Electronic Research Archive

https://www.aimspress.com/journal/era

ERA, 33(4): 2118–2147.
DOI: 10.3934/era.2025093
Received: 31 December 2024
Revised: 28 March 2025
Accepted: 01 April 2025
Published: 15 April 2025

Research article

Motives meet SymPy: studying λ-ring expressions in Python

Daniel Sanchez[1], **David Alfaya**[2,*] **and Jaime Pizarroso**[3]

[1] Institute for Research in Technology, ICAI School of Engineering, Comillas Pontifical University, Calle del Rey Francisco 4, Madrid 28015, Spain

[2] Department of Applied Mathematics and Institute for Research in Technology, ICAI School of Engineering, Comillas Pontifical University, C/Alberto Aguilera 25, Madrid 28015, Spain

[3] Department of Telematics and Computation and Institute for Research in Technology, ICAI School of Engineering, Comillas Pontifical University, C/Alberto Aguilera 25, Madrid 28015, Spain

* **Correspondence:** Email: dalfaya@comillas.edu; Tel: +34-91-542-28-00.

Abstract: We present a new Python package called "motives", a symbolic manipulation package based on SymPy capable of handling and simplifying motivic expressions in the Grothendieck ring of Chow motives and other types of λ-rings. The package is able to manipulate and compare arbitrary expressions in λ-rings and, in particular, it contains explicit tools for manipulating motives of several types of commonly used moduli schemes and moduli stacks of decorated bundles on curves. We have applied this new tool to advance in the verification of Mozgovoy's conjectural formula for the motive of the moduli space of twisted Higgs bundles, proving that it holds in rank 2 and 3 for any curve of genus up to 18 and any twisting bundle of small degree.

Keywords: lambda-rings; symbolic computations of motives; Chow motives; moduli spaces; Higgs bundles moduli space

1. Introduction

The Grothendieck motive of a scheme encodes crucial information about its geometry. For instance, if two schemes share the same class in the Grothendieck ring of Chow motives, then their E-polynomials and their Poincaré polynomials for cohomology with compact support are the same. For this reason, there have been numerous efforts to compute closed formulas for the motives of several types of moduli spaces and moduli stacks in terms of elementary components, like moduli of vector bundles [1–4], moduli of pairs and chains [5, 6], moduli of Higgs bundles and twisted Higgs bundles [6–8] and other moduli of Lie algebroid connections [8] or character varieties [9–11] among others. However, handling expressions in the Grothendieck ring of motives can be a challenging task, and even comparing if two

expressions are equal can become a hard problem. For instance, some of these expressions contain motivic sums with increasing numbers of terms [6, 8] or nontrivial plethystic operations on motivic expressions [7, 11], whose manipulation can become nontrivial at times.

In order to tackle this problem, in [12] a theoretical algorithm was proposed for simplifying these types of expressions into polynomials in certain sets of motivic generators, as well as expressions in other types of λ-rings. In that paper, a focus was put on simplifying expressions in the subring of the Grothendieck ring of Chow motives spanned (as a λ-ring) by a finite set of complex projective algebraic curves, and an ad-hoc MATLAB code was implemented to apply this theoretical algorithm to successfully compare two different formulas for the moduli space of L-twisted Higgs bundles: a conjectural equation obtained by Mozgovoy as a solution to the ADHM equations [7] and two formulas proven in [8] for the Grothendieck virtual class of that moduli space in rank 2 and 3 obtained through a Bialynicki-Birula decomposition of the variety. The resulting program was able to verify that Mozgovoy's conjecture for the motive holds for curves of genus at most 11 and any twisting bundle L of low degree.

In this paper, we have further improved the algorithm proposed in [12], and we have built a full-general purpose λ-ring manipulation Python package, integrating the algorithm into the SymPy symbolic framework [13], a Python library for symbolic mathematics. This new library called `motives` defines an abstract λ-ring expression class and provides general simplification algorithms capable of simplifying effectively any λ-ring expression into a polynomial in a finite set of motivic generators. By building upon `SymPy`, these algorithms further integrate into the symbolic manipulation toolbox from `SymPy`, extending the functionality of its general-purpose simplification and manipulation functionality to work on λ-rings and, at the same time, enriching it with new tools explicitly aimed to simplify and compare λ-ring expressions.

The simplification algorithms are designed to work with any user-defined λ-ring class, but the library also includes some out-of-the-box types of λ-rings and modules aimed at the working mathematician.

The current release of the package includes:

- λ-rings of integers.
- Free λ-rings and free λ-ring extensions of a λ-ring.
- Polynomial λ-ring extensions of a λ-ring.
- Grothendieck ring of Chow motives, including the following pre-programmed motives:
  - Complex algebraic curves.
  - Jacobian varieties of curves.
  - Symmetric and alternated products of any variety given its motive.
  - Moduli spaces of vector bundles on curves.
  - Moduli spaces of L-twisted Higgs bundles on curves.
  - Moduli spaces of chain bundles and variations of Hodge structure on curves in low rank.
  - Algebraic groups.
  - Moduli stacks of vector bundles and principal G-bundles on curves.
  - Classifying stacks BG for several groups G.

and we plan to expand the library with more λ-rings and geometric elements in future releases. The code of the library is publicly hosted at `https://github.com/CIAMOD/motives`. Additionally, the `motives` package can be installed directly from PyPI by running `pip install motives` in a terminal.

In order to test the capabilities of this—now general-purpose—symbolic package, we have applied it to the same problem treated originally in [12]: the proof of Mozgovoy's conjectural formula of the moduli space of L-twisted Higgs bundles. The new package was proven to be significantly more efficient than the ad-hoc code implemented for the problem in [12]. Using the new algorithm, we were able to prove that Mozgovoy's conjecture holds for curves of genus up to 18 (increasing significantly the current verification, limited to genus 11 curves). Concretely, we prove the following (see Section 5 and Theorem 5.3 for details).

For each integer $n \geq 1$, let

$$\mathcal{H}_n(t) = \sum_{a \in \mathcal{P}(n)} \prod_{s \in d(a)} (-t^{a(s)-l(s)} \mathbb{L}^{a(s)})^p t^{(1-g)(2l(s)+1)} Z_X(t^{h(s)} \mathbb{L}^{a(s)}),$$

where $\mathcal{P}(n)$ denotes the set of ordered partitions of n, $d(a)$ is the Young diagram of the partition $a \in \mathcal{P}(n)$, and, given $s \in d(a)$, $a(s)$, $l(s)$, and $h(s)$ denote the arm, leg and hook lengths of s in the partition, respectively (see Sections 2 and 5 for details). From $\mathcal{H}_n(t)$, define $H_r(t)$ for each $r \geq 1$ as follows:

$$\sum_{r \geq 1} H_r(t) T^r = (1-t)(1-\mathbb{L}t) \sum_{j \geq 1} \sum_{k \geq 1} \frac{(-1)^{k+1} \mu(j)}{jk} \left(\sum_{n \geq 1} \psi_j[\mathcal{H}_n(t)] T^{jn} \right)^k.$$

Let us compute

$$M^{\mathrm{ADHM}}_{g,r,p} := (-1)^{pr} \mathbb{L}^{r^2(g-1)+p\frac{r(r+1)}{2}} H_r(1).$$

Then Mozgovoy's conjecture states that $M^{\mathrm{ADHM}}_{g,r,p}$ yields the motive of the moduli space of L-twisted Higgs bundles of rank r and degree d on a genus g curve. On the other hand, for $r = 1, 2, 3$ consider the following motivic formulas $M^{\mathrm{BB}}_{g,r,p}$.

i) For $r = 1$

$$M^{\mathrm{BB}}_{g,1,p} := [\mathrm{Jac}(X) \times H^0(X, L^\vee)] = \mathbb{L}^{g-1+p} P_X(1)$$

ii) For $r = 2$, if $(2, d) = 1$,

$$M^{\mathrm{BB}}_{g,2,p} := \frac{\mathbb{L}^{4g-4+4p} \left(P_X(1) P_X(\mathbb{L}) - \mathbb{L}^g P_X(1)^2 \right)}{(1-\mathbb{L})(1-\mathbb{L}^2)} + \mathbb{L}^{4g-4+3p} P_X(1) \sum_{i=1}^{\lfloor \frac{2g-1+p}{2} \rfloor} \lambda^{2g-1+p-2i}([X]).$$

iii) For $r = 3$, if $(3, d) = 1$,

$$M^{\mathrm{BB}}_{g,3,p} = \frac{\mathbb{L}^{9g-9+9p} P_X(1)}{(\mathbb{L}-1)(\mathbb{L}^2-1)^2(\mathbb{L}^3-1)} \Big(\mathbb{L}^{3g-1}(1+\mathbb{L}+\mathbb{L}^2) P_X(1)^2 - \mathbb{L}^{2g-1}(1+\mathbb{L})^2 P_X(1) P_X(\mathbb{L}) + P_X(\mathbb{L}) P_X(\mathbb{L}^2) \Big) + \frac{\mathbb{L}^{9g-9+7p} P_X(1)^2}{\mathbb{L}-1} \sum_{i=1}^{\lfloor \frac{1}{3} + \frac{2g-2+p}{2} \rfloor} \left(\mathbb{L}^{i+g} \lambda^{-2i+2g-2+p}([X]+\mathbb{L}^2) - \lambda^{-2i+2g-2+p}([X]\mathbb{L}+1) \right)$$

$$+\frac{\mathbb{L}^{9g-9+7p}P_X(1)^2}{\mathbb{L}-1}\sum_{i=1}^{\lfloor\frac{2}{3}+\frac{2g-2+p}{2}\rfloor}\left(\mathbb{L}^{i+g-1}\lambda^{-2i+2g-1+p}([X]+\mathbb{L}^2)-\lambda^{-2i+2g-1+p}([X]\mathbb{L}+1)\right)$$
$$+\mathbb{L}^{9g-9+6p}P_X(1)\sum_{i=1}^{2g-2+p}\sum_{j=\max\{2-2g-p+i,1-i\}}^{\lfloor(2g-1+p-i)/2\rfloor}\lambda^{-i+j+2g-2+p}([X])\lambda^{-i-2j+2g-1+p}([X]).$$

These equations were proven in [8] to give expressions for the motives of L-twisted Higgs bundles on a genus $g \geq 2$ curve if the rank and degree of the underlying bundle are coprime. By simplifying computationally both expressions and showing that they are equal ($M^{\mathrm{ADHM}}_{g,r,p} = M^{\mathrm{BB}}_{g,r,p}$) for low g and p, we proved the following result, significantly expanding the results obtained in [12].

Theorem 1.1 (Theorem 5.3)**.** *Mozgovoy's conjectural formula for the motive of the moduli spaces of L-twisted Higgs bundles hold in the Grothendieck ring of Chow motives in rank at most 3 for any smooth complex projective curve of genus g such that $2 \leq g \leq 18$ and any line bundle L on the curve such that $2g-1 \leq \deg(L) \leq 2g+18$.*

To put the increment in the genus bound in perspective, the newly obtained motivic polynomials are 5 times larger than the largest motive that the algorithm in [12] was capable of processing before reaching the memory limit of the machine used for the test (128 GB), and the new library is able to perform these simplifications up to an order of magnitude faster than its task-specific predecessor, showcasing an enhanced scaling behavior.

The paper is structured as follows: Section 2 includes some generalities on λ-rings and introduces some explicit algebraic relations between operators in λ-rings, which will be used by the simplification algorithm. Section 3 explores the properties of the λ-rings of Grothendieck motives (Grothendieck ring of varieties, Grothendieck rings of Chow motives, and other related rings). The description of the main simplification algorithms and the implementation details are described in Section 4.2. Section 5 explores the application of the `motives` library to proving Mozgovoy's conjecture in low genus and the comparison between the results of this library and the ones obtained in [12]. Finally, as the development of the package continues, some future lines of work and expected future additions to the package are described in Section 6.

2. λ-rings

Let R be a unital commutative ring. A *λ-ring structure on R* is a set of maps $\lambda = \{\lambda^n : R \longrightarrow R\}_{n\in\mathbb{N}}$ such that for each $x, y \in R$ and each $n \in \mathbb{N}$, the following hold.

