diffstrata—A Sage Package for Calculations in the Tautological Ring of the Moduli Space of Abelian Differentials

Matteo Costantini\textsuperscript{a}, Martin Möller\textsuperscript{b}, and Jonathan Zachhuber\textsuperscript{b}

\textsuperscript{a}Universität Duisburg-Essen Fakultät für Mathematik, Essen, Germany; \textsuperscript{b}Institut für Mathematik, Goethe-Universität Frankfurt, Frankfurt am Main, Germany

\textbf{ABSTRACT}

The boundary of the multi-scale differential compactification of strata of abelian differentials admits an explicit combinatorial description. However, even for low-dimensional strata, the complexity of the boundary requires use of a computer. We give a description of the algorithms implemented in the SageMath package \texttt{diffstrata} to enumerate the boundary components and perform intersection theory on this space. In particular, the package can compute the Euler characteristic of strata using the methods developed by the same authors.

\textbf{KEYWORDS}

Strata of abelian differentials; intersection theory; sage package

\section{Introduction}

An important tool in studying the moduli space of (meromorphic) abelian differentials $\mathbb{P}\Omega \mathcal{M}_g(\mu)$ is its modular compactification, the space of multi-scale differentials $\mathbb{P}\Xi \mathcal{M}_{g,n}(\mu)$ for $\mu$ an integer partition of $2g-2$. The boundary is of a combinatorial nature, parameterized, for any $\mu$, by finitely many labeled level graphs \cite{3}. However, already listing isomorphism classes of these graphs is a non-trivial task, and even for $g = 3$ the number of components becomes so large that just listing them is unfeasible by hand.

Moreover, in \cite{6} the tautological ring of $\mathbb{P}\Xi \mathcal{M}_{g,n}(\mu)$ is described and calculations therein may be expressed purely in terms of the combinatorics of the boundary. Again, even the simplest calculations are extremely cumbersome to perform without the assistance of a computer.

The \texttt{diffstrata} package provides a framework for calculations in the tautological ring of $\mathbb{P}\Xi \mathcal{M}_{g,n}(\mu)$. It is implemented in \texttt{sage} \cite{17} and it is inspired by the package \texttt{admcycles} \cite{8} for calculations in the tautological ring of $\mathcal{M}_{g,n}$. However, due to the differences in the structure of the boundary, the implementation and interface of the two packages have only little in common. A key step in the evaluation process is performed by \texttt{admcycles}, though.

\subsection{Applications}

First, \texttt{diffstrata} may be used naively for basic inquiries about strata and the compactification constructed in \cite{3}.

\begin{verbatim}
sage: from admcycles.diffstrata import *
sage: X=Stratum((2,2))
sage: X.info()
Stratum: (2, 2) with residue conditions: []
Genus: [3]
Dimension: 6
Boundary Graphs (without horizontal edges):
Codimension 0: 1 graph
Codimension 1: 20 graphs
Codimension 2: 86 graphs
Codimension 3: 147 graphs
Codimension 4: 110 graphs
Codimension 5: 30 graphs
Total graphs: 394
\end{verbatim}

Second, several important invariants of strata can be computed via intersection theory. Recall from \cite[Theorem 1.3]{6} that the (orbifold) Euler characteristic of the (projectivized, but not compactified) moduli space $B = \mathbb{P}\Omega \mathcal{M}_{g,n}(\mu)$ is the dimension-weighted
sum over all level graphs \( \Gamma \in LG_{L}(B) \) without horizontal nodes

\[
\chi(B) = (-1)^d \sum_{L=0}^{d} \sum_{\Gamma \in LG_{L}(B)} \xi_{\Gamma} N_{i}^{\Gamma} \sum_{i=0}^{L} \prod_{\ell=0}^{\Gamma} (\xi_{\ell}^{i})^{d_{i+1}}
\]

of the product of the top power of the of the first Chern class \( \xi_{\ell}^{i} \) of the tautological bundle at each level, where \( d_{i+1} \) is the dimension of the projectivized moduli space at level \( i \) of the boundary stratum \( D_{\ell} \), and where \( d = \dim(B) = N - 1 \). The following example contains the complete code required to calculate the (orbifold) Euler characteristic of \( \mathbb{P} \Omega \mathcal{M}_{2}(2) \):

```python
sage: from admcycles.diffstrata import *
sage: X=Stratum((2,1))
sage: X.euler_characteristic()
-1/40
```

Another example for such a quantity is the Masur–Veech volume, that, according to [5], can be written as

\[
\text{vol}(\Omega \mathcal{M}_{g,n}(m_{1}, \ldots, m_{n})) = -\frac{2(2\pi)^{2g}}{(2g - 3 + n)!} \int_{\mathbb{P}\mathcal{L} \Omega_{g,n}(\mu)} \xi_{2g-2} \cdot \prod_{i=1}^{n} \psi_{i}.
\]

In `diffstrata` the computation (without the prefactor) is accomplished by:

```python
sage: from admcycles.diffstrata import *
sage: X=Stratum((1,1))
sage: (X.xi^2 * X.psi(1) * X.psi(2)).evaluate()
-1/720
```

We remark that the recursive algorithms in [5] to evaluate Masur–Veech volume are much faster than the direct evaluation via intersection numbers. This is quite parallel to the evaluation of Weil-Petersson volumes as \( \kappa \)-integrals via `admcycles`, which is much slower than specialized recursions as e.g. in [13].

Third, and most important, the program should be used as development tool for testing formulas. Many topics well-studied for the moduli space of curves (such as the Picard group, relations in the tautological ring, divisor class computations, Kodaira dimension, ample cone) are hardly understood for \( \mathbb{P} \mathcal{L} \Omega \mathcal{M}_{g,n}(\mu) \) at the time of writing. E.g. any relation in the tautological ring of strata can immediately be tested for plausibility in `diffstrata` by intersecting with all classes of complementary dimension.

### 1.2. Algorithmic aspects

The new algorithmic concept of `diffstrata` is to work with (enhanced) profiles to encode boundary strata. We compare this to the situation of \( \mathcal{M}_{g,n}^{\ast} \) (and `admcycles`), where boundary strata are given by stable graphs. Their number grows rapidly with \( (g, n) \) and presents the main performance obstacle. The boundary strata of \( \mathbb{P} \mathcal{L} \Omega \mathcal{M}_{g,n}(\mu) \) are encoded by enhanced level graphs, whose precise definition is recalled in Section 2. Due to the level structure and enhancements, there are even more of these graphs. However, all the calculations of the Chern classes of the logarithmic cotangent bundle, in particular of the Euler characteristic ([6]), and also the computation of Masur-Veech volumes ([5]) happen in the tautological ring defined by clutching of non-horizontal divisors. The `diffstrata` package treats exclusively graphs without horizontal edges. The number of levels of an enhanced level graph (plus the number of horizontal edges) encodes the codimension of the boundary stratum defined by the graph, and the (enhanced) profile is a way of encoding which boundary divisors have to be intersected to define such locus. Using this method, we can avoid often to invoke expensive graph comparison algorithms.

Implementing the compactification \( \mathbb{P} \mathcal{L} \Omega \mathcal{M}_{g,n}(\mu) \) poses a series of challenges not encountered in the boundary of \( \mathcal{M}_{g,n} \). Constructing all level graphs satisfying precisely the conditions of Bainbridge et al. [3, Def. 1.1] is a nontrivial problem. First, checking the validity of a graph requires checking the Global Residue Condition (GRC) discovered in [2]. The corresponding algorithm is given in Proposition 2.2, which refines the combinatorial criterion of Möller et al. [14]. Second, while generating a codimension-one degeneration of a stable graph inside \( \mathcal{M}_{g,n} \) is straight-forward (add an edge either as a loop or by splitting a vertex subject to the stability condition), for a level graph this is the more subtle problem of splitting a level. To solve it, we must construct all divisors in any generalized stratum, i.e., parameterizing meromorphic differentials on possibly disconnected curves and with residue conditions, as these appear as levels already inside low-genus holomorphic strata. An efficient algorithm for this is given in Section 2.2. We usually refer to two-level graphs, which define all divisors considered here, for brevity as BICs (`bicoloured graphs`, as in [9]).

Having generated all BICs, this allows us to construct any graph by recursively clutching BICs (one for each level-crossing). As a by-product, this yields discrete coordinates for the enhanced level graphs, as any graph splits uniquely into a product of distinct BICs: we therefore number the BICs of a stratum and associate to each level graph its profile, the tuple of indices of BICs appearing as levels of this graph. Unfortunately, this is not always injective: profiles may be reducible and we need an enhanced profile to refer to a graph uniquely.
The tautological ring of $\overline{\mathcal{M}_{g,n}}(\mu)$ is abstractly defined as the smallest subring of the Chow ring that contains $\psi$-classes and is stable under push-forward of clutching maps and marked point forgetful maps. It is finitely generated by boundary strata decorated with products of $\psi$-classes on each of its levels, see [6, Thm. 1.5] for details. Performing computations in the tautological ring requires intersecting such decorated boundary strata and evaluating elements of dimension zero. Algorithms for these steps are given in Section 3.

The novel ingredient in $\text{diffstrata}$ is that all standard operations in the intersection ring, computations of normal bundles, pullbacks and the implementation of the excess intersection formula for $\overline{\mathcal{M}_g}$ and admcycles, since normal bundles of boundary components in $\overline{\mathcal{M}_g}$ are plain $\psi$-class expressions that do not need recursive treatment.

For implementing the multiplication of tautological classes the notion of profile is again key, since it allows us to efficiently determine which graphs appear as degenerations and is essential in the computation of normal bundles and general intersections, see Section 3.

In Section 4 we give a brief introduction to working with the package, including some sample calculations and a summary of the implemented algorithms for calculating the Euler characteristic. We end with an example illustrating the caching mechanisms of $\text{diffstrata}$ and giving a flavor of the complexity of the Euler characteristic calculation even for fairly small strata. We also describe some crosschecks for the algorithms of Section 3. More details on the subtleties and caveats of the implementation and caching, as well as an extended guide to the interface are given in the longer version of this text available as [7].

