ADMCYCLES - A SAGE PACKAGE FOR COMPUTATIONS WITH TAUTOLOGICAL CLASSES AND ADMISSIBLE COVER CYCLES

JOHANNES SCHMITT AND JASON VAN ZELM

The tautological ring $RH^*(\overline{\mathcal{M}}_{g,n}) \subset H^*(\overline{\mathcal{M}}_{g,n})$ of the moduli space of stable curves has been studied extensively in the last decades. It has an explicit additive set of generators and a large (conjecturally complete) set of relations between these generators. Many operations in this ring (like intersection products, pullbacks by boundary morphisms of $\overline{\mathcal{M}}_{g,n}$, etc.) admit complete, combinatorial descriptions in terms of the generators.

Given geometrically defined loci in $\overline{\mathcal{M}}_{g,n}$, their fundamental class is often (though not always) contained in the tautological ring. One example are loci of admissible covers (such as the locus of hyperelliptic or bielliptic curves), which are tautological in many cases (for low g and n).

In the Sage program `admcycles.sage` we implement natural operations in the tautological ring, together with a function identifying formulas for many cycles of admissible covers in terms of the generating set of $RH^*(\overline{\mathcal{M}}_{g,n})$. This identification works by computing intersection numbers of admissible cover cycles with tautological classes and by pulling the cycles back via boundary morphisms. Details will be given in a forthcoming paper. This document serves as a user manual for `admcycles.sage`.

We are indebted to Aaron Pixton for letting us use and modify a previous implementation of operations in the tautological ring by him. The two main functions of his program that we use are the computation of intersection numbers of κ and ψ classes and a function computing the generalized Faber-Zagier relations between the generators of the tautological ring. For details on the functions that we use see Appendix A.

1. Features of the program

1.1. Data types.

- stable graphs
- tautological classes (on products of moduli spaces of stable curves)
- admissible cover cycles (i.e. loci of curves $(C, p_1, \dots, p_r)$ admitting G-Galois covers $C \to D$ with ramification points $p_1, \dots, p_r$, for fixed genera, finite group[1] G and ramification data)

Date: November 26, 2018.

[1]Currently, most functions are only implemented for finite cyclic groups G.

1.2. **Supported operations.**

- intersecting tautological classes (Section 3.1), computing degrees of tautological zero-cycles (Section 3.2)
- computing a basis of the tautological ring $\mathrm{RH}^{2d}(\overline{\mathcal{M}}_{g,n})$ and writing tautological classes in terms of this basis[2] (Section 3.3)
- computing pullbacks and pushforwards of tautological classes under gluing morphism associated to a stable graph (Section 3.4) and forgetful morphisms (Section 3.2)
- identifying admissible cover cycles in terms of tautological cycles (Section 4)

1.3. **Background computations[3].**

- handling tautological classes on products of moduli spaces of stable curves, computing decomposition of diagonals
- intersecting admissible cover cycles with tautological cycles, pulling them back to the boundary

2. Getting started

Open a command line, go to a directory containing the files `admcycles.sage` and `DR.py` and start sage. Then type

```
load("admcycles.sage")
```

The file `examples.sage` contains several sample computations (with comments), which can be copied and pasted into the command line. We will describe these computations in more detail in the following.

3. Tautological classes

3.1. **Creating tautological classes.** There are different ways to enter tautological classes in the program and depending on the example, some are more convenient than others.

For boundary divisors as well as ψ, κ and λ-classes, there are predefined functions.

- `sepbdiv(h,A,g,n)` gives the pushforward $\xi_*[\overline{\mathcal{M}}_\Gamma]$ of the boundary gluing map
$$\xi : \overline{\mathcal{M}}_\Gamma = \overline{\mathcal{M}}_{h,A\cup\{p\}} \times \overline{\mathcal{M}}_{g-h,(\{1,\dots,n\}\setminus A)\cup\{p'\}} \to \overline{\mathcal{M}}_{g,n},$$
where `A` can be a list, set or tuple of numbers from 1 to n. Be careful that tuples of length 1 must be entered as `(a,)` in Python, instead of `(a)`.

[2]This works in the case where Pixton's generalized Faber-Zagier relations give all relations between the generators of the tautological ring.

[3]These are operations that are implemented and run in the background of some of the other functions. Currently they are not yet optimized for user access, but once this is done, we will add a description here.

