R );;
gap> C := Cohomology( c, R );
------------------------------------------------->>>>  Z^(1 x 1)
------------------------------------------------->>>>  0
```

```
------------------------------------------------>>>>  Z/< 2 > + Z/< 2 >
------------------------------------------------>>>>  Z/< 2 >
------------------------------------------------>>>>  Z/< 2 > + Z/< 2 > + Z/< 2\
 >
<A graded cohomology object consisting of 5 left modules at degrees
[ 0 .. 4 ]>
```

This is the V_4 group cohomology up to degree 4.

3.3 Example 3: The "Teardrop" orbifold

You have seen a manifold in example 1, and group cohomology in example 2. Now we will meet our first proper orbifold, the Teardrop. This is the example Moerdijk and Pronk used in their paper [MP99] on which my work is based. It is an easy example, but includes both nontrivial isotropy and μ-maps. We take the isotropy at the top to be C_2. The triangulation looks like this, with the gluing being at [1,3].

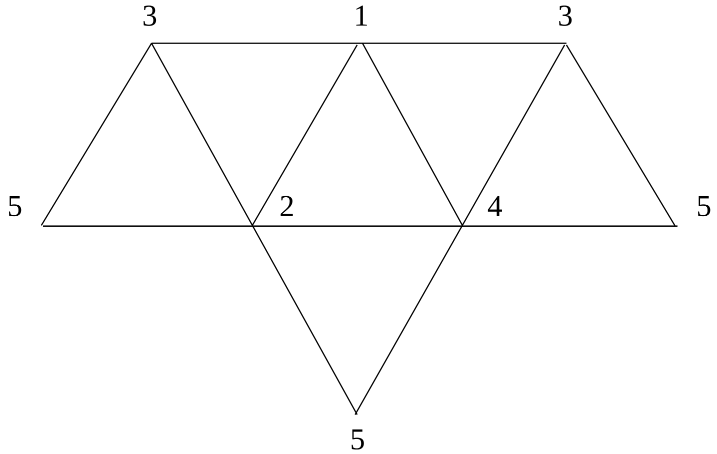

Figure 3.2: triangulation

The source code:

Example
```
gap> M := [ [1,2,3], [1,2,4], [1,3,4], [2,3,5], [2,4,5], [3,4,5] ];;
gap> iso := rec( 1 := Group( (1,2) ) );;
gap> mu := [
>             [ [3], [1,3], [1,2,3], [1,3,4], x -> (1,2) ],
>             [ [3], [1,3], [1,3,4], [1,2,3], x -> (1,2) ]
>           ];;
gap> ot := OrbifoldTriangulation( M, iso, mu, "Teardrop" );
<OrbifoldTriangulation "Teardrop" of dimension 2. 6 simplices on 5 vertices wi\
th Isotropy on 1 vertex and nontrivial mu-maps>
gap> ss := SimplicialSet( ot );
<The simplicial set of the orbifold triangulation "Teardrop", computed up to d\
imension 0 with Length vector [ 6 ]>
gap> R := HomalgRingOfIntegers();
Z
```

```
gap> c := CreateCoboundaryMatrices( ss, 6, R );;
gap> C := Cohomology( c, R );
---------------------------------------------->>>>  Z^(1 x 1)
---------------------------------------------->>>>  0
---------------------------------------------->>>>  Z^(1 x 1)
---------------------------------------------->>>>  0
---------------------------------------------->>>>  Z/< 2 >
---------------------------------------------->>>>  0
---------------------------------------------->>>>  Z/< 2 >
<A graded cohomology object consisting of 7 left modules at degrees
[ 0 .. 6 ]>
```

This is the Teardrop cohomology.

Chapter 4

SCO methods and functions

4.1 Methods and functions for orbifold triangulations

4.1.1 OrbifoldTriangulation

▷ `OrbifoldTriangulation(M[, I, mu_data, info])` (function)

Returns: OrbifoldTriangulation

The constructor for OrbifoldTriangulations. Needs the list *M* of maximal simplices, the Isotropy at certain vertices as a record *I*, and the list *mu_data* that encodes the function mu. If only one argument is given, *I* and *mu_data* are supposed to be empty. In case of two arguments, *mu_data* is supposed to be empty. If the last argument *info* is given as a string, it is stored in the info component of the orbifold triangulation and does not count towards the total number of arguments.

Example