### 1.3. Performance

We present data about the time and memory consumption of some algorithms, in order to give to the reader an idea of what is possible to compute on a standard laptop. The growth of complexity is illustrated on the algorithm that generates all BICs of a stratum. Table 1 shows that the growth is roughly governed by the dimension of the stratum.

More relevant in practice is the construction of the whole boundary. This is significantly more time and memory expensive, since it is built by recursively using the BICs generation algorithm for every level graph. While for the minimal stratum $\mu = (4)$ in genus three this is instantaneous, we can already see that for the principal stratum $\mu = (1^4)$ the memory used is around 3.7 GB.

```sage
sage: before = get_memory_usage()
sage: %time len(flatten(Stratum((1,1,1,1)).all_graphs))
CPU times: user 3min 9s, sys: 4.68 s, total: 3min 13s
Wall time: 3min 16s
22976
sage: get_memory_usage(before)
3669.03515625
```

### 1.4. Open questions

For the moduli space of (pointed) curves $\overline{\mathcal{M}_{g,n}}$ enumerating the boundary strata, also known as tropical curves, and providing a tight estimate for their growth rate has been discussed at various places ([4], [11]), but we are aware of a complete solution only in genus zero ([12]). It seems interesting to address the analogous problem in $\mathbb{P}^\infty \overline{\mathcal{M}_{g,n}}(\mu)$, to count the number of boundary strata (see above for the case $\mu = (2,2)$) or at least provide tight estimates for their number. Such estimates would also be the basis for serious run-time analysis of the algorithms in $\text{diffstrata}$.\(^1\)

| $\mu$ | $\dim$ | $\#$BICs | memory | CPU time |
|-------|--------|----------|--------|---------|
| (6)   | 7      | 23       | 3 MB   | 0.20 s  |
| (1^4) | 8      | 102      | 17 MB  | 1.14 s  |
| (3,2,1)| 9     | 103      | 10 MB  | 2.01 s  |
| (2^3) | 9      | 194      | 69 MB  | 16.1 s  |
| (3,1^3)| 10    | 316      | 60 MB  | 16.0 s  |
| (2^2,1^2)| 10  | 374      | 144 MB | 24.5 s  |
| (2,1^4)| 11    | 1308     | 800 MB | 2 min 41 s |
| (1^6) | 12     | 5846     | 10711 MB | 2 h 13 s |

\(^1\)In practice, one runs out of memory before speed becomes a serious issue: Calculations in genus four take several hours but need several TB of RAM.
1.5. Acknowledgements

We thank Vincent Delecroix and Johannes Schmitt for many helpful suggestions for implementing this package. We are also grateful to the Mathematical Sciences Research Institute (MSRI, Berkeley) and the Hausdorff Institute for Mathematics (HIM, Bonn), where significant progress on this project was made during their programs and workshops. The authors thank the MPIM, Bonn, for hospitality and support for SageMath [17] computations.

2. Generalized Strata

The moduli space of multi-scaled differentials is a compactification of strata of differentials which has many desirable properties similar to the Deligne-Mumford compactification of the moduli space of curves. For example it is a smooth modular compactification with normal crossing divisors and its boundary components parametrize products of lower dimensional moduli spaces of multi-scaled differentials. In this section, we use these properties, especially the recursive structure of the boundary, in order to encode all boundary components of a stratum by iteratively cluturing 2-level graphs. These graphs appear as BICs boundary components of level strata, which in general parametrize meromorphic differentials on disconnected curves with residue conditions. We begin by briefly recalling the notions from [3] and [6] in the generality that we here require.

2.1. Strata with residue conditions

To obtain a recursive structure on the boundary of $\mathbb{P} \mathcal{X} \mathcal{M}_{g,n}(\mu)$, recall the definition of a generalized stratum, introduced in [6, §4] to cover the case of a level of an enhanced level graph. More precisely, we allow differentials on disconnected surfaces: denote by $\mu_i = (m_{i,1}, \ldots, m_{i,n_i}) \in \mathbb{Z}^{n_i}$ the type of a differential, i.e., we require that $\sum_{j=1}^{n_i} m_{i,j} = 2g_i - 2$ for some $g_i \in \mathbb{Z}$ and $i = 1, \ldots, k$. Then we define, for a tuple $g = (g_1, \ldots, g_k)$ of genera and a tuple $n = (n_1, \ldots, n_k)$ together with $\mu = (\mu_1, \ldots, \mu_k)$, the disconnected stratum

$$\Omega \mathcal{M}_{g,n}(\mu) = \prod_{i=1}^{k} \Omega \mathcal{M}_{g_i,n_i}(\mu_i).$$

Note that the projectivized stratum $\mathbb{P} \Omega \mathcal{M}_{g,n}(\mu)$ is the quotient by the diagonal action of $\mathbb{C}^\ast$, not the quotient by the action of $(\mathbb{C}^\ast)^k$.

Moreover, we consider subspaces of these cut out by residue conditions. More precisely, denote by $H_p \subseteq \bigcup_{i=1}^{k} \{(i, 1), \ldots, (i, n_i)\}$ the subset of the marked points such that $m_{i,j} < -1$. Now consider vector spaces $\mathfrak{R}$ of the following special shape, modeled on the global residue condition from [2]: for $\lambda$ a partition of $H_p$, with parts denoted by $\lambda^{(k)}$, and a subset $\lambda_{\mathfrak{R}}$ of the parts of $\lambda$, we define the $\mathbb{C}$-vector space

$$\mathfrak{R} := \left\{ r = (r_{ij})_{(ij) \in H_p} \in \mathbb{C}^{H_p} \quad \text{and} \quad \sum_{(ij) \in \lambda^{(k)}} r_{ij} = 0 \quad \text{for all} \quad \lambda^{(k)} \in \lambda_{\mathfrak{R}} \right\}. \quad (1)$$

We denote the subspace of surfaces with residues in $\mathfrak{R}$ by $\Omega \mathcal{M}_{g,n}^{\mathfrak{R}}(\mu)$.

In [6, Prop. 4.2] a modular compactification $\mathbb{P} \mathcal{X} \mathcal{M}_{g,n}(\mu)[[\mathfrak{g}, \mathfrak{n}]][\mathfrak{R}]$ of $\mathbb{P} \Omega \mathcal{M}_{g,n}^{\mathfrak{R}}(\mu)$ is constructed in analogy to [3]. Consequently, the boundary components are parametrized by enhanced level graphs. More precisely, a level graph is defined to be a stable graph together with a level function. Recall that a stable graph is a tuple $\Gamma = \bigsqcup_{i=1}^{k} \Gamma_i$, where $\Gamma_i = (V_i, H_i, E_i, g_i, v_i, \iota_i)$ are the connected components of the graph consisting of vertices $V_i$, a genus map $g_i$: $V_i \to \mathbb{Z}_{\geq 0}$, legs $H_i$ that are associated to the vertices by a vertex map $v_i$: $H_i \to V_i$, and come with an involution $\iota_i$: $H_i \to H_i$, the two-cycles of which form the oriented edges $E_i \subseteq H_i \times H_i$ while the fixed points (denoted $H^m_i$) are in bijection with the $n_i$ marked points. The $k$ graphs $\Gamma_j = (V_j, H_j, E_j, g_j, v_j, \iota_j)$ are required to be connected and satisfy the usual stability conditions. Moreover, we set $g = \sqcup g_i, v = \sqcup v_i, E = \sqcup E_i, H = \sqcup H_i$, and $V = \sqcup V_i$. Note that this data induces a unique bijection $o$: $\bigsqcup H^m_i \to \mu$ associating to each marked point $(i, j)$ the order $m_{ij}$ of the differential.

A level function on the vertices is a map $\ell: V \to \mathbb{Z}$, which we normalise to take values in $\{0, -1, \ldots, -L\}$, and require that, for all edges $e \in E$, $\ell(v(e^+)) = \ell(v(e^-))$, where we write $e := (e^+, e^-) \in H \times H$.

Moreover, an enhancement is a map $\kappa: E \to \mathbb{Z}_{\geq 0}$ such that $\kappa(e) = 0$ if and only if $e$ is horizontal (i.e. $\ell(v(e^+)) = \ell(v(e^-))$), subject to the following stability condition: define the degree of a vertex $v$ in $V$ to be

$$\deg(v) = \sum_{h \in H_i, v(h) = v} o(h) + \sum_{e \in E, v(e^+) = v} (\kappa(e) - 1) - \sum_{e \in E, v(e^-) = v} (\kappa(e) + 1).$$

Then the enhancement is admissible, if $\deg(v) = 2g(v) - 2$ holds for every vertex $v$ of $\Gamma$.

The enhanced level graph then consists of the triple $(\Gamma, \ell, \kappa)$. We denote the corresponding boundary component of $\mathbb{P} \mathcal{X} \mathcal{M}_{g,n}(\mu)[[\mathfrak{g}, \mathfrak{n}]][\mathfrak{R}]$ by $D_P$. All the figures in this text give examples of enhanced level graphs, see in particular the box in Figure 2 bottom left for a disconnected stratum and the reason for decorating it with residue conditions.
Remark 2.1. Note that $\kappa(e)$ corresponds to the number of prongs at $e$ and, for a non-horizontal edge, the associated differential has a zero of order $\kappa(e) - 1$ on the vertex at the top end of the edge and a pole of order $-\kappa(e) - 1$ on the vertex at the bottom end. (For horizontal edges there is a simple pole on each adjacent vertex.) This gives an extension of $o$ to $H$. Using this identification, the stability condition of [3, §2] is simply the requirement that the orders of zeros and poles sum to $2g_{ij} - 2$ on each vertex. See Bainbridge et al. [3, §2] and Costantini et al. [6, §3.2] for details.

