The Origami Package

Computing Veechgroups of Origamis

Version 1.0.0

13.11.2018

Pascal Kattler
Andrea Thevis

Pascal Kattler Email: kattler@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

Andrea Thevis Email: thevis@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

We thank Sergio Siccha for his support and valuable input at the beginning of this project. Further, we would like to thank Vincent Delecroix and Samuel Lelièvre for fruitful discussions about algorithms implemented in the surface_dynamics package in sage. Some of the functionality of the surface_dynamics package can be used in this package as well. For this we use an interface between GAP and sage. We are thankful to Mohamed Barakat and Markus Pfeiffer for helping using the interfaces. Moreover, we thank Thomas Breuer for helpful ideas and comments on some algorithms implemented in this package.

This software package is part of the project I.8 'Algorithmic approaches to Teichmüller curves' (AG Weitze, Saarland University) 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 Overview

This package provides calculations with a certain class of translation surfaces called origamis. An origami (also known as square-tiled surface) is a finite covering of a torus which is ramified at most over one point. It can be described in the following way from two permutations $\sigma_x, \sigma_y \in S_d$. We take d squares $Q_1, \ldots, Q_d$ and glue the lower side of Q_i to the upper side of $Q_{\sigma_y(i)}$ and the right side of Q_i to the left side of $Q_{\sigma_x(i)}$. We require origamis to be connected and thus the group generated by σ_x and σ_y acts transitively on $\{1, \ldots, d\}$. In this package we identify an origami with a pair of permutations, which acts transitively on $\{1, \ldots, d\}$ up to simultaneous conjugation. This corresponds to renumbering the labels of the squares. By choosing a certain numbering in a canonical way one can achieve a canonical representative.

In this package we are mainly interested in Veech groups of origamis. They are subgroups of finite index of $SL_2(\mathbb{Z})$ which carry a lot of information about the geometric and dynamic properties of the underlying translation surface. For further information about origamis and translation surfaces in general see e.g.[Zmi11], [Sch05], [HL06], [Ran17] and [Zor06].

1.2 Notations

Since we are mainly interested in the Veech group of an origami, which are a subgroups of $SL_2(\mathbb{Z})$ of finite index, we fix two generators of $SL_2(\mathbb{Z})$

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

and

$$T = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}.$$

In this package we fix the free group F generated by $\tilde{S}$ and $\tilde{T}$. We consider the canonical epimorphism $\pi : F \to SL_2(\mathbb{Z})$ with $\pi(\tilde{S}) = S$ and $\pi(\tilde{T}) = T$.

Chapter 2

The functionality of this package

2.1 The Origami Object

In this section we introduce a new type called 'origami', namely origami objects, which are created by this two permutations. The degree of an origami is the number of squares. Origamis are stored as such objects.

2.1.1 Origami

▷ `Origami(`*`permX, permY`*`)` (function)

Returns: An Origami.

This function generates a new origami object with $\sigma_x =$ *permX* , $\sigma_y =$ *permY* . The function tests whether (*permX* ,*permY*) defines a connected surface and returns false otherwise.

Example

```
gap> Origami((1,2), (2,3));
Origami((1,2), (2,3), 3)
```

▷ `OrigamiNC(`*`permX, permY, d`*`)` (function)

Returns: An Origami.

This function does the same as `Origami`, but in contrast it does not test, whether the origami describes a connected surface.

▷ `OrigamiNormalForm(`*`origami`*`)` (function)

Returns: An Origami.

This function calculates a canonical representation of *origami*. Two origamis are equivalent, if they have the same canonical representation. This function has been implemented by Luca Junk and the algorithm is from [McK].

Example

```
gap> OrigamiNormalForm(Origami((1,3,5,7)(2,8,4,10)(6,9),
>(1,5,2,6,3)(4,10)(7,9) ));
Origami((1,2)(3,5,4,6)(7,8,9,10), (1,3)(2,4,5,6,7)(9,10), 10)
```

▷ `OrigamiFamily` (family)

Since origamis do not fit in any existing family in GAP, we introduce a new family for origami objects called OrigamiFamily.