```
gap> M := [ [1,2,3], [1,2,4], [1,3,4], [2,3,4] ];;
gap> S2 := OrbifoldTriangulation( M, "S^2" );
<OrbifoldTriangulation "S^2" of dimension 2. 4 simplices on 4 vertices without\
 Isotropy>
gap> I := rec( 1 := Group( (1,2) ) );;
gap> mu_data := [
> [ [2], [1,2], [1,2,3], [1,2,4], x->x*(1,2) ],
> [ [2], [1,2], [1,2,4], [1,2,3], x->x*(1,2) ]
> ];;
gap> Teardrop := OrbifoldTriangulation( M, I, mu_data, "Teardrop" );
<OrbifoldTriangulation "Teardrop" of dimension 2. 4 simplices on 4 vertices wi\
th Isotropy on 1 vertex and nontrivial mu-maps>
```

4.1.2 Vertices

▷ `Vertices(ot)` (method)

Returns: List *V*

This returns the list of vertices *V* of the orbifold triangulation *ot*. Should be preferred to the equivalent `ot!.vertices`.

4.1.3 Simplices

▷ `Simplices(ot)` (method)

Returns: List *M*

This returns the list of maximal simplices *M* of the orbifold triangulation *ot*. Should be preferred to the equivalent `ot!.max_simplices`.

4.1.4 Isotropy

▷ `Isotropy(ot)` (method)

Returns: Record *I*

This returns the isotropy record *I* of the orbifold triangulation *ot*. Should be preferred to the equivalent `ot!.isotropy`.

4.1.5 Mu

▷ `Mu(ot)` (method)

Returns: Function *mu*

This returns the function *mu* of the orbifold triangulation *ot*. Should be preferred to the equivalent `ot!.mu`.

4.1.6 MuData

▷ `MuData(ot)` (method)

Returns: List *mu_data*

This returns the list *mu_data* that encodes the function mu of the orbifold triangulation *ot*. Should be preferred to the equivalent `ot!.mu_data`.

4.1.7 InfoString

▷ `InfoString(ot)` (method)

Returns: String *info*

This return the string *info* of the orbifold triangulation *ot*. Should be preferred to the equivalent `ot!.info`.

4.2 Methods and functions for simplicial sets

4.2.1 SimplicialSet (constructor)

▷ `SimplicialSet(ot)` (method)

Returns: SimplicialSet

The constructor for simplicial sets based on an orbifold triangulation *ot*. This just sets up the object without any computations. These can be triggered later, either explicitly or by `SimplicialSet` (4.2.2).

Example

