`diffstrata` – A SAGE PACKAGE FOR CALCULATIONS IN THE TAUTOLOGICAL RING OF THE MODULI SPACE OF ABELIAN DIFFERENTIALS

MATTEO COSTANTINI, MARTIN MÖLLER, AND JONATHAN ZACHHUBER

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 `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 in [CMZ20b].

Contents

1. 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 $\mathbb{P}\Xi\overline{\mathcal{M}}_{g,n}(\mu)$ for μ an integer partition of $2g-2$. The boundary is of a combinatorial nature, parameterised, for any μ, by finitely many labeled *level graphs* [BCGGM3]. 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 [CMZ20b] the tautological ring of $\mathbb{P}\Xi\overline{\mathcal{M}}_{g,n}(\mu)$ 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 `diffstrata` package provides a framework for calculations in the tautological ring of $\mathbb{P}\Xi\overline{\mathcal{M}}_{g,n}(\mu)$. It is implemented in `sage` [SageMath] and it is inspired by the package `admcycles` [DSZ20] for calculations in the tautological ring of $\overline{\mathcal{M}}_{g,n}$. However, due to the differences in the structure of the boundary, the implementation

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".

and interface of the two packages have only little in common. A key step in the evaluation process is performed by `admcycles`, though.

1.1. **Applications.** First, `diffstrata` may be used naively for basic inquiries about strata and the compactification constructed in [BCGGM3].

```
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
```

Second, several important invariants of strata can be computed via intersection theory. Recall from [CMZ20b, Theorem 1.3] 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 $\Gamma \in \mathrm{LG}_L(B)$ without horizontal nodes

$$\chi(B) \;=\; (-1)^d \sum_{L=0}^{d} \sum_{\Gamma \in \mathrm{LG}_L(B)} \ell_\Gamma N_\Gamma^\top \int_{D_\Gamma} \prod_{i=0}^{L} (\xi_\Gamma^{[i]})^{d_\Gamma^{[i]}}$$

of the product of the top power of the of the first Chern class $\xi_\Gamma^{[i]}$ of the tautological bundle at each level, where $d_\Gamma^{[i]}$ is the dimension of the projectivized moduli space at level i of the boundary stratum D_Γ, and where $d = \dim(B) = N - 1$. The following example contains the complete code required to calculate the (orbifold) Euler characteristic of $\mathbb{P}\Omega\mathcal{M}_2(2)$:

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

Another example for such a quantity is the Masur–Veech volume, that, according to [CMSZ19], can be written as

$$\mathrm{vol}\left(\Omega\mathcal{M}_{g,n}(m_1,\dots,m_n)\right) \;=\; -\frac{2(2i\pi)^{2g}}{(2g-3+n)!} \int_{\mathbb{P}\Xi\overline{\mathcal{M}}_{g,n}(\mu)} \xi^{2g-2} \cdot \prod_{i=1}^{n} \psi_i \,.$$

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

```
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 [CMSZ19] 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 κ-integrals via `admcycles`, which is much slower than specialised recursions as e.g. in [MZ11].

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 $\mathbb{P}\Xi\overline{\mathcal{M}}_{g,n}(\mu)$ 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 $\overline{\mathcal{M}}_g$ (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 $\mathbb{P}\Xi\overline{\mathcal{M}}_{g,n}(\mu)$ 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 ([CMZ20b]), and also the computation of Masur-Veech volumes ([CMSZ19]) 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 $\mathbb{P}\Xi\overline{\mathcal{M}}_{g,n}(\mu)$ poses a series of challenges not encountered in the boundary of $\overline{\mathcal{M}}_{g,n}$. Constructing all level graphs satisfying precisely the conditions of [BCGGM3, Def. 1.1] is a non-trivial problem. First, checking the validity of a graph requires checking the Global Residue Condition (GRC) discovered in [BCGGM1]. The corresponding algorithm is given in Proposition 2.2, which refines the combinatorial criterion of [MUW17]. Second, while generating a codimension-one degeneration of a stable graph inside $\overline{\mathcal{M}}_{g,n}$ 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 *generalised* 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 [FP18]).

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 $\mathbb{P}\Xi\overline{\mathcal{M}}_{g,n}(\mu)$ is abstractly defined as the smallest subring of the Chow ring that contains ψ-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 ψ-classes on each of its levels, see [CMZ20b, 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 `diffstrata` is that all standard operations in the intersection ring, computations of normal bundles, pullbacks and the implementation of the excess intersection formula for $\mathbb{P}\Xi\overline{\mathcal{M}}_{g,n}(\mu)$ [CMZ20b, §8] have to be dealt with simultaneously in an algorithm that terminates thanks to a dimension induction. This is in contrast with $\overline{\mathcal{M}}_g$ and `admcycles`, since normal bundles of boundary components in $\overline{\mathcal{M}}_g$ are plain ψ-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 `diffstrata` and giving a flavour 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 [CMZ20a].

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 build 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: 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
```

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 ([Cha12], [MP11]), but we are aware of a complete solution only in genus zero ([McM14]). It seems interesting to address the analogous problem in $\mathbb{P}\Xi\overline{\mathcal{M}}_{g,n}(\mu)$, to count the number of boundary

μ	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

TABLE 1. Performance of the BICs generation algorithm, run on a 2018 laptop.

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 `diffstrata`.[1]

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] computations.

2. Generalised 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 clutching 2-level graphs. These graphs appear as BICs boundary components of level strata, which in general parametrize meromorphic differentials on disconnected curves and with residue conditions. We begin by briefly recalling the notions from [BCGGM3] and [CMZ20b] in the generality that we here require.

[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.

