UNIVERSIDAD PONTIFICIA COMILLAS
Higher Technical School of Engineering ICAI

MATHEMATICAL ENGINEERING AND ARTIFICIAL INTELLIGENCE

Final Degree Project

Computation of Grothendieck motives of moduli spaces and the Mozgovoy Conjecture

author
Daniel Sánchez Sánchez - 202108855

director
David Alfaya Sánchez

Madrid, June 2025

I hereby declare, under my own responsibility, that the Project presented with the title

Computation of Grothendieck motives of moduli spaces and the Mozgovoy Conjecture

at the High Technical School of Engineering - ICAI of Universidad Pontificia Comillas in the academic year **2024/25** is my own work, original and unpublished, and has not been previously submitted for any other purpose.

The Project is not a copy of someone else's work, neither totally nor partially, and any information taken from other documents has been properly referenced.

Signed: **Daniel Sánchez Sánchez** Date: 18/06/2025

Daniel

Project submission authorized
PROJECT SUPERVISOR

Signed: **David Alfaya Sánchez** Date: 18/06/2025

David

Project Summary

Abstract

We present `motives`, a new Python package integrated into the SymPy environment designed to simplify and manipulate equations in λ-rings. The package's aim is to facilitate the study of motivic expressions, and to this end it includes many pre-programmed motives, as well as implementing and optimizing an algorithm that turns any expression into a polynomial of its λ or its Adams operations. The package was used to study Mozgovoy's conjectural formula for the motive of the moduli space of L-twisted Higgs bundles by comparing it to a proven expression that represents the same motive.

Keywords: *Mozgovoy conjecture, λ-rings, motives, symbolic computation, Grothendieck ring, generating functions.*

In [Moz12], Mozgovoy proposed a conjectural formula that represents the motive of the moduli space of L-twisted Higgs bundles. On the other hand, in [AO24], a separate equation was proven that represents the exact same motive. However, despite modeling the same object, the two formulas differ fundamentally. Mozgovoy's conjecture is a generating function, from which one has to extract a specific coefficient and find its limit, while the proven equation is a finite algebraic sum of products.

In this bachelor thesis, we present a Python package called `motives` which implements, among other things, a simplification algorithm introduced in [Alf22]. This algorithm is generalized and optimized, so it is valid for any λ-ring. The package integrates with `SymPy` to leverage some of its capabilities, such as its implemented objects and properties. Additionally, `motives` comes with some built-in structures, such as the Grothendieck ring of Chow motives. This makes it a powerful tool for research in algebraic geometry and representation theory.

In order to optimize the simplification algorithm, we derive explicit formulas that relate λ, σ and ψ, and which speed up the algorithm by more than ten-fold.

There are two main goals for this project:

- **To provide a generalized, open-source implementation of the λ-ring simplification algorithm.** The implementation uses Python and `SymPy`, making it accessible, extendable, and integrable with existing mathematical software. It can be installed directly from PyPI by running `pip install motives` in a terminal. It was built with its simplicity in mind, so that it would be easy to use.
- **To verify the Mozgovoy conjecture for higher parameters.** Specifically, the `motives` library was used to simplify and prove the equality of Mozgovoy's generating function and the proven algebraic formula from [AO24]. Furthermore, specific optimizations were made for this particular case, so that the computation of both motives becomes feasible.

The library is structured around a class system that abstracts λ-rings, morphisms, and motives. It allows defining custom λ-ring structures, building free and polynomial extensions, and performing symbolic computations over the Grothendieck ring of Chow motives. Built-in motives include those of algebraic curves, vector bundles and moduli stacks. Specifically, the current release of the library 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 project combines theoretical and computational approaches:

- **Theoretical background:** The design of the simplification algorithm is grounded in the theory of λ-rings, which generalize operations like symmetric powers. Additionally, many implemented equations, such as the equations used to turn a λ, σ or ψ operation into any other, come from exploiting certain properties from λ-rings and leveraging generating functions.
- **Software development:** The implementation was done in Python, and it extends `SymPy` by inheriting from its base classes. The design follows object-oriented principles, with abstract base classes for all nodes in the expression tree.
- **Performance optimization:** Several improvements were made to the original MATLAB algorithm. These include explicit formulas to compute the polynomial relations amongst opposite λ structures and Adams operations, better specific canceling techniques, and capitalizing on `SymPy`'s sparse polynomials to save memory and computation time.
- **Validation:** The library was validated against known motivic formulas and identities. It was also applied to Mozgovoy's conjecture and several groups and curves identities.

The research developed in this bachelor thesis has lead to a paper in Electronic Research Archive (namely, [SAP25]), a scientific poster presented at *Reunión Temática de Geometría y Física* annual meeting (namely, [San24]) and a Python package, `motives`, uploaded to PyPI.

One of the biggest optimizations to the library came from implementing the following equations, derived in this bachelor thesis, into the simplification algorithm:

Proposition 0.1 (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 to σ. Then*

$$\sigma^n(x) = \sum_{i=0}^{n} \sum_{a=(a_1,\dots,a_i)\in\mathcal{P}_i(n)} \left(\frac{(-1)^{n+i}}{n_1(a)!n_2(a)!\dots n_n(a)!} \prod_j \frac{\psi^{a_j}(x)}{a_j} \right). \tag{0.1}$$

Proposition 0.2 (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 to σ. Then*

$$\lambda^n(x) = \sum_{i=0}^{n} \sum_{a=(a_1,\dots,a_i)\in\mathcal{P}_i(n)} \left(\frac{1}{n_1(a)!n_2(a)!\dots n_n(a)!} \prod_j \frac{\psi^{a_j}(x)}{a_j} \right). \tag{0.2}$$

Proposition 0.3 (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 to σ. Then*

$$\psi^n(x) = \sum_{i=1}^{n}(-1)^{i+n+1} \sum_{l=0}^{n-1} \sum_{a=(a_1,\dots,a_i)\in p_i(l)} (n-l)\frac{i!}{n_1(a)!n_2(a)!\dots n_l(a)!}\sigma^{n-l}(x)\prod_j \sigma^{a_j}(x). \tag{0.3}$$

Proposition 0.4 (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 to σ. Then*

$$\psi^n(x) = \sum_{i=1}^{n}(-1)^{i} \sum_{l=0}^{n-1} \sum_{a=(a_1,\dots,a_i)\in p_i(l)} (n-l)\frac{i!}{n_1(a)!n_2(a)!\dots n_l(a)!}\lambda^{n-l}(x)\prod_j \lambda^{a_j}(x). \tag{0.4}$$

The improvement in efficiency can be seen in Figure 1.

Using `motives`, we prove the following theorem:

Theorem 0.5 (Theorem 5.3). *Mozgovoy's conjectural formula for the motive of the moduli spaces of L-twisted Higgs bundles holds 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 understand the improvement in efficiency, in Figure 4, one can see the comparison between the ad-hoc MATLAB algorithm and the `motives` implementation applied to Mozgovoy's conjecture.

The library `motives` is very easy to use. In fact, to perform the comparison specified above, we just run the following code:

```
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")
```

This project presents a concrete contribution to the symbolic study of Grothendieck motives and λ-rings. Through the design and implementation of the `motives` library, the simplification of motivic expressions, which is central to the study of moduli spaces, has been significantly advanced. It fills a gaping hole in the symbolic computation ecosystem, and facilitates the manipulation of any λ-ring.

The library enables researchers to define new λ-rings, simplify any expression with a λ-ring structure, and manipulate motivic formulas symbolically. The verification of Mozogovoy's conjectural formula strengthens string theory, as it is derived as a solution of the ADHM equation, based on [CDP11].

Optimizing the simplification algorithm further, by performing partial simplifications by heuristics, could be explored by using artificial intelligence. This is approach is left as future work.

Ultimately, `motives` stands as a versatile and general tool for researchers working on motives, in order to check different steps of their proofs, to verify conjectures, and many other uses.

Resumen del Proyecto

Resumen

Presentamos `motives`, un nuevo paquete de Python integrado en el entorno `SymPy`, diseñado para simplificar y manipular ecuaciones en λ-anillos. El objetivo del paquete es facilitar el estudio de expresiones motívicas, y para ello incluye muchos motivos preprogramados, así como la implementación y optimización de un algoritmo que convierte cualquier expresión en un polinomio en sus operaciones λ o Adams. El paquete ha sido utilizado para estudiar la fórmula conjetural de Mozgovoy para el motivo del espacio de moduli de L-twisted Higgs bundles, comparándola con una expresión probada que representa el mismo motivo.

Palabras clave: *Conjetura de Mozgovoy, λ-anillos, motivos, cálculo simbólico, anillo de Grothendieck, funciones generatrices.*

En [Moz12], Mozgovoy propuso una fórmula conjetural que representa el motivo del espacio de moduli de L-twisted Higgs bundles. Por otro lado, en [AO24], se demostró una ecuación distinta que representa exactamente el mismo motivo. Sin embargo, a pesar de representar el mismo objeto, ambas fórmulas difieren fundamentalmente. La conjetura de Mozgovoy es una función generatriz, de la cual se debe extraer un coeficiente específico y luego calcular su límite, mientras que la fórmula demostrada es una suma algebraica finita de productos.

