The ModularGroup Package

Finite-index subgroups of $(P)SL_2(\mathbb{Z})$

Version 0.0.1

16 February 2018

Luca Leon Junk

Luca Leon Junk
Email: junk@math.uni-sb.de
Homepage: http://www.math.uni-sb.de/ag/weitze/
Address: AG Weitze-Schmithüsen
FR 6.1 Mathematik
Universität des Saarlandes
D-66041 Saarbrücken

Copyright

Acknowledgements

Supported by Project I.8 of SFB-TRR 195 'Symbolic Tools in Mathematics and their Application' of the German Research Foundation (DFG).

Contents

Chapter 1

Introduction

1.1 General aims of the ModularGroup package

This GAP package provides methods for computing with finite-index subgroups of the modular groups $SL_2(\mathbb{Z})$ and $PSL_2(\mathbb{Z})$. This includes, but is not limited to, computation of the generalized level, index or cusp widths. It also implements algorithms described in [Hsu96] and [HL14] for testing if a given group is a congruence subgroup.

1.2 Installation

1.3 Technicalities

A convenient way to represent finite-index subgroups of $SL_2(\mathbb{Z})$ is by specifying the action of generator matrices on the right cosets by right multiplication. For example, one could choose the generators

$$S = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \quad T = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

and represent a subgroup as a tuple of transitive permutations (σ_S, σ_T) describing the action of S and T. This is exactly the way this package internally treats such subgroups. We employ the convention that 1 corresponds to the coset of the identity matrix.

Our choice of S and T as generators of $SL_2(\mathbb{Z})$ is quite arbitrary, but we plan to implement methods for displaying results and inputting data in terms of different generators in the near future.

Chapter 2

Subgroups of $SL_2(\mathbb{Z})$

For representing finite-index subgroups of $SL_2(\mathbb{Z})$, this package introduces the new type `ModularSubgroup`. As stated in the introduction, a `ModularSubgroup` essentially consists of the two permutations σ_S and σ_T describing the coset graph with respect to the generator matrices S and T (with the convention that 1 corresponds to the identity coset). So explicitly specifying these permutations is the canonical way to construct a `ModularSubgroup`.
Though you might not always have a coset graph of your subgroup at hand, but rather a list of generator matrices. Therefore we implement two different constructors for `ModularSubgroup`, one that takes as input two permutations describing the coset graph, and one that takes a list of $SL_2(\mathbb{Z})$ matrices as generators.

2.1 Construction of modular subgroups

2.1.1 Constructors

▷ `ModularSubgroup(s, t)` (operation)

Returns: A modular subgroup.

Constructs a `ModularSubgroup` object corresponding to the finite-index subgroup of $SL_2(\mathbb{Z})$ described by the permutations `s` and `t`.
This constructor tests if the given permutations actually describe the coset action of the matrices

$$S = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \quad T = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

by checking that they act transitively and satisfy the relations

$$s^4 = (s^3t)^3 = s^2ts^{-2}t^{-1} = 1$$

Upon creation, the cosets are renamed in a standardized way to make the internal interaction with extisting GAP methods easier.

Example

```
gap> G := ModularSubgroup(
> (1,2)(3,4)(5,6)(7,8)(9,10),
> (1,4)(2,5,9,10,8)(3,7,6));
<modular subgroup of index 10>
```

▷ `ModularSubgroup(gens)` (operation)

Returns: A modular subgroup.

Constructs a `ModularSubgroup` object corresponding to the finite-index subgroup of $SL_2(\mathbb{Z})$ generated by the matrices in *gens*.
No test is performed to check if the generated subgroup actually has finite index!
This constructor implicitly computes a coset table of the subgroup. Hence it might be slow for very large index subgroups.

Example
```
gap> G := ModularSubgroup([
> [[1,2], [0,1]],
> [[1,0], [2,1]],
> [[-1,0], [0,-1]]
> ]);
<modular subgroup of index 6>
```

2.1.2 Getters for the coset action

▷ `SAction(G)` (operation)

Returns: A permutation.

Returns the permutation σ_S describing the action of the matrix S on the cosets of *G*.

▷ `TAction(G)` (operation)

Returns: A permutation.

Returns the permutation σ_T describing the action of the matrix T on the cosets of *G*.

▷ `CosetActionOf(A, G)` (operation)

Returns: A permutation.

Returns the permutation σ_A describing the action of the matrix $A \in SL_2(\mathbb{Z})$ on the cosets of *G*.

2.2 Computing with modular subgroups

2.2.1 Index

▷ `Index(G)` (attribute)

Returns: A natural number.

For a given modular subgroup *G* this method returns its index in $SL_2(\mathbb{Z})$. As *G* is internally stored as permutations (s,t) this is just

```
LargestMovedPoint(s,t)
```

(or 1 if the permutations are trivial).