2.1. **Strata with residue conditions.** To obtain a recursive structure on the boundary of $\mathbb{P}\Xi\overline{\mathcal{M}}_{g,n}(\mu)$, recall the definition of a *generalised stratum*, introduced in [CMZ20b, §4] to cover the case of a level of an enhanced level graph. More precisely, we allow differentials on disconnected surfaces: denote by $\mu_i = (m_{i,1}, \dots, m_{i,n_i}) \in \mathbb{Z}^{n_i}$ 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 $i = 1, \dots, k$. Then we define, for a tuple $\mathbf{g} = (g_1, \dots, g_k)$ of genera and a tuple $\mathbf{n} = (n_1, \dots, n_k)$ together with $\boldsymbol{\mu} = (\mu_1, \dots, \mu_k)$, the disconnected stratum

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

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

Moreover, we consider subspaces of these *cut out by residue conditions*. More precisely, denote by $H_p \subseteq \cup_{i=1}^{k}\{(i,1), \cdots (i,n_i)\}$ the subset of the marked points such that $m_{i,j} < -1$. Now consider vector spaces $\mathfrak{R}$ of the following special shape, modelled on the global residue condition from [BCGGM1]: for λ a partition of H_p, with parts denoted by $\lambda^{(k)}$, and a subset $\lambda_{\mathfrak{R}}$ of the parts of λ, we define the $\mathbb{C}$-vector space

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

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

In [CMZ20b, Prop. 4.2] a modular compactification $\mathbb{P}\Xi\overline{\mathcal{M}}^{\mathfrak{R}}_{\mathbf{g},\mathbf{n}}(\boldsymbol{\mu})$ of $\mathbb{P}\Omega\mathcal{M}^{\mathfrak{R}}_{\mathbf{g},\mathbf{n}}(\boldsymbol{\mu})$ is constructed in analogy to [BCGGM3]. Consequently, the boundary components are parametrised 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* $g_i \colon V_i \to \mathbb{Z}_{\geq 0}$, *legs* H_i that are associated to the vertices by a *vertex map* $v_i \colon H_i \to V_i$ and come with an involution $\iota_i \colon H_i \to H_i$, the two-cycles of which form the *oriented edges* $E_i \subseteq H_i \times H_i$ while the fixed points (denoted H_i^m) are in bijection with the n_i marked points. The k graphs $\Gamma_i = (V_i, H_i, E_i, g_i, v_i, \iota_i)$ 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 $o \colon \bigsqcup H_i^m \to \boldsymbol{\mu}$ associating to each marked point (i,j) the order $m_{i,j}$ of the differential.

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

Moreover, an *enhancement* is a map $\kappa \colon E \to \mathbb{Z}_{\geq 0}$ 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

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

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

The *enhanced level graph* then consists of the triple (Γ, ℓ, κ). We denote the corresponding boundary component of $\mathbb{P}\Xi\overline{\mathcal{M}}^{\mathfrak{R}}_{\mathbf{g},\mathbf{n}}(\boldsymbol{\mu})$ by D_Γ. 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 [BCGGM3, §2] is simply the requirement that the orders of zeros and poles sum to $2g_{i,j} - 2$ on each vertex. See [BCGGM3, §2] and [CMZ20b, §3.2] for details.

To determine, for a generalised stratum $\mathbb{P}\Xi\overline{\mathcal{M}}^{\mathfrak{R}}_{\mathbf{g},\mathbf{n}}(\boldsymbol{\mu})$, which enhanced level graphs give non-empty boundary components, we recall the $\mathfrak{R}$-GRC from [CMZ20b, §4] using notation from (1): Starting with an enhanced level graph Γ, we construct a new *auxiliary level graph* $\widetilde{\Gamma}$ by adding, for each $\lambda^{(k)} \in \lambda_{\mathfrak{R}}$, a new vertex $v_{\lambda^{(k)}}$ to Γ at level ∞ 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 Γ satisfies the *$\mathfrak{R}$-global residue condition ($\mathfrak{R}$-GRC)* if the tuple of residues at the legs in H_p belongs to $\mathfrak{R}$ and for every level $J < \infty$ of $\widetilde{\Gamma}$ and every connected component Y of the subgraph $\widetilde{\Gamma}_{>J}$ one of the following conditions holds.

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

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

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

By [CMZ20b, Prop. 4.2], the boundary components D_Γ of $\mathbb{P}\Xi\overline{\mathcal{M}}^{\mathfrak{R}}_{\mathbf{g},\mathbf{n}}(\boldsymbol{\mu})$ are parametrised by enhanced level graphs (Γ, ℓ, κ) satisfying the $\mathfrak{R}$-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 [MUW17]. The following proposition strips the tropical language away in the criterion [MUW17, Theorem 1] and generalises it to meromorphic strata with residue conditions.

Let (Γ, ℓ, κ) be an enhanced level graph. We call a vertex v of Γ *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 $\mathfrak{p}(v)$ 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 $p \in \mathfrak{p}(v)$ such that

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

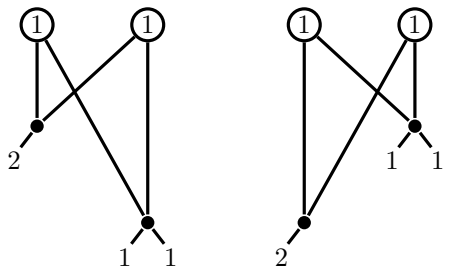


FIGURE 1. An illegal (left) and legal (right) graph in the boundary of $\Omega\mathcal{M}_3(2,1,1)$.

Proposition 2.2. The boundary stratum D_Γ of $\mathbb{P}\Xi\overline{\mathcal{M}}^{\mathfrak{R}}_{\mathbf{g},\mathbf{n}}(\boldsymbol{\mu})$ associated with the enhanced level graph (Γ,ℓ,κ) is non-empty if and only if both of the following conditions are satisfied

(1) For every inconvenient vertex v of Γ 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_p whose residue is not constrained to zero for all elements of $\mathfrak{R}$.

(2) For every horizontal edge e of Γ 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_p whose residue is not constrained to zero for all elements of $\mathfrak{R}$.

Proof. Consider first the case that $\widetilde{\Gamma}$ is connected. We view $\widetilde{\Gamma}$ as an enhanced level graph of an auxiliary stratum $\widetilde{X}$ as follows: we add prongs to the new edges of $\widetilde{\Gamma}$ in accordance with the pole orders of the half-edges on Γ. For each vertex v at level ∞, 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 $\widetilde{X}$, we are now reduced to the situation of [MUW17, Theorem 1].