En esta tesis fin de grado, presentamos un paquete de Python llamado `motives`, que implementa, entre otras cosas, un algoritmo de simplificación introducido en [Alf22]. Este algoritmo ha sido generalizado y optimizado, por lo que es válido para cualquier λ-anillo. El paquete se integra con `SymPy` para aprovechar sus capacidades, como métodos y propiedades ya implementadas. Además, `motives` incluye estructuras predefinidas, como el anillo de Grothendieck de motivos de Chow, lo que lo convierte en una herramienta poderosa para la investigación en geometría algebraica y teoría de representaciones.

Con el fin de optimizar el algoritmo de simplificación, derivamos fórmulas explícitas que relacionan las operaciones λ, σ y ψ, usando las cuales se consigue un aumento de eficiencia gracias al cual el nuevo algoritmo es más de 10 veces más rápido.

Este proyecto tiene dos objetivos principales:

- **Proporcionar una implementación generalizada y open-source del algoritmo de simplificación para λ-anillos.** La implementación está hecha en Python y usa `SymPy`, lo que la hace accesible para cualquier matemático. Puede instalarse directamente desde PyPI ejecutando `pip install motives` en una terminal. Se ha construido intentando que sea lo más simple posible, para facilitar su uso.
- **Verificar la conjetura de Mozgovoy para parámetros más altos.** En particular, se ha utilizado la librería `motives` para simplificar y demostrar la igualdad entre la función generatriz conjetural de Mozgovoy y la fórmula algebraica demostrada de [AO24]. Además, se han realizado optimizaciones específicas para este caso particular, lo que hace viable el cálculo de ambos motivos.

La biblioteca está estructurada en torno a un sistema de clases que abstrae λ-anillos, morfismos y motivos. Permite definir estructuras personalizadas de λ-anillos, construir variables generales y polinomiales, y realizar cálculos simbólicos sobre el anillo de Grothendieck de motivos de Chow. Los motivos integrados incluyen los de curvas algebraicas, grupos y stacks, entre otros. La versión actual de la biblioteca incluye:

- λ-anillos de enteros.
- λ-anillos generales y extensiones generales de λ-anillos.
- Extensiones polinomiales de λ-anillos.
- Anillo de Grothendieck de motivos de Chow, incluyendo los siguientes motivos preprogramados:
  - Curvas algebraicas complejas.
  - Variedades jacobianas de curvas.

- Productos simétricos y alternados de cualquier variedad dado su motivo.
- Espacios de moduli de vector bundles sobre curvas.
- Espacios de moduli de L-twisted Higgs bundles sobre curvas.
- Espacios de moduli de chain bundles y variaciones de estructuras de Hodge sobre curvas de bajo rango.
- Grupos algebraicos.
- Moduli stacks de vector bundles y G-bundles principales sobre curvas.
- Stacks clasificadores BG para varios grupos G.

y se planea expandir la biblioteca con más λ-anillos y elementos geométricos en versiones futuras.

El proyecto combina enfoques teóricos y computacionales:

- **Fundamentos teóricos:** El diseño del algoritmo de simplificación se basa en la teoría de λ-anillos, que generalizan operaciones como las potencias simétricas. Además, muchas ecuaciones implementadas, como aquellas que convierten una operación λ, σ o ψ en otra, se derivan explotando ciertas propiedades de los λ-anillos y el uso de funciones generatrices.
- **Desarrollo de software:** La implementación se ha realizado en Python, y extiende `SymPy` heredando de sus clases base. El diseño sigue principios de programación orientada a objetos, con clases abstractas para todos los nodos del árbol de expresión.
- **Optimización del rendimiento:** Se han hecho varias mejoras respecto al algoritmo original en MATLAB. Estas incluyen fórmulas explícitas para calcular los polinomios de equivalencia entre estructuras λ y Adams, mejores técnicas de cancelación específica, y se aprovechan los polinomios sparse de `SymPy` para ahorrar memoria y tiempo de computación.
- **Validación:** La biblioteca se ha comprobado comparándola con identidades y fórmulas motívicas conocidas, como la conjetura de Mozgovoy.

La investigación desarrollada en esta tesis ha dado lugar a un artículo en *Electronic Research Archive* ([SAP25]), un póster científico presentado en la *Reunión Temática de Geometría y Física* ([San24]) y un paquete de Python, `motives`, subido a PyPI.

Una de las mayores optimizaciones de la biblioteca proviene de implementar las siguientes ecuaciones, derivadas en esta tesis, dentro del algoritmo de simplificación:

Proposition 0.6 (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 to σ. Then*

$$\sigma^n(x) = \sum_{i=0}^{n} \sum_{a=(a_1,\dots,a_i)\in\mathcal{P}_i(n)} \left(\frac{(-1)^{n+i}}{n_1(a)!n_2(a)!\dots n_n(a)!} \prod_j \frac{\psi^{a_j}(x)}{a_j} \right). \tag{0.5}$$

Proposition 0.7 (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 to σ. Then*

$$\lambda^n(x) = \sum_{i=0}^{n} \sum_{a=(a_1,\dots,a_i)\in\mathcal{P}_i(n)} \left(\frac{1}{n_1(a)!n_2(a)!\dots n_n(a)!} \prod_j \frac{\psi^{a_j}(x)}{a_j} \right). \tag{0.6}$$

Proposition 0.8 (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 to σ. Then*

$$\psi^n(x) = \sum_{i=1}^{n} (-1)^{i+n+1} \sum_{l=0}^{n-1} \sum_{a=(a_1,\dots,a_i)\in p_i(l)} (n-l) \frac{i!}{n_1(a)!n_2(a)!\dots n_l(a)!} \sigma^{n-l}(x) \prod_j \sigma^{a_j}(x). \tag{0.7}$$

Proposition 0.9 (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 to σ. Then*

$$\psi^n(x) = \sum_{i=1}^{n}(-1)^i \sum_{l=0}^{n-1} \sum_{a=(a_1,\ldots,a_i)\in p_i(l)} (n-l)\frac{i!}{n_1(a)!n_2(a)!\ldots n_l(a)!}\lambda^{n-l}(x)\prod_j \lambda^{a_j}(x). \tag{0.8}$$

La mejora en eficiencia puede verse en la Figura 1.

Usando `motives`, hemos demostrado el siguiente teorema:

Theorem 0.10 (Teorema 5.3)**.** *La fórmula conjetural de Mozgovoy para el motivo del espacio de moduli de L-twisted Higgs bundles se cumple en el anillo de Grothendieck de motivos de Chow en rango máximo 3 para cualquier curva proyectiva smooth compleja de género g tal que $2 \leq g \leq 18$ y cualquier line bundle L sobre la curva tal que $2g-1 \leq \deg(L) \leq 2g+18$.*

Para entender la mejora en eficiencia, en la Figura 4 se muestra la comparación entre el algoritmo ad-hoc en MATLAB y la implementación en `motives` aplicada a la conjetura de Mozgovoy.

La biblioteca `motives` es muy fácil de usar. De hecho, para realizar la comparación especificada anteriormente, solo se necesita ejecutar el siguiente código:

```python
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")
```

Este proyecto representa una contribución concreta al estudio simbólico de motivos de Grothendieck y λ-anillos. Mediante el diseño e implementación de la biblioteca `motives`, se ha avanzado significativamente en la simplificación de expresiones motívicas, las cuales son fundamentales para el estudio de espacios de moduli. Llena un vacío importante en el ecosistema del cálculo simbólico y permite la manipulación de cualquier λ-anillo.

La biblioteca permite a los investigadores definir nuevos λ-anillos, simplificar cualquier expresión con estructura de λ-anillo, y manipular fórmulas motívicas simbólicamente. La verificación de la fórmula conjetural de Mozgovoy fortalece la teoría de cuerdas, ya que se deriva como solución de la ecuación ADHM, basada en [CDP11].

Una posible mejora futura sería optimizar aún más el algoritmo de simplificación, realizando simplificaciones parciales guiadas por heurísticas mediante el uso de inteligencia artificial.

En definitiva, `motives` se presenta como una herramienta versátil y general para investigadores que trabajen con motivos, permitiéndoles verificar pasos de demostraciones, comprobar conjeturas y muchas otras aplicaciones.

Contents

1. Introduction

In [Moz12], Mozgovoy comes up with a conjectural formula for the moduli space of L-twisted Higgs bundles. On the other hand, in [AO24], certain equations which model the exact same moduli space were proven. However, these two equations—one a conjecture, the other a theorem—are very dissimilar in nature. Mozgovoy's conjectural equation appears as a generating function from which one has to extract a coefficient, in order to find its limit as t approaches 1. The equation proposed in [AO24] is, in contrast, a sum of products. Thus, it is not straightforward to prove that they are the same. In order to do that, an ad-hoc algorithm was devised in [Alf22] to simplify λ-rings. The algorithm was implemented in MATLAB, and it was applied to these equations, due to the fact that the formulas represent Grothendieck motives with a λ-ring structure.

In this work, we introduce the Python library `motives`, which implements said algorithm, generalizes it to any λ-ring, and optimizes it. Additionally, `motives` is built into the `SymPy` environment, which means that almost any operation, function or object available in `SymPy` is also available in `motives`. Although `motives` works for any user-defined λ-ring, and provides tools to define them, it comes with a set of λ-rings already built into it and ready to use. In the current release of the library, those are:

- 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.