1) $\lambda^0(x) = 1$
2) $\lambda^1(x) = x$,
3) $\lambda^n(x+y) = \sum_{i=0}^{n}\lambda^i(x)\lambda^{n-i}(y)$

The pair (R, λ) is called a λ-ring. A λ-ring structure is called *special* if, moreover, for each $x, y \in R$ and each $n, m \in \mathbb{N}$, we have

4) $\lambda^n(xy) = P_n(\lambda^1(x), \ldots, \lambda^n(x), \lambda^1(y), \ldots, \lambda^n(y))$,
5) $\lambda^n(\lambda^m(x)) = P_{n,m}(\lambda^1(x), \ldots, \lambda^{nm}(x))$,

where P_n and $P_{n,m}$ are the Grothendieck universal polynomials, defined as follows (for more information, see [14, 15]). If

$$s_d(\bar{X}) = s_d(X_1, \ldots, X_n) = \sum_{1 \le i_1 < \ldots < i_d \le n} \prod_{k=1}^{d} X_{i_k}$$

is the d-th elementary symmetric polynomial in the variables given by $\bar{X} = (X_1, \ldots, X_n)$, then P_n and $P_{n,m}$ are the unique integral polynomials such that

$$P_n(s_1(\bar{X}), \ldots, s_n(\bar{X}), s_1(\bar{Y}), \ldots, s_n(\bar{Y})) = \operatorname*{coeff}_{t^n} \prod_{i,j=1}^{n} (1 + tX_iY_j),$$

$$P_{n,m}(s_1(\bar{Z}), \ldots, s_{nm}(\bar{Z})) = \operatorname*{coeff}_{t^n} \sum_{\substack{I \subset \{1, \ldots, mn\} \\ |I| = m}} (1 + t \prod_{i \in I} Z_i),$$

where $\bar{X} = (X_1, \ldots, X_n)$, $\bar{Y} = (Y_1, \ldots, Y_n)$ and $\bar{Z} = (Z_1, \ldots, Z_{nm})$. For example, if we denote $\lambda^n(xy) = P_n = P_n(\lambda^1(x), \ldots, \lambda^n(x), \lambda^1(y), \ldots, \lambda^n(y))$ and $\lambda^n(\lambda^m(x)) = P_{n,m} = P_{n,m}(\lambda^1(x), \ldots, \lambda^{nm}(x))$, then

$P_0 = 1,$

$P_1 = \lambda^1(x)\lambda^1(y),$

$P_2 = \lambda^1(x)^2\lambda^2(y) + \lambda^1(y)^2\lambda^2(x) - 2\lambda^2(x)\lambda^2(y),$

$P_3 = \lambda^1(x)^3\lambda^3(y) + \lambda^1(x)\lambda^1(y)\lambda^2(x)\lambda^2(y) - 3\lambda^1(x)\lambda^2(x)\lambda^3(y) + \lambda^1(y)^3\lambda^3(x) - 3\lambda^1(y)\lambda^2(y)\lambda^3(x) + 3\lambda^3(x)\lambda^3(y),$

$P_{0,5} = 1,$

$P_{5,0} = 0,$

$P_{1,1} = \lambda^1(x),$

$P_{1,2} = \lambda^2(x),$

$P_{2,1} = \lambda^2(x),$

$P_{2,2} = \lambda^1(x)\lambda^3(x) - \lambda^4(x),$

$P_{2,3} = -\lambda^1(x)\lambda^5(x) + \lambda^2(x)\lambda^4(x) + \lambda^6(x),$

$P_{2,4} = \lambda^1(x)\lambda^7(x) - \lambda^2(x)\lambda^6(x) + \lambda^3(x)\lambda^5(x) - \lambda^8(x).$

For more examples, see, for instance, [15]. Further examples can also be computed using the Python package `motives` proposed in this work.

Given an element $x \in R$, let

$$\lambda_t(x) = \sum_{n \ge 0} \lambda^n(x)t^n \ \in \ 1 + tR[[t]]$$

denote the generating series of the λ-powers of x in the variable t. Condition (3) of the definition of λ-ring is equivalent to

$$\lambda_t(x + y) = \lambda_t(x)\lambda_t(y). \tag{2.1}$$

Given a λ-ring structure on R, we define its opposite λ-ring structure σ as follows. For each $x \in R$, take

$$\sigma_t(x) := (\lambda_{-t}(x))^{-1}. \tag{2.2}$$

It is easy to verify that if λ is a λ-ring structure, then $\sigma_t(x) = \sum_{n\geq 0} \sigma^n(x)t^n$ defines another λ-ring structure σ on R. Thus, any λ-ring structure on a ring R defines a triple (R, λ, σ). In this work, we will assume that at least one of the mutually opposite λ-ring structures, λ or σ, is special, and, without loss of generality, we will assume that σ always denotes a special structure and that λ is a structure that may not be special (as will be described in Section 3, this will be the situation in the Grothendieck ring of Chow motives).

If we assume that σ is a special λ-ring structure, we can associate to it a set of Adams operations $\psi^n : R \longrightarrow R$ for each $n \geq 1$ taking

$$\psi^n(x) = N_n(\sigma^1(x), \ldots, \sigma^n(x)), \tag{2.3}$$

for each $x \in R$, where $N_n(X_1, \ldots, X_n)$ is the Hirzebruch-Newton polynomial, which is the unique polynomial such that

$$\sum_{i=1}^{n} X_i^n = N_n(s_1(\bar{X}), \ldots, s_n(\bar{X}))$$

for $\bar{X} = (X_1, \ldots, X_n)$. For example, if we denote $\psi^n(x) = N_n = N_n(\sigma^1(x), \ldots, \sigma^n(x)$, then,

$N_1 = \sigma^1(x)$,
$N_2 = \sigma^1(x)^2 - 2\sigma^2(x)$,
$N_3 = \sigma^1(x)^3 - 3\sigma^1(x)\sigma^2(x) + 3\sigma^3(x)$,
$N_4 = \sigma^1(x)^4 - 4\sigma^1(x)^2\sigma^2(x) + 4\sigma^1(x)\sigma^3(x) + 2\sigma^2(x)^2 - 4\sigma^4(x)$.

Further examples can also be computed using the proposed Python package `motives`. Furthermore, for each $x \in R$, let

$$\psi_t(x) := \sum_{n\geq 1} \psi^n(x)t^n$$

denote the generating function for the sequence $\{\psi^n(x)\}_{n\geq 0}$ in the variable t. Then, it can be shown (see, for instance, [15]) that

$$\psi_t(x) = -t\frac{d}{dt}\log\left(\sigma_{-t}(x)\right). \tag{2.4}$$

Notice that, as $\sigma_{-t}(x) = (\lambda_t(x))^{-1}$, we have

$$\psi_t(x) = t\frac{d}{dt}\log\left(\lambda(x)\right). \tag{2.5}$$

By [15, Theorem 9.2], if σ is special, then the following hold.

1) For each $n \in \mathbb{N}$, $\psi^n : (R, \sigma) \longrightarrow (R, \sigma)$ is a λ-ring homomorphism.

2) $\psi^1(x) = x$.

3) For each $i, j \in \mathbb{N}$, $\psi^i \circ \psi^j = \psi^{ij}$.

In [12], several recurrence formulas were inferred and used for computing certain multivariate polynomials, which allowed expressing λ, σ, and ψ in terms of each other. An efficient computation and manipulation of these polynomials is fundamental to applying the algorithm described in [12] for simplifying algebraic expressions in λ-rings. However, some of the expressions found in [12] were based on iterated substitutions of some multivariate polynomials inside other multivariate polynomial expressions. This was highly inefficient, and we found that it was one of the computational bottlenecks for the application of the cited algorithm (for instance, to the verification of Mozgovoy's conjecture, described in Section 5).

In order to solve this issue, we will start this work by presenting some alternative equations providing algebraic relations between λ, σ and ψ.

Let $\mathcal{P}_k(n)$ denote the set of ordered partitions of a natural number $n \in \mathbb{N}$ into a sum of k positive natural numbers ordered decreasingly, i.e., the set of decompositions of a number n into a sum of the form

$$\begin{cases} n = a_1 + a_2 + \cdots + a_k, \\ a_1 \geq a_2 \geq \ldots \geq a_k > 0, \\ a_i \in \mathbb{N} \quad \forall i = 1, \ldots, k. \end{cases}$$

We will use the notation $a = (a_1, \ldots, a_k) \in \mathcal{P}_k(n)$ to denote the partition $n = a_1 + a_2 + \ldots + a_k$. Let $\mathcal{P}(n) = \coprod_{k=1}^{n} \mathcal{P}_k(n)$ denote the set of all partitions of the number n into sums of positive natural numbers, with an arbitrary number of summands. Given a partition $a = (a_1, a_2, \ldots, a_k) \in \mathcal{P}_k(n)$, let $n_i(a)$ denote the number of times that i appears in the partition a. For example, $a = (3, 3, 2, 1, 1) \in p_5(10)$ satisfies $n_1(a) = 2$, $n_2(a) = 1$, $n_3(a) = 2$, and $n_i(a) = 0$ for any other $i > 3$.

On the other hand, recall that we say that a ring R has additive torsion whenever there exists an element $x \in R$ and integer $n > 0$ such that $nx := \underbrace{x + \ldots + x}_{n} = 0$.

Proposition 2.1. *Let σ be a special λ-ring structure on a ring R with no additive torsion and let ψ^n be the Adams operations associated with σ. Then*

$$\sigma^n(x) = \sum_{i=0}^{n} \sum_{a=(a_1,\ldots,a_i)\in\mathcal{P}_i(n)} \left(\frac{(-1)^{n+i}}{n_1(a)!n_2(a)!\ldots n_n(a)!} \prod_j \frac{\psi^{a_j}(x)}{a_j} \right). \tag{2.6}$$

Proof. We solve for $\sigma_{-t}(x)$ in Eq (2.4).

$$\sigma_{-t}(x) = e^{-\int \frac{\psi_t(x)}{t} dt}. \tag{2.7}$$

Now, we expand $-\int \frac{\psi_t(x)}{t} dt$:

$$-\int \frac{\psi_t(x)}{t} dt = -\int \sum_{n=0}^{\infty} \frac{\psi^n(x) \cdot t^n}{t} dt = -\sum_{n=0}^{\infty} \int \psi^n(x) t^{n-1} dt = -\sum_{n=0}^{\infty} \frac{1}{n} \psi^n(x) t^n.$$

By using this equality and applying the Taylor series expansion of e^x to the right-hand side of Eq (2.7), we obtain:

$$\sigma_{-t}(x) = \sum_{i=0}^{\infty} \frac{1}{i!} \left(-\int \frac{\psi_t(x)}{t} dt \right)^i = \sum_{i=0}^{\infty} \frac{1}{i!} \left(-\sum_{n=0}^{\infty} \frac{1}{n} \psi^n(x) t^n \right)^i.$$

Changing t for $-t$ yields

$$\sigma_t(x) = \sum_{i=0}^{\infty} \frac{1}{i!}(-1)^i \left(\sum_{n=0}^{\infty} (-1)^n \frac{1}{n} \psi^n(x) t^n \right)^i.$$

We can now expand $(\sum_{n=0}^{\infty}(-1)^n \frac{1}{n}\psi^n(x)t^n)^i$ by using the multinomial theorem.

$$\begin{aligned}
\sigma_t(x) &= \sum_{i=0}^{\infty} \frac{1}{i!} \sum_{n=0}^{\infty} \left(\sum_{a=(a_1,a_2,\ldots,a_i)\in\mathcal{P}_i(n)} (-1)^i \binom{i}{n_1(a), n_2(a), \ldots, n_n(a)} \frac{(-1)^{a_1}}{a_1}\psi^{a_1}(x) \frac{(-1)^{a_2}}{a_2}\psi^{a_2}(x) \ldots \frac{(-1)^{a_i}}{a_i}\psi^{a_i}(x) \right) t^n \\
&= \sum_{n=0}^{\infty} \sum_{i=0}^{n} \sum_{a=(a_1,\ldots,a_i)\in\mathcal{P}_i(n)} \frac{(-1)^{i+\sum_j a_j}}{i! \prod_j a_j} \binom{i}{n_1(a), n_2(a), \ldots, n_n(a)} \prod_j \psi^{a_j}(x) t^n \\
&= \sum_{n=0}^{\infty} \sum_{i=0}^{n} \sum_{a=(a_1,\ldots,a_i)\in\mathcal{P}_i(n)} \frac{(-1)^{n+i} \cdot i! \prod_j \psi^{a_j}(x)}{i! \cdot n_1(a)! n_2(a)! \ldots n_n(a)! \prod_j a_j} t^n.
\end{aligned}$$