To determine, for a generalized stratum $\mathbb{P} \Xi M_{g,n}(\mu)[\mathbf{g}, \mathbf{n}][\mathcal{R}]$, which enhanced level graphs give non-empty boundary components, we recall the $\mathcal{R}$-GRC from [6, §4] using notation from (1): Starting with an enhanced level graph $\Gamma$, we construct a new auxiliary-level graph $\bar{\Gamma}$ by adding, for each $\lambda^{(k)} \in \mathcal{R}$, a new vertex $v_{\lambda^{(k)}}$ to $\bar{\Gamma}$ at level $\infty$ and converting a tuple $(i,j) \in \lambda^{(k)}$ into an edge from the marked point $(i,j)$ to the vertex $v_{\lambda^{(k)}}$. We then say that $\bar{\Gamma}$ satisfies the $\mathcal{R}$-global residue condition ($\mathcal{R}$-GRC) if the tuple of residues at the legs in $H_{\bar{\Gamma}}$ belongs to $\mathcal{R}$ and for every level $J < \infty$ of $\bar{\Gamma}$ and every connected component $Y$ of the subgraph $\bar{\Gamma}_{> J}$ one of the following conditions holds.

1. The component $Y$ contains a marked point with a prescribed pole that is not in $\lambda_{\mathcal{R}}$.
2. The component $Y$ contains a marked point with a prescribed pole $(i,j) \in H_{\bar{\Gamma}}$ and there is an $r \in \mathcal{R}$ with $r_{(i,j)} \neq 0$.
3. Let $e_1, \ldots, e_b$ denote the set of edges where $Y$ intersects $\bar{\Gamma}_{= f}$. Then

$$\sum_{j=1}^{b} \text{Res}_{e_j}^{-} \eta_{v(e_j^-)} = 0,$$

where $v(e_j^-) \in \bar{\Gamma}_{= f}$.

By Costantini et al. [6, Prop. 4.2], the boundary components $D_\Gamma$ of $\mathbb{P} \Xi M_{g,n}(\mu)[\mathbf{g}, \mathbf{n}][\mathcal{R}]$ are parameterized by enhanced level graphs $(\Gamma, \ell, \kappa)$ satisfying the $\mathcal{R}$-GRC.

For applications such as listing all graphs in the boundary of a stratum, it is convenient to have a purely graph-theoretic criterion in analogy to the one shown for the classical GRC in [14]. The following proposition strips the tropical language away in the criterion [14, Theorem 1] and generalizes it to meromorphic strata with residue conditions.

Let $(\Gamma, \ell, \kappa)$ be an enhanced level graph. We call a vertex $v$ of $\Gamma$ inconvenient if $g(v) = 0$, if it is not adjacent to any edge $e$ with enhancement $\kappa(e) = 0$ and if it is adjacent to a leg with a very high enhancement in the following precise sense: denote by $p(v)$ the set of half-edges on the vertex $v$ that are poles (in the sense of Remark 2.1). Then the condition is that there is a $p \in p(v)$ such that

$$o(p) > \sum_{p' \in p(v)} (o(p') - 1) - 1.$$

Proposition 2.2. The boundary stratum $D_\Gamma$ of $\mathbb{P} \Xi M_{g,n}(\mu)[\mathbf{g}, \mathbf{n}][\mathcal{R}]$ associated with the enhanced level graph $(\Gamma, \ell, \kappa)$ is non-empty if and only if both of the following conditions are satisfied:

1. For every inconvenient vertex $v$ of $\Gamma$ there is
   
   (a) a simple cycle based at $v$ that does not pass through any vertex of level smaller than $v$, or
   
   (b) the graph of levels $\geq \ell(v)$ deprived of the vertex $v$ has two components, each of which has a marked pole in $H_{\bar{\Gamma}}$ whose residue is not constrained to zero for all elements of $\mathcal{R}$.

2. For every horizontal edge $e$ of $\Gamma$ there is
   
   (a) a simple cycle based through $e$ that does not pass through any vertex of level smaller than $\ell(e)$, or
   
   (b) the graph of levels $\geq \ell(e)$ deprived of the edge $e$ has two components, each of which has a marked pole in $H_{\bar{\Gamma}}$ whose residue is not constrained to zero for all elements of $\mathcal{R}$.

Proof. Consider first the case that $\bar{\Gamma}$ is connected. We view $\bar{\Gamma}$ as an enhanced level graph of an auxiliary stratum $\bar{X}$ as follows: we add prongs to the new edges of $\bar{\Gamma}$ in accordance with the pole orders of the half-edges on $\Gamma$. For each vertex $v$ at level $\infty$, we then extend the genus function by $g_v$ setting $2g_v - 2$ as the sum of orders of the half-edges on $v$ induced by the newly added prongs, possibly adding an extra simple zero to fix parity issues. Note that all new vertices are of positive genus, so stability is not an issue. Therefore, for the stratum $\bar{X}$, we are now reduced to the situation of [14, Theorem 1].

In a product of strata, clearly a graph is admissible if and only if each component is admissible. \hfill $\square$

Example 2.3. Consider the graphs in the boundary of the stratum $\mathbb{P} \Xi M_{g,n}(\mu)(2, 1, 1) | \{3, 3\}$ that are depicted in Figure 1. Note that the left graph is illegal, as the bottom-left vertex is inconvenient: it is a stratum with signature $(2, -2, -2)$ and both residues are forced zero, as there is no cycle or pole in the graph above to rectify this. This problem does not occur in the right graph.

Consider the situation now as a degeneration of the bottom level of the "zigzag-graph," the BIC that is the common undesegeneration of the two graphs depicted in Figure 2. Note that there are residue conditions intertwining the simple zeros on the two components. These imply that the middle graph is legal, while the right one is not.
Figure 1. An illegal (left) and legal (right) graph in the boundary of $\Omega M_3(2, 1, 1)$.

Figure 2. The “zigzag-graph” (left) in the boundary of the stratum $\Omega M_3(2, 1, 1)$ and a legal (center) and illegal (right) degeneration of its bottom level.

Figure 3. An example of a reducible profile: the two non-isomorphic three-level graphs $G$ and $H$ have the same two-level graphs as undegenerations. The level crossings are denoted by dotted lines.

This criterion allows us to explicitly construct all graphs in a given stratum. In fact, we can construct all graphs with no horizontal edges recursively from the two-level graphs.

### 2.2. Constructing Level Graphs

Recall the undegeneration maps $\delta_i$ [6, §3.3], contracting all level crossing of an enhanced level graph $\Gamma$ without horizontal edges except for the $i$th-level crossing, yielding a two-level graph. The boundary component defined by $\Gamma$ is contained in the intersection of the divisors defined by $\delta_i(\Gamma)$, but in general this intersection can contain also other components defined by other level graphs.

**Definition 2.4.** Let $(\Gamma, \ell, \kappa)$ be an enhanced level graph with $L$ levels and without horizontal edges. We define the profile of $\Gamma$ to be the tuple $(\delta_1(\Gamma), \ldots, \delta_L(\Gamma))$. The enhanced profile of $\Gamma$ is its profile together with an assignment of a number recovering uniquely $\Gamma$ among all enhanced level graph with same profile.

**Example 2.5.** Note that the association of a profile to a graph is not injective. Indeed, consider the following level graphs in the boundary of $\Omega M_3(4)$ depicted in Figure 3.

A profile is primarily useful to encode efficiently how graphs degenerate. By definition, the information of an enhanced profile of $\Gamma$ is equivalent to the data of $\Gamma$.

Generating all non-horizontal graphs inside a stratum is thus equivalent to listing all non-empty enhanced profiles. We do this recursively. For $X$ a generalized stratum, denote by $BIC(X)$ the (non-horizontal) two-level graphs parametrizing the boundary divisors of $X$.

**Definition 2.6.** Let $X$ be a generalized stratum. For each $\Gamma \in BIC(X)$ we define
(1) the generalized strata $X^\top_\Gamma$ and $X^\bot_\Gamma$, the top and bottom levels of the divisor $D_\Gamma$ defined by $\Gamma$;
(2) a map $\beta^\top_\Gamma : \text{BIC}(X^\top_\Gamma) \to \text{BIC}(X)$ that associates to a 2-level graph $\Gamma' \in \text{BIC}(X^\top_\Gamma)$ the graph $\delta_1(\Lambda) \in \text{BIC}(X)$ where $\Lambda$ is the graph obtained by clutching $\Gamma''$ to $\Gamma$;
(3) a map $\beta^\bot_\Gamma : \text{BIC}(X^\bot_\Gamma) \to \text{BIC}(X)$ that associates to a 2-level graph $\Gamma' \in \text{BIC}(X^\bot_\Gamma)$ the graph $\delta_2(\Lambda) \in \text{BIC}(X)$ where $\Lambda$ is the graph obtained by clutching $\Gamma''$ to $\Gamma$.

The following proposition is the most important feature of the boundary that we need in order to efficiently code the components. It also implies that the non-horizontal boundary components are simple normal crossings.

Proposition 2.7. Let $X$ be a generalized stratum and $\Gamma, \Gamma' \in \text{BIC}(X)$.

(1) The images of $\beta^\top_\Gamma$ and $\beta^\bot_\Gamma$ are disjoint.
(2) The profile $(\Gamma', \Gamma)$ is non-empty if and only if $\Gamma'$ is in the image of $\beta^\top_\Gamma$.
(3) The profile $(\Gamma, \Gamma')$ is non-empty if and only if $\Gamma'$ is in the image of $\beta^\bot_\Gamma$.

Proof. The proof follows directly from [6, Prop. 5.1]. We recall that the argument consists of checking the dimensions of the top and bottom levels of the undegenerations of boundary strata, which determine uniquely the profile. \qed

As a consequence, we may define a partial order $\prec$ on $\text{BIC}(X)$ by defining $\Gamma \prec \Gamma'$ if and only if $(\Gamma', \Gamma)$ is non-empty.