▷ HorizontalPerm(*origami*) (attribute)

Returns: A permutation.

This function returns the horizontal permutation σ_y of an origami.

Example

```
gap> HorizontalPerm(Origami((1,3,5), (1,3)(2,4,5)));
(1,3,5)
```

▷ VerticalPerm(*origami*) (attribute)

Returns: A permutation.

This function returns the vertical permutation σ_x of an origami.

Example

```
gap> VerticalPerm( Origami((1,3,5), (1,3)(2,4,5) ));
(1,3)(2,4,5)
```

▷ DegreeOrigami(*origami*) (attribute)

Returns: An Int.

This function returns the degree of an origami.

Example

```
gap> DegreeOrigami(Origami((1,3,5), (1,3)(2,4,5) ));
5
```

▷ Stratum(*origami*) (attribute)

Returns: A list of Ints.

This function calculates the stratum of an origami. The stratum of an origami is a list of the nonzero degrees of the singularities. For a singularity of cone angle $2\pi k$ the degree of the singularity is $k-1$.

Example

```
gap> Stratum(Origami((1,6,4,7,5,3)(2,8), (1,4,5,3,8,2,6) ));
[ 1, 5 ]
```

▷ Genus(*origami*) (attribute)

Returns: An Int.

This function calculates the genus of the origami surface.

Example

```
gap> Genus( Origami((1,2,3,4),(1,2)(3,4) ) );
2
```

2.2 SL_2(Z)-Action on Origamis

The group $SL_2(\mathbb{Z})$ acts on the set of origamis of fixed degree. The following methods help calculating this operation both for matrices and words in the free group F.

2.2.1 ActionOfSpecialLinearGroup

▷ `ActionOfSpecialLinearGroup(word, origami)` (method)

Returns: An Origami object.

Given a word *word* in the free group $F = Group(\tilde{S},\tilde{T})$ this function computes $\pi(word) \in SL_2(\mathbb{Z})$ and returns $\pi(word).origami$. The word is given as a string, as shown in the following example.

Example

```
gap> ActionOfSpecialLinearGroup("ST",Origami((1,3,5), (1,3)(2,4,5)));
Origami((1,3)(2,5,4), (2,4,5,3), 5)
```

Returns: An Origami.

▷ `ActionOfSpecialLinearGroup(matrix, origami)` (method)

Given matrix in $SL_2(\mathbb{Z})$ this function returns *matrix.origami*.

Example

```
gap> ActionOfSpecialLinearGroup([ [ 0, -1 ], [ 1, 1 ] ], Origami((1,3,5),
>(1,3)(2,4,5)));
Origami((1,3)(2,5,4), (2,4,5,3), 5)
```

▷ `ActionOfF2ViaCanonical(origami, word)` (function)

Returns: An Origami.

Given a word *word* in the free group $Group(\tilde{S},\tilde{T})$ this function computes $\pi(word) \in SL_2(\mathbb{Z})$ and returns $\pi(word).origami$. But in contrast to `ActionOfSpecialLinearGroup` the result is stored in the canonical representation. ATTENTION: the order of arguments is switched compared to the order of the arguments in the function `ActionOfSpecialLinearGroup`.

Example

```
gap> ActionOfF2ViaCanonical(Origami((1,2), (1,3)), "S");
Origami((1,2), (2,3), 3)
```

▷ `RightActionOfF2ViaCanonical(origami, word)` (function)

Returns: An Origami.

This function computes the right action of the projection of a word *word* in the free group $Group(\tilde{S},\tilde{T})$ on an origami *origami*. It returns $origami.\pi(word) = \pi(word)^{-1}.origami$, where the left action is the common action of $SL_2(\mathbb{Z})$ on origamis of a given degree. This action has the same orbits as the left action. For the Veech group computation both actions can be used and give the same result. In contrast to `ActionOfSpecialLinearGroup` the result is stored in the canonical representation. ATTENTION: the order of arguments is switched compared to the order of the arguments in the function `ActionOfSpecialLinearGroup`.