By equating the coefficients and cancelling the factorials, we achieve the desired equality.

$$\sigma^n(x) = \sum_{i=0}^{n} \sum_{a=(a_1,\ldots,a_i)\in\mathcal{P}_i(n)} \left(\frac{(-1)^{n+i}}{n_1(a)! n_2(a)! \ldots n_n(a)!} \prod_j \frac{\psi^{a_j}(x)}{a_j} \right).$$

□

Proposition 2.2. *Let λ and σ be two opposite λ-ring structures on a ring R with no additive torsion and such that σ is special, and let ψ^n be the Adams operations associated with σ. Then*

$$\lambda^n(x) = \sum_{i=0}^{n} \sum_{a=(a_1,\ldots,a_i)\in\mathcal{P}_i(n)} \left(\frac{1}{n_1(a)! n_2(a)! \ldots n_n(a)!} \prod_j \frac{\psi^{a_j}(x)}{a_j} \right). \tag{2.8}$$

Proof. The proof is analogous to the previous lemma, using Eq (2.5) instead of Eq (2.4) and adjusting the signs accordingly. □

Proposition 2.3. *Let σ be a special λ-ring structure on a ring R with no additive torsion, and let ψ^n be the Adams operations associated with σ. Then*

$$\psi^n(x) = \sum_{i=1}^{n} (-1)^{i+n+1} \sum_{l=0}^{n-1} \sum_{a=(a_1,\ldots,a_i)\in\mathcal{P}_i(l)} (n-l) \frac{i!}{n_1(a)! n_2(a)! \ldots n_l(a)!} \sigma^{n-l}(x) \prod_j \sigma^{a_j}(x). \tag{2.9}$$

Proof. We apply the Taylor series expansion of the natural logarithm to the right-hand side of Eq (2.4):

$$\psi_t(x) = -t \cdot \frac{d}{dt} \log(\sigma_{-t}(x)) = -t \cdot \frac{d}{dt} \sum_{i=1}^{\infty} \frac{1}{i} (\sigma_{-t}(x) - 1)^i (-1)^{i-1}.$$

Next, we expand the derivative.

$$\psi_t(x) = -t \sum_{i=1}^{\infty} \frac{1}{i} i \, (\sigma_{-t}(x) - 1)^{i-1} \left(\frac{d}{dt} \sigma_{-t}(x) \right) (-1)^{i-1}$$

$$= -\sum_{i=0}^{\infty}(-1)^i\left(\sum_{n=1}^{\infty}\sigma^n(x)(-t)^n\right)^i\sum_{n=1}^{\infty}(-1)^n n\sigma^n(x)t^n.$$

Now we use the multinomial theorem to expand the sum $(\sum_{n=1}^{\infty}\sigma^n(x)(-t)^n)^i$.

$$\psi_t(x) = -\sum_{i=0}^{n}(-1)^i\left(\sum_{n=1}^{\infty}\left(\sum_{a=(a_1,\dots,a_i)\in\mathcal{P}_i(n)}\binom{i}{n_1(a),n_2(a),\dots,n_n(a)}\right)\prod_j(-1)^{a_j}\sigma^{a_j}(x)\right)t^n\sum_{n=1}^{\infty}(-1)^n n\sigma^n(x)t^n$$

By applying the convolution, we obtain

$$\begin{aligned}\psi_t(x) &= -\sum_{i=0}^{n}(-1)^i\sum_{n=1}^{\infty}\sum_{l=0}^{n}(n-l)\sigma^{n-l}(x)\left(\sum_{a=(a_1,\dots,a_i)\in\mathcal{P}_i(l)}\binom{i}{n_1(a),n_2(a),\dots,n_l(a)}\right)(-1)^l\prod_j\sigma^{a_j}(x)\Bigg)t^n\\ &= -\sum_{n=1}^{\infty}\sum_{i=0}^{n}(-1)^i\sum_{l=0}^{n-1}(-1)^n(n-l)\sigma^{n-l}(x)\left(\sum_{a=(a_1,\dots,a_i)\in\mathcal{P}_i(l)}\binom{i}{n_1(a),n_2(a),\dots,n_l(a)}\right)\prod_j\sigma^{a_j}(x)\Bigg)t^n,\end{aligned}$$

and, equating the coefficients, we arrive at the desired formula

$$\psi^n(x) = \sum_{i=1}^{n}(-1)^{i+n+1}\sum_{l=0}^{n-1}\sum_{a=(a_1,\dots,a_i)\in\mathcal{P}_i(l)}(n-l)\frac{i!}{n_1(a)!n_2(a)!\dots n_l(a)!}\sigma^{n-l}(x)\prod_j\sigma^{a_j}(x).$$

□

Proposition 2.4. *Let λ and σ be two opposite λ-ring structures on a ring R with no additive torsion, and such that σ is special and let ψ^n be the Adams operations associated with σ. Then*

$$\psi^n(x) = \sum_{i=1}^{n}(-1)^i\sum_{l=0}^{n-1}\sum_{a=(a_1,\dots,a_i)\in\mathcal{P}_i(l)}(n-l)\frac{i!}{n_1(a)!n_2(a)!\dots n_l(a)!}\lambda^{n-l}(x)\prod_j\lambda^{a_j}(x). \tag{2.10}$$

Proof. The proof is analogous to the previous lemma, using Eq (2.5) instead of Eq (2.4) and adjusting the signs accordingly. □

Proposition 2.5. *Let λ and σ be two opposite λ-ring structures on a ring R such that σ is special, and let ψ^n be the Adams operations associated with σ. Then*

$$\psi^n(x) = \sum_{i=1}^{n}(-1)^{n-i}i\sigma^{n-i}(x)\lambda^i(x).$$

Proof. As λ is a λ-ring structure, by (2.1) we have

$$\lambda_t(x+y) = \lambda_t(x)\lambda_t(y).$$

By substituting x and $-x$, we obtain

$$\lambda_t(0) = \lambda_t(x)\lambda_t(-x),$$

and so

$$\lambda_t(-x) = (\lambda_t(x))^{-1}. \tag{2.11}$$

Next, we use the derivative of the logarithm on the right-hand side of Eq (2.5).

$$\psi_t(x) = t \cdot \frac{d}{dt}\log(\lambda_t(x)) = t \cdot \frac{\frac{d}{dt}\lambda_t(x)}{\lambda_t(x)} = t \cdot \frac{\sum_{n=1}^{\infty} n\lambda^n(x)t^{n-1}}{\lambda_t(x)} = \lambda_t(x)^{-1}\sum_{n=1}^{\infty} n\lambda^n(x)t^n.$$

Now, we apply Eq (2.11) to this result, yielding

$$\psi_t(x) = \lambda_t(-x)\sum_{k=1}^{\infty} k\lambda^k(x)t^k = \left(\sum_{j=0}^{\infty}\lambda^j(-x)t^j\right)\left(\sum_{k=1}^{\infty} k\lambda^k(x)t^k\right).$$

We apply the convolution

$$\psi_t(x) = \sum_{n=1}^{\infty}\sum_{i=1}^{n} i\lambda^i(x)\lambda^{n-i}(-x)t^n.$$

and, by equating the coefficients, we obtain

$$\psi^n(x) = \sum_{i=1}^{n} i\lambda^i(x)\lambda^{n-i}(-x).$$

Now we use the fact that $(-1)^n\sigma^n(x) = \lambda^n(-x)$, which follows from (2.11) and the definition of opposite structure, to obtain

$$\psi^n(x) = \sum_{i=1}^{n}(-1)^{n-i} i\sigma^{n-i}(x)\lambda^i(x).$$

□

Remark 2.6. *We can further use the equations from [12, Proposition 2] to express $\sigma^n(x)$ in terms of $\lambda^n(x)$ (or vice versa) in order to expand the right-hand side of the equation from Proposition 2.5 into a polynomial formula in σ or λ that represents a specific degree of ψ. We can obtain it by substituting the corresponding operator by the polynomial $P_n^{op} \in \mathbb{Z}[X_1, \ldots, X_n]$, which expresses λ in terms of σ and vice versa (see [12, Proposition 2] for details)*

$$\sigma^n = P_n^{op}(\lambda^1(x), \ldots, \lambda^n(x)), \qquad \lambda^n = P_n^{op}(\sigma^1(x), \ldots, \sigma^n(x)), \quad \forall x \in R \tag{2.12}$$

and which can be defined recursively as follows:

$$P_0^{op} = 1$$
$$P_n^{op} = \sum_{i=0}^{n-1} P_i^{op} X_{n-i} \quad \forall n \geq 1.$$

This yields a polynomial expression

$$\psi^n(x) = \sum_{i=1}^{n}(-1)^{n-i} i\sigma^{n-i}(x)P_i^{op}(\sigma^1(x), \ldots, \sigma^i(x))$$

for $\psi^n(x)$ in terms of σ operations of x or, equivalently, a polynomial

$$\psi^n(x) = \sum_{i=1}^{n}(-1)^{n-i} iP_{n-i}^{op}(\lambda_1(x), \ldots, \lambda_{n-i}(x))\lambda^i(x)$$

in terms of λ operations of x.

Finally, let us recall the notion of dimension of an element $x \in R$ in a λ-ring. We say that x is d-dimensional for the structure λ (respectively, for the structure σ) if $\lambda_t(x)$ (respectively $\sigma_t(x)$) is a degree d-polynomial. This implies that $\lambda^n(x) = 0$ or $\sigma^n(x) = 0$ for each $n > d$, respectively.

Remark 2.7. *If $x \in R$ is a d-dimensional object for λ, then $\sigma^n(x)$ and $\psi^n(x)$ can be expressed as polynomials in d variables depending on $\lambda^1(x), \ldots, \lambda^d(x)$ through* (2.10) *and* (2.12).

3. λ-ring structures in the Grothendieck ring of Chow motives

Let $\mathcal{V}ar_{\mathbb{C}}$ denote the category of quasi-projective varieties over $\mathbb{C}$. The Grothendieck ring of varieties, denoted by $K_0(\mathcal{V}ar_{\mathbb{C}})$, is the quotient of the free commutative group generated by isomorphism classes of varieties in $\mathcal{V}ar_{\mathbb{C}}$ quotiented by the following relation. If Z is a closed subvariety of X, then

$$[X] = [X \backslash Z] + [Z],$$

where $[X]$ denotes the class of X in $K_0(\mathcal{V}ar_{\mathbb{C}})$. The product of the ring is defined by taking

$$[X][Y] = [X \times Y]$$

for any pair of varieties X and Y.

Similarly, let $\mathcal{CM}_{\mathbb{C}}$ denote the category of Chow motives and let $K_0(\mathcal{CM}_{\mathbb{C}})$ denote the Grothendieck ring of Chow motives. For both rings, we will denote by $\mathbb{L} := [\mathbb{A}^1]$ the Lefschetz object, and we will also consider the dimensional completions of the localizations of these rings with respect to $\mathbb{L}$. Concretely, let

$$\hat{K}_0(\mathcal{V}ar_{\mathbb{C}}) := \left\{ \sum_{r \geq 0} [X_r]\mathbb{L}^{-r} \,\middle|\, [X_r] \in \mathcal{V}ar_{\mathbb{C}}, \text{ and } \lim_{r \to \infty}(\dim(X_r) - r) = -\infty \right\}$$

denote the completion of the localized ring $K_0(\mathcal{V}ar_{\mathbb{C}})[\mathbb{L}^{-1}]$ and let $\hat{K}_0(\mathcal{CM}_{\mathbb{C}})$ be the completion of $K_0(\mathcal{CM}_{\mathbb{C}})[\mathbb{L}^{-1}]$ defined in analogous terms (see [16] or [17] and [1] for more details).

These rings admit two naturally opposite λ-ring structures of geometric nature. Given a variety $X \in \mathcal{V}ar_{\mathbb{C}}$, let

$$\mathrm{Sym}^n(X) := \frac{\overbrace{X \times X \times \ldots X}^{n}}{S_n}$$

denote its symmetric product. The map Sym^n extends to a map $\lambda^n : \hat{K}_0(\mathcal{V}ar_{\mathbb{C}}) \longrightarrow \hat{K}_0(\mathcal{V}ar_{\mathbb{C}})$ induced by

$$\lambda^n([X]) = [\mathrm{Sym}^n(X)].$$

By [18], λ defines a λ-ring structure on $K_0(\mathcal{V}ar_{\mathbb{C}})$, but it is not a special λ-ring structure. Instead, its opposite λ-ring structure, which we will denote by σ, is special.