In a product of strata, clearly a graph is admissible if and only if each component is admissible. □

Example 2.3. Consider the graphs in the boundary of the stratum $\mathbb{P}\Xi\overline{\mathcal{M}}_{3,3}((2,1,1))$ 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 undegeneration of the two graphs depicted in Figure 2. Note that there are residue conditions intertwining the simple zeros on

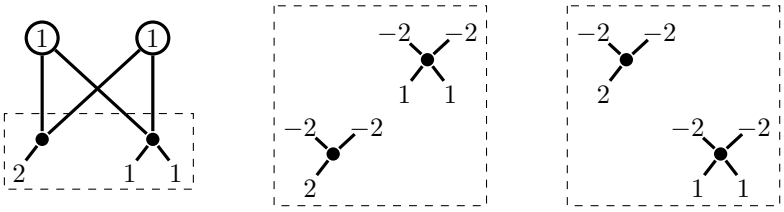


FIGURE 2. The "zigzag-graph" (left) in the boundary of the stratum $\Omega\mathcal{M}_3(2,1,1)$ and a legal (center) and illegal (right) degeneration of its bottom level.

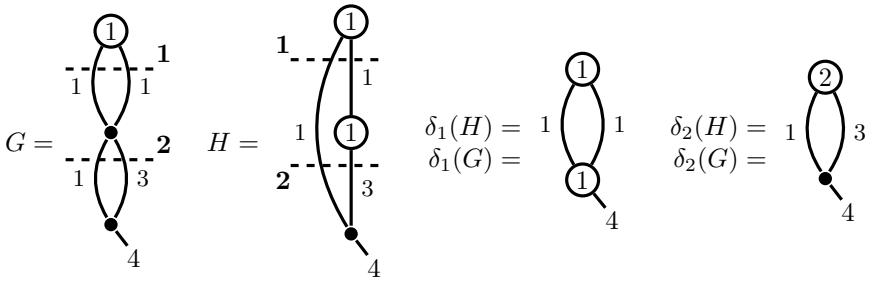


FIGURE 3. An example of a reducible profile: the two non-isomorphic three-level graphs G and H have the same two-level graphs as undegenerations. The level crossings are denoted by dotted lines.

the two components. These imply that the middle graph is legal, while the right one is not.

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 δ_i [CMZ20b, §3.3], contracting all level crossing of an enhanced level graph Γ without horizontal edges except for the i-th level crossing, yielding a two-level graph. The boundary component defined by Γ 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 (Γ, ℓ, κ) be an enhanced level graph with L levels and without horizontal edges. We define the *profile* of Γ to be the tuple $(\delta_1(\Gamma), \ldots, \delta_L(\Gamma))$. The *enhanced profile* of Γ is its profile together with an assignment of a number recovering uniquely Γ 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 $\Omega\mathcal{M}_3(4)$ depicted in Figure 3.

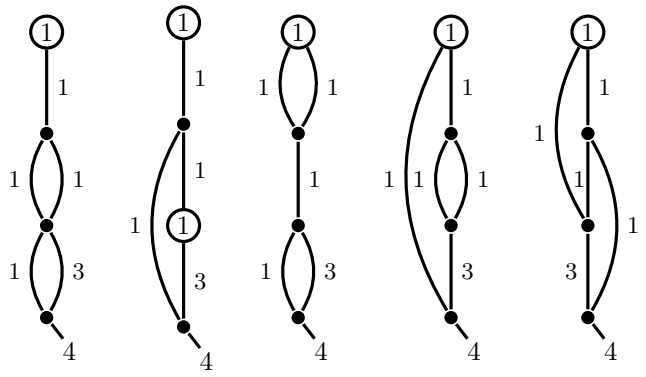


FIGURE 4. The four-level graphs degenerating from the *reducible* profile consisting of the graphs G and H above. Note that the left two graph share a (reducible) profile, as do the right two. The middle graph, however, is unique inside its *irreducible* profile. Observe also that the right two graphs cannot be distinguished by their levels.

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

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 generalised stratum, denote by $\mathrm{BIC}(X)$ the (non-horizontal) two-level graphs parametrizing the boundary divisors of X.

Definition 2.6. Let X be a generalised stratum. For each $\Gamma \in \mathrm{BIC}(X)$ we define:

(1) the generalised strata $X_\Gamma^\top$ and $X_\Gamma^\perp$, the top and bottom levels of the divisor D_Γ defined by Γ;
(2) a map $\beta_\Gamma^\top \colon \mathrm{BIC}(X_\Gamma^\top) \to \mathrm{BIC}(X)$ that associates to a 2-level graph $\Gamma' \in \mathrm{BIC}(X_\Gamma^\top)$ the graph $\delta_1(\Lambda) \in \mathrm{BIC}(X)$ where Λ is the graph obtained by clutching Γ' to Γ;
(3) a map $\beta_\Gamma^\perp \colon \mathrm{BIC}(X_\Gamma^\perp) \to \mathrm{BIC}(X)$ that associates to a 2-level graph $\Gamma' \in \mathrm{BIC}(X_\Gamma^\perp)$ the graph $\delta_2(\Lambda) \in \mathrm{BIC}(X)$ where Λ is the graph obtained by clutching Γ' to Γ.

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 generalised stratum and $\Gamma, \Gamma' \in \mathrm{BIC}(X)$.

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

Proof. The proof follows directly from [CMZ20b, 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. □

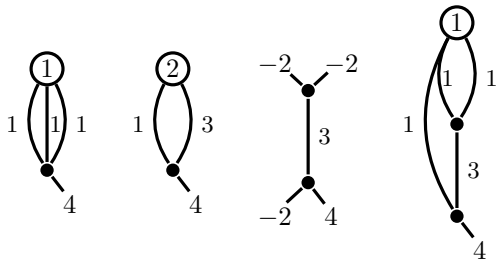


FIGURE 5. From left to right: the "triple banana"; the "asymmetric banana"; the compact type BIC inside $\Omega\mathcal{M}_0(4,-2,-2,-2)$, the bottom level of the "triple banana"; the unique graph in the intersection of the triple and asymmetric banana.

As a consequence, we may define a partial order $\prec$ on $\mathrm{BIC}(X)$ by defining $\Gamma \prec \Gamma'$ if and only if (Γ', Γ) 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 (Γ_1, Γ_2) is reducible, the corresponding maps $\beta^{\top}_{\Gamma_2}$ and $\beta^{\perp}_{\Gamma_1}$ 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 generalised 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 $\Omega\mathcal{M}_3(4)$. The bottom level is the stratum $\Omega\mathcal{M}_0(4,-2,-2,-2)$, 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 $\beta^{\perp}$ 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 generalised stratum.