Remark 2.8. Profiles can be reducible and determining the reducibility of a profile is a delicate issue. In particular, each of the following may occur:

(1) a degeneration of an irreducible profile can be reducible (Figure 3);
(2) a degeneration of a reducible profile can be irreducible (Figure 4);
(3) Whenever a profile $(\Gamma_1, \Gamma_2)$ is reducible, the corresponding maps $\beta^\top_{\Gamma_2}$ and $\beta^\bot_{\Gamma_1}$ are not injective. However, non-injectivity does not imply reducibility. Degenerating a level in different ways may give isomorphic graphs.

In the sequel, we will call 2-level graphs of banana type if they have only two vertices which are joined by more than one edge. We will call them of compact type if they have two vertices joined by only one edge. (The name is justified by the fact that the generalized Jacobians of the associated nodal curves are of compact type).

Example 2.9. We give an example for the last point: consider the “triple banana”, the BIC with a non-trivial $S_3$ action (cf. Figure 5) in the boundary of $\Omega M_3(4)$. The bottom level is the stratum $\Omega M_0(4, -2, -2, -2)$, which is one-dimensional and contains three BICs of compact type, distinguished only by the numbering of their marked points. However, no matter how we glue the compact type graph into the bottom level, the resulting graphs are always isomorphic: each gives the intersection with the “asymmetric banana.” Thus the profile of the intersection is irreducible even though $\beta^\bot$ is not injective (all three compact type BICs map to the asymmetric banana).

The previous proposition allows us to recursively compute all profiles.

Proposition 2.10. Let $X$ be a generalized stratum.

(1) The number of enhanced 2-level graphs in $\text{BIC}(X)$ is finite and the graphs can be listed explicitly.

![Figure 4](image-url) The four-level graphs degenerating from the reducible profile consisting of the graphs $G$ and $H$ above. Note that the left two graphs share a (reducible) profile, as do the right two. The middle graph, however, is unique inside its irreducible profile. Observe also that the right two graphs cannot be distinguished by their levels.
Figure 5. From left to right: the "triple banana"; the "asymmetric banana"; the compact type BIC inside $\Omega M_{4}(4,-2,-2,-2)$, the bottom level of the "triple banana"; the unique graph in the intersection of the triple and asymmetric banana.

(2) All non-empty profiles in $X$ can be constructed recursively from BIC($X$).

(3) All graphs inside a profile can be constructed explicitly from the profile’s components.

**Proof.** The generation of BIC($X$) may be accomplished in several steps: first, a list of all combinatorially possible 2-level graphs is generated for each component of the stratum, then the GRGs are checked for each graph and these are then combined to give all possible BICs for a Generalized Stratum. Finally, they are sorted by isomorphism class and checked against the $\mathcal{R}$-GRG. Since for each step there are explicit bounds (see Algorithm 2.11), it follows that the number of BICs is finite.

The non-empty profiles are constructed recursively: a non-empty profile $(\Gamma_1, \ldots, \Gamma_l)$ of length $l$ may be extended to a non-empty profile $(\Gamma_0, \Gamma_1, \ldots, \Gamma_l)$ if and only if $\Gamma_1 < \Gamma_0$. Indeed, $\Gamma_l^\top$ is also the top level of any graph $\Lambda$ in $(\Gamma_1, \ldots, \Gamma_l)$ and thus a preimage $(\beta_1^\top)^{-1}(\Gamma_0)$ exists in BIC($X_{\Gamma_0}^\top$) and can be clutched to $\Lambda$ to yield a graph in $(\Gamma_0, \Gamma_1, \ldots, \Gamma_l)$. Similarly, the profile may be extended to $(\Gamma_1, \ldots, \Gamma_l, \Gamma_{l+1})$ if and only if $\Gamma_{l+1} < \Gamma_l$.

The converse direction is simply given by the undegeneration obtained by the contraction of a level crossing.

To get all the graphs we follow the above procedure, noting that we might obtain several graphs in the same profile if $\beta$ is non-injective (we clutch each preimage with each other graph).

Next, we provide an efficient algorithm for listing BIC($X$). We stress that the naive approach of considering all stable graphs in the corresponding moduli space of curves and trying to put all possible enhanced level structures on each one of them is a much less efficient procedure.

We start with some elementary bounds: We denote the maximal sum of genera of vertices on bottom level by $g_{\max}$ and the minimal sum of genera of vertices on top level by $g_{\min}$. As every top-level vertex $v$ with $g_v = 0$ requires at least one pole, $g_{\max} = g - 1$ and $g_{\min} = 1$ for holomorphic strata and correspondingly $g_{\max}^{\top} = g$ and $g_{\min}^{\top} = 0$ for meromorphic strata.

**Algorithm 2.11 (BIC Generation).**

Step 1. We begin by iterating over the possible number $v^{\perp}$ of vertices on bottom level. As every bottom-level vertex needs at minimum either a zero or (if it’s genus 0 and has only one edge going up) two marked points, we note that $z + n$ is an upper bound for $v^{\perp}$.

Step 2. Next, we distribute the zeros between upper and lower level by iterating over all 2-length partitions of $z$ (and also include the case where all zeros are on bottom level), ensuring at each step that we have enough zeros to satisfy $v^{\perp}$.

Step 3. The zeros are distributed onto the bottom components. If there are no marked points, every component needs at least one zero, otherwise we are more flexible.

Step 4. The genus is partitioned into the contribution $g^{\top}$ from the top vertices, $g^{\perp}$ from the bottom vertices and the graph. For this, we iterate over $g^{\top}$ and $g^{\perp}$ (using the bounds $g_{\max}^{\top}$ and $g_{\min}^{\top}$).

Step 5. $g^{\perp}$ is distributed onto the bottom components. At this point, we have added all zeros and will have to add (at least) double poles for the edges. Thus, if the orders on any vertex $v$ sum up to less than $2g_v$, we can move on to the next iteration.

Step 6. We now distribute the poles $p$. This is again achieved via 2-length partitions. Note that every $g = 0$ vertex on top needs at least one pole (to compensate the edge(s) going down), so this gives an immediate check for the partitions.

Step 7. Next, we distribute the poles among the bottom components. At this point, we also check that the difference of $2g_v - 2$ and the orders distributed to $v$, i.e. the "space" left for half-edges, is at least $-2$ on each component, so that there is potential for at least one edge going up for every vertex.

Step 8. Now we consider the top level for the first time. We iterate through the number of top-level vertices, $v^{\top}$, which is bounded by the sum of $g^{\perp}$ and the number of poles on top-level. As we know the number of vertices and the distribution of genus, the Euler formula determines the number of edges, $|E|$. This gives a "global" check for the "spaces" left on the bottom components: they must sum up to $-2 \cdot |E|$.

Step 9. Similar to above, we now distribute genus, poles and zeros on top level. We also do the obvious checks on the orders and record the spaces on top (there are much fewer constraints here, as the spaces may well be zero).
Step 10. We now place the half-edges. We again start on bottom-level, because the poles give stronger constraints. Moreover, the half-edge orders on bottom determine those on top.

Step 11. Next we check that the graph we have created is connected, if not we proceed with the next iteration.

Step 12. The only thing missing are the marked points. These are distributed and we check for stability.

Step 13. In the final step, we check the GRC.

While we do not claim that algorithm is optimal, the implementation in diffstrata certainly runs in a reasonable time:

```
sage: from admcycles.diffstrata.bic import bic_alt_noiso
sage: %time len(bic_alt_noiso((1,1,1,1)))
CPU times: user 678 ms, sys: 9.38 ms, total: 688 ms
Wall time: 691 ms
384
```

This algorithm does not yet sort BICs into isomorphism classes yet, which is a subsequent time-consuming step. In the stratum \( \mu = (1^4) \) there are 102 non-isomorphic BICs among the 384 generated above, see Table 1. An algorithm that directly produces only non-isomorphic BICs could lead to significantly improved performance.

The key observation for generating all graphs is that BICs and three-level graphs are sufficient for constructing the entire stratum. In particular, all levels are seen by these.

Remark 2.12. Let \( X \) be a generalized stratum. Note that any level \( L \) appearing in any graph in \( X \) is one of the following three types:

1. a top level of a BIC,
2. a bottom level of a BIC, or
3. a middle level of a three-level graph.

Indeed, given a graph \( \Gamma \), level \( l \) of \( \Gamma \) remains unchanged by contracting any level crossing not adjacent to \( l \). Contracting all nonadjacent levels results either in a BIC (if \( l \) is top or bottom level) or in a three-level graph around level \( l \).

3. Calculations in the tautological ring

The purpose of the diffstrata package is to facilitate calculations in the tautological ring of strata. In light of [6, Theorem 1.5], any tautological class may be expressed as a formal sum of additive generators given by products of \( \psi \)-classes supported on single level graphs.

Let \( X \) be a generalized stratum. Any tautological classes on \( X \) can be added, multiplied and evaluated (i.e. integrated against the class of \( X \)). The intersection of tautological classes is developed by implementing the excess intersection formula proven in [6, Proposition 8.1]. In particular we present three intertwined algorithms, the intersection, pull-back and normal bundle algorithms, which together implement a recursive procedure to intersect tautological classes. Finally, the evaluation works by breaking down the expression into \( \psi \)-products on meromorphic strata and using the admcycles package [8] to evaluate these. We describe more in detail in this section the algorithms involved.

3.1. Evaluation

Any tautological class can be evaluated, i.e. integrated against the fundamental class of the corresponding stratum. If an additive generator is not of top degree, it automatically evaluates to 0.

```
sage: X=Stratum((2,))
sage: (X.psi(1)).evaluate()
0
sage: (X.psi(1)^3).evaluate()
1/1920
```

The key to evaluating a tautological class is to split it into pieces that can be evaluated by admcycles, using the expression of the class of a stratum in the tautological ring of \( \overline{M}_{g,n} \) by Sauvaget [15]. For this purpose, the stack factor \( sf \) of an additive generator is defined as follows: let \( G \) be the associated graph, then \( sf \) is the quotient of the product of the prongs of \( G \) and the product of \( \ell_B \) for every \( B \) appearing in the profile of \( G \) and the number of automorphisms of \( G \). For a BIC \( B \), the number \( \ell_B \) is the lcm of the prongs of \( B \). By [6, Lemma 9.12] this factor is necessary to convert an integral over a boundary stratum \( D_\Gamma \) into the product of level-wise evaluations.