2.2.2 GeneralizedLevel

▷ `GeneralizedLevel(G)` (attribute)

Returns: A natural number.

This method calculates the general Wohlfahrt level (i.e. the lowest common multiple of all cusp widths) of `G`.

2.2.3 RightCosetRepresentatives

▷ `RightCosetRepresentatives(G)` (attribute)

Returns: A list of words.

This function returns a list of representatives of the (right) cosets of `G` as words in S and T.

Example
```
gap> G := ModularSubgroup((1,2),(2,3));
<modular subgroup of index 3>
gap> RightCosetRepresentatives(G);
[ <identity ...>, S, S*T ]
```

2.2.4 GeneratorsOfGroup

▷ `GeneratorsOfGroup(G)` (attribute)

Returns: A list of words.

Calculates a list of generators (as words in S and T) of `G`. This list might include redundant generators (or even duplicates).

Example
```
gap> G := ModularSubgroup((1,2)(3,5)(4,6), (1,3)(2,4)(5,6));
<modular subgroup of index 6>
gap> GeneratorsOfGroup(G);
[ S^-2, T^-2, S*T^-2*S^-1 ]
```

2.2.5 MatrixGeneratorsOfGroup

▷ `MatrixGeneratorsOfGroup(G)` (attribute)

Returns: A list of matrices.

Calculates a list of generator matrices of `G`. This list might include redundant generators (or even duplicates).

Example
```
gap> G := ModularSubgroup((1,2)(3,5)(4,6), (1,3)(2,4)(5,6));
<modular subgroup of index 6>
gap> MatrixGeneratorsOfGroup(G);
[ [ [ -1, 0 ], [ 0, -1 ] ], [ [ 1, -2 ], [ 0, 1 ] ], [ [ 1, 0 ], [ 2, 1 ] ] ]
```

2.2.6 IsCongruence

▷ `IsCongruence(G)` (attribute)

Returns: True or false.

This method test whether a given modular subgroup *G* is a congruence subgroup. It is essentially an implementation of an algorithm described in [HL14].

Example

```
gap> G := ModularSubgroup([
> [[1,2],[0,1]],
> [[1,0],[2,1]]
> ]);
<modular subgroup of index 12>
gap> IsCongruence(G);
true
```

2.2.7 Cusps

▷ `Cusps(G)` (attribute)

Returns: A list of rational numbers and infinity.

This method computes a list of inequivalent cusp representatives with respect to *G*.

Example

```
gap> G := ModularSubgroup(
> (1,2)(3,6)(4,8)(5,9)(7,11)(10,13)(12,15)(14,17)(16,19)(18,21)(20,23)(22,24),
> (1,3,7,4)(2,5)(6,9,8,12,14,10)(11,13,16,20,18,15)(17,21,22,19)(23,24)
> );
<modular subgroup of index 24>
gap> Cusps(G);
[ infinity, 0, 1, 2, 3/2, 5/3 ]
```

2.2.8 CuspWidth

▷ `CuspWidth(c, G)` (operation)

Returns: A natural number.

This method takes as input a cusp *c* (a rational number or infinity) and a modular group *G* and calculates the width of this cusp with respect to *G*.

Example

```
gap> G := ModularSubgroup(
> (1,2,6,3)(4,11,15,12)(5,13,16,14)(7,17,9,18)(8,19,10,20)(21,24,22,23),
> (1,4,5)(2,7,8)(3,9,10)(6,15,16)(11,20,21)(12,19,22)(13,23,17)(14,24,18)
> );
<modular subgroup of index 24>
gap> CuspWidth(-1, G);
3
gap> CuspWidth(infinity, G);
3
```

2.2.9 CuspsEquivalent

▷ `CuspsEquivalent(p, q, G)` (operation)

Returns: True or false.

Takes two cusps `p` and `q` and a modular subgroup `G` and checks if they are equivalent modulo `G`, i.e. if there exists a matrix $A \in G$ with $Ap = q$.

Example

```
gap> G := ModularSubgroup(
> (1,2,6,3)(4,11,15,12)(5,13,16,14)(7,17,9,18)(8,19,10,20)(21,24,22,23),
> (1,4,5)(2,7,8)(3,9,10)(6,15,16)(11,20,21)(12,19,22)(13,23,17)(14,24,18)
> );
<modular subgroup of index 24>
gap> CuspsEquivalent(infinity, 1, G);
false
gap> CuspsEquivalent(-1, 1/2, G);
true
```

2.2.10 IndexModN

▷ `IndexModN(G, N)` (operation)

Returns: A natural number.

For a modular subgroup `G` and a natural number `N` this method calculates the index of the projection $\bar{G}$ of G in $SL_2(\mathbb{Z}/N\mathbb{Z})$.

2.2.11 Deficiency

▷ `Deficiency(G, N)` (operation)

Returns: A natural number.