Although the package was built with the verification of Mozgovoy's conjecture in mind, it fills a hole in the symbolic manipulation environment and proves to be an invaluable tool for the mathematician working with moduli spaces and motives. By studying Grothendieck motives, one can extract highly useful information about their associated moduli spaces, and the geometry of schemes. Thus, the study of these motives is a very impactful branch of mathematics, and in the past decades, the motives of numerous shapes have been a subject of great interest. In fact, many moduli spaces have been studied with the aim of finding formulas for their motives [dB02, BD07, Sán14, GPHS14, Moz12, Lee18, GP18, GL20, FNZ21, AO24, SAP25]. However, these motives tend to be extremely complex, and handling them leads to many errors. Just checking for equality between two motivic expressions is far from trivial, albeit very useful. Studying λ-rings is a good way to understand motivic expressions, as many have a λ-ring structure. Hence, `motives` has many applications outside the verification of this specific conjecture, and it would be an instrumental tool in the verification of other conjectures in the field. In fact, any mathematician would find it useful for checking their steps when manipulating motives, and to simplify their equations.

In [Alf22], it was proven that Mozgovoy's conjecture holds for curves of genus up to 11, but using `motives` that result has been significantly extended. Specifically, by simplifying computationally both expressions and showing that they are equal for low genus and degree, we proved the following result (see Section 5 and 5.3 for more details), expanding significantly the results obtained in [Alf22]:

Theorem 1.1 (Theorem 5.3)**.** *Mozgovoy's conjectural formula for the motive of the moduli spaces of L-twisted Higgs bundles holds 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 understand the growing complexity of these polynomials and the increase in efficiency of `motives` compared to the ad hoc implementation, the polynomial for genus 18 is five times larger than the maximum polynomial reached by the MATLAB algorithm before collapsing the memory of the cluster in which it was run (which had a RAM of 128GB). Additionally, it now takes an order of magnitude less time to compute said polynomial with this enhanced, generalized library.

This bachelor thesis is structured the following way: we first define λ-rings in Section 2, their associated operations, and we derive important results related to them. In Section 2.3, we define and prove specific equations that relate ψ, λ and σ. Section 2.4 introduce moduli spaces and Grothendieck motives. In Section 3, we explain the structure of the library, its classes and how the implementation with `SymPy` works. The simplification algorithm is described in Section 4, and in Section 5, the verification of Mozgovoy's conjecture is examined in depth.

The research developed in this bachelor thesis has lead to:

- A paper in the special issue *Applications of symbolic computation* of Electronic Research Archive (namely, [SAP25]).
- A scientific poster presented at *Reunión Temática de Geometría y Física* annual meeting, at ICMAT (namely, [San24]).
- A Python package, `motives`, uploaded to PyPI.

2. λ-rings

A λ-ring is an abelian ring for which we define a set of λ operations $\lambda^n : R \to R$, where n is a natural number representing its degree and R the abelian ring. This operation has certain properties, but its defined by the following:

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

More technically, a λ-ring is a pair (R, λ), in which R is a unital abelian ring and λ is a set of operations $\{\lambda^n : R \to R\}$ that satisfy the properties mentioned (see [Gri19] for more details about λ-rings). Now, consider the generating function $\lambda_t(x)$ of the λ-ring operation, which we define as

$$\lambda_t(x) = \sum_{n=0}^{\infty} \lambda^n(x)t^n.$$

This representation is helpful for various reasons, but one of them is the property that the multiplication of polynomials is equal to the convolution of its terms, because from property 3,

$$\lambda_t(x)\lambda_t(y) = \sum_{n=0}^{\infty} \lambda^n(x)t^n \sum_{n=0}^{\infty} \lambda^n(y)t^n = \sum_{n=0}^{\infty}\sum_{i=0}^{n} \lambda^i(x)\lambda^{n-i}(x)t^n = \sum_{n=0}^{\infty} \lambda^n(x+y)t^n = \lambda_t(x+y),$$

so

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

In fact, the opposite direction is also true, i.e. $\lambda_t(x+y) = \lambda_t(x)\lambda_t(y) \implies \lambda^n(x+y) = \sum_{i=0}^{n} \lambda^i(x)\lambda^{n-i}y$, so it is sufficient to prove (2.1) instead of 3.

A natural follow up would be to evaluate $y = -x$ on (2.1), which yields

$$\begin{aligned} \lambda_t(x + (-x)) &= \lambda_t(x)\lambda_t(-x) \\ 1 &= \lambda_t(x)\lambda_t(-x) \\ (\lambda_t(x))^{-1} &= \lambda_t(-x). \end{aligned} \tag{2.2}$$

We will now look into an example of a λ-ring. Let R be the ring of integers with the normal operations. We need an operation with a degree that takes and returns an integer x, and satisfies the three conditions for a λ-structure. Let

$$\lambda^n(x) = \binom{x}{n}, \tag{2.3}$$

considering that $\binom{x}{n} = 0$ if and only if $n > x$. Properties 1 and 2 are apparent. Now, to prove property 3, first notice that

$$(1+t)^x = \sum_{n=0}^{x} \binom{x}{n} t^n = \sum_{n=0}^{\infty} \binom{x}{n} t^n = \lambda_t(x),$$

which means that $(1+t)^x$ is the generating function of the defined λ-operation. Now, instead of proving property 3, we can prove equation (2.1), which is simple enough, as we have that

$$\lambda_t(x+y) = (1+t)^{x+y} = (1+t)^x(1+t)^y = \lambda_t(x)\lambda_t(y).$$

We can extend the binomial coefficient to any real number, by defining it as

$$\binom{x}{n} = \frac{x^{\underline{n}}}{n!} = \frac{x(x-1)(x-2)\dots(x-n+1)}{n!}.$$

Then, for a rational number $\frac{p}{q}$,

$$\lambda^n\left(\frac{p}{q}\right) = \binom{\frac{p}{q}}{n} = \frac{\frac{p}{q}\frac{p-q}{q}\frac{p-2q}{q}\dots\frac{p-nq+q}{q}}{n!} = \frac{\prod_{i=0}^{n-1}(p-iq)}{n!q^n}.$$

2.1. The σ structure. Considering its generating function, λ admits a natural opposite operation, that we will denote as σ, and that is defined as follows:

$$\sigma_t(x) = (\lambda_{-t}(x))^{-1}. \tag{2.4}$$

Applying equation (2.2) to this definition, we can derive that

$$\begin{aligned}\sigma_t(x) &= (\lambda_{-t}(x))^{-1}\\ \sigma_t(x) &= \lambda_{-t}(-x)\\ \sum_{n=0}^{\infty}\sigma^n(x)t^n &= \sum_{n=0}^{\infty}\lambda^n(-x)(-t)^n\\ \sum_{n=0}^{\infty}\sigma^n(x)t^n &= \sum_{n=0}^{\infty}(-1)^n\lambda^n(-x)t^n\\ \sigma^n(x) &= (-1)^n\lambda^n(-x)\end{aligned} \tag{2.5}$$

We are going to prove that σ is a λ-ring structure. For properties 1 and 2, we can see that

$$\sigma^0(x) = (-1)^0\lambda^0(-x) = 1$$

and

$$\sigma^1(x) = (-1)\lambda^1(-x) = (-1)(-x) = x.$$

Again, instead of proving property 3, we will prove equation (2.1). By its definition,

$$\sigma_t(x)\sigma_t(y) = (\lambda_{-t}(x))^{-1}(\lambda_{-t}(y))^{-1} = (\lambda_{-t}(x)\lambda_{-t}(y))^{-1} = (\lambda_{-t}(x+y))^{-1} = \sigma_t(x+y),$$

so σ is indeed a λ ring structure.

The σ operation of the rationals is quite straightforward to compute. We will use equation (2.5). See that

$$\begin{aligned}\sigma^n(x) &= (-1)^n\lambda^n(-x)\\ &= (-1)^n\binom{-x}{n}\\ &= (-1)^n\frac{-x(-x-1)(-x-2)\dots(-x-n+1)}{n}\\ &= \frac{x(x+1)(x+2)\dots(x+n-1)}{n}\\ &= \binom{x+n-1}{n}\\ &= \left(\!\!\binom{n}{x}\!\!\right),\end{aligned}$$

so

$$\sigma^n(x) = \left(\!\!\binom{n}{x}\!\!\right). \tag{2.6}$$

An interesting result is that if we start from (2.4) and apply this formula, we arrive at a very elegant proof of the following:

$$\begin{aligned}
(\lambda_{-t}(x))^{-1} &= \sigma_t(x) \\
(1-t)^{-x} &= \sigma_t(x) \\
\left(\frac{1}{1-t}\right)^x &= \sum_{n=0}^{\infty} \sigma^n(x)t^n \\
\left(\sum_{n=0}^{\infty} t^n\right)^x &= \sum_{n=0}^{\infty} \left(\!\binom{n}{x}\!\right) t^n.
\end{aligned}$$

2.2. The Adams operations. We will now introduce the concept of a special λ-ring. A λ-ring is special if its associated λ-structure satisfies the following properties:

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

where P_n and $P_{n,m}$ are the Grothendieck polynomials. They are defined as follows (for more information, see [Knu73], [Gri19]). If

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

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

$$\begin{aligned}
P_n(s_1(\bar{X}), \ldots, s_n(\bar{X}), s_1(\bar{Y}), \ldots, s_n(\bar{Y})) &= \operatorname*{coeff}_{t^n} \prod_{i,j=1}^{n} (1 + tX_iY_j), \\
P_{n,m}(s_1(\bar{Z}), \ldots, s_{nm}(\bar{Z})) &= \operatorname*{coeff}_{t^n} \sum_{\substack{I \subset \{1, \ldots, mn\} \\ |I| = m}} (1 + t\prod_{i \in I} Z_i),
\end{aligned}$$

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, [Gri19]. Further examples can also be computed using the Python package `motives` proposed in this work.