```
gap> Teardrop;
<OrbifoldTriangulation "Teardrop" of dimension 2. 4 simplices on 4 vertices wi\
th Isotropy on 1 vertex and nontrivial mu-maps>
gap> S := SimplicialSet( Teardrop );
<The simplicial set of the orbifold triangulation "Teardrop", computed up to d\
imension 0 with Length vector [ 4 ]>
```

4.2.2 SimplicialSet (data access)

▷ `SimplicialSet(S, i)` (method)

Returns: List `S_i`

This returns the components of dimension `i` of the simplicial set `S`. Should be used to access existing data instead of using `S!.simplicial_set[ i + 1 ]`, as it has the additional side effect of computing `S` up to dimension `i`, thus always returning the desired result.

Example
```
gap> S := SimplicialSet( Teardrop );
<The simplicial set of the orbifold triangulation "Teardrop", computed up to d\
imension 0 with Length vector [ 4 ]>
gap> S!.simplicial_set[1];
[ [ [ 1, 2, 3 ] ], [ [ 1, 2, 4 ] ], [ [ 1, 3, 4 ] ], [ [ 2, 3, 4 ] ] ]
gap> S!.simplicial_set[2];;
Error, List Element: <list>[2] must have an assigned value
gap> SimplicialSet( S, 0 );
[ [ [ 1, 2, 3 ] ], [ [ 1, 2, 4 ] ], [ [ 1, 3, 4 ] ], [ [ 2, 3, 4 ] ] ]
gap> SimplicialSet( S, 1 );;
gap> S;
<The simplicial set of the orbifold triangulation "Teardrop", computed up to d\
imension 1 with Length vector [ 4, 12 ]>
```

4.2.3 ComputeNextDimension

▷ `ComputeNextDimension(S)` (method)

Returns: `S`

This computes the component of the next dimension of the simplicial set `S`. `S` is extended as a side effect.

Example
```
gap> S;
<The simplicial set of the orbifold triangulation "Teardrop", computed up to d\
imension 1 with Length vector [ 4, 12 ]>
gap> ComputeNextDimension( S );
<The simplicial set of the orbifold triangulation "Teardrop", computed up to d\
imension 2 with Length vector [ 4, 12, 22 ]>
```

4.2.4 Extend

▷ `Extend(S, i)` (method)

Returns: `S`

This computes the components of the simplicial set `S` up to dimension `i`. `S` is extended as a side effect. This method is equivalent to calling `ComputeNextDimension` (4.2.3) the appropriate number of times.

Example
```
gap> S;
<The simplicial set of the orbifold triangulation "Teardrop", computed up to d\
imension 2 with Length vector [ 4, 12, 22 ]>
gap> Extend( S, 5 );
<The simplicial set of the orbifold triangulation "Teardrop", computed up to d\
imension 5 with Length vector [ 4, 12, 22, 33, 51, 73 ]>
```

4.3 Methods and functions for matrix creation and computation

4.3.1 BoundaryOperator

▷ `BoundaryOperator(i, L, mu)` (function)

Returns: List B

This returns the `i`th boundary of `L`, which has to be an element of a simplicial set. `mu` is the function μ that has to be taken into account when computing orbifold boundaries. This function is used for matrix creation, there should not be much reason for calling it independently.

4.3.2 CreateBoundaryMatrices

▷ `CreateBoundaryMatrices(S, d, R)` (method)

Returns: List `M`

This returns the list `M` of homalg matrices over the homalg ring `R` up to dimension `d`, corresponding to the boundary matrices induced by the simplicial set `S`. If `d` is not given, the current dimension of `S` is used.

Example
```
gap> S := SimplicialSet( Teardrop );
<The simplicial set of the orbifold triangulation "Teardrop", computed up to d\
imension 0 with Length vector [ 4 ]>
gap> M := CreateBoundaryMatrices( S, 4, HomalgRingOfIntegers() );;
gap> S;
<The simplicial set of the orbifold triangulation "Teardrop", computed up to d\
imension 5 with Length vector [ 4, 12, 22, 33, 51, 73 ]>
```

4.3.3 Homology

▷ `Homology(M[, R])` (method)

Returns: a homalg complex

This returns the homology complex of a list `M` of homalg matrices over the homalg ring `R`.

Example
```
gap> S := SimplicialSet( Teardrop );
<The simplicial set of the orbifold triangulation "Teardrop", computed up to d\
imension 0 with Length vector [ 4 ]>
gap> R := HomalgRingOfIntegers();
Z
gap> M := CreateBoundaryMatrices( S, 4, R );;
gap> Homology( M, R );
----------------------------------------------->>>>  Z^(1 x 1)
----------------------------------------------->>>>  0
----------------------------------------------->>>>  Z^(1 x 1)
----------------------------------------------->>>>  Z/< 2 >
----------------------------------------------->>>>  0
<A graded homology object consisting of 5 left modules at degrees [ 0 .. 4 ]>
```

4.3.4 CreateCoboundaryMatrices

▷ `CreateCoboundaryMatrices(S[, d], R)` (method)

Returns: List `M`

This returns the list *M* of homalg matrices over the homalg ring *R* up to dimension *d*, corresponding to the coboundary matrices induced by the simplicial set *S*. If *d* is not given, the current dimension of *S* is used.

Example
```
gap> S := SimplicialSet( Teardrop );
<The simplicial set of the orbifold triangulation "Teardrop", computed up to d\
imension 0 with Length vector [ 4 ]>
gap> M := CreateCoboundaryMatrices( S, 4, HomalgRingOfIntegers() );;
gap> S;
<The simplicial set of the orbifold triangulation "Teardrop", computed up to d\
imension 5 with Length vector [ 4, 12, 22, 33, 51, 73 ]>
```

4.3.5 Cohomology

▷ Cohomology(*M*[, *R*]) (method)

Returns: a homalg complex

This returns the cohomology complex of a list *M* of homalg matrices over the homalg ring *R*.

Example
```
gap> S := SimplicialSet( Teardrop );
<The simplicial set of the orbifold triangulation "Teardrop", computed up to d\
imension 0 with Length vector [ 4 ]>
gap> R := HomalgRingOfIntegers();
Z
gap> M := CreateCoboundaryMatrices( S, 4, R );;
gap> Cohomology( M, R );
----------------------------------------------->>>>  Z^(1 x 1)
----------------------------------------------->>>>  0
----------------------------------------------->>>>  Z^(1 x 1)
----------------------------------------------->>>>  0
----------------------------------------------->>>>  Z/< 2 >
<A graded cohomology object consisting of 5 left modules at degrees
[ 0 .. 4 ]>
```

4.3.6 SCO_Examples

▷ SCO_Examples() (function)

Returns: nothing

This is just an easy way to call the script examples.g, which is located in gap/pkg/SCO/examples/.

Example
```
gap> SCO_Examples();
@@@@@@@@ SCO @@@@@@@@