Analogous λ-ring structures also exist in $\hat{K}_0(\mathcal{CM}_{\mathbb{C}})$. Given a Chow motive $[X] \in \hat{K}_0(\mathcal{CM}_{\mathbb{C}})$, define $\lambda^n([X]) = \mathrm{Sym}^n(X)$ as the image of the map

$$\frac{1}{n!} \sum_{\alpha \in S_n} \alpha \, : \, X^{\otimes n} \longrightarrow X^{\otimes n}$$

and define $\sigma^n([X]) = \mathrm{Alt}^n(X)$ as the image of the map

$$\frac{1}{n!}\sum_{\alpha\in S_n}(-1)^{|\alpha|}\alpha \,:\, X^{\otimes n} \longrightarrow X^{\otimes n}.$$

Then, λ and σ are opposite λ-ring structures on $\hat{K}_0(\mathcal{CM}_{\mathbb{C}})$ and σ is a special λ-ring structure (see, for instance, [19]). Moreover, the natural map $\hat{K}_0(\mathcal{V}ar_{\mathbb{C}}) \longrightarrow \hat{K}_0(\mathcal{CM}_{\mathbb{C}})$ is a λ-ring map for these structures and, as a consequence of [20, Proposition 2.7], the localizations $\hat{K}_0(\mathcal{V}ar_{\mathbb{C}})$ and $\hat{K}_0(\mathcal{CM}_{\mathbb{C}})$ have no additive torsion, and, therefore, the identities from Section 2 relating the λ structures λ and σ with the Adams operations ψ in these rings hold.

Given a variety X, it is common to use the notation

$$Z_X(t) := \sum_{n\geq 0}[\mathrm{Sym}^n(X)]t^n = \lambda_t(X).$$

This is usually called the motivic zeta-function of X [19].

4. The `motives` package

4.1. Overview of the package

The `motives` package is organized around two principal subpackages:

i. **Core.** This subpackage provides the fundamental abstract classes for working with λ-rings and integrates `Sympy`'s capabilities. These classes inside the `motives.core` subpackage are not expected to be used directly, but instead they serve as base classes for the rest of the objects implemented in the package. The main modules in this subpackage are:

- **`LambdaRingExpr`.** This is the abstract class from which all of the implemented classes inherit. It sets the layout that all subclasses should follow. It expands the functionality from SymPy's Expr in order to be able to manage λ-rings.
- **`Operand`.** This is the abstract class from which all of the implemented operands inherit. Operands are all classes that act as leaf nodes in the expression tree. It specifies all of the necessary methods operands should implement, as well as defining default behaviours for some of them. These necessary methods are `get_adams_var`, `get_lambda_var`, `_to_adams`, `_apply_adams`, `_to_adams_lambda` and `_subs_adams`. The role of these functions in the main simplification algorithm will be described in Section 4.2.
- **Operators.** All nodes in an expression tree that are not operands are operators, i.e., all non-root nodes. These include the ring operators `Sigma` (σ), `Lambda_` (λ), and `Adams` (ψ). The ring operators are implemented as their own classes, inheriting from `RingOperator`, which itself extends `LambdaRingExpr`. The `SymPy` operators, such as `Add`, `Mul`, and `Pow`, are not implemented as subclasses. Instead, the necessary methods are added directly to their classes by defining them as external functions and associating them with the appropriate `SymPy` methods. This approach ensures compatibility with `SymPy`'s internal instantiation mechanisms, which directly create these classes and would otherwise lack the necessary methods. Note that `Lambda_` has purposely a final "_" due to "`lambda`" being a reserved word in the Python programming language.

- **`LambdaRingContext`.** A singleton (i.e., globally shared) class that computes and caches the universal polynomials relating λ, σ, and ψ operators of a λ-ring, computed through the equations described in Propositions 2.2, 2.1, 2.4, 2.3, 2.5, and [12, §2]. The polynomials are computed on demand either recursively or using the explicit formulas described in Section 2 and saved for future use each time a new polynomial is generated.
- **`Object1Dim`.** Denotes a 1-dimensional object in a λ-ring for the structure σ. Recall that if an element x in a λ-ring (R, λ, σ) is d-dimensional (for the λ-structure σ), then $\sigma^n(x) = 0$ for each $n > d$. Thus, a 1-dimensional object x satisfies

$$\sigma_t(x) = 1 + x$$

and, as a consequence of Eqs (2.2) and (2.5), we have

$$\lambda^n(x) = \psi^n(x) = x^n.$$

ii. **Grothendieck motives subpackage.** On top of these core abstractions, this subpackage defines specific `Operand` subclasses for elements in the λ-ring of Grothendieck motives (e.g., `Curve`, `Jacobian`, `Lefschetz`, etc.). These can implement specialized rules about Adams or λ-operations, thereby integrating geometric relations with the library's symbolic manipulations. Other domain-specific subpackages can follow the same pattern, using the core λ-ring machinery to handle rewriting and caching. See Section 4.4 for more details on each of the elements of the λ-ring of Grothendieck motives implemented.

iii. Other base λ-rings implemented not in a subpackage:

- `Polynomial1Var`: Abstract polynomial symbolic variable yielding a polynomial extension of a λ-ring. Given a λ-ring (R, λ), the ring of polynomials $R[T]$ acquires a natural λ-ring structure in which $\lambda^n(T) = T^n$ (in particular, it becomes a 1-dimensional object for the opposite λ-ring structure to λ). It allows defining expressions in polynomial extensions of rings of motives, like the Grothendieck ring of motives.
- `Free`: Element in a free λ-ring. If no further information is provided, all its λ-powers will be treated as independent algebraic elements in the ring.

Figure 1 shows the relationship among these classes and subpackages.

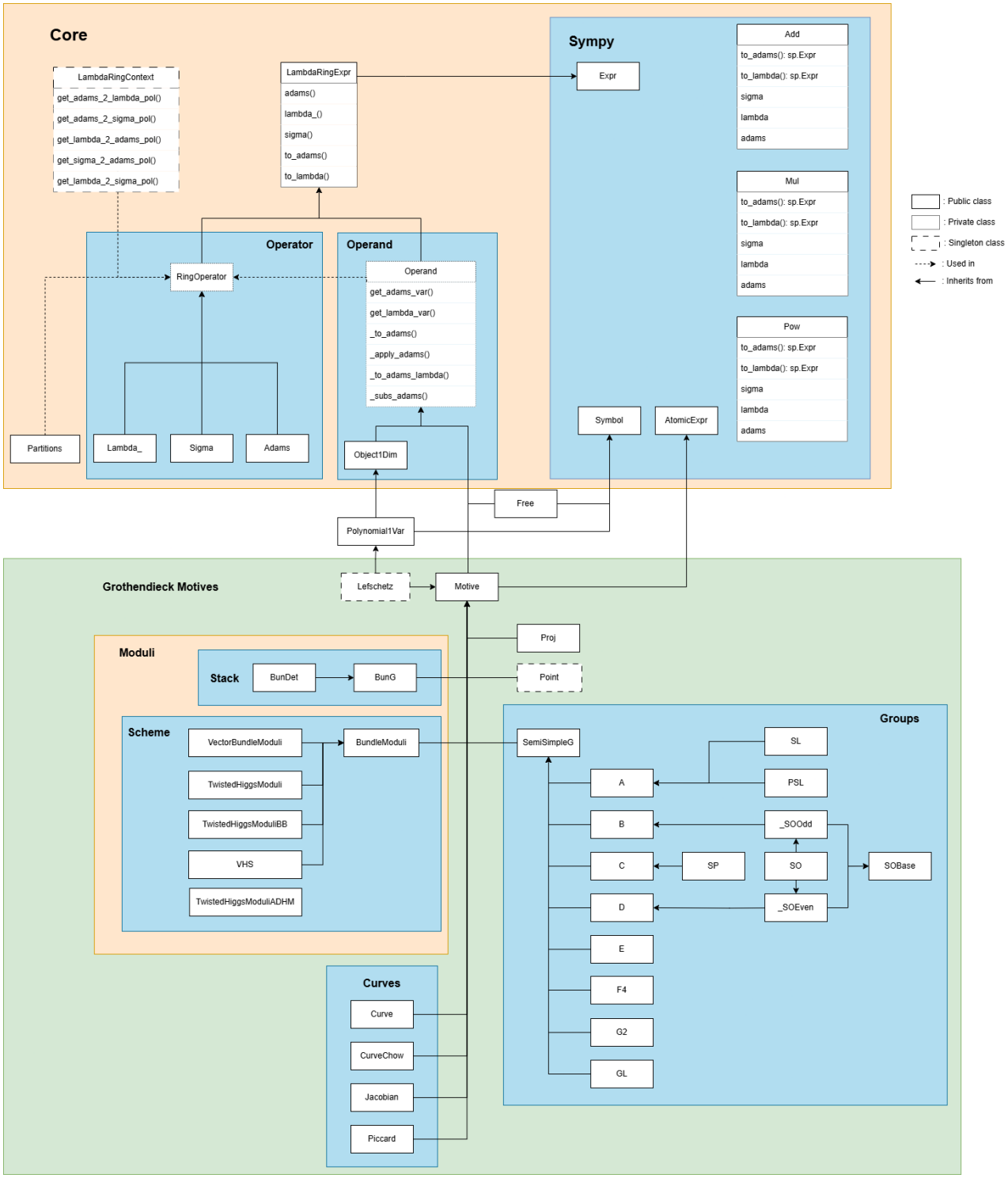

Figure 1. High-level class diagram of the `motives` library and its integration with Sympy. Only the most important public methods are shown; all other methods and internal details have been omitted for clarity. Classes are organized into two colored regions: **Core** (foundation of the lambda-ring framework, including `LambdaRingExpr`, `Operand`, and ring operators), and **Grothendieck Motives** (domain-specific classes for algebraic curves, groups, and moduli schemes/stacks). Solid arrows indicate inheritance, dashed arrows denote usage/composition, dashed borders identify singleton classes, dotted borders indicate private classes and solid borders indicate public classes.

4.1.1. Integration with SymPy

As mentioned before, the `motives` package is built on top of `SymPy` [13]. By leveraging `SymPy`'s existing capabilities for symbolic computation, the `motives` package extends its functionality to support operations using λ-rings and Grothendieck motives. This integration allows for seamless manipulation of λ-ring expressions using familiar symbolic expressions while introducing specialized classes and methods to handle the motivic expressions. Let us give some details on this integration.

The `motives` package introduces a new class, `LambdaRingExpr`, to define an abstract λ-ring expression that inherits and expands the functionality of `SymPy`'s `Expr`. As stated in the previous section, this class serves as the base for all expressions in the λ-ring context. Subclasses of `LambdaRingExpr` implement methods to handle specific types of expressions, ensuring that they conform to the operations and properties of λ-rings.

The other base classes that have been extended are the basic `SymPy` operators (`Add` for addition, `Mul` for element-wise multiplication, and `Pow` for power) and the `Rational` class, which represents all rational numbers. For these classes, no new specific extension has been created, but instead they implement specific methods that are needed to manipulate λ-rings and motives. For instance, they all need the `to_adams` method, which converts an arbitrary expression into a polynomial of Adams operators and operands. For rational numbers, an extension of the following natural λ-ring structures on the integers has been implemented:

$$\lambda^n(x) = \binom{x+n-1}{n}, \qquad \sigma^n(x) = \binom{x}{n}, \qquad \psi^n(x) = x, \quad \forall x \in \mathbb{Z}.$$

The bulk of the module is on the operands. All of them inherit either from `SymPy`'s `Symbol`, or from `SymPy`'s `AtomicExpr`, which is an abstraction of `Symbol`. This gives them many useful properties, such as the associative and commutative properties, but it also allows the expression to handle more powerful methods, such as expand or simplify. The λ-ring operators (`Adams`, `Lambda` and `Sigma`) are extensions of `SymPy`'s `Function`.