- `irrbdiv(g,n)` gives the pushforward $\xi_*[\overline{\mathcal{M}}_{g-1,n+2}]$ of the boundary gluing map
$$\xi : \overline{\mathcal{M}}_{g-1,n+2} \to \overline{\mathcal{M}}_{g,n}.$$
- `psiclass(i,g,n)` gives the class ψ_i on $\overline{\mathcal{M}}_{g,n}$, defined by
$$\pi : \overline{\mathcal{M}}_{g,n+1} \to \overline{\mathcal{M}}_{g,n}, \sigma_i : \overline{\mathcal{M}}_{g,n} \to \overline{\mathcal{M}}_{g,n+1}, \psi_i = c_1(\sigma_i^*\omega_\pi).$$
- `kappaclass(a,g,n)` gives the (Arbarello-Cornalba) κ-class κ_a on $\overline{\mathcal{M}}_{g,n}$, defined by
$$\kappa_a = \pi_*\psi_{n+1}^{a+1}.$$
- `lambdaclass(d,g,n)` gives the class λ_d on $\overline{\mathcal{M}}_{g,n}$, defined as the d-th Chern class $\lambda_d = c_d(\mathbb{E})$ of the Hodge bundle $\mathbb{E}$.

When working with a fixed space $\overline{\mathcal{M}}_{g,n}$, it is furthermore possible to specify the desired value of the global variables `g` and `n` to avoid giving them as an argument each time.

These tautological classes can be combined in the usual way by operations `+`, `-`, `*` and raising to a power `^`.

```
sage: t1=3*sepbdiv(1,(1,2),3,4)-psiclass(4,3,4)^2
sage: g=2;n=1;
sage: t2=-1/3*irrbdiv()*lambdaclass(1)
```

To enter more complicated classes coming from decorated boundary strata, it is often convenient to first list all such decorated strata of a certain degree and then select the desired ones from the list. To give a list of all generators of $R^r(\overline{\mathcal{M}}_{g,n})$ use the function `list_tautgens(g,n,r)`. The list itself is created by `tautgens(g,n,r)`, from which one can then select the classes.

```
sage: list_tautgens(2,0,2)
[0] : Graph :      [2] [[]] []
Polynomial : 1*(kappa_2^1 )_0
[1] : Graph :      [2] [[]] []
Polynomial : 1*(kappa_1^2 )_0
[2] : Graph :      [1, 1] [[2], [3]] [(2, 3)]
Polynomial : 1*(kappa_1^1 )_0
[3] : Graph :      [1, 1] [[2], [3]] [(2, 3)]
Polynomial : 1*psi_2^1
[4] : Graph :      [1] [[2, 3]] [(2, 3)]
Polynomial : 1*(kappa_1^1 )_0
[5] : Graph :      [1] [[2, 3]] [(2, 3)]
Polynomial : 1*psi_2^1
[6] : Graph :      [0, 1] [[3, 4, 5], [6]] [(3, 4), (5, 6)]
Polynomial : 1*
[7] : Graph :      [0] [[3, 4, 5, 6]] [(3, 4), (5, 6)]
Polynomial : 1*
```

```
sage: L=tautgens(2,0,2);
sage: t3=2*L[3]+L[4]
sage: t3

Graph :      [1, 1] [[2], [3]] [(2, 3)]
Polynomial : 2*psi_2^1

Graph :      [1] [[2, 3]] [(2, 3)]
Polynomial : 1*(kappa_1^1 )_0
```

The output above should be interpreted as follows: each `tautclass` consists of a sum of decorated boundary strata (represented by data type `decstratum`), which consist of a graph (datatype `stgraph`) and a polynomial in κ and ψ-classes (datatype `kppoly`).

Note: For decorated stratum classes, we have the convention of *not* dividing by the order of the automorphism group of the stable graph. Thus the elements of the list `L` above are really just pushforwards of κ and ψ classes on products of moduli spaces $\overline{\mathcal{M}}_{g,n}$ under appropriate gluing maps.

Let us look at the example of generator `L[3]` above.

```
[3] : Graph :      [1, 1] [[2], [3]] [(2, 3)]
Polynomial : 1*psi_2^1
```

Its stable graph is represented by three lists.