More important, though, is that when we have a special λ-structure, we can define an Adams operation ψ associated to it, which has very useful properties. From now on we will assume that at least one of λ or σ is special, and without loss of generality we will let σ be special. Then, we can associate to it a set of Adams operations $\{\psi^n : R \longrightarrow R\}$ for each $n \geq 1$ taking

$$\psi^n(x) = N_n(\sigma^1(x), \dots, \sigma^n(x)), \tag{2.7}$$

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

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

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

If $\psi_t(x) = \sum_{n=0}^{\infty} \psi^n(x)t^n$, it can be shown (see [Gri19]) that

$$\psi_t(x) = -t\frac{d}{dt}\log(\sigma_{-t}(x)) \tag{2.8}$$

and so, applying equation (2.4) to the formula,

$$\psi_t(x) = t\frac{d}{dt}\log(\lambda_t(x)). \tag{2.9}$$

Additionally, ψ^n is a λ-ring homomorphism, so

1. $\psi^n(x + y) = \psi^n(x) + \psi^n(y)$
2. $\psi^n(xy) = \psi^n(x)\psi^n(y)$.

Not only that, but it also is multiplicative under composition, i.e.

$$\psi^n(\psi^m(x)) = \psi^{n \cdot m}(x).$$

For exhaustive proofs of all these facts, see [Gri19].

As a final note, we will calculate the Adams operation for the rationals, letting $\sigma^n(x) = \binom{x}{n}$, and so $\lambda^n(x) = \left(\binom{x}{n}\right)$. We will leave proving that σ is special as an exercise for the reader. Starting from equation (2.9), we have that

$$\begin{aligned}
\psi_t(x) &= t\frac{d}{dt}\log(\lambda_t(x)) \\
&= t\frac{\frac{d}{dt}(\lambda_t(x))}{\lambda_t(x)} \\
&= t\frac{\frac{d}{dt}(1-t)^{-x}}{(1-t)^{-x}} \\
&= t\frac{x(1-t)^{-x-1}}{(1-t)^{-x}} \\
&= \frac{tx}{1-t} \\
&= tx\sum_{n=0}^{\infty} t^n \\
&= \sum_{n=1}^{\infty} xt^n.
\end{aligned}$$

Which means that

$$\psi^n(x) = x \tag{2.10}$$

for the λ-ring specified, so ψ^n is the identity. Trivially, it satisfies all properties specified.

2.3. Relations between λ, σ and ψ. It is apparent that the Adams operation is substantially easier to work with than any λ-structure. Thus, the formulas that convert between λ and σ, and ψ, are fundamental to the algorithm described in [Alf22]. However, the Hirzebruch-Newton polynomial N_n is very expensive to calculate. In [Alf22], a recurrent equation to compute N_n was derived, as well as a recurrent equation to compute L_n such that $\sigma^n(x) = L_n(\psi^1(x), \dots, \psi^n(x))$, which were substantially faster. Additionally, a recurrent equation for the polynomials $L_n^{op}(\psi^1(x), \dots, \psi^n(x)) = \lambda^n(x)$ and $N_n^{op}(\lambda^1(x), \dots, \lambda^n(x)) = \psi^n(x)$ was also found. However, the computation of these polynomials turned out to still be a cause of significant slow-down for the algorithm, because they involved the substitution of polynomials inside other polynomials, which made their expansion very computationally expensive. Thus, we present explicit formulas for these equations, plus other helpful propositions.

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

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

We will use the notation $a = (a_1, \dots, a_k) \in \mathcal{P}_k(n)$ 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 $a = (a_1, a_2, \dots, a_k) \in \mathcal{P}_k(n)$, 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 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 to σ. Then*

$$\sigma^n(x) = \sum_{i=0}^{n} \sum_{a=(a_1,\dots,a_i)\in\mathcal{P}_i(n)} \left(\frac{(-1)^{n+i}}{n_1(a)!n_2(a)!\dots n_n(a)!} \prod_j \frac{\psi^{a_j}(x)}{a_j} \right). \tag{2.11}$$

Proof. We solve for $\sigma_{-t}(x)$ in equation (2.8).

$$\sigma_{-t}(x) = e^{-\int \frac{\psi_t(x)}{t} dt}. \tag{2.12}$$

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 equation (2.12), we get

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

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.

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

By equating the coefficients and canceling 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 to σ. Then*

$$\lambda^n(x)=\sum_{i=0}^{n}\sum_{a=(a_1,\ldots,a_i)\in\mathcal{P}_i(n)}\left(\frac{1}{n_1(a)!n_2(a)!\ldots n_n(a)!}\prod_j\frac{\psi^{a_j}(x)}{a_j}\right). \tag{2.13}$$

Proof. The proof is analogous to the previous lemma, using equation (2.9) instead of equation (2.8) 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 to σ. Then*

$$\psi^n(x)=\sum_{i=1}^{n}(-1)^{i+n+1}\sum_{l=0}^{n-1}\sum_{a=(a_1,\ldots,a_i)\in p_i(l)}(n-l)\frac{i!}{n_1(a)!n_2(a)!\ldots n_l(a)!}\sigma^{n-l}(x)\prod_j\sigma^{a_j}(x). \tag{2.14}$$

Proof. We apply the Taylor series expansion of the natural logarithm to the right hand side of equation (2.8):

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

Next, we expand the derivative and adjust indexes.

$$\begin{aligned}\psi_t(x) &= -t\sum_{i=1}^{\infty}\frac{1}{i}i\left(\sigma_{-t}(x)-1\right)^{i-1}\left(\frac{d}{dt}\sigma_{-t}(x)\right)(-1)^{i-1}\\ &= -\sum_{i=1}^{\infty}(-1)^{i-1}\left(\sum_{n=1}^{\infty}\sigma^n(x)(-t)^n\right)^{i-1}\left(t\frac{d}{dt}\sigma_{-t}(x)\right)\\ &= -\sum_{i=0}^{\infty}(-1)^i\left(\sum_{n=1}^{\infty}\sigma^n(x)(-t)^n\right)^i\sum_{n=1}^{\infty}n\sigma^n(x)(-t)^n.\end{aligned}$$

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

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

By applying the convolution, we obtain

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

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

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

□

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 to σ. Then*

$$\psi^n(x) = \sum_{i=1}^{n}(-1)^i\sum_{l=0}^{n-1}\sum_{a=(a_1,\dots,a_i)\in p_i(l)}(n-l)\frac{i!}{n_1(a)!n_2(a)!\dots n_l(a)!}\lambda^{n-l}(x)\prod_j \lambda^{a_j}(x). \tag{2.15}$$

Proof. The proof is analogous to the previous lemma, using equation (2.9) instead of equation (2.8) 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 to 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 (2.2) we have

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

Next, we use the derivative of the logarithm on the right hand side of equation (2.9).

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

Now, we apply equation (2.2) 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 equaling the coefficients, we get

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

Now we use equation (2.5) to obtain

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

□

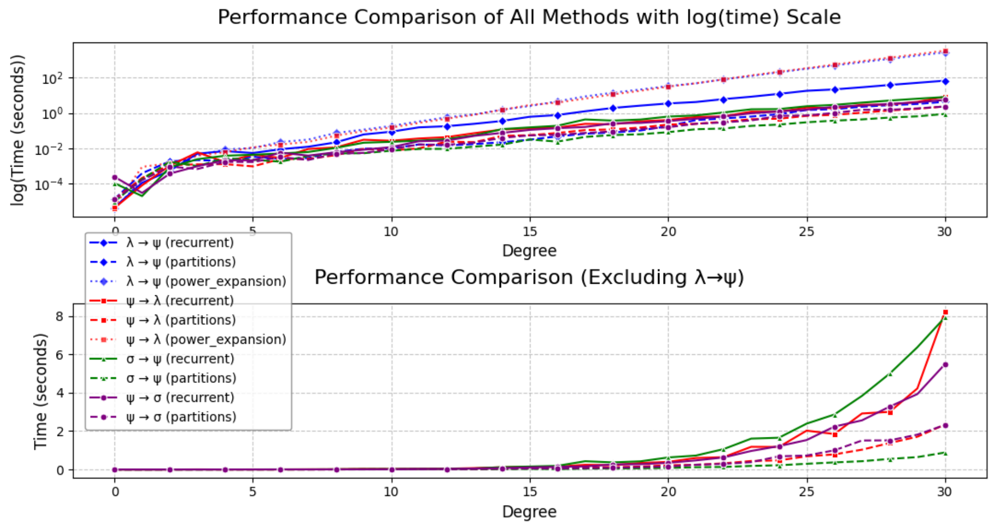


Figure 1. Time comparison between different strategies to compute N_n, represented as $\psi \to \sigma$; L_n, represented as $\sigma \to \psi$; N_n^{op}, represented as $\psi \to \lambda$, and L_n^{op}, represented as $\lambda \to \psi$. The `partitions` strategy employs the explicit equations derived in [SAP25], `recurrent` uses equations (2.17), (2.18), and the equations for N_n and L_n from [Alf22], and `power_expansion` refers to the equations for N_n^{op} and L_n^{op} from [Alf22].