4.2. The simplification algorithm

The main strategy followed for the simplification algorithm is based on [12, Algorithm 1 and Theorem 4], but some further abstractions and simplifications have been made in order to yield a general implementation of such an algorithm. In order to simplify arbitrary algebraic expressions possibly combining λ, σ, and ψ in non-trivial ways, we use the fact that Adams operations are ring homomorphisms (see, for instance, [15, Theorem 9.2]) and that we can use the expressions obtained in Section 2 and in [12, Section 2] to rewrite any other operators λ^n or σ^n as polynomials in Adams operations. More concretely, as ψ^n is a ring homomorphism for any n, if $P(\psi^{k_1}(x_1), \dots, \psi^{k_s}(x_s)) \in \mathbb{Q}[\psi^{k_1}(x_1), \dots, \psi^{k_s}(x_s)]$ is a rational polynomial expression depending on Adams operations of some $x_1, \dots, x_s \in R$, then,

$$\psi^n(P(\psi^{k_1}(x_1), \dots, \psi^{k_s}(x_s))) = P(\psi^{nk_1}(x_1), \dots, \psi^{nk_s}(x_s)). \tag{4.1}$$

This could also be done if P is a rational expression instead of a polynomial. We can use this principle recursively to transform any expression tree formed by a composition of ring operations and Adams

operations into a polynomial (respectively, rational expression) in a finite set of Adams operations of the leaves of the given expression tree.

Furthermore, the Eqs (2.8) and (2.6) provide a way to compute $\lambda^n(x)$ and $\sigma^n(x)$ as polynomials depending on the Adams operations $\psi^k(x)$ for $k \leq n$ for any $x \in R$. The `motives` package combines these two methods to recursively transform any λ-ring expression into a polynomial in Adams operations as follows.

As introduced in Section 4.1, each expression tree is a `LambdaRingExpr` whose leaf nodes are subclasses of `Operand`. The `to_adams` method rewrites any such expression as a `SymPy` polynomial depending on the Adams operations of its leaves. The `Operand` class has been defined to characterize admissible "leaf objects" or "operands" for λ-ring expressions. In order for the simplification algorithm to take into account possible non-trivial and domain-specific algebraic relations between different Adams operations of a certain type of objects, the computation and symbolic representation of Adams operations of an element are always delegated to the corresponding `Operand` object in the following sense.

Any `Operand` object representing an element $x \in R$ in the λ-ring must implement an `_apply_adams` method capable of receiving a positive integer n and a polynomial P of the form

$$P(\psi^1(x), \ldots, \psi^k(x), \psi^{k_1}(x_1), \ldots, \psi^{k_s}(x_s)) \tag{4.2}$$

depending on Adams operations of x and possibly other elements $x_1, \ldots, x_s$ and performs a partial Adams operation on the polynomial, by replacing the instances of the Adams operations of x on which P depends, namely $\psi^1(x), \ldots, \psi^k(x)$, with the result of applying ψ^n to them, leaving any other Adams operation in the expression P unaffected. Concretely, if P is a polynomial as in (4.2), then `x._apply_adams(n,P)` would yield

$$P(\psi^n(x), \ldots, \psi^{nk}(x), \psi^{k_1}(x_1), \ldots, \psi^{k_s}(x_s)).$$

Depending on the specific context of what x represents, the concrete λ-ring where this expression lives and possible algebraic relations existing between the Adams operations of x, the computed elements $\psi^{kn}(x)$ substituted into P might be new algebraically independent symbolic variables created to represent such Adams operations, some expressions (depending possibly on other Adams operations of x or other elements in the ring), etc. Some more complex manipulations of the polynomials could also be possible. This is left to depend on the implementation of the specific `Operand`.

Notice that if $P = P(\psi^{k_1}(x_1), \ldots, \psi^{k_s}(x_s))$ is an Adams polynomial depending on $x_1, \ldots, x_s \in R$, then applying `P=`x_i`._apply_adams(n,P)` iteratively for each $i = 1, \ldots, s$ would yield exactly $\psi^n(P)$, by (4.1). This behaviour is actually what defines functionally the `_apply_adams` method. Each `Operand` must implement it so that an iterated application of the method for each element on which an Adams polynomial depends results in the application of ψ^n to such a polynomial.

Counting on the existence of this delegated `_apply_adams` method on the leaves of an expression tree, the simplification `to_adams` algorithm performs a recursive walk through the tree acting as follows on each traversed node.

1) If the node is a built-in `SymPy` operator node corresponding to a ring operation (`Add`, `Mul`, `Pow`), then perform `to_adams` at the operand children of the node to transform the operand expressions into Adams polynomials and perform the operation designated by the node on the corresponding results.

2) If the node is an `Adams` operation, representing $\psi^n(T)$ for some child expression tree T, then apply `to_adams` to the child T to obtain a polynomial (respectively, a rational expression)

$$P(\psi^{k_{1,1}}(x_1), \ldots, \psi^{k_{1,n_1}}(x_1), \ldots, \psi^{k_{s,n_s}}(x_s))$$

which represents T and depends on the Adams operations of the `Operand` objects $x_1, \ldots, x_n$ appearing in the leaves of the tree T. Then, as described before, iteratively apply `P=`x_i`.to_adams(n,P)` in order to compute $\psi^n(P) = \psi^n(T)$, delegating the computation of the Adams operations to the operands x_i in T.

3) If the node is a `Lambda_` or `Sigma` operator, i.e., it represents $\lambda^n(T)$ or $\sigma^n(T)$ respectively for some child expression tree T, then apply `to_adams` to T in order to obtain an equivalent Adams polynomial P to T. Then, compute $\psi^k(P)$ for each $k = 1, \ldots, n$ by applying the same technique as for the `Adams` nodes as follows. For each k, apply iteratively x_i`._apply_lambda(k,P)` for each leaf x_i in T to obtain a polynomial $P_k := \psi^k(P)$. Then, use the polynomials L_n and L_n^{op} from [12, Proposition 3 and Corollary 1], computed through Eqs (2.8) and (2.6), respectively, to compute

$$\lambda^n(T) = \lambda^n(P) = L_n^{op}(\psi^1(P), \ldots, \psi^n(P)) = L_n^{op}(P_1, \ldots, P_k)$$

or

$$\sigma^n(T) = \sigma^n(P) = L_n(\psi^1(P), \ldots, \psi^n(P)) = L_n(P_1, \ldots, P_k)$$

as necessary.

4) Finally, for terminal `Operand` nodes (that is, a leaf of the tree), the implementation of the `to_adams` method depends on the specific λ-ring and element being computed. For example, an integer or free symbolic variable will simply return itself.

Figure 2 shows an overview of the steps taken to convert the expression $\lambda^2(\psi^2(x) - \frac{y}{2})$ to a polynomial of Adams operators of x and y following the algorithm described.

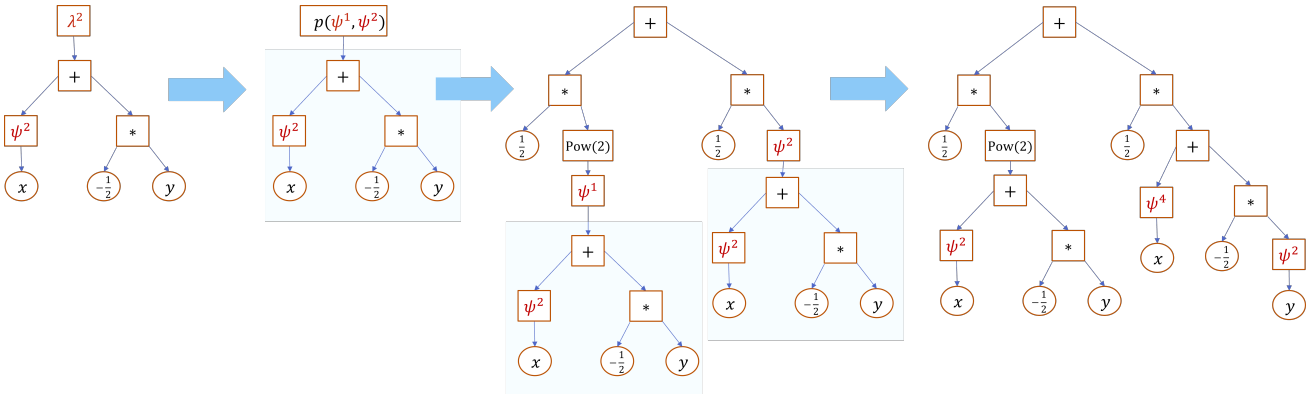

Figure 2. Step-by-step illustration of the `to_adams()` algorithm. Starting with the expression $\lambda^2(\psi^2(x) - \frac{y}{2})$, the algorithm iteratively expands the λ^2 and ψ^2 operations following universal λ-ring relations. Each intermediate tree replaces higher-level operators with expanded terms in the corresponding ψ-powers, until the final expression is a polynomial purely in Adams operators and basic ring operations.

An additional `to_lambda` method was also implemented for transforming the expression into a polynomial depending only on λ-powers instead of depending on λ-operations. This becomes useful when the expression depends on objects that are known to be finite-dimensional for the λ structure, so that the resulting simplified polynomial only depends on a known finite set of generators (see Remark 2.7). Moreover, as explained in Section 3, in the Grothendieck rings of varieties and Chow motives λ-operations have an actual geometric meaning, and a decomposition in terms of polynomials of such objects can sometimes be interpreted as very useful geometric decompositions of a variety. To compute `to_lambda`, first `to_adams` is computed so that the tree T is transformed into a polynomial of the form $T = P(\psi^{k_1}(x_1), \dots, \psi^{k_s}(x_s))$. Then Propositions 2.4 and 2.5 are used to compute $\psi^{k_i}(x_i)$ as a polynomial in terms of $\lambda^1(x_i), \dots, \lambda^{k_i}(x_i)$ for each i. These polynomials are then substituted into P to yield the desired polynomial expression for the tree T. This substitution is again delegated to the leaf operands, which must implement a method called `_subs_adams`. Analogously to `_apply_adams`, the `_subs_adams` method is called iteratively on the Adams polynomial for each leaf `Operand` appearing in the expression tree, and calling it for a leaf substitutes all instances of Adams operations of the given operand by their corresponding polynomial expressions in terms of its λ-powers.

One could remark that, if the starting expression only depends on λ-operations to begin with, then transforming them completely to Adams operations ψ^k in order to transform them back to λ-operations λ^k at the end could be unnecessary for simplifying it in some situations. For instance, if an expression has some parts that are already polynomials in λ, then it would be very inefficient to transform that part of the tree to Adams just to transform it back to λ. This would actually only be needed if there is another operator (λ, σ, or ψ) acting on the subtree, which would require transforming all operators to Adams in order to compute the corresponding composition. In order to avoid this, an optimization was made in the code to detect these types of situations and only transform λ operators into Adams when needed, implemented through a `_to_adams_lambda` method.

As the polynomials relating λ, σ, and ψ are universal, a singleton class `LambdaRingContext` was implemented to compute these polynomials and save them cached for reuse to improve efficiency.

4.3. Usage example

`motives` can simplify any expression into a polynomial in Adams or λ operations by calling the methods `to_adams()` or `to_lambda()` on the expression, respectively. An example goes as follows. To perform the simplification of the expression

$$\lambda^2\left(\psi^2(x) - \frac{y}{2}\right),$$

Using the `motives` package, we can run the following Python code:

```
x, y = Free("x"), Free("y")

expr = (x.adams(2) - y / 2).lambda_(2)
print(expr.to_adams())
```

which yields the following equivalent polynomial expression depending solely on x, y and the necessary

Adams operations of x and y.

$$-\frac{\psi^2(y)}{4}+\frac{\psi^4(x)}{2}+\frac{(\psi^2(x)-\frac{y}{2})^2}{2}.$$

In the code, x and y were declared as `Free` objects, so no further assumptions were made on them, and the computation was performed in the free λ-ring spanned by x and y. If `to_lambda()` is called on the expression instead of `to_adams()`, we get the following polynomial depending on the λ powers of x and y instead of their Adams operations

$$\frac{-x^4}{2}+2x^2\lambda^2(x)-2x\lambda^3(x)+\frac{y^2}{4}-(\lambda^2(x))^2-\frac{\lambda^2(y)}{2}+2\lambda^4(x)+\frac{(-x^2-\frac{y}{2}+2\lambda^2(x))^2}{2}.$$

A graphical depiction of this process can be seen in Figure 2. This expression can be further simplified using the `simplify()` method provided by `Sympy`:

```
print(expr.to_lambda().simplify())
```

which yields the following polynomial still depending on the λ powers of x and y.

$$\frac{x^2y}{2}-2x\lambda^3(x)+\frac{3y^2}{8}-y\lambda^2(x)+\lambda^2(x)^2-\frac{\lambda^2(y)}{2}+2\lambda^4(x).$$

4.4. Grothendieck motives

Let us describe the main classes implemented in the `motives.grothendieck` subpackage for handling expressions in extensions of the Grothendieck ring of Chow motives $\hat{K}_0(\mathcal{CM}_{\mathbb{C}})$. It contains some classes for handling basic motives, as well as some sub-packages for handling motives depending on algebraic curves, algebraic groups, and moduli schemes and stacks. The package includes the following base classes.