Example

```
gap> RightActionOfF2ViaCanonical(Origami((2,3), (1,3,2)),"T");
Origami((1,2), (2,3), 3)
```

2.3 Veech groups

The Veech group G of an origami is a subgroup $SL_2(\mathbb{Z})$ of finite index. The group is stored as a ModularSubgroup from the ModularSubgroup package. It is represented by two permutations σ_S and σ_T describing how the generators S and T of $SL_2(\mathbb{Z})$ act on the cosets of G in $SL_2(\mathbb{Z})$. E.g, if $SH_i = H_j$ and H_i, H_j are the cosets associated to integers i, j, respectively, then $\sigma_S(i) = j$. See [Sch04] for a detailed version of the algorithm.

2.3.1 VeechGroup

▷ VeechGroup(*origami*) (attribute)

Returns: A ModularSubGroup.

This function calculates the Veech group of an origami. You get the coset permutations using the ModularSubgroup package as in the following example.

Example

```
gap> SAction(VeechGroup(Origami((1,2,5)(3,4,6), (1,2)(5,6) )));
(1,3)(2,5)(4,7)(6,8)(9,10)
gap> TAction(VeechGroup(Origami((1,2,5)(3,4,6), (1,2)(5,6) )));
(1,2,4)(3,6)(5,8,7,9,10)
```

▷ Cosets(*origami*) (attribute)

Returns: A list of element of the free group with generators S and T.

This function calculates the right cosets of the Veech group of an origami as a list of words in S and T.

Example

```
gap> Cosets(Origami((1,2,5)(3,4,6), (1,2)(5,6), 6));
[ <identity ...>, S, T, T^-1, S*T, S*T^-1, T*S, T^-1*S, S*T*S, S*T^-1*S ]
```

Returns: A boolean.

▷ EquivalentOrigami(*origami1, origami2*) (function)

This function tests whether *origami1* is equal to *origami2* up to renumbering of the squares.

Example

```
gap> EquivalentOrigami(Origami((1,4)(2,6,3), (1,5)(2,3,6,4) ), Origami((1,4,3)
>(2,5), (1,5,3,4)(2,6) ));
true

gap> EquivalentOrigami(Origami((1,4)(2,6,3), (1,5)(2,3,6,4) ), Origami((1,2,5)
>(3,4,6), (1,2)(5,6) ));
false
```

At this point we thank Thomas Breuer for the idea to implement the latter function.

2.4 Lists of Origamis

The following functions generate lists of origamis of a given degree (and stratum).

2.4.1 OrigamiList

▷ OrigamiList(*d*) (function)

Returns: A list of Origamis.

This function calculates a list of all origamis with a given degree *d*.

Example

```
gap> OrigamiList(2);
[ Origami((), (1,2), 2), Origami((1,2), (), 2), Origami((1,2), (1,2), 2) ]
```

▷ `OrigamiListWithStratum(`*`d, stratum`*`)` (function)

Returns: A list of Origamis.

This function calculates a list of all origamis with a given degree *d* and stratum *stratum*.

Example

```
gap> OrigamiListWithStratum(5, [1,1]);
[ Origami((1,2), (1,3)(2,4,5), 5), Origami((1,2), (1,3,2,4,5), 5),
  Origami((1,2,3), (1,2)(3,4,5), 5), Origami((1,2,3), (2,3,4,5), 5),
  Origami((1,2,3), (2,4,5,3), 5), Origami((1,2)(3,4,5), (2,3), 5),
  Origami((1,2)(3,4,5), (1,2,3), 5), Origami((1,2)(3,4,5), (2,3,4,5), 5),
  Origami((1,2)(3,4,5), (2,3,5,4), 5), Origami((1,2)(3,4,5), (1,2,3,4,5), 5),
  Origami((1,2)(3,4,5), (1,2,3,5,4), 5), Origami((1,2,3,4), (3,4,5), 5),
  Origami((1,2,3,4), (3,5,4), 5), Origami((1,2,3,4), (1,2,3)(4,5), 5),
  Origami((1,2,3,4), (1,3,2)(4,5), 5), Origami((1,2,3,4), (2,3,4,5), 5),
  Origami((1,2,3,4), (2,5,4,3), 5), Origami((1,2,3,4), (1,2,4,5,3), 5),
  Origami((1,2,3,4), (1,3,5,4,2), 5), Origami((1,2,3,4,5), (3,5), 5),
  Origami((1,2,3,4,5), (1,2)(3,4,5), 5), Origami((1,2,3,4,5), (1,2)(3,5,4), 5)
    , Origami((1,2,3,4,5), (2,4,3,5), 5), Origami((1,2,3,4,5), (2,5,3,4), 5) ]
```