Remark 2.6. *We can further use the equations from* [Alf22, 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* $P_n^{op} \in \mathbb{Z}[X_1, \dots, X_n]$*, which expresses* λ *in terms of* σ *and vice versa (see* [Alf22, Proposition 2] *for details)*

$$\sigma^n = P_n^{op}(\lambda^1(x), \dots, \lambda^n(x)), \qquad \lambda^n = P_n^{op}(\sigma^1(x), \dots, \sigma^n(x)), \quad \forall x \in R \tag{2.16}$$

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

$$\psi^n(x) = \sum_{i=1}^{n} (-1)^{n-i} i \sigma^{n-i}(x) P_i^{op}(\sigma^1(x), \dots, \sigma^i(x)) \tag{2.17}$$

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

$$\psi^n(x) = \sum_{i=1}^{n} (-1)^{n-i} i P_{n-i}^{op}(\lambda_1(x), \dots, \lambda_{n-i}(x)) \lambda^i(x) \tag{2.18}$$

in terms of λ *operations of* x.

Intuitively, the explicit, partition based equations should be the fastest, followed by the recurrent equations, and equations where one has to expand powers of polynomials should be the slowest. The algorithm described in [Alf22] uses recurrent equations for computing N_n and L_n, but in order to compute N_n^{op} and L_n^{op} it uses the formulas

$$\lambda^n(x) = L_n^{op}(\psi^1(x), \dots, \psi^n(x)) = P_n^{op}(L_1(\psi^1(x)), \dots, L_n(\psi^1(x), \dots, \psi^n(x)))$$

and

$$\psi^n(x) = N_n^{op}(\lambda^1(x), \dots, \lambda^n(x)) = N_n(P_1^{op}(\lambda^1(x)), \dots, P_n^{op}(\lambda^1(x), \dots, \lambda^n(x))).$$

These formulas rely on repeated substitution of polynomials into other polynomials, which makes the later expansion very expensive to calculate. In Figure 1, we can see the comparison in time taken to compute these polynomials. It is apparent from the graphs that the explicit equations are substantially faster, while the `power_expansion` computations are orders of magnitude slower (taking up to 2616 seconds for degree 30 compared to less than 3 seconds using the explicit equations for the same degree), which is what we expected.

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.

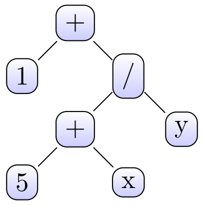
Remark 2.7. *If $x \in R$ is a d-dimensional object for λ, then $\sigma^n(x)$ and $\psi^n(x)$ can be expressed as polynomials in d variables depending on $\lambda^1(x), \ldots, \lambda^d(x)$ through (2.15) and (2.16).*

2.4. Moduli and motives. A moduli space is a geometric space that represents the solution to a classification problem (see [New78] for more information on the theory of moduli spaces). For example, the collection of all lines in $\mathbb{R}^2$ that pass through the origin form a motive known as $\mathrm{P}^1(\mathbb{R})$. These moduli spaces naturally arise as solutions to classification problems, and model the geometry of important spaces in other fields, such as in physics (specially in string theory, see [Moz12]).

The Grothendieck motive is a highly useful tool that facilitates the study of the geometry of different moduli spaces. It is an algebraic invariant of a scheme that encodes most of its geometric information. A complete formal definition is out of the scope of this work, but one can find it in [Man68]. Among other things, if two moduli spaces share the same class in the Grothendieck ring of Chow motives, then their Poincaré polynomials are the same. Incidentally, many Grothendieck motives have a λ-ring structure, which means that having a general tool to manipulate λ-rings would greatly aid in the study of Grothendieck motives, and thus of moduli spaces.

3. Structure of the `motives` package

The `motives` package implements all the needed functionalities to be able to operate with symbolic expressions in λ-rings. It extends the library `SymPy` by inheriting from its core classes and adding to them. Then, all expressions are implicitly represented as a tree. An expression tree is a data class that is based on a tree graph, in which each node represents a part of an expression. Every leaf node is an operand, while every non-leaf node represents an operator. For example, $\frac{5+x}{y} + 1$ would be represented as



Operators can be n-ary (variable amount of children), binary (two children), unary (one children), or any other. For example, division (/) is binary, while addition is n-ary and negation is unary. This data structure is useful because it lets us navigate the tree and do transformations more hierarchically than if we stored the expression as it is read.

`motives` is divided into two clear sub-packages.

3.1. Core. In core, the main logic behind the general functionality of the package is implemented. It models how λ-rings behave and integrates this with `SymPy`. The classes in this section are not meant to be used directly, but instead act as the base of most of the other classes in `motives`. The implemented core modules are

- `LambdaRingExpr`. The `LambdaRingExpr` class is the homologous abstract class to `Sympy`'s *`Expr`*, and it inherits from it. It's the parent class of all other classes that represent nodes in the tree. It defines what methods all other classes should implement, which are `get_max_adams_degree` (returns the degree of the Adams operation with the highest degree of its subtree), `get_max_groth_degree` (returns the maximum Adams degree that the equivalent Adams polynomial would have), `_to_adams` (turns the subtree into its equivalent Adams polynomial) and `_to_adams_lambda` (turns the subtree into its equivalent Adams polynomial using some heuristic optimizations).

- `Operand`. The `Operand` class is also an abstract class, from which all root nodes in the tree inherit. It defines what methods operands should implement, which are `get_adams_var` (returns the operand with an Adams operation applied to it), `get_lambda_var` (returns the operand with a λ operation applied to it), `_apply_adams` and `_subs_adams` (turns every Adams operation of the operand in a given polynomial into its equivalent polynomial of λ operations using `LambdaRingContext`). These methods, plus the ones defined in `LambdaRingExpr`, will be better explained in Section 4.
- **Conventional operators.** Conventional operators, i.e. multiplication, addition, and exponentiation, are also extended from their respective versions in the `SymPy` package. However, a new child class is not created for any of them. Conventional methods implemented in `SymPy`, such as `expand` or `cancel`, directly instantiate these *parent* classes, and they wouldn't have the necessary methods specified by `LambdaRingExpr`. So instead, these methods are defined externally and are attached to the appropriate class, which conveniently causes that any new instance implements them.
- **Ring operators.** These operators are `Sigma` (σ), `Lambda_` (λ) and `Adams` (ψ). They all share a parent class, `RingOperator`. They also inherit from the `SymPy` class `Function`. As a note, `Lambda_` is spelled that way because "`lambda`" is a reserved keyword in the `Python` language.
- **Rational operand.** This is the equivalent to the `Sympy` class `Rational`. It is implemented in the same manner as the conventional operators for the same reasons. Equations (2.6) and (2.10) respectively are used to compute the λ and the Adams operation of this operand. Note that this implicitly encodes the λ-ring of integers.
- `Object1Dim`. This is an abstract class representing a 1-dimensional object in a λ-ring for the special structure σ. That means that if x is a 1-dimensional object, it satisfies that $\sigma^n(x) = 0 \ \forall\ n > 1$. Thus,
$$\sigma_t(x) = 1 + xt,$$
and therefore,
$$\lambda^n(x) = \psi^n(x) = x^n.$$
The proof of this result is very similar to the proof of equation (2.10).
- `LambdaRingContext`. `LambdaRingContext` is a helper class that contains methods for computing the universal polynomials relating the λ, σ and Adams operations. One can change how it computes these polynomials by modifying the parameter `mode` (*partitions* to use equations (2.11), (2.13), (2.14) and (2.15); *recurrent* to use equations (2.17), (2.18) and equations from [Alf22, §2], and *old* to exclusively use the implemented equations in the MATLAB ad-hoc version of the simplification algorithm from [Alf22], described more in detail in Section 4). It is a singleton, and it keeps a cache of the already computed polynomials.

3.2. Grothendieck motives.

While core lays the groundwork for the package, this section implements most of the concrete operands that are used to build expression trees. They inherit from the abstract class `Motive`, which represents a motive in the Grothendieck λ-ring of varieties, the Grothendieck ring of Chow motives or in any extension or completion of such rings. The more general classes defined in this sub-package are

- `Point`. Represents the motive of a closed point. It inherits from the `SymPy` class `AtomicExpr` and is a singleton. It satisfies that $\lambda^n(\mathrm{pt}) = \psi^n(\mathrm{pt}) = 1 \ \forall\ n$.
- `Lefschetz`. Represents the Lefschetz motive ($\mathbb{L}$), which models the complex line $\mathbb{A}^1$. It is a 1-dimensional object, so it inherits from `Polynomial1Var`. It is also a singleton.
- `Proj`. Represents the motive of a complex projective space P^n. It equals the sum $1 + \mathbb{L} + ...\mathbb{L}^n$, and inherits from `AtomicExpr`.

The rest of the classes of the sub-package are divided into 3 groups: curves, groups and moduli.

3.2.1. *Curves.*

This submodule includes classes for handling and operating with abstract smooth complex algebraic curves, which we denote X, of genus 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 $h^1(X)$ is $2g$-dimensional for λ by [Kap00]. Concretely, in the following formulas we will use the notation

$$Z_X(t) = \lambda_t([X]) = \sum_{n=0}^{\infty} \lambda^n([X])t^n$$

and

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

Also notice that

$$Z_X(t) = \lambda_t([X]) = \lambda_t(1 + h^1(X) + \mathbb{L}) = \lambda_t(1)\lambda_t(h^1(X))\lambda_t(\mathbb{L}) = \frac{P_X(t)}{(1-t)(1-\mathbb{L}t)}.$$

The following classes are ascribed to curves.