**Algorithm 3.1 (Evaluation).**

Step 1. Take the sum of the evaluation of the additive generators multiplied with their respective coefficients.
Step 2. Each additive generator is evaluated by sorting the $\psi$-classes by level and taking the product of the evaluations of these $\psi$-polynomials on each level. This product is then multiplied with the stack factor.

Step 3. If the residue space $\mathcal{R}$ is empty, then we evaluate as follows. If the level is disconnected, it evaluates to 0; if it is 0-dimensional, it evaluates to 1; otherwise we use the function of admcycles that computes the tautological class of the image of the stratum in $\mathcal{M}_{g,n}$ and integrate this against the $\psi$-classes, using admcycle's evaluation method.

Step 4. If the level splits as a product (with residue conditions, i.e., with respect to the underlying graph $\tilde{\Gamma}$), then it evaluates to 0, since the fiber dimension to the product of moduli spaces is positive and the $\psi$-classes are pullbacks from there.

Example 3.2. We illustrate the calculation of a class cut out by a residue condition:

```
sage: X=GeneralisedStratum([Signature((23,5,-13,-17))])
sage: assert X.res_stratum_class([[0,2]]).evaluate() == 5
```

In fact, this stratum has three boundary points corresponding to graphs $\Gamma_1$, $\Gamma_2$ and $\Gamma_3$ (see Figure 6) that have respectively the marked points of order $(23,5), (23,-13)$ and $(23,-17)$ on lower level. By [6, Prop. 8.2] we can express $\xi$ using the first $\psi$-class as $\int_X \xi = 24 - 29 - 11 - 7 = -23$. Now in [6, Prop. 8.3] the contributing boundary graphs are $\Gamma_2$ (because the point with order $-13$ is on lower level) and $\Gamma_3$ (because the zero residue at the point of order $-13$ on upper level is automatic), but not $\Gamma_1$. We find that the evaluation of the boundary stratum is $-(-23) - 11 - 7 = 5$, as claimed.

### 3.2. Intersections

While adding two additive generators is straight-forward, expressing the intersection of two additive generators again as a sum of additive generators is subtle, in particular if the intersection is not transversal.

A first approximation is finding common degenerations of two graphs, but if these graphs have a common undegeneration, there is a normal bundle contribution. The precise answer is the excess intersection formula [6, Prop. 8.1].

In this section, we implement the general version of the excess intersection formula [6, eq. (61)].

Let $\Lambda_1$ and $\Lambda_2$ be degenerations of a $k$-level graph $\Gamma$. A level graph $\Pi$ is a $(\Lambda_1, \Lambda_2)$-graph if there are undegeneration morphisms $\rho_l : \Pi \to \Lambda_l$, i.e. edge contraction morphisms with the property that there are subsets $I$ and $J$ of levels such that $\delta_I(\Pi) = \Lambda_1$ and $\delta_J(\Pi) = \Lambda_2$. Denote the associated boundary components by $D_\Pi, D_{\Lambda_1}, D_{\Lambda_2}$ and $D_\Gamma$, as in the following diagram:

\[
\begin{array}{ccc}
D_\Pi & \xrightarrow{j_{\Pi,\Lambda_2}} & D_{\Lambda_2} \\
\downarrow \downarrow & & \downarrow \downarrow \\
D_{\Lambda_1} & \xrightarrow{j_{\Lambda_1,\Gamma}} & D_\Gamma
\end{array}
\]

Furthermore, we define $v_{(\Lambda_1 \cap \Lambda_2)/\Gamma}$ to be the product of the pullback to $D_\Pi$ of the normal bundles $N_{\Pi', \Gamma}$ where $\Pi'$ ranges over all $k+1$-level graphs $\Pi'$ that are a degeneration of $\Gamma$ and that are moreover common to $\Lambda_1$ and $\Lambda_2$, see Section 3.3. For any $\alpha \in \text{CH}^*(D_{\Lambda_2})$ we can then express its push-forward pulled back to $\Lambda_1$ as

\[
j_{\Lambda_1,\Gamma}^* j_{\Lambda_2,\Gamma} \cdot \alpha = \sum_\Pi j_{\Pi,\Lambda_1} \cdot \left( v_{(\Lambda_1 \cap \Lambda_2)/\Gamma} \cdot i_{\Pi,\Lambda_2}^* \alpha \right),
\]

where the sum ranges over all $(\Lambda_1, \Lambda_2)$-graphs $\Pi$. This corresponds to the product of $\alpha$ with the class of $\Lambda_2$ in $\text{CH}(D_\Gamma)$. For more details, see [6, §8.1].
Of course, the product of two additive generators on the same graph is simply the product of their $\psi$-polynomials and the product of two tautological classes is the sum of the products of their additive generators.

Therefore, the excess intersection formula is given by a pullback as the base case (no common intersection) and recursive multiplication with normal bundles. Note that the codimension of the ambient stratum increases at each step and therefore the normal bundle becomes trivial after finitely many iterations. In Example 4.1, we illustrate the intersection procedure in more detail.

### 3.3. Normal bundles

The key ingredient for the multiplication is the calculation of normal bundles. The first Chern class of the normal bundle of a divisor is computed in [6, Thm 7.1]. In `diffstrata`, this can be computed using `normal_bundle`:

```python
sage: X=Stratum((2,))
sage: X.normal_bundle(((0,),0)) == X.taut_from_graph((0,),0)^2
True
sage: X.normal_bundle(((1,),0)) == X.taut_from_graph((1,),0)^2
True
```

However, for the excess intersection formula, we need to compute $\nu/\pi_1(\Lambda_1 \cap \Lambda_2)/\Gamma$. The situation is summarized in the following diagram of the involved graphs:

```
\[
\begin{array}{c}
\Lambda_1 \cap \Lambda_2 \\
\downarrow \\
\Lambda_1 \\
\end{array}
\rightarrow
\begin{array}{c}
\Lambda_2 \\
\downarrow \\
\Lambda_1 \cap \Lambda_2 \\
\end{array}
\rightarrow
\begin{array}{c}
\Gamma'' \\
\Gamma' \\
\end{array}
\rightarrow
\begin{array}{c}
\Gamma \\
\end{array}
\end{array}
\]
```

where the arrows represent undegeneration maps. The graph $\Gamma''$ is the *minimal common undegeneration* of $\Lambda_1$ and $\Lambda_2$, corresponding to the intersection of the profiles, while $\Lambda_1 \cap \Lambda_2$ corresponds to the union of the profiles (up to reducibility issues, cf. Example 2.5). The graph(s) $\Gamma'$ are codimension one degenerations of $\Gamma$ that are common undegenerations of $\Lambda_1$ and $\Lambda_2$.

For the excess intersection formula, we need the product of the normal bundles $\mathcal{N}_{\Gamma,\nu} = \mathcal{N}_{D_{\nu}/D_{\Gamma}}$, where $\Gamma$ is a $k + 1$-level degeneration of the $k$-level graph $\Gamma$. More precisely, the normal bundles $\mathcal{N}_{D_{\nu}/D_{\Gamma}}$ are pulled back to $D_{\nu''}$ and the product is computed in $CH(D_{\nu''})$. The normal bundle is computed in [6, Prop. 7.5]:

$$c_1(\mathcal{N}_{\Gamma,\nu}) = \frac{1}{\ell_{\delta(\Gamma')}} (-\xi^{[i]}_{\Gamma'} - c_1(\mathcal{L}^{[i]}_{\Gamma'}) + \xi^{[i+1]}_{\Gamma'})$$

in $CH^1(D_{\nu''})$,

where

$$\mathcal{L}^{[i]}_{\Gamma'} = O_{D_{\Gamma'}} \left( \sum_{\nu''[\hat{\Delta}]_\Gamma} \ell_{\delta_{i+1}(\hat{\Delta})} D_{\hat{\Delta}} \right),$$

where the sum runs over all graphs $\hat{\Delta} \in LG_{k+2}(\hat{B})$ that yield divisors in $D_{\Gamma'}$ by splitting the $i$th level. These are then pulled back to $D_{\nu''}$, see Section 3.4, and then they are multiplied in $CH(D_{\nu''})$. In Example 4.1, we show how to apply this common normal bundle procedure in `diffstrata`.

**Remark 3.3.** Observe that this recursive procedure terminates: indeed, the product in $CH(D_{\Gamma'})$ has been transformed to a product in $CH(D_{\nu''})$ which is of strictly smaller dimension. Moreover, if the dimension is small enough, transversality is ensured by dimension reasons.

We summarise the normal bundle algorithm:

**Algorithm 3.4 (Normal Bundle).**

1. Compute the minimal common undegeneration $\Gamma''$.
2. If $\Gamma''$ is $\Gamma'$ the intersection is transversal, we are in the base case.
3. Loop through the codimension one common undegenerations $\Gamma'$ of $\Gamma''$.
4. Calculate the level $i$ where $\Gamma'$ and $\Gamma''$ differ.
5. Calculate the normal bundle by computing $\xi^{[i]}_{\Gamma'}$ and $\mathcal{L}^{[i]}_{\Gamma'}$ (see Algorithm 3.5).
Step 6. Pull this normal bundle back to $\Gamma''$, cf. Section 3.4.
Step 7. Return the product of the normal bundles (one for each $\Gamma''$) inside $\Gamma''$.

We explain how to compute $\xi_{B[i]}^\ell$ and $L_{\beta[i]}^\ell$, which was required in the fifth step of the previous algorithm. These objects are computed by clutching the pull back of the corresponding classes $\xi_{B[i]}^\ell$ and $L_{\beta[i]}^\ell$ which are elements of the Chow ring of the level stratum $B[i]^\ell$. Even though, in contrast to the moduli space of curves, there is no clutching morphism from the product of the level strata $B[i]^\ell$ to $D^\ell$, there is a correspondence between the product of level strata and the associated boundary component. The details are explained in [6, Sec. 4.2].