Chapter 3

SageMath functions

3.1 Using SageMath functions

The SageMath Package surface_dynamics from Vincent Delecroix [VD] provides functions to study origamis. To use the functions of this chapter, Sage and the surface_dynamics package must be installed on your operation system. Furthermore, the GAP packages homalgto, io_forhomalg, io and rings must be installed.
The following descriptions and examples are mainly taken from the manual of [VD].

3.1.1 SageMath

▷ `VeechgroupBySage(`*origami*`)` (function)

Returns: A ModularSubGroup.

This function executes the SageMath method veech_group to *origami* and returns its result as GAP object. It does the same as `VeechGroup` (2.3.1).

Example

```
gap> VeechgroupBySage(Origami((1,3,5), (1,3)(2,4,5) ));
<modular subgroup of index 24>
```

▷ `NormalFormBySage(`*origami*`)` (function)

Returns: An Origami.

This function executes the SageMath method to_standard_form to *origami* and returns its result as GAP object. In principle, it calculates a canonical representative by changing the numbering of the squares like the function `OrigamiNormalForm` (2.1.1). But it chooses another representation.

Example

```
gap> NormalFormBySage(Origami((1,3,5), (1,3)(2,4,5) ));
Origami((3,4,5), (1,2,3)(4,5), 5)
```

▷ `IsHyperellipticBySage(`*origami*`)` (function)

Returns: A boolean.

This function executes the SageMath method IsHyperelliptic to *origami* and returns its result as GAP object. It tests, weather *origami* is hyperelliptic.

Example
```
gap> IsHyperellipticBySage(Origami((1,3), (1,2) ));
true
```

▷ IsPrimitiveBySage(*origami*) (function)

Returns: A boolean.

This function executes the SageMath method IsPrimitive to *origami* and returns its result as GAP object. An origami is primitive if it does not cover an other origami. An origami is primitive if the action of the monodromy group has no non trivial block.

Example
```
gap> IsPrimitiveBySage(Origami((1,3), (1,2) ));
true
gap> IsPrimitiveBySage(Origami((1,2)(3,4), (1,3,5,6)(2,4) ));
false
```

▷ ReduceBySage(*origami*) (function)

Returns: An Origami.

This function executes the SageMath method Reduce to *origami* and returns its result as GAP object. It returns a reduced origami isomorphic (up to $SL(2,\mathbb{Q})$ action) to that origami.

Example
```
gap> ReduceBySage(Origami((1,2)(3,4), (1,3,5,6)(2,4) ));
Origami((1,2), (1,3), 6)
```

▷ AbsolutePeriodGeneratorsBySage(*origami*) (function)

Returns: A list of lists of Ints.

This function executes the SageMath method AbsolutePeriodGenerators to *origami* and returns its result as GAP object. It return a generating set of the absolute periods of this origami. To each curve on an origami, we can associate its holonomy (that is an element of $\mathbb{Z}\times\mathbb{Z}$). This function returns a generating set of the module generated by holonomies of closed curves.

Example
```
gap> AbsolutePeriodGeneratorsBySage(Origami((1,2,3,4)(5,6), (1,5)(2,6) ));
[ [ 2, 0 ], [ 2, 0 ], [ 0, 1 ], [ 0, 1 ] ]
```

▷ LatticeOfAbsolutePeriodsBySage(*origami*) (function)

Returns: A list of lists of Ints.