(1) The number of enhanced 2-level graphs in $\mathrm{BIC}(X)$ is finite and the graphs can be listed explicitly.
(2) All non-empty profiles in X can be constructed recursively from $\mathrm{BIC}(X)$.
(3) All graphs inside a profile can be constructed explicitly from the profile's components.

Proof. The generation of $\mathrm{BIC}(X)$ 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 GRC is checked for each graph and these are then combined to give all possible BICs for a Generalised Stratum. Finally, they are sorted by isomorphism class and checked against the $\mathfrak{R}$-GRC. 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 $(\Gamma_1, \dots, \Gamma_l)$ of length l may be extended to a non-empty profile $(\Gamma_0, \Gamma_1, \dots, \Gamma_l)$ if and only if $\Gamma_1 \prec \Gamma_0$. Indeed, $\Gamma_1^\top$ is also the top level of any graph Λ in $(\Gamma_1, \dots, \Gamma_l)$ and thus a preimage $(\beta^\top_{\Gamma_1})^{-1}(\Gamma_0)$ exists in $\mathrm{BIC}(X^\top_{\Gamma_1})$ and can be clutched to Λ to yield a graph in $(\Gamma_0, \Gamma_1, \dots, \Gamma_l)$. Similarly, the profile may be extended to $(\Gamma_1, \dots, \Gamma_l, \Gamma_{l+1})$ if and only if $\Gamma_{l+1} \prec \Gamma_l$.

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 β is non-injective (we clutch each preimage with each other graph). □

Next, we provide an efficient algorithm for listing $\mathrm{BIC}(X)$. 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 $\mathbf{g}^\perp_{\max}$ and the minimal sum of genera of vertices on top level by $\mathbf{g}^\top_{\min}$. As every top-level vertex v with $g_v = 0$ requires at least one pole, $\mathbf{g}^\perp_{\max} = g - 1$ and $\mathbf{g}^\top_{\min} = 1$ for holomorphic strata and correspondingly $\mathbf{g}^\perp_{\max} = g$ and $\mathbf{g}^\top_{\min} = 0$ for meromorphic strata.

Algorithm 2.11 (BIC Generation)**.**

- **Step 1:** We begin by iterating over the possible number $\mathbf{v}^\perp$ 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 $\mathbf{v}^\perp$.
- **Step 2:** Next, we distribute the zeros between upper and lower level by iterating over all 2-length partitions of $\mathbf{z}$ (and also include the case where all zeros are on bottom level), ensuring at each step that we have enough zeros to satisfy $\mathbf{v}^\perp$.
- **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 $\mathbf{g}^\top$ from the top vertices, $\mathbf{g}^\perp$ from the bottom vertices and the graph. For this, we iterate over $\mathbf{g}^\top$ and $\mathbf{g}^\perp$ (using the bounds $\mathbf{g}^\perp_{\max}$ and $\mathbf{g}^\top_{\min}$).
- **Step 5:** $\mathbf{g}^\perp$ 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 $\mathbf{p}$. 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, $\mathbf{v}^\top$, which is bounded by the sum of $\mathbf{g}^\perp$ 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 generalised 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 Γ, level l of Γ remains unchanged by contracting any level crossing not adjacent to l. Contracting all non-adjacent 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 [CMZ20b, Theorem 1.5], any tautological class may be expressed as a formal sum of additive generators given by products of ψ-classes supported on single level graphs.

Let X be a generalised 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 [CMZ20b, 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 ψ-products on meromorphic strata and using the `admcycles` package [DSZ20] 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 $\overline{\mathcal{M}}_{g,n}$ by Sauvaget [Sau19]. For this purpose, the stack factor $\mathfrak{sf}$ of an additive generator is defined as follows: let G be the associated graph, then $\mathfrak{sf}$ is the quotient of the product of the prongs of G and the product of ℓ_B for every B appearing in the profile of G and the number of automorphisms of G. For a BIC B, the number ℓ_B is the lcm of the prongs of B. By [CMZ20b, Lemma 9.12] this factor is necessary to convert an integral over a boundary stratum D_Γ 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 ψ-classes by level and taking the product of the evaluations of these ψ-polynomials on each level. This product is then multiplied with the stack factor.
- **Step 3:** If the residue space $\mathfrak{R}$ is empty, 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 $\overline{\mathcal{M}}_{g,n}$ and integrate this against the ψ-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 $\widetilde{\Gamma}$), it evaluates to 0, since the fiber dimension to the product of moduli spaces is positive and the ψ-classes are pullbacks from there.

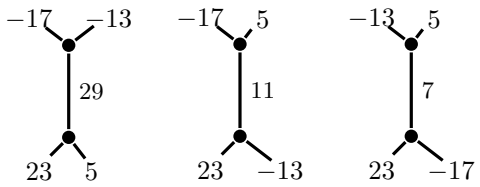


FIGURE 6. The boundary divisors Γ_1, Γ_2 and Γ_3 of the stratum $\mathbb{P}\Xi\overline{\mathcal{M}}_{0,4}(23, 5, -13, -17)$.

Otherwise, we create a new Generalised Stratum with one residue condition removed. We repeat until this condition is non-trivial (or $\mathfrak{R} = \emptyset$) and evaluate the product of the (original) ψ-expression and the class cut out by this residue condition (using [CMZ20b, Prop. 8.3]).

The evaluations in Step 3 are based on the algorithms in [DSZ20]. In particular the `Strataclass` function is based on the description of fundamental classes of (non-generalised) strata conjectured in [FP18] and [Sch18] and proven recently in [BHPSS20] based on results from [HS19].

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 Γ_1, Γ_2 and Γ_3 (see Figure 6) that have respectively the marked points of order $(23, 5)$, $(23, -13)$ and $(23, -17)$ on lower level. By [CMZ20b, Prop. 8.2] we can express ξ using the first ψ-class as $\int_X \xi = 24 - 29 - 11 - 7 = -23$. Now in [CMZ20b, Prop. 8.3] the contributing boundary graphs are Γ_2 (because the point with order -13 is on lower level) and Γ_3 (because the zero residue at the point of order -13 on upper level is automatic), but not Γ_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 [CMZ20b, Prop. 8.1].