It remains to explain only how to “glue” or clutch a tautological class on $B[i]^\ell$ into $D^\ell$, in order to obtain an element of the Chow ring of the original stratum $X$. Since we are only interested in $\xi_{B[i]}^\ell$ and $L_{\beta[i]}^\ell$, we can focus on clutching $\psi$-classes and divisors, and ignore classes of higher codimension.

Inserting $\psi$-classes on a level is straightforward, since we only need to take care of the numbering of the marked points in the level stratum $B[i]^\ell$ and the corresponding half-edge in $\Gamma$. This information is stored by $\text{diffstrata}$ during level extraction and is available.

Consider the situation that $\Gamma'$ is a BIC in $B[i]^\ell$. We want to clutch this in, obtaining a graph $\Lambda$ in $X$ with one more level than $\Gamma$. However, the implementation of this is rather delicate: in light of Remark 2.12, we can realise $B[i]^\ell$ as a level in a BIC or three-level graph and for these we can use the maps $\beta$ of Definition 2.6 to translate the BICs of the level into BICs of $X$. But for graphs with several levels, we must first undegenerate to the appropriate BIC or three-level graph and thus the extracted levels could differ by an automorphism from the levels used by $\beta$. See [7] for more examples and details on how this is resolved.

We summarise how to determine the enhanced profile of the one-level degeneration $\Lambda$ of $\Gamma$:

**Algorithm 3.5 (Gluing in a BIC).**

Step 1. Determine the new profile. This is given by the degeneration graph via the maps $\beta$ of Definition 2.6, depending on the location of $B[i]^\ell$.
Step 2. Determine the graph by “gluing” in the BIC on the level of graphs.
Step 3. Find the enhanced profile by locating the isomorphism class of the clutched graph inside the new profile.

Moreover, we need to weight the clutched in class with the contribution from comparison of multiplicities in the level projections as given in [6, Prop. 4.7]. The correction factor is the product of the edge contribution and the automorphism contribution. The edge contribution is the quotient of $\ell_{\Lambda''}$, where $\Lambda''$ is the BIC of $X$ that extended the profile of $\Gamma'$ and $\ell_{\Gamma''}$ of the BIC $\Gamma''$ of $B[i]^\ell$ that was inserted. The automorphism factor is the quotient of the number of automorphisms of the glued graph and the product of the number of automorphisms of $\Gamma$ and $\Gamma''$.

### 3.4. Pulling back classes

To calculate the pullback of an additive generator, we consider first the base case and then the generalized case.

Consider a graph $\Lambda$ in $X$, $\alpha$ an additive generator on corresponding to a class on $D_{\Lambda}$ and a degeneration $\Pi$ of $\Lambda$, i.e. we obtain $\Lambda$ by contracting some of the level-crossings of $\Pi$. Then there are finitely many contraction morphisms $\rho: \Pi \to \Lambda$ and each of these gives a well-defined pullback map of $\alpha$. The pullback is the weighted sum of these.

**Example 3.6.** Consider the class $S$ of one of the top half-legs of the banana graph in the minimal stratum in genus 2. Pulling it back to the intersection will also vanish for dimension reasons, but pulling it back to the banana graph itself illustrates the weighted sum: there are two graph morphisms (switching the edges).

```sage
sage: X=Stratum((2,))
sage: S=X.additive_generator(((0,),0), {1:1}); print(S)
Psi class 1 with exponent 1 on level 0 * Graph ((0,), 0)
sage: print(S.pull_back(((0,),0)))
Tautological class on Stratum: (2,)
with residue conditions: []
1/2 * Psi class 2 with exponent 1 on level 0 * Graph ((0,), 0) +
1/2 * Psi class 1 with exponent 1 on level 0 * Graph ((0,), 0) +
```

However, for the excess intersection formula, we require a more general notion of pullback. As above, let $\Lambda$ be a graph in $X$ and $\alpha$ a class on $D_{\Lambda}$, but now we do not require the “target” graph $\Lambda'$ to be a degeneration of $\Lambda$. Instead, we will pull $\alpha$ back to the
intersection $\Lambda \cap \Lambda'$:

$$
\Lambda \cap \Lambda' \xrightarrow{\Gamma^\prime} \Lambda
$$

$$
\Lambda' \xrightarrow{\Gamma''} \Lambda
$$

Note that this must include the normal bundle contribution of the minimal common undegeneration $\Gamma''$ of $\Lambda$ and $\Lambda'$ (in $D_\Gamma$).

More precisely, the algorithm to compute the pullback of $\alpha$ to $\Lambda \cap \Lambda'$ is:

**Algorithm 3.7 (Pullback).**

Step 1. Compute the common normal bundle of $\alpha$ and $\Lambda'$ in $D_\Gamma$ (Section 3.3).

Step 2. If it is transversal, perform the pullback to each graph of $\Lambda \cap \Lambda'$ as described above.

Step 3. Otherwise, multiply (in CH($\Lambda \cap \Lambda'$)) the pullback to each graph of $\Lambda \cap \Lambda'$ with the normal bundle, with ambient $\Gamma''$.

**Remark 3.8.** Observe that this recursive procedure terminates: indeed, as for the normal bundle calculation, the involved intersections are always performed in an ambient stratum of strictly lower dimension, cf. Remark 3.3.

In Example 4.1 we show how to apply this generalized pullback in diffstrata.

### 4. Interface and examples

#### 4.1. Installation

The package **diffstrata** is included with **admcycles** version 1.1 or greater. See [8] for a detailed guide to installing it.

From now on, all examples will assume that the line

```
sage: from admcycles.diffstrata import *
```

has been executed!

#### 4.2. Working with **diffstrata**

We firstly briefly revisit the examples of the introduction. The first step is always generating a stratum:

```
sage: X=Stratum((2,))
sage: print(X)
Stratum: (2,)
with residue conditions: []
```

Here we have defined a **Stratum** object. The argument is a Python tuple and may contain integers that sum to $2g-2$ to define any meromorphic stratum (note the trailing, if there is only one entry!). The print statement displays information about the object and hints that this is in fact an instance of a **GeneralisedStratum** that can be disconnected and have residue conditions at the poles.

Creating a **Stratum** automatically performs a series of calculations. For example, all non-horizontal divisors (BICs) are generated and can now be accessed through $X$:

```
sage: X.bics
[EmbeddedLevelGraph(LG=LevelGraph([1, 1], [[1], [2, 3]], [(1, 3)], [1: 0, 2: 2, 3: -2], [0, -1], True), dmp={2: (0, 0)}, dlevels={0: 0, -1: -1}), EmbeddedLevelGraph(LG=LevelGraph([1, 0], [[1, 2], [3, 4, 5]], [(1, 4), (2, 5)], [1: 0, 2: 0, 3: 2, 4: -2, 5: -2], [0, -1], True), dmp={3: (0, 0)}, dlevels={0: 0, -1: -1})]
```

This illustrates how **diffstrata** represents level graphs internally. The multitude of decorations makes the classes **EmbeddedLevelGraph** and **LevelGraph** a bit unwieldy and there should be little reason to enter them by hand. But they do store the essential information, they are a backbone of **diffstrata**, and they appear frequently in the output.

For a single **EmbeddedLevelGraph**, e.g. an element of $X.bics$, we may use its explain method to produce a human-readable description of the graph:

```
sage: X.bics[0].explain()
LevelGraph embedded into stratum Stratum: (2,)
```
with residue conditions: []

with:

On level 0:
* A vertex (number 0) of genus 1

On level 1:
* A vertex (number 1) of genus 1

The marked points are on level 1.
More precisely, we have:
* Marked point (0, 0) of order 2 on vertex 1 on level 1

Finally, we have one edge. More precisely:
* one edge between vertex 0 (on level 0) and vertex 1 (on level 1) with prong 1.

Instead of entering graphs by hand, we should always use (enhanced) profiles to refer to them inside X. For BICs, this is simply their index in X.bics. Recall that the length of a profile is the number of level crossings of the level graphs with the profile, or equivalently to the codimension of the associated boundary components. We can list all profiles of a given length:

```
sage: X.enhanced_profiles_of_length(2)
(((0, 1), 0),)
sage: X.enhanced_profiles_of_length(3)
()
```

Recall that an enhanced profile is given by a pair, where the first entry is the profile, and the second entry is a number corresponding to the enhancement determining uniquely level graphs with the same profile. For example, the enhanced profiles of the level graphs \(G\) and \(H\) of Example 2.5 are \(((0, 1), 0, )\) and \(((0, 1), 1, )\).

Note that in the running example of the minimal stratum in genus 2 there are no profiles of length 3 even though

```
sage: X.dim()
3
```

The reason is that diffstrata ignores all graphs with horizontal edges in the boundary. We can also retrieve the EmbeddedLevel Graph from an (enhanced) profile:

```
sage: X.lookup_graph((0,1))
EmbeddedLevelGraph(LG=LevelGraph([1, 0, 0], [[1], [2, 3, 4], [5, 6, 7]], [(1, 4), (2, 6), (3, 7)], {1: 0, 2: 0, 3: 0, 4: -2, 5: 2, 6: -2, 7: -2}, [0, -1, -2], True), dmp={5: (0, 0)}, dlevels={0: 0, -1: -1, -2: -2})
```

The examples of the introduction also illustrated working in the tautological ring of \(X\). We may inspect the individual classes. Using print gives a more readable output:

```
sage: print(X.psi(1))
Tautological class on Stratum: (2,)
with residue conditions: []
1 * Psi class 1 with exponent 1 on level 0 * Graph ((0, 0) +
sage: X.psi(1)
ELGTautClass(X=GeneralisedStratum(sig_list=[Signature((2,))], res_cond=[], psi_list=[(1, AdditiveGenerator(X=GeneralisedStratum(sig_list=[Signature((2,))], res_cond=[]), enh_profile=(((), 0), leg_dict={1: 1})))])
```

This illustrates how diffstrata encodes elements of the tautological ring: a tautological class is represented by an ELGTaut Class, which is in turn a sum of AdditiveGenerators. Each AdditiveGenerator corresponds to a \(\psi\)-monomial on a graph and thus carries the information of an enhanced profile and a leg_dict encoding the \(\psi\)-powers: every \(\psi\)-class is associated to a marked point of a level \([6, \text{Theorem 1.5}]\), i.e., a leg of the graph. A \(\psi\)-monomial is thus encoded by a Python dict with entries of the form \(l : n\) where \(l\) is the number of a leg of the graph and \(n\) is the exponent of the \(\psi\)-class associated to this leg. For example, we saw above that \(X.bics[0]\) is the compact-type graph in the boundary of \(\Omega M_2(2)\). We see from the LevelGraph that the marked point is at leg 2, the \(\psi\)-class at this point is therefore represented by the leg_dict \(\{2 : 1\}\). We can enter this into diffstrata as follows:

```
sage: A = X.additive_generator(((0,), 0), {2 : 1})
sage: print(A)
Psi class 2 with exponent 1 on level 1 * Graph ((0,), 0)
```

Note that we had to use the enhanced profile \(((0,), 0)\) to refer to the graph.