(1) The first list `[1, 1]` are the genera of the vertices, so there are two vertices, both of genus 1. Note that vertices are numbered by $0, 1, 2, \ldots$, so in the above case, the vertices are numbers 0 and 1.
(2) the second list gives the legs (that is markings or half-edges) attached to the vertices, so vertex 0 carries the half-edge 2 and vertex 1 the half-edge 3,
(3) the third list gives the edges, that is half-edge pairs that are connected; in the above case, the two half-edges 2 and 3 form an edge, connecting the two vertices

If we wanted to enter this `stgraph` manually, we could use its constructor as follows:

```
sage: stgraph([1,1],[[2],[3]],[(2,3)])
[1, 1] [[2], [3]] [(2, 3)]
```

The polynomial in κ and ψ is `1*psi_2^1` in this case, so the half-edge 2 on the second vertex carries a ψ-class. For the generator `L[4]` the polynomial looks like `1*(kappa_1^1 )_0`, meaning that vertex 0 carries a class $\kappa_1^1 = \kappa_1$.

Finally, it is possible to manually enter tautological classes by using the constructors of the classes `tautclass`, `decstratum` and so on. We refer to comments in the source code `admcycles.sage` for details on this.

3.2. **Basic operations.** Apart from the usual arithmetic operations, we can take forgetful pushforwards and pullbacks of tautological classes and also compute the degree of tautological zero-cycles. In particular, we can compute intersection numbers. Below, for the forgetful map $\pi : \overline{\mathcal{M}}_{1,3} \to \overline{\mathcal{M}}_{1,2}$ forgetting the marking 3 we verify the relations

$$\pi_*\psi_3^2 = \kappa_1, \pi^*\psi_2 = \psi_2 - D_{0,\{2,3\}},$$

and we compute the intersection number

$$\langle \tau_0\tau_1\tau_2 \rangle_{1,3} = \int_{\overline{\mathcal{M}}_{1,3}} \psi_1^0\psi_2\psi_3^2 = 1/12.$$

We check that it agrees with the prediction $\langle \tau_0\tau_1\tau_2 \rangle_{1,3} = \langle \tau_0\tau_2 \rangle_{1,2} + \langle \tau_1^2 \rangle_{1,2}$ by the string equation.

```
sage: s1=psiclass(3,1,3)^2
sage: s1.forgetful_pushforward([3])

Graph :      [1] [[1, 2]] []
Polynomial : 1*(kappa_1^1 )_0
sage: s2=psiclass(2,1,2)
sage: s2.forgetful_pullback([3])

Graph :      [1] [[1, 2, 3]] []
Polynomial : 1*psi_2^1

Graph :      [1, 0] [[1, 4], [5, 3, 2]] [(4, 5)]
Polynomial : -1*
sage: s3=psiclass(2,1,3)*psiclass(3,1,3)^2
sage: s3.evaluate()
1/12
sage: s4=psiclass(2,1,2)^2+psiclass(1,1,2)*psiclass(2,1,2)
sage: s4.evaluate()
1/12
```

3.3. **A basis of the tautological ring and tautological relations.** The function `generating_indices(g,n,r)` computes the indices (with respect to Pixton's generating set) of a basis of $R^r(\overline{\mathcal{M}}_{g,n})$, assuming that the generalized Faber-Zagier relations between the additive generators give a complete set of relations between them. The function `Tautvecttobasis` converts a vector with respect to the whole generating set into a vector in this basis. The function `tautclass.toTautbasis(g,n,r)` converts a `tautclass` into such a vector.

Continuing the example from Section 3.1 we see:

```
sage: generating_indices(2,0,2)
[0, 1]
sage: t3.toTautbasis(2,0,2)
(-48, 22)
```

This means that generators `L[0]`, `L[1]` form a basis of $R^2(\overline{\mathcal{M}}_2)$ and the tautclass `t3=2*L[3]+L[4]` is equivalent to `-48*L[0]+22*L[1]`.

We can also directly verify tautological relations using the built-in function `is_zero` of `tautclass`. Below we verify the known relation $\kappa = \psi - \delta_0 \in R^1(\overline{\mathcal{M}}_{1,n})$ for $n = 4$. Here ψ is the sum of all ψ_i and δ_0 is the sum of all separating boundary divisors, i.e. those having a genus 0 component. For this, we list all stable graphs with one edge via `list_strata(g,n,1)`. We exclude the graph `gamma` with a self-loop by requiring that the number of vertices `gamma.numvert()` is at least 2. Then we can convert these graphs to tautclases by using `to_tautclass`.

```
sage: g=1;n=4;
sage: bgraphs=[bd for bd in list_strata(g,n,1) if bd.numvert()>1]
sage: del0=sum([bd.to_tautclass() for bd in bgraphs])
sage: psisum=sum([psiclass(i) for i in range(1,n+1)])
sage: rel=kappaclass(1)-psisum+del0
sage: rel.is_zero()
True
```

In practice, much of the time in some computations is spent on calculating generalized Faber-Zagier relations between tautological cycles on $\overline{\mathcal{M}}_{g,n}$. However, once computed, the relations can be saved to a file and be reloaded in a later session using the functions `save_FZrels()` and `load_FZrels()`. Careful: the function `save_FZrels()` creates (and overwrites previous version of) a file `geninddb.pkl` which depending on the previous computations can be quite large.