- `Motive`: Base class for all Grothendieck motives. All other classes inherit from it. It implements methods for computing the symmetric and alternated powers of a motive.
- `Lefschetz`: Lefschetz object $\mathbb{L}=[\mathbb{A}^1]$.
- `Point`: Class of a point.
- `Proj`: Class of the projective space $[\mathbb{P}^n]$. Even though this motive can be clearly expressed in terms of $\mathbb{L}$ as $[\mathbb{P}^n]=\mathbb{L}^n+\ldots+\mathbb{L}+1$, we have included it as an independent class as a convenient shortcut that can be used in formulas, as well as to give support to symbolic factorizations that could involve projective factors.

Moreover, the following sub-packages are included.

4.4.1. *Curves*

Submodule with classes for handling and simplifying motives depending on the class of an abstract smooth complex algebraic curve of genus g, for a given g. Every smooth complex projective curve X admits a canonical decomposition for its Chow motive

$$[X]=h^0(X)+h^1(X)+h^2(X)=1+h^1(X)+\mathbb{L}.$$

and, by [21], $h^1(X)$ is $2g$-dimensional for λ. Concretely, in the following formulas, we will use the notation

$$P_X(t) = Z_{h^1(X)}(t) = \sum_{n=0}^{2g} \lambda^n(h^1(X))t^n$$

The module contains the following classes.

- `Curve`: Class representing the motive of an abstract complex projective curve of genus g. It implements the identities and decompositions from [21] and [19], articulating expressions depending on the curve X as algebraic expressions in a finite set of algebraic generators depending on its Chow decomposition $h^1(X)$ (see below for details).
- `CurveChow`: Chow decomposition $h^1(X)$ of a generic genus g curve X. It corresponds to $h^1(X) = [X] - 1 - \mathbb{L}$. Since X is considered as an abstract algebraic curve, the only algebraic relations between the elements $\lambda^k(h^1(X))$ being considered during simplification are the following identities arising from [21] (see also [19]).

  $$\lambda^k(h^1(X)) = \mathbb{L}^{k-g}\lambda^{2g-k}(X) \qquad \forall g < k \leq 2g$$

  $$\lambda^k(h^1(X)) = 0 \qquad \forall k > 2g$$

  In particular, by default, $\lambda^k(X)$ for $k = 1, \ldots, g$ and $\mathbb{L}$ are considered as algebraically independent objects in the ring of motives. So are $\lambda^i(X)$ and $\lambda^j(Y)$ for $i, j = 1, \ldots, g$ if X and Y are different curves.
- `Jacobian`: Jacobian of a curve X. Computed in terms of the motivic zeta function of $h^1(X)$ (see [21] and [19])

  $$[\mathrm{Jac}(X)] = \sum_{k=0}^{2g} \lambda^k(h^1(X)) = P_X(1).$$

- `Piccard`: Picard variety of a curve X. The motive coincides with that of its Jacobian.

4.4.2. *Groups*

Submodule with classes of several complex algebraic groups, based on the formulas from [2]. For a connected semisimple complex group G, the motive is computed as a polynomial in $\mathbb{L}$ through the following equation from [2, Proposition 2.1]

$$[G] = \mathbb{L}^{\dim G} \prod_{i=1}^{r} (1 - \mathbb{L}^{-d_i}), \tag{4.3}$$

where d_i are the degrees of the basic invariant generators of G, and r is its rank. For classical groups, these were obtained from [22, Table 1, pp. 59]. The implemented groups include the following.

- `SemisimpleG`: Class of a general connected semisimple complex algebraic group with exponents $d_1, \ldots, d_n$. Computed through Eq (4.3). It is also used to describe any other possibly non-semisimple group whose motive is nonetheless computed analogously to a that of a connected semisimple complex algebraic group in the sense that it can be calculated in terms of a set of exponents $d_1, \ldots, d_n$ through Eq (4.3) (like GL_n).

- `A`: Motivic class of any connected semisimple complex algebraic group of type A_n.
- `B`: Connected semisimple complex algebraic group of type B_n.
- `C`: Connected semisimple complex algebraic group of type C_n.
- `D`: Connected semisimple complex algebraic group of type D_n.
- `E`: Connected semisimple complex algebraic group of type E_n, for $n = 6, 7, 8$.
- `F4`: Connected semisimple complex algebraic group of type F_4.
- `G2`: Connected semisimple complex algebraic group of type G_2.
- `GL`: $\mathrm{GL}_n(\mathbb{C})$.
- `SL`: $\mathrm{SL}_n(\mathbb{C})$.
- `PSL`: $\mathrm{PSL}_n(\mathbb{C})$.
- `SO`: $\mathrm{SO}_n(\mathbb{C})$.
- `Sp`: $\mathrm{Sp}_{2n}(\mathbb{C})$.
- `Spin`: $\mathrm{Spin}_n(\mathbb{C})$.

Remark 4.1. *In addition to the classical and exceptional groups, the motivic equations computed in the package for groups of types A_n, B_n, C_n, D_n, E_n, F_4, or G_2 remain valid for any other connected semisimple complex algebraic group of the given types, since the invariant polynomials and, therefore, the invariant degrees of two semisimple groups with the same Lie algebra coincide and, thus, Eq* (4.3) *yields the same motivic expression for any of them. For example, the Python class* `E` *in this package can be used to compute the motive of the exceptional group E_6, but also to obtain the motive of the quotient group $E_6/\mathbb{Z}_3$ of E_6 by its center $\mathbb{Z}_3$.*

4.4.3. *Moduli schemes and moduli stacks*

Submodule with classes for describing the motivic class of some moduli spaces of decorated bundles on curves (in the future, other moduli spaces will be added to this package as well; see Section 6). See Section 5 for further details on the considered moduli. It is composed of a submodule for handling moduli schemes and another submodule for handling moduli stacks.

The moduli schemes already implemented are:

- `VectorBundleModuli`: Class of the moduli space of semistable vector bundles of rank r and degree d on a smooth complex projective algebraic curve X of genus $g \geq 2$, computed using the following equations from [6], [5], and [17, Theorem 4.11], assuming that r and d are coprime.

$$
\begin{aligned}
[M(X,2,d)] =& \frac{[\mathrm{Jac}(X)]P_X(\mathbb{L}) - \mathbb{L}^g[\mathrm{Jac}(X)]^2}{(\mathbb{L}-1)(\mathbb{L}^2-1)} \\
[M(X,3,d)] =& \frac{[\mathrm{Jac}(X)]}{(\mathbb{L}-1)(\mathbb{L}^2-1)^2(\mathbb{L}^3-1)}\Big(\mathbb{L}^{3g-1}(1+\mathbb{L}+\mathbb{L}^2)[\mathrm{Jac}(X)]^2 \\
& - \mathbb{L}^{2g-1}(1+\mathbb{L})^2[\mathrm{Jac}(X)]P_X(\mathbb{L}) + P_X(\mathbb{L})P_X(\mathbb{L}^2)\Big) \\
[M(X,r,d)] =& \sum_{s=1}^{r} \sum_{\substack{r_1+\ldots+r_n=r \\ r_i>0}} (-1)^{s-1}\frac{P_X(1)^s}{(1-\mathbb{L})^{s-1}} \prod_{j=1}^{s}\prod_{i=1}^{r_j-1} Z_X(\mathbb{L}^i)
\end{aligned}
$$

$$\prod_{j=1}^{s-1} \frac{1}{1-\mathbb{L}^{r_j+r_{j+1}}} \mathbb{L}^{\sum_{i<j} r_i r_j (g-1) + \sum_{i=1}^{s-1} (r_i+r_{i+1})\{-(r_1+\ldots+r_i)d/n\}}$$

where $\{x\}$ denotes the decimal part of $x \in \mathbb{R}$, i.e., $\{x\} = x - \lfloor x \rfloor$.

- `VHS`: Class of the moduli space of variations of Hodge structure, or moduli space of chains of a given type. The currently implemented types are $(1,1)$, $(1,2)$, $(2,1)$, $(1,1,1)$ and (r) (which corresponds to moduli spaces of vector bundles). Further types will be added in the future. Computed using the equations from [6], [5], and the computations from [8]. See [8, §8] for details.
- `TwistedHiggsModuli` Class of the moduli space of L-twisted Higgs bundles of rank r and degree d on a genus $g \geq 2$ smooth complex projective curve. Two methods have been implemented for computing this class.
  - `TwistedHiggsModuliBB`: Class computed for rank 2 and 3 of the moduli space of L-twisted through a Bialynicki-Birula of the moduli using [8, Corollary 8.1] and [6, Theorem 3]. These formulas were proven when the rank and degree are coprime.
  - `TwistedHiggsModuliADHM`: Conjectural class for the moduli on arbitrary rank and degree. Based on the formulas from [7, Conjecture 3], which are solutions to the motivic ADHM equations [23].

The moduli stacks already implemented are:

- `BG`: Classifying space for the group G, $BG = [pt/G]$ for a connected semisimple complex algebraic group G. It is computed as
$$[BG] = 1/[G],$$
with $[G]$ computed through equation (4.3). This formula is conjectural in general, but it has been proven for special groups (like $\mathrm{GL}_n(\mathbb{C})$, $\mathrm{SL}_n(\mathbb{C})$ and $\mathrm{Sp}_{2n}(\mathbb{C})$) [2, Example 2.6], for $\mathrm{PSL}_n(\mathbb{C})$ if $n = 2, 3$ [24, Theorem A], for $\mathrm{SO}_n(\mathbb{C})$ [25, Theorem 3.7 and Corollary 3.8]), and for O_n [26, Theorem 3.1 and Corollary 3.2].
- `Bun`: Moduli stack of principal G-bundles on a smooth complex projective curve X, computed through the following conjectural formula from [2, Conjecture 3.4]
$$[\mathfrak{Bun}(X,G)] = |\pi_1(G)| \mathbb{L}^{(g-1)\dim G} \prod_{i=1}^{r} Z_C(\mathbb{L}^{-d_i}),$$
where d_i are the exponents of the group and r its rank. The conjecture was proven for $G = \mathrm{SL}_n(\mathbb{C})$ in [2, §6].

5. Applications to the computation of the motive of twisted Higgs bundles

Let X be a smooth complex projective curve of genus $g \geq 2$, and let L be a line bundle on X of degree $\deg(L) = 2g - 2 + p$ with $p > 0$. An L-twisted Higgs bundle of rank r on X is a pair (E, φ) consisting of a rank r vector bundle E and a homomorphism $\varphi \in H^0(\mathrm{End}(E) \otimes L)$. We say that (E, φ) is semistable if for any subbundle $F \subsetneq E$ such that $\varphi(F) \subseteq F \otimes L$, the following inequality holds.

$$\frac{\deg(F)}{\mathrm{rk}(F)} \leq \frac{\deg(E)}{\mathrm{rk}(E)}.$$

Let $\mathcal{M}_L(X,r,d)$ denote the moduli space of semistable L-twisted Higgs bundles on X of degree r and rank d. Through this section, assume that r and d are coprime. There is a natural $\mathbb{C}^*$ action on the moduli space given by $t\cdot(E,\varphi)=(E,t\varphi)$, giving the moduli space the structure of a smooth semiprojective variety of dimension $1+r^2(2g-2+p)$. In [7], Mozgovoy stated the following conjectural formula for this moduli space, which is a solution to the motivic ADHM recursion formula [23].