For a modular subgroup `G` and a natural number `N` this method calculates the so-called *deficiency* of `G` from being a congruence subgroup of level `N`.
The deficiency of a finite-index subgroup Γ of $SL_2(\mathbb{Z})$ was introduced in [WS15]. It is defined as the index $[\Gamma(N) : \Gamma(N) \cap \Gamma]$ where $\Gamma(N)$ is the principal congruence subgroup of level N.

Example

```
gap> G := ModularSubgroup([
> [[1,2],[0,1]],
> [[1,0],[2,1]]
> ]);
<modular subgroup of index 12>
gap> Deficiency(G, 2);
2
gap> Deficiency(G, 4);
1
```

2.2.12 Projection

▷ `Projection(G)` (operation)

Returns: A projective modular subgroup.

For a given modular subgroup `G` this function calculates its image `Ḡ` under the projection $\pi\colon SL_2(\mathbb{Z}) \to PSL_2(\mathbb{Z})$.

2.2.13 NormalCore

▷ `NormalCore(G)` (attribute)

Returns: A modular subgroup.

Calculates the normal core of `G` in $SL_2(\mathbb{Z})$, i.e. the maximal subgroup of `G` that is normal in $SL_2(\mathbb{Z})$.

2.2.14 QuotientByNormalCore

▷ `QuotientByNormalCore(G)` (attribute)

Returns: A finite group.

Calculates the quotient of $SL_2(\mathbb{Z})$ by the normal core of `G`.

2.2.15 AssociatedCharacterTable

▷ `AssociatedCharacterTable(G)` (attribute)

Returns: A character table.

Returns the character table of $SL_2(\mathbb{Z})/N$ where N is the normal core of `G`.

2.2.16 IsElementOf

▷ `IsElementOf(A, G)` (operation)

Returns: True or false.

This function checks if a given matrix `A` is an element of the modular subgroup `G`.

2.3 Miscellaneous

The following functions are mostly helper functions used internally and are only documented for sake of completeness.

2.3.1 DefinesCosetAction

▷ `DefinesCosetAction(s, t)` (operation)

Returns: True or false.

Checks if two given permutations `s` and `t` describe the action of the generator matrices S and T on the cosets of some subgroup. This is the case if they satisfy the relations

$$s^4 = (s^3t)^3 = s^2ts^{-2}t^{-1} = 1$$

and act transitively.

2.3.2 CosetActionFromGenerators

▷ `CosetActionFromGenerators(gens)` (operation)

Returns: A tuple of permutations.

Takes a list of generator matrices and calculates the coset graph (as two permutations σ_S and σ_T) of the generated subgroup of $SL_2(\mathbb{Z})$.

2.3.3 STDecomposition

▷ `STDecomposition(A)` (operation)

Returns: A word in S and T.

Takes a matrix $A \in SL_2(\mathbb{Z})$ and decomposes it into a word in the generator matrices S and T.

Example

```
gap> M := [ [ 4, 3 ], [ -3, -2 ] ];;
gap> STDecomposition(M);
S^2*T^-1*S^-1*T^2*S^-1*T^-1*S^-1
```

2.3.4 STDecompositionAsList

▷ `STDecompositionAsList(A)` (operation)

Returns: A list representing a word in S and T.

Takes a matrix $A \in SL_2(\mathbb{Z})$ and decomposes it into a word in the generator matrices S and T. The word is represented as a list in the format [[generator, exponent], ...]

Example

```
gap> M := [ [ 4, 3 ], [ -3, -2 ] ];;
gap> STDecompositionAsList(M);
[ [ "S", 2 ], [ "T", -1 ], [ "S", -1 ], [ "T", 2 ], [ "S", -1 ], [ "T", -1 ],
  [ "S", -1 ], [ "T", 0 ] ]
```

Chapter 3

Subgroups of $PSL_2(\mathbb{Z})$

Analogous to finite-index subgroups of $SL_2(\mathbb{Z})$, we define a new type `ProjectiveModularSubgroup` for representing subgroups of $PSL_2(\mathbb{Z})$. It consists essentially of two permutations $\sigma_{\overline{S}}$ and $\sigma_{\overline{T}}$ describing the action of $\overline{S}$ and $\overline{T}$ on the cosets of the given subgroup, where $\overline{S}$ and $\overline{T}$ are the images of S and T in $PSL_2(\mathbb{Z})$.

References

[HL14] Thomas Hamilton and David Loeffler. Congruence testing for odd subgroups of the modular group. *LMS Journal of Computation and Mathematics*, 17(1):206–208, 2014. 4, 8

[Hsu96] Tim Hsu. Identifying congruence subgroups of the modular group. *Proceedings of the American Mathematical Society*, 124(5), April 1996. 4

[WS15] Gabriela Weitze-Schmithuesen. The deficiency of being a congruence group for veech groups of origamis. *International Mathematics Research Notices*, 2015(6):1613–1637, 2015. 9

Index