3.4. **Pulling back tautological classes to the boundary.** Below we pull back a generator of $R^2(\overline{\mathcal{M}}_4)$ to the boundary divisor with genus partition $4 = 2 + 2$. This produces an element of type `prodtautclass`, a tautological class on a product of moduli spaces, in this case $\overline{\mathcal{M}}_{2,1} \times \overline{\mathcal{M}}_{2,1}$. Two elements on the same product of spaces can be added and multiplied and further operations like pushforwards under (partial) gluing maps are supported. We will describe this in more detail in a later version of this manual (but some explanation is already provided in comments in the code of `admcycles.sage`).

In our case, we want to express the pullback to $\overline{\mathcal{M}}_{2,1} \times \overline{\mathcal{M}}_{2,1}$ in terms of a basis of $H^2(\overline{\mathcal{M}}_{2,1} \times \overline{\mathcal{M}}_{2,1})$ obtained from the preferred bases of the factors $H^*(\overline{\mathcal{M}}_{2,1})$ given by `generating_indices`. We can either represent the result as a list of matrices (giving the coefficients in the tensor product bases) or as a combined vector (using the option `vecout=true`).

```
sage: bdry=stgraph([2,2],[[1],[2]],[(1,2)])
sage: generator=tautgens(4,0,2)[3]; generator
Graph :      [1, 3] [[2], [3]] [(2, 3)]
Polynomial : 1*psi_3^1
sage: pullback=bdry.boundary_pullback(generator);
sage: pullback.totensorTautbasis(2)
[
                           [-3]
                           [ 1]
                 [0 0 0]   [-3]
                 [0 0 0]   [ 7]
[-3  1 -3  7  1], [0 0 0], [ 1]
]
sage: pullback.totensorTautbasis(2,vecout=true)
(-3, 1, -3, 7, 1, 0, 0, 0, 0, 0, 0, 0, 0, -3, 1, -3, 7, 1)
```

4. Admissible cover cycles

4.1. **Shortcut: Hyperelliptic and bielliptic cycles.** Before we go into details of how to specify general admissible cover cycles, let us mention the important cases of hyperelliptic and bielliptic cycles.

Given $g \geq 0$, $0 \leq n \leq 2g+2$ and $m \geq 0$ with $2g-2+n+2m > 0$, we have the locus $\overline{H}_{g,n,2m} \subset \overline{\mathcal{M}}_{g,n+2m}$ of curves $(C, p_1, \ldots, p_n, q_1, q_1', \ldots, q_m, q_m')$ such that C is hyperelliptic with $p_1, \ldots, p_n$ fixed points of the hyperelliptic involution and the pairs q_i, q_i' being exchanged by this involution. An analogous definition gives the locus $\overline{B}_{g,n,2m} \subset \overline{\mathcal{M}}_{g,n+2m}$ of bielliptic curves with n fixed points and m pairs of points forming orbits under the bielliptic involution.

Then the fundamental class of the (reduced) loci $\overline{H}_{g,n,2m}$ and $\overline{B}_{g,n,2m}$ can (in many cases) be computed by the functions `Hyperell(g,n,m)` and `Biell(g,n,m)` of our program.

As an example, we compute the class $[\overline{H}_3] \in A^1(\overline{\mathcal{M}}_3)$ and verify that we obtain the known result

$$[\overline{H}_3] = 9\lambda - \delta_0 - 3\delta_1,$$

where δ_0 is the class of the divisor of irreducible nodal curves and δ_1 is the divisor of curves with a separating node between a genus 1 and a genus 2 component.

```
sage: H=Hyperell(3,0,0)
sage: H.toTautbasis()
(3/4, -9/4, -1/8)
sage: g=3; n=0;
sage: H2=9*lambdaclass(1)-(1/2)*irrbdiv()-3*sepbdiv(1,())
sage: H2.toTautbasis()
(3/4, -9/4, -1/8)
```

Here we need to divide `irrbdiv()` by two, the degree of the corresponding gluing map.