This function executes the SageMath method LatticeOfAbsolutePeriods to *origami* and returns its result as GAP object. It returns (a,t,u) where ((a,0),(t,u)) is a standard basis for the lattice of the absolute periods of self. The lattice of periods of an origami is the sublattice of $\mathbb{Z}^2$ generated by the holonomy vectors of its saddle connections. Any sublattice of $\mathbb{Z}^2$ has a standard basis consisting of a horizontal vector (a,0) and a nonhorizontal vector (t,u), where a, t, u are integers satisfying $0 < t < a$ and $0 < t$.

Example
```
gap> LatticeOfAbsolutePeriodsBySage(Origami((1,2)(3,4), (2,3) ));
[ [ 2, 0 ], [ 0, 1 ] ]
```

▷ OptimalDegreeBySage(*origami*) (function)

Returns: An Int.

This function executes the SageMath method OptimalDegree to *origami* and returns its result as GAP object. The optimal degree of self is the degree of the map to the largest torus. Any origami $X \to T$ factor as $i \circ \pi_{opt}$ where i is an isogeny. The optimal degree is the degree of π_{opt}.

Example

```
gap> OptimalDegreeBySage(Origami((1,2)(3,4), (2,3) ));
2
```

▷ PeriodGeneratorsBySage(*origami*) (function)

Returns: A list of lists of Ints.

This function executes the SageMath method PeriodGenerators to *origami* and returns its result as GAP object. It return a list of periods that generate the lattice of periods.

Example

```
gap> PeriodGeneratorsBySage(Origami((1,3,6)(2,5,7)(4),(1,2,4,3,5,6,7) ));
[ [ 2, 0 ], [ 1, 0 ], [ 1, 0 ], [ 0, 2 ], [ -1, 2 ], [ 0, 1 ] ]
```

▷ WidthsAndHeightsBySage(*origami*) (function)

Returns: A list of lists of Ints.

This function executes the SageMath method WidthsAndHeights to *origami* and returns its result as GAP object. It return the list of widths and heigths of cylinder.

Example

```
gap> WidthsAndHeightsBySage(Origami((1,2)(3,4),(1,3,5,6)(2,4) ));
[ [ 1, 2 ], [ 2, 2 ] ]
```

▷ SumOfLyapunovExponentsBySage(*origami*) (function)

Returns: A rational number.

This function executes the SageMath method SumOfLyapunovExponents to *origami* and returns its result as GAP object. It returns the sum of Lyapunov exponents for this origami.

Example

```
gap> SumOfLyapunovExponentsBySage(Origami((1,2)(3,4),(2,3) ));
3/2
```

▷ IntermediateCoversBySage(*origami*) (function)

This function executes the SageMath method NormalFormBySage to *origami* and returns its result as GAP object. It return the list of intermediate covers of this origami. Not implemented

References

[HL06] Pascal Hubert and Samuel Lelièvre. Prime arithmetic Teichmüller discs in $H(2)$. *Israel J. Math.*, 151:281–321, 2006. 4

[McK] Brendan McKay. The simultaneous conjugacy problem in the symmetric group s_n. 5

[Ran17] Anja Randecker. Skript zur Vortragsreihe Unendliche Translationsflächen, 2017. 4

[Sch04] Gabriela Schmithüsen. An algorithm for finding the Veech group of an origami. *Experiment. Math.*, 13(4):459–472, 2004. 7

[Sch05] Gabriela Schmithüsen. *Veech Groups of Origamis*. PhD thesis, University Karlsruhe, 2005. 4

[VD] Samuel Lelièvre Vincent Delecroix, Charles Fougeron. surface_dynamics sagemath package version 0.4.0. `https://pypi.org/project/surface_dynamics/`. 10

[Zmi11] David Zmiaikou. *Origamis and permutation groups*. PhD thesis, University Paris-Sud, 2011. 4

[Zor06] Anton Zorich. Flat surfaces. In *Frontiers in number theory, physics, and geometry. I*, pages 437–583. Springer, Berlin, 2006. 4