In `diffstrata`, we implement the general version of the excess intersection formula [CMZ20b, eq. (61)].

Let Λ_1 and Λ_2 be degenerations of a k-level graph Γ. A level graph Π is *a* (Λ_1, Λ_2)*-graph* if there are undegeneration morphisms $\rho_i \colon \Pi \to \Lambda_i$, 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_Π, D_{Λ_1}, D_{Λ_2} and D_Γ, as in the following diagram:

$$\begin{array}{ccc} D_\Pi & \xrightarrow{\mathrm{j}_{\Pi,\Lambda_2}} & D_{\Lambda_2} \\ \downarrow{\scriptstyle \mathrm{j}_{\Pi,\Lambda_1}} & & \downarrow{\scriptstyle \mathrm{j}_{\Lambda_2,\Gamma}} \\ D_{\Lambda_1} & \xrightarrow{\mathrm{j}_{\Lambda_1,\Gamma}} & D_\Gamma \end{array}$$

Furthermore, we define $\nu^\Pi_{(\Lambda_1\cap\Lambda_2)/\Gamma}$ to be the product of the pullback to D_Π of the normal bundles $\mathcal{N}_{\Gamma',\Gamma}$ where Γ' ranges over all $k+1$-level graphs Γ' that are a degeneration of Γ and that are moreover common to Λ_1 and Λ_2, see Section 3.3. For any $\alpha \in \mathrm{CH}^\bullet(D_{\Lambda_2})$ we can then express its push-forward pulled back to Λ_1 as

$$\mathrm{j}^*_{\Lambda_1,\Gamma}\mathrm{j}_{\Lambda_2,\Gamma *}\alpha \;=\; \sum_\Pi \mathrm{j}_{\Pi,\Lambda_1,*}\left(\nu^\Pi_{(\Lambda_1\cap\Lambda_2)/\Gamma}\cdot \mathrm{j}^*_{\Pi,\Lambda_2}\alpha\right),$$

where the sum ranges over all (Λ_1,Λ_2)-graphs Π. This corresponds to the product of α with the class of Λ_2 in $\mathrm{CH}(D_\Gamma)$. For more details, see [CMZ20b, §8.1].

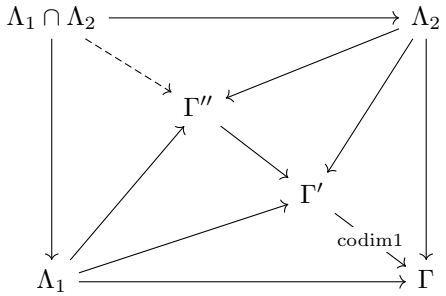
Of course, the product of two additive generators on the same graph is simply the product of their ψ-polynomials and the product of two tautological classes is the sum of the products of their additive generators.