Conjecture 5.1. *[7, Conjecture 3] For each integer $n\geq 1$, let*

$$\mathcal{H}_n(t)=\sum_{a\in\mathcal{P}(n)}\prod_{s\in d(a)}(-t^{a(s)-l(s)}\mathbb{L}^{a(s)})^p t^{(1-g)(2l(s)+1)}Z_X(t^{h(s)}\mathbb{L}^{a(s)}),$$

where $\mathcal{P}(n)$ denotes the set of ordered partitions of n, a partition $a=(a_1,\dots,a_k)\in\mathcal{P}(n)$ is considered as a non-increasing sequence of positive integers $a_1\geq a_2\geq\cdots\geq a_k>0$ summing n, and, for each $a\in\mathcal{P}(n)$, we consider the Young diagram of a, given as

$$d(a)=\{(i,j)\in\mathbb{Z}^2\,|\,1\leq i,\ 1\leq j\leq a_i\},$$

and for each element $(i,j)\in d(a)$, its arm, leg and hook functions, defined as

$$a(i,j)=a_i-j,\quad l(i,j)=\max\{l\,|\,a_l\geq j\}-i,\qquad h(i,j)=a(i,j)+l(i,j)+1.$$

From $\mathcal{H}_n(t)$, define $H_r(t)$ for each $r\geq 1$ as follows:

$$\sum_{r\geq 1}H_r(t)T^r=(1-t)(1-\mathbb{L}t)\sum_{j\geq 1}\sum_{k\geq 1}\frac{(-1)^{k+1}\mu(j)}{jk}\left(\sum_{n\geq 1}\psi_j[\mathcal{H}_n(t)]T^{jn}\right)^k.$$

Then $H_r(t)$ is a polynomial in t and

$$[\mathcal{M}_L(X,r,d)]=M^{\mathrm{ADHM}}_{g,r,p}:=(-1)^{pr}\mathbb{L}^{r^2(g-1)+p\frac{r(r+1)}{2}}H_r(1). \tag{5.1}$$

This conjecture implies analogous formulas for the E-polynomial of the moduli space and for the number of rational points in the moduli space, which were later proven in [27, Theorems 1.1 and 4.6].

On the other hand, in [8], the following formulas were proven for the virtual classes of moduli spaces of L-twisted Higgs bundles of degree d and rank at most 3 by computing the Bialynicki-Birula decomposition of the variety with respect to the previously mentioned natural $\mathbb{C}^*$-action on the L-twisted Higgs moduli scheme.

Theorem 5.2 ([6, Theorem 3], [8, Corollary 8.1])**.** *The following equalities hold in $\hat{K}_0(\mathcal{V}ar_{\mathbb{C}})$.*

1). For $r=1$

$$[\mathcal{M}_L(X,1,d)]=M^{\mathrm{BB}}_{g,1,p}:=[\mathrm{Jac}(X)\times H^0(X,L^\vee)]=\mathbb{L}^{g-1+p}P_X(1) \tag{5.2}$$

2). For $r=2$, if $(2,d)=1$,

$$\begin{aligned}[\mathcal{M}_L(X,2,d)]=M^{\mathrm{BB}}_{g,2,p}:=&\frac{\mathbb{L}^{4g-4+4p}\left(P_X(1)P_X(\mathbb{L})-\mathbb{L}^gP_X(1)^2\right)}{(1-\mathbb{L})(1-\mathbb{L}^2)}\\&+\mathbb{L}^{4g-4+3p}P_X(1)\sum_{i=1}^{\lfloor\frac{2g-1+p}{2}\rfloor}\lambda^{2g-1+p-2i}([X]).\end{aligned} \tag{5.3}$$

3). For $r = 3$, if $(3, d) = 1$,

$$[\mathcal{M}_L(X,3,d)] = M^{\mathrm{BB}}_{g,3,p} := \frac{\mathbb{L}^{9g-9+9p}P_X(1)}{(\mathbb{L}-1)(\mathbb{L}^2-1)^2(\mathbb{L}^3-1)}\Big(\mathbb{L}^{3g-1}(1+\mathbb{L}+\mathbb{L}^2)P_X(1)^2 - \mathbb{L}^{2g-1}(1+\mathbb{L})^2P_X(1)P_X(\mathbb{L}) + P_X(\mathbb{L})P_X(\mathbb{L}^2)\Big)$$
$$+ \frac{\mathbb{L}^{9g-9+7p}P_X(1)^2}{\mathbb{L}-1}\sum_{i=1}^{\lfloor \frac{1}{3}+\frac{2g-2+p}{2}\rfloor}\Big(\mathbb{L}^{i+g}\lambda^{-2i+2g-2+p}([X]+\mathbb{L}^2) - \lambda^{-2i+2g-2+p}([X]\mathbb{L}+1)\Big)$$
$$+ \frac{\mathbb{L}^{9g-9+7p}P_X(1)^2}{\mathbb{L}-1}\sum_{i=1}^{\lfloor \frac{2}{3}+\frac{2g-2+p}{2}\rfloor}\Big(\mathbb{L}^{i+g-1}\lambda^{-2i+2g-1+p}([X]+\mathbb{L}^2) - \lambda^{-2i+2g-1+p}([X]\mathbb{L}+1)\Big)$$
$$+ \mathbb{L}^{9g-9+6p}P_X(1)\sum_{i=1}^{2g-2+p}\sum_{j=\max\{2-2g-p+i,1-i\}}^{\lfloor (2g-1+p-i)/2\rfloor}\lambda^{-i+j+2g-2+p}([X])\lambda^{-i-2j+2g-1+p}([X]). \quad (5.4)$$

A direct comparison between the conjectured expression (5.1) and the proven expressions (5.2)–(5.4) is not immediate, especially due to the Adams operations acting on the terms $\mathcal{H}_n(t)$ in (5.1) and the sums in the Eqs (5.3) and (5.4). In [12], an ad hoc MATLAB code was written for manipulating these specific expressions using the general simplification algorithm described in that article. As a result, it was proven that the conjectural formulas $M^{\mathrm{ADHM}}_{g,r,p}$ from 5.1 and the proved formulas $M^{\mathrm{BB}}_{g,r,p}$ from Theorem 5.2 agree in $\hat{K}_0(C\mathcal{M}_{\mathbb{C}})$ in the following cases:s

- X is any curve of genus g with $2 \leq g \leq 11$,
- L is any line bundle on X of degree $2g-1 \leq \deg(L) \leq 2g+18$, and
- $1 \leq r \leq 3$.

The computations were carried out on an Intel(R) Xeon(R) E5-2680v4@2.40GHz with 128GB of RAM, and it was found that the main limitation for extending the verification beyond these limits was the over-exponential growth of the memory usage of the MATLAB code with respect to the genus. The test for $r = 3$, $g = 11$ curves, and $\deg(L) = 2g + 18$ used all the available RAM in the machine. As a way to test the general-purpose simplification library proposed in this work, we have revisited this problem using the new `motives` package and compared its performance with the previous ad-hoc MATLAB implementation of the algorithm. First, the following simplifications, which were also applied in [12], were performed in Eq (5.4). As λ is a λ-ring structure and $\mathbb{L}$ is a 1-dimensional object, then for any $n \geq 0$

$$\lambda^n([X]+\mathbb{L}^2) = \sum_{k=0}^{n}\lambda^k([X])\lambda^{n-k}(\mathbb{L}^2) = \sum_{k=0}^{n}\lambda^k([X])\mathbb{L}^{2n-2k},$$

$$\lambda^n([X]\mathbb{L}+1) = \sum_{k=0}^{n}\lambda^k([X]\mathbb{L}) = \sum_{k=0}^{n}\lambda^k([X])\mathbb{L}^k.$$

The `to_lambda` method from `motives` was then applied to both expressions for the motive of the moduli space. In the case of (5.1), terms in $t-1$ were collected and cancelled before evaluating $H_r(1)$, and the resulting polynomials were compared through a standard `SymPy` comparison of symbolic expressions.

We run an instance of `motives` in the same machine with the same RAM limitations for this problem. The library was able to greatly surpass the capabilities of the previous ad-hoc code, and it compared both expressions successfully, showing that they are equal in $\hat{K}_0(C\mathcal{M}_{\mathbb{C}})$ whenever

- X is any curve of genus g with $2 \leq g \leq 18$.
- L is any line bundle on X of degree $2g - 1 \leq \deg(L) \leq 2g + 18$, and
- $1 \leq r \leq 3$.

As a consequence, Theorem 5.2 implies that Mozgovoy's conjecture holds under those conditions, yielding the following theorem.

Theorem 5.3. *Let X be any smooth complex projective curve of genus $2 \leq g \leq 18$. Let L be any line bundle on X of degree $\deg(L) = 2g - 2 + p$ with $0 < p \leq 20$. If $r \leq 3$ and d is coprime with r, then, in $\hat{K}_0(C\mathcal{M}_{\mathbb{C}})$,*

$$[\mathcal{M}_L(X, r, d)] = M^{\mathrm{BB}}_{g,r,p}.$$

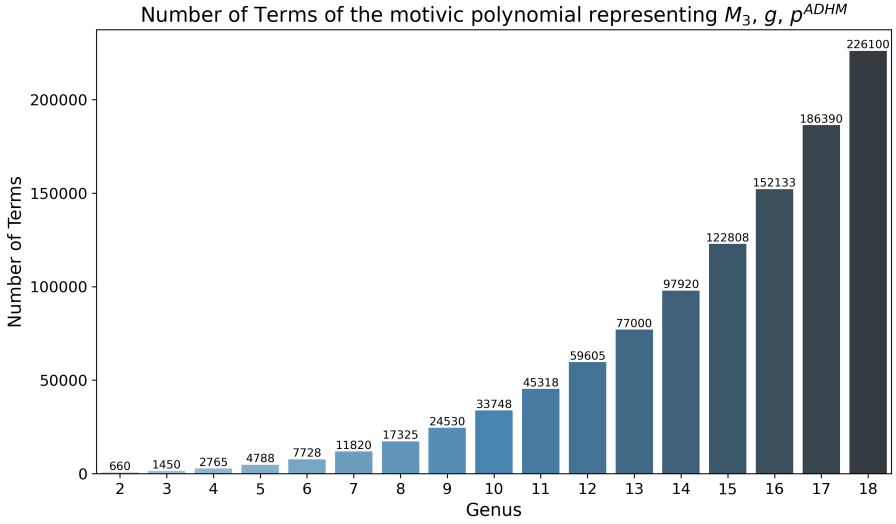

Figure 3. Size comparison between the polynomials generated for $M^{\mathrm{BB}}_{g,3,20} = M^{\mathrm{ADHM}}_{g,3,20}$ after simplification for different g, all taken with $r = 3$ and $p = 20$.

To put in perspective the jump from genus $g = 11$ to genus $g = 18$, one should consider how the expressions grow with respect to the genus. The moduli space has dimension $\dim(\mathcal{M}_L(X, r, d)) = r^2 \deg(L) + 1$, so the largest moduli space processed in [12], corresponding to genus 11, rank 3, and $\deg(L) = 2g + 18 = 40$, had dimension 361. The largest moduli space considered in the current work, corresponding to genus 18, rank 3, and $\deg(L) = 2g + 18 = 54$, has now dimension 487. The algorithm simplifies the motivic expressions into a multivariate polynomial in $g + 1$ (independent) generators, and its degree is at most the dimension of the moduli space. In [12], the maximum test performed resulted in a degree 361 polynomial in 12 variables with 45,318 terms. In contrast, the new results were attained for a moduli space whose motive is represented by a degree 487 polynomial in 19 variables with 226,100 terms, 5 times larger than the largest expression which the old code was able to handle in that given machine. Figure 3 shows the increase in the number of terms of the computed simplified expression. It is important to notice that the computational complexity of manipulating and simplifying

the type of nested polynomial expressions arising naturally from the application of the method to the given expressions raises significantly when the number of variables of the polynomials increases, even slightly. Consider, for instance, that even though the obtained final simplified polynomials are relatively small in this case, polynomials of degree 361 in 12 variables can have at most $\sim 3.4 \cdot 10^{24}$ possible terms, whereas polynomials of degree 487 in 19 variables could have up to $\sim 3.1 \cdot 10^{36}$ terms.