- `Curve`. Represents the motive of an abstract smooth complex algebraic curve X of genus g. It equals the sum of the motive of a point, the Lefschetz motive, and the `CurveChow` component of the curve, as defined before. It inherits from `AtomicExpr`. To compute the λ operation of the curve, we apply the convolution to its components, so

$$\lambda^n(X) = \sum_{i+j+k=n} \mathbb{L}^j \lambda^k(h^1(X)),$$

  and its Adams operation is just the sum of the Adams operation of its components.
- `CurveChow`. Represents the $h^1(X)$ component of the curve. It inherits from `AtomicExpr`. Now, notice that from (2.9) we get

$$\psi_t(h^1(X)) = t\frac{d}{dt}\log(\lambda_t(h^1(X))) = t\frac{\frac{d}{dt}\lambda_t(h^1(X))}{\lambda_t(h^1(X))} = \left(\sum_{n=0}^{\infty} \lambda^n(h^1(X)) \cdot n \cdot t^n\right) \lambda_t(h^1(X))^{-1}.$$

  Let q_n be the coefficient of t^n of $\lambda_t(h^1(X))^{-1}$. Then,

$$\psi_t(h^1(X)) = \left(\sum_{n=0}^{\infty} \lambda^n(h^1(X)) \cdot n \cdot t^n\right) \sum_{n=0}^{\infty} q_n t^n,$$

  and thus

$$\psi^n(h^1(X)) = \sum_{i=1}^{n} \lambda^i(h^1(X)) \cdot i \cdot q^{n-i}.$$

  By definition,

$$\lambda_t(h^1(X))\lambda_t(h^1(X))^{-1} = 1,$$

  so we have that

$$\sum_{i=0}^{n} q^i \lambda^{n-i}(h^1(X)) = 0 \quad \forall\, n \geq 1.$$

  Thus, we can compute q^n using the recursion

$$q^n = -\sum_{i=0}^{n-1} q^i \lambda^{n-i}(h^1(X)).$$

  We use this to generate the λ operation of the `CurveChow` structure.
- `Jacobian`. Represents the Jacobian of a curve. Computed in terms of the motivic zeta function of $h^1(X)$ (see [Kap00] and [Hei07])

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

  It inherits from `AtomicExpr`.
- `Piccard`: Represents the Piccard variety of a curve X. The motive coincides with that of its Jacobian.

3.2.2. Groups. This submodule includes several complex algebraic groups, based on the formulas from [BD07]. For a connected semisimple complex group G, the motive is computed as a polynomial in $\mathbb{L}$ through the following equation from [BD07, Proposition 2.1]

$$[G] = \mathbb{L}^{\dim G} \prod_{i=1}^{r} (1 - \mathbb{L}^{-d_i}), \tag{3.1}$$

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 [Hum90, Table 1, pp. 59]. Several special cases of the previous formula have been implemented, but all inherit from the class `SemisimpleG`, which in turn inherits from `AtomicExpr`. `SemisimpleG` represents a general connected semisimple complex group G, and it is computed through equation (3.1). The special cases of `SemisimpleG` that have been implemented are

- `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`: $\mathrm{Spin}_n(\mathbb{C})$.

3.2.3. Moduli schemes and stacks. This sub-package contains classes that capture the moduli spaces of decorated bundles on curves. It is divided into a submodule for moduli schemes and a submodule for moduli stacks. The included schemes are

- `BundleModuli`. Abstract class describing a moduli space of decorated bundles on a smooth complex projective curve. All other schemes inherit from this class.
- `VectorBundleModuli`. Represents the motive of the moduli space of vector bundles over a curve of genus ≥ 2. It is computed using the following equations from [GPHS14], [Sán14] and [dB01, Theorem 4.11], assuming that r and d are coprime.

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

  where $\{x\}$ denotes the decimal part of $x \in \mathbb{R}$, i.e., $\{x\} = x - \lfloor x \rfloor$.
- `VHS`. Describes the Grothendieck motive of the moduli space of L-twisted Variations of Hodge Structure over a curve. It is currently only implemented for the following ranks of the components of the VHS: (1, 1), (1, 2), (2, 1) and (1, 1, 1). They follow the equations found in [GPHS14], [Sán14] and the computations from [AO24]. See [AO24, §8] for details.

- `TwistedHiggsModuli`. Represents the motive of the moduli space of L-twisted Higgs bundles over a curve of genus ≥ 2 in the completion of the Grothendieck ring of Chow motives over C. To compute this class, two different methods were implemented:
  - `TwistedHiggsModuliADHM`. This uses the conjectural equation from [Moz12, Conjecture 3] as a solution to the ADHM equation, based on [CDP11]. It is able to compute the motive of arbitrary rank and degree, so this is the method used for rank greater than 3. However, the method to compute this motive requires some very expensive calculations, which is covered more in depth in Section 5.
  - `TwistedHiggsModuliBB`. This uses the formula derived from the Bialynicki-Birula decomposition of the moduli space, using [AO24, Corollary 8.1] and [GPHS14, Theorem 3]. The formula only exists for rank 2 and 3, but its computations are more efficient than the previous method, so it is used for rank ≤ 3. The computation of the motive of this formula is also covered in more depth in Section 5.

The current stacks are

- `BunG`. Represents the Grothendieck motivic class of the moduli stack of principal G-bundles on a smooth complex projective curve X, computed through the following conjectural formula from [BD07, 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 [BD07, §6].
- `BunDet`. It represents the moduli stack of $\mathrm{SL}_n(\mathbb{C})$-bundles with fixed determinant on a smooth complex projective curve. In essence, it is an instance of `BunG` with a $\mathrm{SL}_n(\mathbb{C})$ as its group.
- `BG`. Classifying space for the group G, $BG = [pt/G]$ for a connected semisimple complex algebraic group G. It is computed as

$$[BG] = 1/[G].$$

  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 $\mathrm{Sp}_{2n}(\mathbb{C})$) [BD07, Example 2.6], for $\mathrm{PSL}_n(\mathbb{C})$ if $n = 2, 3$ [Ber16, Theorem A], for $\mathrm{SO}_n(\mathbb{C})$ [DY16, Theorem 3.7 and Corollary 3.8]) and for O_n [TV17, Theorem 3.1 and Corollary 3.2]. BG has been implemented as a method of `SemisimpleG`.

3.3. Other base classes. There are two basic λ-rings that are outside of both sub-packages. Those are

- `Free`. Represents an abstract variable node in an expression, treated as a generator of a free λ-ring or a free extension of a λ-ring. It is analogous to `SymPy`'s `Symbol`, and it inherits from it. No assumptions are made about it, so all its Adams and λ-powers will be treated as independent algebraic elements in the ring.
- `Polynomial1Var`. It is an 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.

Figure 2 shows a general view of all the classes in the package, and of their hierarchy.

3.4. Integration with `SymPy`. The library is integrated within the SymPy environment. That means that most SymPy functions can be run the same way on a λ-ring expression constructed using `motives`. Functions such as `simplify`, `expand`, `limit` or `apart` work the exact same way they would on an ordinary `SymPy` expression. In fact, not only do these more complex functions work, but also some natural and convenient behaviours arise naturally from the fact that all `motives` operands inherit from either `SymPy`'s `Symbol` or `AtomicExpr`, such as the commutative or the associative properties.