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 [CMZ20b, Thm 7.1]. In `diffstrata`, this can be computed using `normal_bundle`:

```
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 $\nu^\Pi_{(\Lambda_1\cap\Lambda_2)/\Gamma}$. The situation is summarised in the following diagram of the involved graphs:



where the arrows represent undegeneration maps. The graph Γ'' is the *minimal common undegeneration* of Λ_1 and Λ_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) Γ' are codimension one degenerations of Γ that are common undegenerations of Λ_1 and Λ_2.

For the excess intersection formula, we need the product of the normal bundles $\mathcal{N}_{\Gamma',\Gamma} = \mathcal{N}_{D_{\Gamma'}/D_\Gamma}$ where Γ' is a $k+1$-level degeneration of the k-level graph Γ. More precisely, the normal bundles $\mathcal{N}_{D_{\Gamma'}/D_\Gamma}$ are pulled back to $D_{\Gamma''}$ and the product is computed in $\mathrm{CH}(D_{\Gamma''})$. The normal bundle is computed in [CMZ20b, Prop. 7.5]:

$$c_1(\mathcal{N}_{\Gamma',\Gamma}) \;=\; \frac{1}{\ell_{\delta_i(\Gamma')}}\big(-\xi'^{[i]}_\Gamma - c_1(\mathcal{L}'^{[i]}_\Gamma) + \xi'^{[i+1]}_\Gamma\big) \quad \text{in} \quad \mathrm{CH}^1(D_{\Gamma'})\,,$$

where

$$\mathcal{L}^{[i]}_{\Gamma'} \;=\; \mathcal{O}_{D'_\Gamma}\Big(\sum_{\Gamma' \overset{[i]}{\rightsquigarrow} \widehat{\Delta}} \ell_{\delta_{i+1}(\widehat{\Delta})} D_{\widehat{\Delta}}\Big)\,,$$

where the sum runs over all graphs $\widehat{\Delta} \in \mathrm{LG}_{k+2}(\overline{B})$ that yield divisors in D'_Γ by splitting the i-th level. These are then pulled back to $D_{\Gamma''}$, see Section 3.4, and then they are multiplied in $\mathrm{CH}(D_{\Gamma''})$. 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 $\mathrm{CH}(D_\Gamma)$ has been transformed to a product in $\mathrm{CH}(D_{\Gamma''})$ 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)**.**

- **Step 1:** Compute the minimal common undegeneration Γ''.
- **Step 2:** If Γ'' is Γ the intersection is transversal, we are in the base case.
- **Step 3:** Loop through the codimension one common undegenerations Γ' of Γ''.
- **Step 4:** Calculate the level i where Γ' and Γ'' differ.
- **Step 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 Γ'', cf. Section 3.4.
- **Step 7:** Return the product of the normal bundles (one for each Γ') inside Γ''.

We explain how to compute $\xi^{[i]}_\Gamma$ and $\mathcal{L}^{[i]}_\Gamma$, which was required in the fifth step of the previous algorithm. These objects are computed by clutching the pull back of the corresponding classes $\xi_{B^{[i]}_\Gamma}$ and $\mathcal{L}_{B^{[i]}_\Gamma}$ which are elements of the Chow ring of the level stratum $B^{[i]}_\Gamma$. Even though, in contrast to the moduli space of curves, there is no clutching morphism from the product of the level strata $B^{[i]}_\Gamma$ to D_Γ, there is a correspondence between the product of level strata and the associated boundary component. The details are explained in [CMZ20b, Sec. 4.2].

It remains to explain only how to "glue" or clutch a tautological class on $B^{[i]}_\Gamma$ into D_Γ, in order to obtain an element of the Chow ring of the original stratum X. Since we are only interested in $\xi_{B^{[i]}_\Gamma}$ and $\mathcal{L}_{B^{[i]}_\Gamma}$, we can focus on clutching ψ-classes and divisors, and ignore classes of higher codimension.

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

Consider the situation that Γ' is a BIC in $B_\Gamma^{[i]}$. We want to clutch this in, obtaining a graph Λ in X with one more level than Γ. However, the implementation of this is rather delicate: in light of Remark 2.12, we can realise $B_\Gamma^{[i]}$ as a level in a BIC or three-level graph and for these we can use the maps β 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 β. See [CMZ20a] for more examples and details on how this is resolved.

We summarise how to determine the enhanced profile of the one-level degeneration Λ of Γ:

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

Step 1: Determine the new profile. This is given by the degeneration graph via the maps β of Definition 2.6, depending on the location of $B_\Gamma^{[i]}$.
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 [CMZ20b, 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 Λ' is the BIC of X that extended the profile of Γ and $\ell_{\Gamma'}$ of the BIC Γ' of $B_\Gamma^{[i]}$ 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 Γ and Γ'.

3.4. **Pulling back classes.** To calculate the pullback of an additive generator, we consider first the base case and then the generalised case.

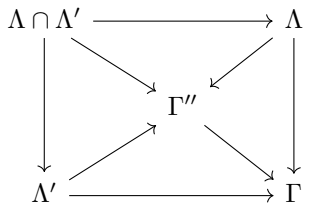
Consider a graph Λ in X, α an additive generator on corresponding to a class on D_Λ and a degeneration Π of Λ, i.e. we obtain Λ by contracting some of the level-crossings of Π. Then there are finitely many contraction morphisms $\rho\colon \Pi \to \Lambda$ and each of these gives a well-defined pullback map of α. 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: 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 Λ be a graph in X and α a class on D_Λ, but now we do not require the "target" graph Λ' to be a degeneration of Λ. Instead, we will pull α back to the intersection $\Lambda \cap \Lambda'$:



Note that this must include the normal bundle contribution of the minimal common undegeneration Γ'' of Λ and Λ' (in D_Γ).

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

Algorithm 3.7 (Pullback)**.**

Step 1: Compute the common normal bundle of α and Λ' in D_Γ (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 $\mathrm{CH}(\Lambda \cap \Lambda')$) the pullback to each graph of $\Lambda \cap \Lambda'$ with the normal bundle, with ambient Γ''.

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

Installation. The package `diffstrata` is included with `admcycles` version 1.1 or greater. See [DSZ20] 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!

4.1. 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 `EmbeddedLevelGraph` 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) +

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 `ELGTautClass`, which is in turn a sum of `AdditiveGenerators`. Each `AdditiveGenerator` corresponds to a ψ-monomial on a graph and thus carries the information of an enhanced profile and a `leg_dict` encoding the ψ-powers: every ψ-class is associated to a marked point of a level [CMZ20b, Theorem 1.5], i.e. a leg of the graph. A ψ-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 ψ-class associated to this leg. For example, we saw above that `X.bics[0]` is the compact-type graph in the boundary of $\Omega\mathcal{M}_2(2)$. We see from the `LevelGraph` that the marked point is at leg `2`, the ψ-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 ψ-class on the stratum with the class of the graph:

```
sage: A == X.psi(1) * X.additive_generator(((0,), 0))
True
```

Moreover, when squaring, e.g., the class of a graph, a normal bundle contribution appears:

```
sage: print(X.additive_generator(((0,), 0))^2)
Tautological class on Stratum: (2,)
with residue conditions: []
```

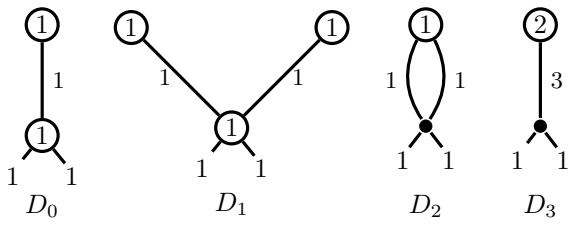


FIGURE 7. The boundary divisors of the principal stratum $\mathbb{P}\Xi\overline{\mathcal{M}}_{2,2}(1,1)$: the compact type with genus 1 on top, the V-graph, the banana graph and the compact type with genus 2 on top.

```
-1 * Psi class 1 with exponent 1 on level 0 * Graph ((0,), 0) +
-1 * Psi class 3 with exponent 1 on level 1 * Graph ((0,), 0) +
```

In the formulas for the Euler characteristic [CMZ20b], the class $\xi = c_1(\mathcal{O}(-1))$ of the tautological bundle and its restriction $\xi_{B_\Gamma^{[i]}}$ to a level i of a graph Γ were key. For a stratum, the class ξ is easily accessible:

```
sage: print(X.xi)
Tautological class on Stratum: (2,)
with residue conditions: []

3 * Psi class 1 with exponent 1 on level 0 * Graph ((), 0) +
-1 * Graph ((0,), 0) +
-1 * Graph ((1,), 0) +
```

Moreover, it is not difficult to compute $\xi_{B_\Gamma^{[i]}}$ (here for the top-level of the compact-type graph):

```
sage: print(X.xi_at_level(0, ((0,),0)))
Tautological class on Stratum: (2,)
with residue conditions: []