4.2. **Creating and identifying admissible cover cycles.** In general, an admissible cover cycle is specified by a genus, a group as well as monodromy data. Currently, intersections are only implemented for cyclic groups. Below we will study bielliptic curves in genus 2, which are double covers of elliptic curves branched over two points. As a first step we enter the ramification data.

```
sage: G=PermutationGroup([(1,2)])
sage: list(G)
[(), (1,2)]
sage: H=HurData(G,[G[1],G[1]])
```

The function `HurData` takes the group G as the first argument and as the second a list of group elements $\alpha \in G$, each of which corresponds to the G-orbit of some marking $p \in C$. Here α is a generator of the stabilizer of p under the group action $G \curvearrowright C$, which gives the monodromy around p. In other words, the natural action of the stabilizer $G_p = \langle \alpha \rangle$ on a tangent vector $v \in T_pC$ is given by

$$\alpha.v = \exp(2\pi i/\mathrm{ord}(h))v.$$

Thus in the example above, we have two markings, both with stabilizer generated by `G[1]=(1,2)` which acts by multiplication of -1 on the tangent space.

To identify the admissible cover cycle (inside the moduli space $\overline{\mathcal{M}}_{g,n}$ with n the total number of marked points from the ramification data) in terms of tautological classes, one can use the function `Hidentify`.

It pulls back the admissible cover cycle to all boundary divisors and (recursively) identifies the pullback itself in terms of tautological classes. It compares this pullback to the pullback of a basis of the tautological ring. Often this pullback map is injective in cohomology such that one can then write the admissible cover cycle in terms of the basis using linear algebra. Sometimes, it is necessary to additionally intersect with some monomials in κ and ψ-classes.

To apply `Hidentify` one gives the genus and the monodromy data as arguments. The standard output format is an instance of the class `tautclass`. For those users familiar with Aaron Pixton's implementation of the tautological ring, there is the option `vecout=true` which returns instead a vector with respect to the generating set of the tautological ring provided by this program.

```
sage: vbeta=Hidentify(2,H,vecout=true)
sage: vector(vbeta)
(517/4, -33, 11/4, 243/4, -125/4, 15/2, 41/4, 125, 99/4, -41,
   -1137/4, -285/4, 0, 0, 0, 0, 0, 0, -57/8, -3/8, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0)
```

The output above means specifically, that inside $\overline{\mathcal{M}}_{2,2}$ the locus of bielliptic curves with the two points fixed by the involution being the marked points is given (in the generating set `gens=all_strata(2,3,(1,2))` produced by Pixton's program) as $517/4 \cdot$ `gens[0]` $- 33 \cdot$ `gens[1]` $+ \ldots$.

If we instead wanted to have bielliptic curves with two marked fixed-points of the involution and one pair of markings that are exchanged by the involution, we would need to use the monodromy data

```
sage: H2=HurData(G,[G[1],G[1],G[0]])
```

in which case `Hidentify(2,H2)` would live inside $R^4(\overline{\mathcal{M}}_{2,4})$.

If we only want to remember a subset of the markings, we can use the optional parameter `marking` to give this subset. For instance, the command `Hidentify(2,H,markings=[])` would give the pushforward of `Hidentify(2,H)` in $\overline{\mathcal{M}}_{2,2}$ to the space $\overline{\mathcal{M}}_2$ under the forgetful morphism (see also Section 4.3).

4.3. **Example: Specifying and Identifying $[\overline{B}_2]$ by hand.** The locus $\overline{B}_2 \subset \overline{\mathcal{M}}_2$ of bielliptic curves is a divisor. A bielliptic genus 2 curve is ramified over two points. In the following we want to use the methods of the previous section to identify its cycle class.

Now when treating admissible cover cycles in general, our program a priori handles the cycle where all possible ramification points are marked. In this case, this is the cycle $[\overline{B}_{2,2,0}] \in R^3(\overline{\mathcal{M}}_{2,2})$ of bielliptic curves C with the two ramification points p_1, p_2 marked. By specifying `markings=[]` when calling `Hidentify`, we tell it to remember none of the markings, in other words to push forward under the map $\pi : \overline{\mathcal{M}}_{2,2} \to \overline{\mathcal{M}}_2$ forgetting the markings.

```
sage: G=PermutationGroup([(1,2)])
sage: H=HurData(G,[G[1],G[1]])
sage: Biell=Hidentify(2,H,markings=[])
sage: Biell.toTautbasis(2,0,1)
(30, -9)
```