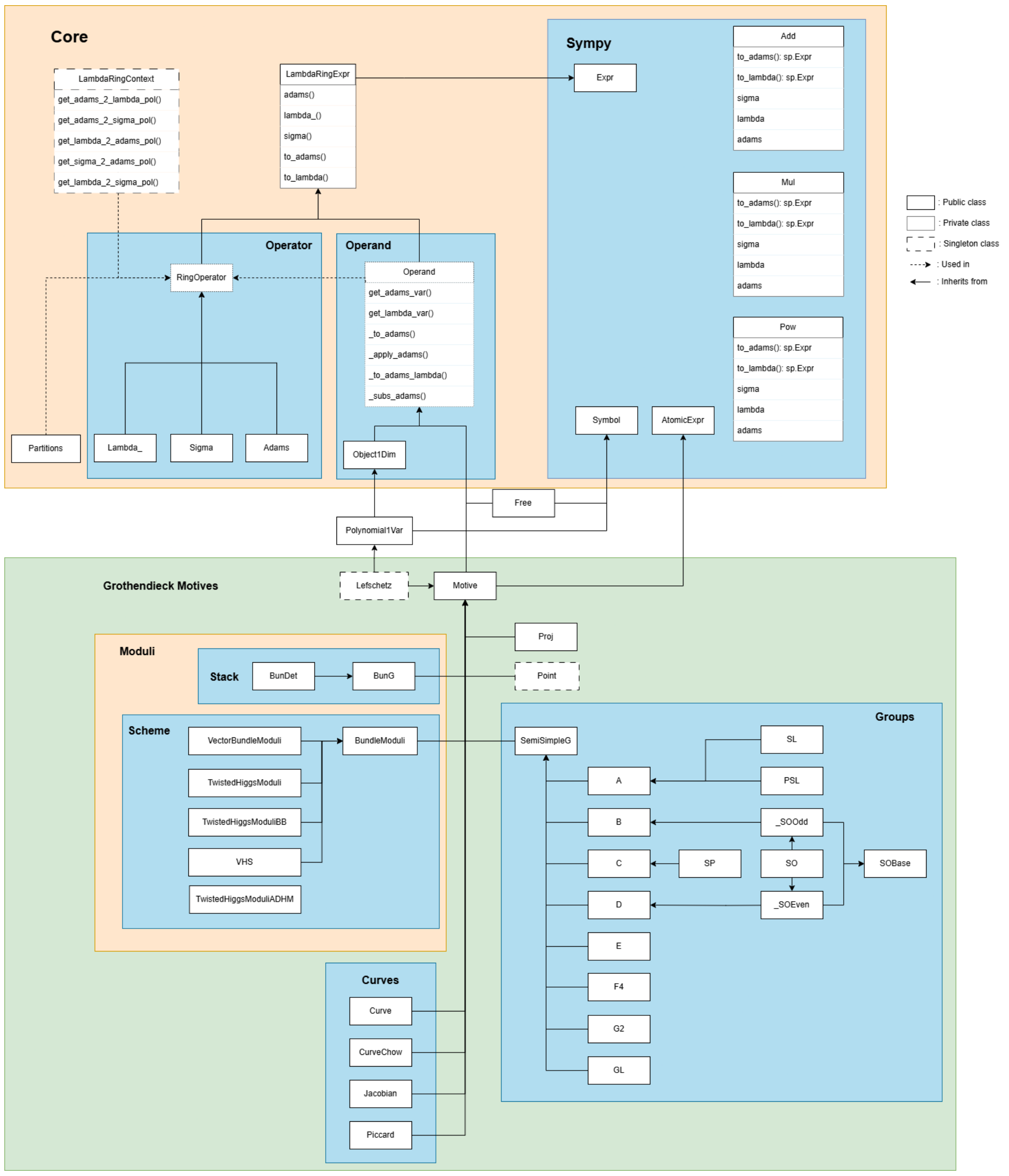


Figure 2. High-level class diagram of the `motives` library and its integration with Sympy. Only the most important public methods are shown; all other methods and internal details have been omitted for clarity. Classes are organized into two colored regions: **Core** (foundation of the lambda-ring framework, 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. The simplification algorithm

The idea of the simplification algorithm implemented in `motives` comes from [Alf22, Algorithm 1 and Theorem 4], but it has been optimized and generalized. We are able to transform any algebraic expression combining λ, σ and ψ into a polynomial of λ or of Adams operations of the base operands.

4.1. `to_adams` algorithm. If an algebraic expression in a λ-ring contains the generators

$$\{a_1, a_2, ..., a_k\},$$

the `to_adams` method is capable of transforming it into a polynomial with generators

$$\{\psi^1(a_1), \ldots, \psi^{n_1}(a_1), \psi^1(a_2), \ldots, \psi^{n_2}(a_2), \ldots, \psi^1(a_k), \ldots, \psi^{n_k}(a_k)\},$$

i.e. a polynomial of Adams operations of its generators. This can be done by leveraging the fact that ψ^k is a ring homomorphism for any k, so we have that

$$\psi^k(P(\psi^{s_1}(a_1), \ldots, \psi^{s_n}(a_n))) = P(\psi^{k\cdot s_1}(a_1), \ldots, \psi^{k\cdot s_n}(a_n)))$$

and this equality holds even if P is a rational polynomial. To be able to compute this, all operands x in `motives` have to implement a method called `_apply_adams`. This method receives a polynomial P of Adams operations of the form

$$P(\psi^1(x), \ldots, \psi^i(x), \psi^{s_1}(a_1), \ldots, \psi^{s_k}(a_k))$$

and a degree n and returns the polynomial P with a partial operation ψ^n applied to the variable, i.e. it returns

$$P(\psi^n(x), \ldots, \psi^{n\cdot i}(x), \psi^{s_1}(a_1), \ldots, \psi^{s_k}(a_k)).$$

The algorithm is agnostic, so each operand is allowed to implement this method in the way it sees convenient, so it can substitute $\psi^i(x)$ for a new algebraically independent symbolic variable, or for an equivalent expression, etc.

Furthermore, notice that we can use equations (2.13) and (2.11) to convert any λ^n or σ^n operation respectively into a polynomial of ψ operations (up to degree n).

Now, we will prove by induction that the algorithm works. The base case is trivial, since an expression of only one element has to be a single operand, which is a polynomial of Adams operations (ψ^1(operand), to be specific) by itself. Now, given a polynomial of Adams operations, we will prove that any operation applied to this polynomial can be transformed into another polynomial of Adams operations. We will cover three distinct cases:

1. **Conventional algebraic operations.** It is trivial to prove that the sum of polynomials of Adams operations is itself a polynomial of Adams operations, and the multiplication of polynomials of Adams operations, by the distributive property, is also a polynomial of Adams operations. Exponentiating a polynomial also yields a polynomial, for the same reason.
2. **ψ operation.** Let $\{a_1, \ldots, a_k\}$ be the operands of P. We have

$$P(\psi^1(a_1), \ldots, \psi^{i_1}(a_1), \psi^1(a_2), \ldots, \psi^{i_2}(a_2), \ldots, \psi^1(a_k), \ldots, \psi^{i_k}(a_k)),$$

and we want to compute

$$\begin{aligned}\psi^n(P(\psi^1(a_1), \ldots, \psi^{i_1}(a_1), \psi^1(a_2), \ldots, \psi^{i_2}(a_2), \ldots, \psi^1(a_k), \ldots, \psi^{i_k}(a_k))) =\\ = P(\psi^n(a_1), \ldots, \psi^{n\cdot i_1}(a_1), \psi^n(a_2), \ldots, \psi^{n\cdot i_2}(a_2), \ldots, \psi^n(a_k), \ldots, \psi^{n\cdot i_k}(a_k)).\end{aligned}$$

Recall that a_1.`_apply_adams(n,P)` returns

$$P(\psi^n(a_1), \ldots, \psi^{n\cdot i_1}(a_1), \psi^1(a_2), \ldots, \psi^{i_2}(a_2), \ldots, \psi^1(a_k), \ldots, \psi^{i_k}(a_k)).$$

Then, running `_apply_adams` iteratively for every operand yields the result we want.
3. **σ and λ operations.** First, we convert the operation into a polynomial of Adams operations using its respective equation. So, assuming we have a λ operation (for a σ operation the process is the same, but using L_n instead of L_n^{op}) we have that

$$\lambda^n(P) = L_n^{op}(\psi^1(P), \psi^2(P), \ldots, \psi^n(P)).$$

Now, we can apply the same process we described before to turn $\psi^i(P)$ into an Adams polynomial, which we will denote by P_i. Thus,

$$\lambda^n(P) = L_n^{op}(P_1, P_2, \ldots, P_n).$$

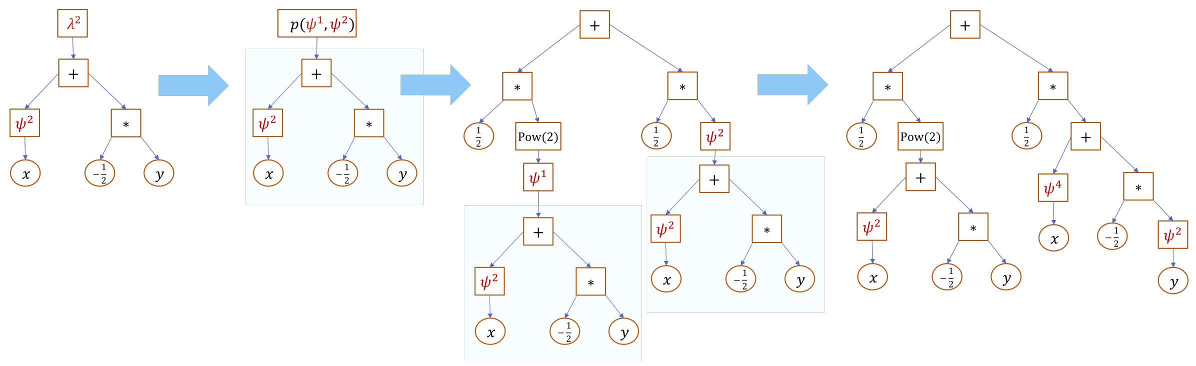


FIGURE 3. Step-by-step illustration of the `to_adams()` algorithm. Starting with the expression $\lambda^2(\psi^2(x) - \frac{y}{2})$, the algorithm iteratively expands the λ^2 and ψ^2 operations following universal λ-ring relations. In this case, $\lambda^2(x)$ is substituted by $\frac{(\psi^1(x))^2}{2} + \frac{\psi^2(x)}{2}$. 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.

Finally, we know that a polynomial of polynomials is itself a polynomial, since all operations in a polynomial are part of the first case, so we are finished.

The algorithm, then, works recursively. Each node calls the algorithm on its child, and then the appropriate transformation, according to which case we are on, is applied. A sketch of the algorithm goes as follows:

```
def to_adams(self):
    child_to_adams = self.child.to_adams()
    return transformation(child_to_adams)
```

Figure 3 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.