1 * Psi class 1 with exponent 1 on level 0 * Graph ((0,), 0) +
```

We end this section by illustrating, in `diffstrata`, the methods explained in Section 3.

Example 4.1. Consider the stratum $\Omega\mathcal{M}_2(1,1)$ and let us look at the boundary divisors (see Figure 7):

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

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

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 `^` are performed in the Chow ring of the stratum. We can instead work in the Chow ring of any boundary stratum D_Γ by specifying as `ambient` the enhanced profile of Γ. As the stratum `Y` is four-dimensional, we can multiply this graph with ξ^3 to obtain a top-degree class that we may evaluate to find a number:

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

This matches the observation that the top level is the minimal stratum in genus two where ξ^3 evaluates to $-\frac{1}{640}$, the bottom level is a point and there are no prongs or automorphisms involved, compare [CMZ20b, §4.3 and Lemma 9.12]:

```
sage: Y.bics[3].top
LevelStratum(sig_list=[Signature((2,))],res_cond=[],leg_dict={1: (0, 0)})
sage: (Y.bics[3].top.xi^3).evaluate()   # calculating on Y.bics[3].top
-1/640
```

We can perform the same calculation in `Y` using `xi_at_level`. However, as the class $\xi_\Gamma^{[i]}$ correspoding to `xi_at_level` lives not on `Y` but on the BIC `3`, performing the normal multiplication with `*` or `^` will not yield the correct result: we must use our BIC Γ as the ambient stratum, as we want to perform the multiplication in $\mathrm{CH}(D_\Gamma)$, 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 `top_xi_at_level` which computes and evaluates the top-power of ξ:

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

In fact, whenever possible this method should be used (even for ξ 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 $\mathcal{M}_2(2)$. 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) +
-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 ψ-class on top-level:

```
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:

```
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 `ELGTautClass` instead of an `AdditiveGenerator`!

Consider now the normal bundle `N0` 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:

```
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 `N0` pulled back to `(0, 3)` *inside* BIC number two:

```
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) +

sage: print(Y.gen_pullback_taut(N0, ((0,3),0), ((0,),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) +
```

The generalised pullback in `Y` is this pullback multiplied with `N0` inside the BIC `0`:

```
sage: print(Y.intersection(Y.gen_pullback_taut(N0, ((0,3),0), ((0,),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) +
```

4.2. **Euler Characteristics and Caching.** We illustrate the methods described above to implement [CMZ20b, Thm. 1.3] for computing the Euler characteristics of strata. Recall from Section 1 that the (orbifold) Euler characteristic of the moduli space $\mathbb{P}\Omega\mathcal{M}_{g,n}(\mu)$ is the dimension-weighted sum over all level graphs $\Gamma \in \mathrm{LG}_L(B)$ without horizontal nodes of the product of the top power of the of the first Chern class $\xi_\Gamma^{[i]}$ of the tautological bundle at each level. The equivalence of this formula and the one stated in [CMZ20b, Eq. (2)], which involves $\xi_{B_\Gamma^{[i]}}$ instead, follows from [CMZ20b, 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-characterstic 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, `AdditiveGenerator`s 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):

```
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 `sig` of a stratum (note that `admcycles` works only with connected strata without residue conditions, cf. Section 3.1) and a ψ-polynomial `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 `psis` transformed into a `tuple`. To avoid recomputations, the signature is sorted and `psis` renumbered accordingly:

```
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 ξ, we cache this, as the Euler characteristic can be computed purely from the degeneration graph and this information. The ξ-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_key`s in the stratum $\Omega\mathcal{M}_2(1,1)$. For the V-graph, the top stratum is disconnected and produces:

```
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:

```
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()
(((-2, -2, 1, 1),), (((0, 0),), ((0, 1),)))
```

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

```
sage: VT= Y.bics[2].top
sage: print(BB)
Stratum: Signature((1, 1, -2, -2))
with residue conditions: [(0, 2), (0, 3)]
dimension: 1
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: BB.dict_key()
(((-2, -2, 1, 1),), (((0, 0), (0, 1)),))
```

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
-------------------------------------------------------------------
(-4, -2, 4)        | [(0, 0), (0, 1)]              | 1
(-4, 0, 2)         | [(0, 0)]                      | 1
(-4, 1, 1)         | [(0, 0)]                      | 1
(-4, 4)            | [(0, 0)]                      | -15/8
(-3, -3, 4)        | [(0, 0), (0, 1)]              | 1
(-2, -2, -2, 4)    | [(0, 0), (0, 1), (0, 2)]      | -4
(-2, -2, -2, 4)    | [[(0, 0), (0, 2)], [(0, 1)]] | 1
```

```
(-2, -2, 0, 2)     | [(0, 0), (0, 1)]             | -2
(-2, -2, 1, 1)     | [[(0, 0)], [(0, 1)]]         | 1
(-2, -2, 1, 1)     | [(0, 0), (0, 1)]             | -1
(-2, -2, 2)        | [(0, 0), (0, 1)]             | 1
(-2, -2, 4)        | [[(0, 0)], [(0, 1)]]         | -11/12
(-2, -2, 4)        | [(0, 0), (0, 1)]             | 13/8
(-2, 0, 0)         | [(0, 0)]                     | 1
(-2, 0, 0, 0)      | [(0, 0)]                     | -1
(-2, 0, 2)         | [(0, 0)]                     | 1/8
(-2, 1, 1)         | [(0, 0)]                     | 0
(-2, 2)            | [(0, 0)]                     | -1/8
(-2, 4)            | [(0, 0)]                     | -23/1152
(0,)               | ()                           | 1/24
[(0,), (-2, 0, 0)] | [(1, 0)]                     | -1/24
[(0,), (0,)]       | ()                           | -1/576
[(0,), (0, 0)]     | ()                           | 0
(0, 0)             | ()                           | 0
(0, 0, 0)          | ()                           | 0
(0, 2)             | ()                           | 0
(1, 1)             | ()                           | 0
(2,)               | ()                           | -1/640
(4,)               | ()                           | 305/580608
sage: print_adm_evals()
Stratum            | Psis                         | eval
-------------------------------------------------------------------
(0,)               | {1: 1}                       | 1/24
(-2, 2)            | {1: 1}                       | 1/8
(-2, 0, 0, 0)      | {1: 1}                       | 1
(-2, -2, -2, 4)    | {1: 1}                       | 1
(-2, -2, 1, 1)     | {1: 1}                       | 1
(-4, 4)            | {1: 1}                       | 5/8
(-2, -2, 0, 2)     | {1: 1}                       | 1
(0, 0)             | {1: 1, 2: 1}                 | 1/24
(-2, -2, 4)        | {1: 1, 2: 1}                 | 11/12
(-2, 0, 2)         | {1: 1, 2: 1}                 | 1/4
(-2, 1, 1)         | {1: 1, 2: 1}                 | 1/6
(0, 0, 0)          | {1: 1, 2: 1, 3: 1}           | 1/12
(-2, 4)            | {1: 1, 2: 2}                 | 73/1152
(0, 0, 0)          | {1: 1, 2: 2}                 | 1/12
(1, 1)             | {1: 1, 2: 3}                 | 1/720
(-2, -2, 4)        | {1: 1, 3: 1}                 | 11/12
(0, 0)             | {1: 2}                       | 1/24
(-2, -2, 4)        | {1: 2}                       | 19/24
(-2, 0, 2)         | {1: 2}                       | 1/8
(-2, 1, 1)         | {1: 2}                       | 1/24
(-2, 4)            | {1: 2, 2: 1}                 | 97/1152
(1, 1)             | {1: 2, 2: 2}                 | 1/720
(2,)               | {1: 3}                       | 1/1920
(0, 0, 0)          | {1: 3}                       | 1/24
(-2, 4)            | {1: 3}                       | 43/1152
(0, 2)             | {1: 4}                       | 11/1920
(1, 1)             | {1: 4}                       | 1/720
(4,)               | {1: 5}                       | 13/580608
```

```
(-4, 4)            | {2: 1}                    | 5/8
(-2, 2)            | {2: 1}                    | 1/8
(-2, 0, 0, 0)      | {2: 1}                    | 1
(-2, 1, 1)         | {2: 1, 3: 1}              | 1/6
(-2, 0, 2)         | {2: 2}                    | 1/4
(-2, 1, 1)         | {2: 2}                    | 1/6
(-2, 4)            | {2: 3}                    | 19/1152
(-2, -2, 0, 2)     | {3: 1}                    | 1
(-2, -2, 1, 1)     | {3: 1}                    | 1
(-2, -2, 4)        | {3: 2}                    | 7/24
(-2, -2, -2, 4)    | {4: 1}                    | 1
```