Tautological classes may be added and multiplied. We can check that the class \(A\) we defined agrees with the product of the \(\psi\)-class on the stratum with the class of the graph:
Moreover, when squaring, e.g., the class of a graph, a normal bundle contribution appears:

\[ \text{True} \]

In the formulas for the Euler characteristic \([6]\), the class \(\xi = c_1(O(-1))\) of the tautological bundle and its restriction \(\xi_{B_i}/\Gamma_1\) to a level \(i\) of a graph \(\Gamma\) were key. For a stratum, the class \(\xi\) is easily accessible:

\[ \text{True} \]

Moreover, it is not difficult to compute \(\xi_{B_i}/\Gamma_1\) (here for the top-level of the compact-type graph):

\[ \text{True} \]

We end this section by illustrating, in different strata, the methods explained in Section 3.

**Example 4.1.** Consider the stratum \(\Omega M_{2}(1,1)\) and let us look at the boundary divisors (see Figure 7):

\[ \text{True} \]

We first of all show the use and necessity of intersection inside an ambient stratum. Consider and the boundary divisor represented by a compact-type graph with a genus two component on top and a genus zero component on bottom level. By inspecting the list of BICs we find that it is the third BIC. The standard operations \(*\) and \(\hat{\cdot}\) are performed in the Chow ring of the stratum. We can instead work in the Chow ring of any boundary stratum \(D_i/\Gamma_1\) by specifying as ambient the enhanced profile of \(\Gamma\). As the stratum \(Y\) is four-dimensional, we can multiply this graph with \(\xi^3\) to obtain a top-degree class that we may evaluate to find a number:

\[ \text{True} \]
This matches the observation that the top level is the minimal stratum in genus two where $\xi^3$ evaluates to $-\frac{1}{640}$, the bottom level is a point and there are no prongs or automorphisms involved, compare \cite[§4.3 and Lemma 9.12]{6}:

```
sage: Y.dim()
4
sage: (Y.xi^3*Y.additive_generator(((3,),0))).evaluate()
1/640
```

We can perform the same calculation in $Y$ using $\xi_{at\, level}$. However, as the class $\xi_t^{[i]}$ corresponding to $\xi_{at\, level}$ lives not on $Y$ but on the BIC 3, performing the normal multiplication with $\ast$ or $\hat{\ast}$ will not yield the correct result: we must use our BIC $\Gamma$ as the ambient stratum, as we want to perform the multiplication in $\text{CH}(D_\Gamma)$, i.e. before pushing forward to $Y$.

Indeed, while the evaluation of the cube of $\xi_{at\, level}$ performed in $Y$ is

```
sage: (Y.xi_at_level(0, ((3,),0))^3).evaluate()
0
```

we obtain the expected result when using the correct ambient stratum, the BIC 3:

```
sage: CT_xi_top = Y.xi_at_level(0, ((3,),0))
sage: (Y.intersection(Y.intersection(CT_xi_top, CT_xi_top, ((3,),0)), CT_xi_top, ((3,),0))).evaluate()
-1/640
```

In the special case of taking exponents of $\xi_{at\, level}$, we can use the method $\xi_{at\, level\, pow}$, which computes the exponent with the correct ambient graph:

```
sage: (Y.xi_at_level_pow(0, ((3,),0), 3)).evaluate()
-1/640
```

Indeed, in this case we may even use the method $\text{top}\_\xi_{at\, level}$ which computes and evaluates the top-power of $\xi$:

```
sage: Y.top_xi_at_level(((3,),0), 0)
-1/640
```

In fact, whenever possible this method should be used (even for $\xi$ on the whole stratum), as the results are cached and reused.

We continue in the same setting and show how to use the common normal bundle method. Recall that $Y.bics[3]$ is the compact-type graph with top-level an $M_2(2)$. We may intersect this graph with the banana graph and the other compact-type graph in $Y$. Calculating the common normal bundle of these intersections we obtain:

```
sage: print(Y.cnb(((0,3),0),((2,3),0)))
Tautological class on Stratum: (1, 1)
with residue conditions: []
-1 * Psi class 1 with exponent 1 on level 0 * Graph ((3,), 0) +
```

As expected, this is the normal of the third BIC, as the other two BICs intersect transversally:

```
sage: print(Y.normal_bundle(((3,),0)))
Tautological class on Stratum: (1, 1)
with residue conditions: []
-1 * Psi class 1 with exponent 1 on level 0 * Graph ((3,), 0) +
```

Calculating, e.g.,

```
sage: print(Y.cnb(((2,3),0),((2,),0)))
Tautological class on Stratum: (1, 1)
with residue conditions: []
-1/2 * Psi class 2 with exponent 1 on level 0 * Graph ((2,), 0) +
```

Intermediate results include:

```
-1/2 * Psi class 1 with exponent 1 on level 0 * Graph ((2,), 0) +
-1/2 * Psi class 5 with exponent 1 on level 1 * Graph ((2,), 0) +
-1/2 * Psi class 6 with exponent 1 on level 1 * Graph ((2,), 0) +
```
gives the normal bundle of the banana graph.

We finally illustrate the *generalized pullback* method. The normal bundle of the compact type graph with genus 2 on top, is simply a \(\psi\)-class on top-level:

```python
sage: N = Y.normal_bundle(((3,),0))
sage: print(N)
Tautological class on Stratum: (1, 1)
with residue conditions: []
-1 * Psi class 1 with exponent 1 on level 0 * Graph ((3,), 0) +
```

Here, the \(V\)-graph is BIC number one. It has empty intersection with the third BIC. We may still perform the pullback and obtain the ZERO class:

```python
sage: print(Y.gen_pullback_taut(N, ((1,),0)))
Tautological class on Stratum: (1, 1)
with residue conditions: []
```

Note that we must use `gen_pullback_taut` if we want to pull back an ELG TautClass instead of an AdditiveGenerator!

Consider now the normal bundle \(N_0\) of the BIC number zero (the other compact type graph). Pulling this back to the third BIC, we obtain the normal bundle on the intersection:

```python
sage: N0=Y.normal_bundle(((0,),0))
sage: print(Y.gen_pullback_taut(N0, ((3,),0)))
Tautological class on Stratum: (1, 1)
with residue conditions: []
-1 * Psi class 1 with exponent 1 on level 0 * Graph ((0, 3), 0) +
-1 * Psi class 3 with exponent 1 on level 1 * Graph ((0, 3), 0) +
```

However, this is \(N_0\) pulled back to \((0, 3)\) *inside* BIC number two:

```python
sage: print(Y.gen_pullback_taut(N0, ((0,3),0)))
Tautological class on Stratum: (1, 1)
with residue conditions: []
2 * Psi class 1 with exponent 1 on level 0 * Psi class 3 with exponent 1 on level 1 * Graph ((0, 3), 0) +
```

The generalized pullback in \(Y\) is this pullback multiplied with \(N_0\) inside the BIC 0:

```python
sage: print(Y.intersection(Y.gen_pullback_taut(N0, ((0,3),0)),N0, ((0,),0)))
Tautological class on Stratum: (1, 1)
with residue conditions: []
2 * Psi class 1 with exponent 1 on level 0 * Psi class 3 with exponent 1 on level 1 * Graph ((0, 3), 0) +
```

### 4.3. Euler Characteristics and Caching

We illustrate the methods described above to implement [6, Thm. 1.3] for computing the Euler characteristics of strata. Recall from Section 1 that the (orbifold) Euler characteristic of the moduli space \(\mathcal{P}\Omega M_{g,n}(\mu)\) is the dimension-weighted sum over all level graphs \(\Gamma \in \text{LG}_B(B)\) without horizontal nodes of the product of the top power of the of the first Chern class \(\xi^{[i]}_1\) of the tautological bundle at each level. The equivalence of this formula and the one stated in [6, Eq. (2)], which involves \(\xi^{[i]}_{R^{(i)}}\) instead, follows from [6, Lemma 9.12], or equivalently from the discussion about clutching BICs of Section 3.3.

Since we already described how to construct all level graphs and how to compute top powers of level-wise tautological classes, the Euler-characteristic algorithm consists in a simple loop that sums over all levels of all enhanced levels graphs. The main computational issue is that boundary strata grow in size very quickly. Even for holomorphic strata in genus 3, these types of calculations would not be possible without extensive caching.
We already emphasized the use of enhanced profiles instead of graphs. This way, each graph and (essentially) also each additive generator is created only once and then reused. Therefore, AdditiveGenerators should always be created and used via X.additive_generator as this stores them in the _AGs dictionary of X and allows them to be reused (instead of being created newly on each call):

```python
sage: a=X.additive_generator(((0,),0))
sage: a is X.additive_generator(((0,),0))
True
sage: a is AdditiveGenerator(X, ((0,),0))
False
```

This allows computations in strata of genus 3 and 4 in feasible time. However, the memory footprint is considerable: already in genus 3, the larger strata use about 20GB, while in genus 4 already more than a TB is required.