Figure 4 shows a comparison between the time taken by the algorithm in [12] and the `motives` library to simplify and compare the motivic expressions $M^{BB}_{g,r,p}$ and $M^{ADHM}_{g,r,p}$ for $r = 3$ and $p \leq 20$ for different values of g. The variation in execution time with respect to other choices of p is relatively small, so only the time needed for computing the biggest moduli for each g ($r = 3$ and $p = 20$) with the two methods is shown in the figure for clarity. The chart for the [12] algorithm ends at $g = 11$ as the memory limit for the machine was reached at that point. Table 1 shows the actual time taken to perform the calculations. During the experiments, running the `motives` library for $g = 18$ also used a significant part of the memory, but there was still room to continue increasing the genus. The test of the library was stopped at $g = 18$ due to the exponential growth of the run time of the algorithm for the given problem.

Table 1. Comparison between the computation times in seconds for the verification of Mozgovoy's conjectural formula using the simplification algorithm from the `Motives` Python package and the MATLAB implementation of the algorithm from [12]. From $g = 11$, the MATLAB program reached the memory limit of the machine.

g	`Motives` Python package implementation (s)	MATLAB implementation of [12] (s)
2	4.48e+01	7.58e+01
3	9.96e+01	1.09e+02
4	2.41e+02	2.01e+02
5	4.06e+02	4.18e+02
6	7.78e+02	8.51e+02
7	1.47e+03	1.79e+03
8	2.74e+03	3.44e+03
9	4.96e+03	1.01e+04
10	8.80e+03	4.60e+04
11	1.58e+04	>2.85e+05
12	2.87e+04	-
13	5.14e+04	-
14	1.02e+05	-
15	2.18e+05	-
16	3.83e+05	-
17	6.09e+05	-
18	1.79e+06	-

It is worth remarking that the new (general purpose) library took less time to compute the biggest $g = 15$ case (whose simplified polynomial has 122,808 terms) than it took for the code in [12] to compute the $g = 11$ case (with 45,318 terms, almost three times less), and it computed the $g = 11$ cases an order of magnitude faster than its predecessors ad-hoc implementation for the problem.

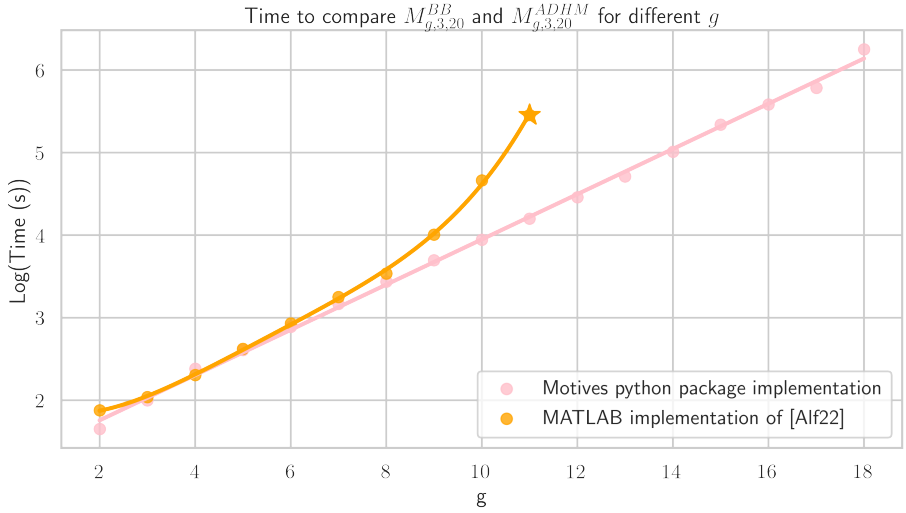

Figure 4. Time comparison between algorithm [12] and `motives` package for the computational verification of Mozgovoy's conjectural formula. [12] data is limited to $g \leq 11$ because the program reached the memory limit for the machine for that g. The y-axis represents the logarithm with base 10 of the time in seconds spent running the computations.

Another great benefit of the library is its simplicity of use. For example, the code needed to simplify and check the equality of $M^{BB}_{g,r,p}$ and $M^{ADHM}_{g,r,p}$ is as follows.

```
cur = Curve("x", g=g)

# Compute the motive of rank r using ADHM derivation
adhm = TwistedHiggsModuli(x=cur, p=p, r=r, method="ADHM")
eq_adhm = adhm.compute(verbose=verbose)

# Compute the motive of rank r using BB derivation
bb = TwistedHiggsModuli(x=cur, p=p, r=r, method="BB")
eq_bb = bb.compute(verbose=verbose)

# Compare the two polynomials
if eq_adhm - eq_bb == 0:
    print("Polynomials are equal")
```

6. Future work

We plan to continue the development of the library `motives`, expanding its functionalities in different directions and exploring new applications of its symbolic manipulation capabilities to other problems related to motives of moduli spaces.

Some features to be implemented in future releases of `motives` include the following.

- Tools for computing E-polynomials, Poincaré polynomials with compact support, and other

invariants (like the dimension) from a motive.
- Further manipulation tactics capable of partial simplification or manipulation of expression trees in λ-rings, which complement the full simplification algorithm described in this paper.
- Implementations for equations of new types of motives of commonly used geometric constructions, like character or representation varieties, general formulas for moduli spaces of chain bundles, and other additional moduli spaces and moduli stacks of bundles and decorated bundles on curves. For instance, we are considering including equations for the motives of Hilbert schemes deduced in [28], or motives for moduli spaces of stable curves of genus 0 with n marked points derived in [29].
- Implementations for other commonly used λ-rings. For instance, the implementation of a sub-package for handling expressions in the K-theory of a variety could be a useful tool for handling and simplifying expressions in characteristic classes. The inclusion of sub-packages oriented for the manipulation of symmetric polynomials or some purely combinatorial problems is also being considered.

Furthermore, the methodology used to verify computationally Mozgovoy's conjecture for curves of small genus could also be used to verify other existing conjectures on the motives of certain moduli spaces, like the conjectures on the geometry of Hilbert schemes described in [28].

Use of AI tools declaration

The authors declare they have not used Artificial Intelligence (AI) tools in the creation of this article.

Acknowledgments

This research was supported by project CIAMOD (Applications of computational methods and artificial intelligence to the study of moduli spaces, project PP2023_9) funded by Convocatoria de Financiación de Proyectos de Investigación Propios 2023, Universidad Pontificia Comillas, and by grants PID2022-142024NB-I00 and RED2022-134463-T funded by MCIN/AEI/10.13039/501100011033. The authors would like to thank Tomás Gómez for useful discussions.

Conflict of interest

The authors declare there are no conflicts of interest.

References

1. S. del Baño, On the motive of moduli spaces of rank two vector bundles over a curve, *Compos. Math.*, **131** (2002), 1–30. https://doi.org/10.1023/A:1014756205008
2. K. Behrend, A. Dhillon, On the motivic class of the stack of bundles, *Adv. Math.*, **212** (2007), 617–644. https://doi.org/10.1016/j.aim.2006.11.003
3. K. S. Lee, Remarks on motives of moduli spaces of rank 2 vector bundles on curves, preprint, arXiv:1806.11101. https://doi.org/10.48550/arXiv.1806.11101

4. T. L. Gómez, K. S. Lee, Motivic decompositions of moduli spaces of vector bundles on curves, preprint, arXiv:2007.06067. https://doi.org/10.48550/arXiv.2007.06067
5. J. Sánchez, *Motives of Moduli Spaces of Pairs and Applications*, PhD thesis, Universidad Complutense, Madrid, 2014.
6. O. García-Prada, J. Heinloth, A. Schmitt, On the motives of moduli of chains and Higgs bundles, *J. Eur. Math. Soc.*, **16** (2014), 2617–2668. https://doi.org/10.4171/jems/494
7. S. Mozgovoy, Solutions of the motivic ADHM recursion formula, *Int. Math. Res. Not.*, **2012** (2012), 4218–4244. https://doi.org/10.1093/imrn/rnr187
8. D. Alfaya, A. Oliveira, Lie algebroid connections, twisted Higgs bundles and motives of moduli spaces, *J. Geom. Phys.*, **201** (2024), 105195. https://doi.org/10.1016/j.geomphys.2024.105195
9. Á. González-Prieto, Motivic theory of representation varieties via topological quantum field theories, preprint, arXiv:1810.09714. https://doi.org/10.48550/arXiv.1810.09714
10. Á. González-Prieto, Virtual classes of parabolic $SL_2(\mathbb{C})$-character varieties, *Adv. Math.*, **368** (2020), 107148. https://doi.org/10.1016/j.aim.2020.107148
11. C. Florentino, A. Nozad, A. Zamora, Serre polynomials of SL_n- and PGL_n-character varieties of free groups, *J. Geom. Phys.*, **161** (2021), 104008. https://doi.org/10.1016/j.geomphys.2020.104008
12. D. Alfaya, Simplification of λ-ring expressions in the Grothendieck ring of Chow motives, *Appl. Algebra Eng. Commun. Comput.*, **33** (2022), 599–628. https://doi.org/10.1007/s00200-022-00558-3
13. A. Meurer, C. P. Smith, M. Paprocki, O. Čertík, S. B. Kirpichev, M. Rocklin, et al., Sympy: symbolic computing in python, *PeerJ Comput. Sci.*, **3** (2017), e103, https://doi.org/10.7717/peerj-cs.103 .
14. D. Knutson, *λ-Rings and the Representation Theory of the Symmetric Group*, Springer Berlin Heidelberg, Berlin, Heidelberg, 1973, https://doi.org/10.1007/BFb0069217 .
15. D. Grinberg, λ-rings: Definitions and basic properties, 2019. Available from: Https://www.cip.ifi.lmu.de/ grinberg/algebra/lambda.pdf.
16. J. Manin, Correspondences, motifs and monoidal transformations, *Mat. Sb.*, **77** (1968), 475–507, https://doi.org/10.1070/SM1968v006n04ABEH001070
17. S. del Baño, On the Chow motive of some moduli spaces, *J. Reine Angew. Math.*, **532** (2001), 105–132, https://doi.org/10.1515/crll.2001.019
18. M. Larsen, V. A. Lunts, Rationality criteria for motivic zeta functions, *Compos. Math.*, **140** (2004), 1537–1560. https://doi.org/10.1112/S0010437X04000764
19. F. Heinloth, A note on functional equations for zeta functions with values in Chow motives, *Ann. Inst. Fourier*, **57** (2007), 1927–1945. https://doi.org/10.5802/aif.2318
20. M. Larsen, V. A. Lunts, Motivic measures and stable birational geometry, *Moscow Math. J.*, **3** (2003), 85–95. https://doi.org/10.17323/1609-4514-2003-3-1-85-95
21. M. Kapranov, The elliptic curve in the S-duality theory and Eisenstein series for Kac-Moody groups, preprint, arXiv:math/0001005. https://doi.org/10.48550/arXiv.math/0001005
22. J. E. Humphreys, *Reflection Groups and Coxeter Groups*, Cambridge Studies in Advanced Mathematics, Cambridge University Press, 1990, https://doi.org/10.1017/CBO9780511623646

23. W. Y. Chuang, D. E. Diaconescu, G. Pan, Wallcrossing and cohomology of the moduli space of Hitchin pairs, *Commun. Number Theory Phys.*, **5** (2011), 1–56, https://doi.org/10.4310/CNTP.2011.v5.n1.a1

24. D. Bergh, Motivic classes of some classifying stacks, *J. London Math. Soc.*, **93** (2016), 219–243. https://doi.org/10.1112/jlms/jdv059

25. A. Dhillon, M. B. Young, The motive of the classifying stack of the orthogonal group, *Mich. Math. J.*, **65** (2016), 189–197, https://doi.org/10.1307/mmj/1457101817

26. M. Talpo, A. Vistoli, The motivic class of the classifying stack of the special orthogonal group, *Bull. London Math. Soc.*, **49** (2017), 818–823. https://doi.org/10.1112/blms.12072

27. S. Mozgovoy, R. O'Gorman, Counting twisted Higgs bundles, *Math. Res. Lett.*, **29** (2022), 1551–1570. https://doi.org/10.4310/MRL.2022.v29.n5.a11

28. M. Graffeo, S. Monavari, R. Moschetti, A. T. Ricolfi, The motive of the Hilbert scheme of points in all dimensions, preprint, arXiv:2406.14321. https://doi.org/10.48550/arXiv.2406.14321

29. P. Aluffi, M. Marcolli, E. Nascimento, Explicit formulas for the Grothendieck class of $\overline{\mathcal{M}}_{0,n}$, preprint, arXiv:2406.13095. https://doi.org/10.48550/arXiv.2406.13095

AIMS Press