Of course this affects any future calculations:

```
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 [CMZ20b, 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:

```
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 cancellations happening for dimension reasons.

```
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) +
```

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

4.3. **Crosschecks.** The module `tests` includes some more cross-checks and example computations using `diffstrata`. For example, `leg_tests` tests on each one-dimensional graph Γ of a stratum if the evaluation of the $\xi_\Gamma^{[i]}$ at that level is the

same (for every leg!) as the product of Γ with ξ. Note that these expressions can be evaluated and the numbers compared.

The class `BananaSuite` tests the strata $\Omega\mathcal{M}_1(k, 1, -k-1)$ (cf. [CMZ20b, §10.3]) implementing the D-notation introduced there and including a method to test [CMZ20b, 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.

References

[BCGGM1] M. Bainbridge et al. "Compactification of strata of Abelian differentials". In: *Duke Math. J.* 167.12 (2018), pp. 2347–2416.

[BCGGM3] M. Bainbridge et al. *The moduli space of multi-scale differentials*. Preprint. 2019. arXiv: 1910.13492.

[BHPSS20] Y. Bae et al. *Pixtons formula and Abel-Jacobi theory on the Picard stack*. (2020). arXiv: 2004.08676.

[Cha12] M. Chan. "Combinatorics of the tropical Torelli map". In: *Algebra Number Theory* 6.6 (2012), pp. 1133–1169.

[CMSZ19] D. Chen et al. *Masur-Veech volumes and intersection theory on moduli spaces of abelian differentials*. (2019). arXiv: 1901.01785. to appear in Invent. Math.

[CMZ20a] M. Costantini, M. Möller, and J. Zachhuber. *diffstrata – a Sage package for calculations in the tautological ring of the moduli space of Abelian differentials*. Long version, including manual. (2020). arXiv: 2006.12815.

[CMZ20b] M. Costantini, M. Möller, and J. Zachhuber. *The Chern classes and the Euler characteristic of the moduli spaces of abelian differentials*. (2020). arXiv: 2006.12803.

[DSZ20] V. Delecroix, J. Schmitt, and J. van Zelm. *admcycles – a Sage package for calculations in the tautological ring of the moduli space of stable curves*. (2020). arXiv: 2002.01709.

[FP18] G. Farkas and R. Pandharipande. "The moduli space of twisted canonical divisors". In: *J. Inst. Math. Jussieu* 17.3 (2018), pp. 615–672.

[HS19] D. Holmes and J. Schmitt. *Infinitesimal structure of the pluricanonical double ramification locus.* (2019). arXiv: 1909.11981.

[McM14] C. McMullen. "Moduli spaces in genus zero and inversion of power series". In: *Enseign. Math.* 60.1-2 (2014), pp. 25–30.

[MP11] S. Maggiolo and N. Pagani. "Generating stable modular graphs". In: *J. Symbolic Comput.* 46.10 (2011), pp. 1087–1097.

[MUW17] M. Möller, M. Ulirsch, and A. Werner. "Realizability of tropical canonical divisors". In: (2017). arXiv: arXiv:1710.0640. to appear in: J. Eur. Math. Soc.

[MZ11] M Mirzakhani and P. Zograf. "Towards large genus asymtotics of intersection numbers on moduli spaces of curves." In: (2011). Preprint. arXiv: 1112.1151.

[SageMath] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 9.0).* https://www.sagemath.org. 2020.

[Sau19] A. Sauvaget. "Cohomology classes of strata of differentials". In: *Geom. Topol.* 23.3 (2019), pp. 1085–1171.

[Sch18] J. Schmitt. "Dimension theory of the moduli space of twisted k-differentials". In: *Doc. Math.* 23 (2018), pp. 871–894.

E-mail address: costanti@math.uni-bonn.de

Institut für Mathematik, Universität Bonn, Endenicher Allee 60, 53115 Bonn, Germany

E-mail address: moeller@math.uni-frankfurt.de

E-mail address: zachhuber@math.uni-frankfurt.de

Institut für Mathematik, Goethe-Universität Frankfurt, Robert-Mayer-Str. 6–8, 60325 Frankfurt am Main, Germany