As described in Section 3.1, diﬀstrata uses the package admcycles to evaluate top-degree ELGTautClasses. As the computations of admcycles are also very involved, we cache every use and, in fact, (attempt to) write any computed value into a local ﬁle that is automatically (attempted to be) reused.

More precisely, given the signature $\text{sig}$ of a stratum (note that admcycles works only with connected strata without residue conditions, cf. Section 3.1) and a $\psi$-polynomial $\text{psis}$ (as a dict mapping points of the stratum to exponents), the method adm_evaluate uses adm_key to compute a key consisting of the signature and $\text{psis}$ transformed into a tuple. To avoid recomputations, the signature is sorted and $\text{psis}$ renumbered accordingly:

```python
sage: from admcycles.diffstrata.levelstratum import adm_key
sage: adm_key((2,-2), {1: 2, 2: 1})
((-2, 2), ((1, 1), (2, 2)))
```

If the key exists in the cache, we return its value, otherwise we use admcycles to compute the value, store it in the cache and write this back into the ﬁle.

Similarly, whenever we evaluate a top-power of $\xi$, we cache this, as the Euler characteristic can be computed purely from the degeneration graph and this information. The $\xi$-cache uses LevelStratum's method dict_key to compute a key, again with the aim of performing as few evaluations as necessary.

**Example 4.2.** We illustrate the dict_keys in the stratum $\Omega M_2(1, 1)$. For the V-graph, the top stratum is disconnected and produces:

```python
sage: VT= Y.bics[1].top
sage: print(VT)
Product of Strata:
Signature((0,))
Signature((0,))
with residue conditions:
dimension: 3
leg dictionary: {1: (0, 0), 2: (1, 0)}
leg orbits: [[[1, 0), (0, 0)]]

sage: VT.dict_key()
(((0,), (0,)), ())
```

The bottom stratum has residue conditions:

```python
sage: VT= Y.bics[1].top
sage: print(VB)
Stratum: Signature((1, 1, -2, -2))
with residue conditions: [(0, 3), (0, 2)]
dimension: 0
leg dictionary: {3: (0, 0), 4: (0, 1), 5: (0, 2), 6: (0, 3)}
leg orbits: [[[0, 0)], [[0, 1), [0, 3), (0, 2)]]

sage: VB.dict_key()
(((0, 0),), (0, 1), (0, 3), (0, 2)))
```

By contrast, the bottom level of the banana graph produces:

```python
sage: VT= Y.bics[2].top
sage: print(BB)
Stratum: Signature((1, 1, -2, -2))
with residue conditions: [(0, 3), (0, 2)]
dimension: 1
leg dictionary: {3: (0, 0), 4: (0, 1), 5: (0, 2), 6: (0, 3)}
```
Example 4.3. The computation of the Euler characteristic of strata is implemented by `euler_characteristic`. We run this with an empty cache:

```
sage: print_adm_evals()
Stratum | Psis | eval
--------|------|------
```

```
sage: print_top_xis()
Stratum | Residue Conditions | xi^dim
--------|--------------------|------
```

```
sage: X=Stratum((4,))
sage: %time X.euler_characteristic()
CPU times: user 7.31 s, sys: 108 ms, total: 7.41 s
Wall time: 7.43 s
-55/504
```

Re-inspecting the cache, we see that it has been filled:

```
sage: print_top_xis()
Stratum | Residue Conditions | xi^dim
--------|--------------------|------
```

```
sage: print_adm_evals()
Stratum | Psis | eval
--------|------|------
```

---

**leg orbits**: $\{(0, 0), (0, 1), [(0, 3), (0, 2)]\}$

```
sage: BB.dict_key()
((-2, -2, 1, 1), ((0, 0), (0, 1),))
```

```
sage: euler_characteristic
```

---
Of course this affects any future calculations:

```python
sage: %time X.euler_characteristic()
CPU times: user 2.52 ms, sys: 5.39 ms, total: 7.91 ms
Wall time: 22.3 ms
-55/504
```

In fact, the cache was even used in the first calculation, as all levels appear already in the two and three-level graphs (see Remark 2.12).

**Example 4.4.** The Euler characteristic can also be computed by calculating the Chern character of the logarithmic cotangent bundle using [6, Thm. 1.2] and using Newton's identity to calculate the Chern polynomial. Of course, this is a longer calculation and there is less caching (top_xi_at_level is not called), but it may be used to check the consistency of the formulas:

```python
sage: X=Stratum((2,))
sage: X.top_chern_class().evaluate()
1/40
sage: X.euler_char()
-1/40
```

By the same method we can compute in general all Chern classes of the logarithmic cotangent bundle in terms of additive generators, even though as explained before, this takes more time than the Euler characteristic since the direct formula for that makes use of many cancelations happening for dimension reasons.

```python
sage: print(X.chern_class(1))
Tautological class on Stratum: (2,) with residue conditions: []
12 * \psi class 1 with exponent 1 on level 0 * Graph (((), 0) + -2 * Graph ((0,), 0) + -3 * Graph ((1,), 0) + -2 * Graph ((2,), 0) + -2 * Graph ((3,), 0) + -2 * Graph ((4,), 0) +
```

The result will be a tautological class of the correct codimension, written as the sum of \( \psi \)-classes on graphs.

### 4.4. Crosschecks

The module tests includes some more cross-checks and example computations using diffstrata. For example, leg_tests tests on each one-dimensional graph \( \Gamma \) of a stratum if the evaluation of the \( \xi^{[i]} \) at that level is the same (for every leg!) as the product of \( \Gamma \) with \( \xi \). Note that these expressions can be evaluated and the numbers compared.

The class BananaSuite tests the strata \( \Omega M_1(k, 1, -k - 1) \) (cf. [6, §10.3]) implementing the \( D \)-notation introduced there and including a method to test [6, Prop. 10.2].
Example 4.5. We illustrate the tests on some small strata:

```
sage: leg_test((4,))
Graph ((3, 6, 7, 2), 0): xi evaluated: 1/48 (dim of Level 0: 1)
level: 0, leg: 1, xi ev: 1/48
Graph ((3, 6, 7, 2), 1): xi evaluated: 1/24 (dim of Level 0: 1)
level: 0, leg: 1, xi ev: 1/24
Graph ((3, 6, 5, 2), 0): xi evaluated: 1/48 (dim of Level 0: 1)
level: 0, leg: 1, xi ev: 1/48
Graph ((3, 6, 5, 4), 0): xi evaluated: 1/48 (dim of Level 0: 1)
level: 0, leg: 1, xi ev: 1/48
sage: B=BananaSuite(2)
sage: B.check()
D(1,1)^2 = -1, RHS = -1
D(1,2)^2 = -1, RHS = -1
D(5,1)^2 = -3/2, RHS = -3/2
True
```

Finally, the method `commutativity_check` runs an extensive commutativity check on a stratum, i.e. multiplying products of BICs in various orders to give top-level classes and check that these evaluate to the same number, testing the normal bundle and intersection formulas along the way.

Funding

Research of the second and third author is supported by the DFG-project MO 1884/2-1 and by the LOEWE-Schwerpunkt ”Uniformisierte Strukturen in Arithmetik und Geometrie.”

Declaration of Interest

No potential conflict of interest was reported by the author(s).

References

[1] Bae, Y., Holmes, D., Pandharipande, R., Schmitt, J., Schwarz, R. (2020). Pixton’s formula and abel-jacobi theory on the picard stack.
[2] Bainbridge, M., Chen, D., Gendron, Q., Grushevsky, S., Möller, M. (2018). Compactification of strata of Abelian differentials. *Duke Math. J.* 167(12): 2347–2416.
[3] Bainbridge, M., Chen, D., Gendron, Q., Grushevsky, S., Möller, M. (2019). The moduli space of multi-scale differentials.
[4] Chan, M. (2012). Combinatorics of the tropical Torell map. *Algebra Number Theory*, 6(6): 1133–1169.
[5] Chen, D., Möller, M., Sauvaget, A., Zagier, D. (2020). Masur-Veech volumes and intersection theory on moduli spaces of Abelian differentials. *Invent. Math.* 222(1): 283–373.
[6] Costantini, M., Möller, M., Zachhuber, J. (2020). The chern classes and the euler characteristic of the moduli spaces of abelian differentials.
[7] Costantini, M., Möller, M., Zachhuber, J. (2020). diffrstrata – a sage package for calculations in the tautological ring of the moduli space of Abelian differentials. Long version, including manual.
[8] Delecroix, V., Schmitt, J., van Zelm, J. (2020). `admcyclus` – a Sage Package for Calculations in the Tautological Ring of the moduli Space of Stable Curves.
[9] Farkas, G., Pandharipande, R. (2018). The moduli space of twisted canonical divisors. *J. Inst. Math. Jussieu*, 17(3): 615–672.
[10] Holmes, D., Schmitt, J. (2019). Infinitesimal structure of the pluricanonical double ramification locus.
[11] Maggioni, S., Pagani, N. (2011). Generating stable modular graphs. *J. Symbolic Comput.*, 46(10): 1087–1097.
[12] McMullen, C. (2014). Moduli spaces in genus zero and inversion of power series. *Enseign. Math.* 60(1/2): 25–30.
[13] Mirzakhani, M., Zograf, P. (2015). Towards large genus asymptotics of intersection numbers on moduli spaces of curves. *Geom. Funct. Anal.* 25(4): 1258–1289.
[14] Möller, M., Ulirsch, M., Werner, A. (2021). Realizability of tropical canonical divisors. *J. Eur. Math. Soc. (JEMS)*, 23(1): 185–217.
[15] Sauvaget, A. (2019). Cohomology classes of strata of differentials. *Geom. Topol.*, 23(3): 1085–1171.
[16] Schmitt, J. (2018). Dimension theory of the moduli space of twisted k-differentials. *Doc. Math.* 23: 871–894.
[17] The Sage Developers. (2020). *SageMath, the Sage Mathematics Software System (Version 9.0)*. Available at: https://www.sagemath.org.