Select base ring:
 1) Integers (default)
 2) Rationals
 3) Z/nZ
:1

Select Computer Algebra System:
 1) GAP (default)
```

```
 2) External GAP
 3) MAGMA
 4) Maple
 5) Sage
:3
---------------------------------------------------------------
Magma V2.14-14    Tue Aug 19 2008 08:36:19 on evariste [Seed = 1054613462]
Type ? for help.  Type <Ctrl>-D to quit.
----------------------------------------------------------------


Select Method:
 1) Full syzygy computation (default)
 2) matrix creation and rank computation only
:1

Select orbifold (default="C2")
:Torus

Select mode:
 1) Cohomology (default)
 2) Homology
:1

Select dimension (default = 4)
:4
Creating the coboundary matrices ...
Starting cohomology computation ...
----------------------------------------------->>>>  Z^(1 x 1)
----------------------------------------------->>>>  Z^(1 x 2)
----------------------------------------------->>>>  Z^(1 x 1)
----------------------------------------------->>>>  0
----------------------------------------------->>>>  0
```

Appendix A

An Overview of the SCO package source code

Filename	Content
OrbifoldTriangulation.gi	Definitions and methods for orbifold triangulations
SimplicialSet.gi	Definitions and methods for simplicial sets
Matrices.gi	Methods for (Co-)homology matrix creation
SCO.gi	(Co-)homology computations and `SCO_Examples` (4.3.6)

Table: *The SCO package files.*

References

[BR08] M. Barakat and D. Robertz. *homalg project*, 2003-2008. http://wwwb.math.rwth-aachen.de/homalg. 2

[Gör08a] S. Görtzen. Simplicial cohomology of orbifolds. Master's thesis, RWTH Aachen University, 2008. in preparation. 2, 4, 5

[Gör08b] S. Görtzen. Wallpaper group cohomology. http://wwwb.math.rwth-aachen.de/goertzen/, 2008. 5

[MP99] I. Moerdijk and D. A. Pronk. Simplicial cohomology of orbifolds. *Indag. Math. (N.S.)*, 10(2):269–293, 1999. 2, 9

Index