We want to compare the result with the known formulas for $[\overline{B}_2]$. For δ_0 the class of the irreducible boundary of $\overline{\mathcal{M}}_2$ and δ_1 the class of the boundary

divisor with genus-splitting $(1,1)$, it is known that $[\overline{B}_2] = \frac{3}{2}\delta_0 + 6\delta_1$. If we want to enter this combination of δ_0 and δ_1, we have to be careful about conventions, though: the corresponding gluing maps $\xi : \overline{\mathcal{M}}_{1,2} \to \overline{\mathcal{M}}_2$ and $\xi' : \overline{\mathcal{M}}_{1,1} \times \overline{\mathcal{M}}_{1,1} \to \overline{\mathcal{M}}_2$ both have degree 2. This corresponds to the fact that the associated stable graphs both have an automorphism group of order 2. Hence we have to divide by a factor of two and obtain:

```
sage: g=2;n=0
sage: Biell2=3/4*irrbdiv()+ 3*sepbdiv(1,())
sage: Biell2.toTautbasis(2,0,1)
(15/2, -9/4)
```

We see that up to a factor of 4 the two vectors `(30, -9)` and `(15/2, -9/4)` agree. Where does this factor come from?

For this recall that the cycle `Biell` above is equal to $\pi_*[\overline{B}_{2,2,0}]$. Since for the generic bielliptic curve C there are two choices of orderings for marking p_1, p_2, this explains a factor of 2. On the other hand, the hyperelliptic involution $\sigma : C \to C$ on C exchanges p_1 and p_2. Thus $\sigma \in \mathrm{Aut}(C)$, but $\sigma \notin \mathrm{Aut}(C, p_1, p_2)$. This missing automorphism factor explains another factor of 2 in the pushforward under π, so in fact $[\overline{B}_2] = \frac{1}{4}\pi_*[\overline{B}_{2,2,0}]$.

Note that since the cycles of bielliptic loci are implemented via the function `Biell`, we could have taken a shortcut above.

```
sage: B=Biell(2,0,0); B.toTautbasis()
(15/2, -9/4)
```

As an application, we can check the Hurwitz-Hodge integral

$$\int_{[\overline{B}_{2,2,0}]} \lambda_2\lambda_0 = \int_{\pi_*[\overline{B}_{2,2,0}]} \lambda_2 = \frac{1}{48}$$

predicted by [JPT11].

```
sage: (Biell*lambdaclass(2,2,0)).evaluate()
1/48
```

In principle the corresponding integrals for $g = 3, 4$ can also be verified like this, but the amount of time and memory needed grows drastically.

Appendix A. Functions from Aaron Pixton's program

Many functions of our program rely in essential ways on a previous implementation by Aaron Pixton. It is imported in the form of the precompiled file `DRpython.pyc`. The original code (slightly modified by us) can be downloaded from our homepage.

Below we give an overview which functions we use.

- `all_strata`/ `num_strata` - This computes a list/the number of all decorated stratum classes in a given degree on $\overline{\mathcal{M}}_{g,n}$. For the sake of compatibility, we chose to use the same order in our own program.
- `FZ_matrix` - Computes the generalized Faber-Zagier relations between the decorated stratum classes in a given degree. We use this to identify tautological classes in terms of a basis and to verify tautological relations.
- `socle_formula` - Evaluates the integral of a monomial in κ and ψ-classes. We use this when computing intersection numbers, both for tautological classes and for the intersection of a tautological class with an admissible cover cycle.
- `pairing_submatrix` - Computes a submatrix of the pairing matrix between complementary degrees in the tautological ring. We implemented our own version of the intersection in the tautological ring, so using `socle_formula` we could compute these numbers ourselves. However, for small g, n the implementation by Pixton is faster, so currently we use his version of the function.
- We use a few more functions for converting tautological classes and relations from Pixton's program to our own.

References

[JPT11] P. Johnson, R. Pandharipande, and H.-H. Tseng. Abelian Hurwitz-Hodge integrals. *Michigan Math. J.*, 60(1):171–198, 2011.

Departement Mathematik, ETH Zürich, Rämistrasse 101, 8092 Zürich, Switzerland

E-mail address: `johannes.schmitt@math.ethz.ch`

Deparment of Mathematics, Humboldt-Universität zu Berlin, Rudower Chaussee 25, Room 1.415, 12489 Berlin, Germany

E-mail address: `jasonvanzelm@outlook.com`