4.2. `to_lambda` algorithm. This algorithm turns any algebraic expression in a λ-ring into a polynomial in λ operations of its generators. First, it converts the expression into a polynomial in Adams using `to_adams`, yielding $P(\psi^{k_1}(x_1), \dots, \psi^{k_s}(x_s))$, and then it uses the method `_subs_adams` to turn it into a polynomial in terms of $\lambda^1(x_i), \dots, \lambda^{k_i}(x_i)$ for each operand x_i. Similarly to `apply_adams`, `subs_adams` must be implemented by every operand in the expression tree. It accepts a polynomial P and substitutes any instances of Adams of the operand in the polynomial for the equivalent polynomial in λ. A general, free variable uses (2.15), but other motives may have a more efficient way of performing the substitutions. More specifically, running x_i`._subs_adams(P)` yields

$$P(N_n^{op}(\lambda^1(x_1), \dots, \lambda^{k_1}(x_1)), \dots, \psi^{k_s}(x_s)),$$

so, analogously to `apply_adams`, `subs_adams` is applied repeatedly for every operand in the tree, yielding a polynomial on only λ-powers.

It must be observed that these two methods grows very rapidly in complexity as the degrees of the ring operations grow. Especially `to_lambda`, due to the fact that if we have a σ or a λ operation of degree n of a `Free` operand in the original tree, we have to first convert it to a polynomial in Adams, which has n variables, and then each of those variables has to be converted to a polynomial in λ. A quite trivial optimization can be made. If when computing `to_lambda` we have a part of the expression that already is a polynomial in λ, and it has no other λ-ring structures higher in the tree, we will leave it as it is. This is implemented through a `_to_adams_lambda` method.

As explained in Section 3.1, the polynomials used for converting between λ-ring structures are computed using the class `LambdaRingContext`.

Finally, both `to_adams` and `to_lambda` have a parameter `as_symbol`, which specifies how to return the polynomial. If `as_symbol` is `False`, $\lambda^n(x)$ and $\psi^n(x)$ will be instances of `Lambda_` or `Adams` respectively. Otherwise, they will be `SymPy` symbols. The first is useful when we just want to manipulate an expression, but if we want to turn the expression into an actual instance of a `SymPy` polynomial, the second is needed.

4.3. Usage example. We have the following expression:

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

To simplify it using `motives`, we can run the following code

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

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

which yields the following polynomial exclusively in Adams variables (and in x and y):

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

As we can see in the code, x and y are set as `Free` variables. Then, the expression is constructed, and `to_adams()` is called on it. This is the same procedure that is followed for any motivic expression. A graphical depiction of this process can be seen in Figure 3. If instead of `to_adams()`, `to_lambda()` were to be called, it would yield the following polynomial:

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

Notice how in this case, the polynomial exclusively depends on λ-powers of x and y, as expected. 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.

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

5. Verification of Mozgovoy's conjecture

The library has several uses for the working mathematician. One of them is simplifying motivic expressions, and through it enabling the verification of conjectures. To test the library, we computed the motive of twisted Higgs bundles for specific parameters.

In [Moz12], the following conjectural formula for the motive of the moduli space of L-twisted Higgs bundles was derived as a solution to the ADHM equation, based on [CDP11]. We denote the motive as $\mathcal{M}_L(X,r,d)$, with X being a smooth complex algebraic curve of genus g, r being the rank of the Higgs bundle and d its degree. Note that for the following section, r and d are assumed to be coprime.

Conjecture 5.1. [Moz12, Conjecture 3] *Let $\mathcal{P}(n)$ denote the set of ordered partitions of n, where 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*

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

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

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

Then, for each integer $n\geq 1$, let

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

From $\mathcal{H}_n(t)$, define $H_r(t)$ for each $r \geq 1$ as the coefficient of T^r of

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

So

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

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

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

Additionally, in [AO24], we have the following proven formula for computing the motive for rank up to 3.

Theorem 5.2 ([GPHS14, Theorem 3], [AO24, Corollary 8.1])**.** *The following equalities hold in $\hat{K}_0(\mathcal{V}ar_{\mathbb{C}})$.*

1. *For $r = 1$*
$$[\mathcal{M}_L(X,1,d)] = M^{\mathrm{BB}}_{g,1,p} := \mathbb{L}^{g-1+p}P_X(1) \tag{5.2}$$
2. *For $r = 2$, if $(2,d) = 1$,*
$$\begin{aligned}[\mathcal{M}_L(X,2,d)] = M^{\mathrm{BB}}_{g,2,p} := {} & \frac{\mathbb{L}^{4g-4+4p}\Big(P_X(1)P_X(\mathbb{L}) - \mathbb{L}^g P_X(1)^2\Big)}{(1-\mathbb{L})(1-\mathbb{L}^2)} \\ & + \mathbb{L}^{4g-4+3p}P_X(1)\sum_{i=1}^{\lfloor\frac{2g-1+p}{2}\rfloor}\lambda^{2g-1+p-2i}([X]).\end{aligned} \tag{5.3}$$
3. *For $r = 3$, if $(3,d) = 1$,*

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

Proving that the equations from Theorem 5.2 and from Conjecture 5.1 represent the same motive is far from trivial. The equations grow exponentially in time of compute and in number of terms, and prior to this work, it had only been verified up to genus 11 and $p \leq 20$, for both rank 2 and 3. Specifically, in [Alf22], an ad hoc MATLAB algorithm following the algorithm described in Sections 4.1 and 4.2 managed to verify the conjecture with the following conditions:

- 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 in an Intel(R) Xeon(R) E5-2680v4@2.40GHz with 128GB of RAM. It was observed that the main limitation was the RAM, due to the fact that during the cancellation process of equation (5.1), the expression displays an over-exponential growth. In fact, the test for $r = 3$, $g = 11$ and $\deg(L) = 2g+18$ used up all the RAM of the cluster, and took over 3 days to complete.

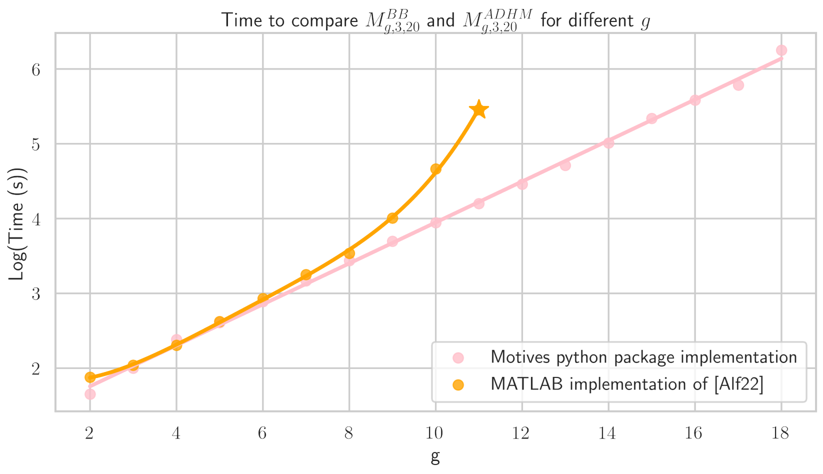


FIGURE 4. Time comparison between algorithm [Alf22] and `motives` package for the computational verification of Mozgovoy's conjectural formula. [Alf22] data is limited to $g \leq 11$ because the program reached the memory limit for the machine for that g. The y-axis represents the logarithm with base 10 of the time in seconds spent running the computations.

Equations (5.3) and (5.4) are comparably faster to simplify, after a few transformations to the equation for rank 3. Namely, we apply the convolution to the lambda operations, and thanks to the optimization mentioned in Section 4.2, applying `to_lambda()` becomes trivial. Specifically, we use the fact that

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

and

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

This last step works because, by (2.9),

$$\begin{aligned} t\frac{d}{dt}\log(\lambda_t([X]\mathbb{L})) &= \psi_t([X]\mathbb{L}) \\ &= \sum_{n=0}^{\infty} \psi^n([X]\mathbb{L})t^n \\ &= \sum_{n=0}^{\infty} \psi^n([X])\psi^n(\mathbb{L})t^n \\ &= \sum_{n=0}^{\infty} \psi^n([X])\mathbb{L}^n t^n \\ &= \psi_{\mathbb{L}t}([X]) \\ &= t\frac{d}{dt}\log(\lambda_{\mathbb{L}t}([X])). \end{aligned}$$

So

$$\begin{aligned} t\frac{d}{dt}\log(\lambda_t([X]\mathbb{L})) &= t\frac{d}{dt}\log(\lambda_{\mathbb{L}t}([X])) \\ \lambda_t([X]\mathbb{L}) &= \lambda_{\mathbb{L}t}([X]) \\ \sum_{n=0}^{\infty}\lambda^n([X]\mathbb{L})t^n &= \sum_{n=0}^{\infty}\lambda^n([X])(\mathbb{L}t)^n \\ \sum_{n=0}^{\infty}\lambda^n([X]\mathbb{L})t^n &= \sum_{n=0}^{\infty}\lambda^n([X])\mathbb{L}^n t^n \\ \lambda^n([X]\mathbb{L}) &= \lambda^n([X])\mathbb{L}^n. \end{aligned} \tag{5.5}$$

This works with any motive, not just $[X]$.

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 [Alf22]. From $g = 11$ the MATLAB program reached the memory limit of the machine.

g	`Motives` Python package implementation (s)	MATLAB implementation of [Alf22] (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	-

Equation (5.1) is the bottleneck of the process. It is not straightforward to evaluate $H_r(t)$ at $t = 1$. When evaluating $\mathcal{H}_n(t)$, $Z_X(t)$ terms appear in the expression, and they introduce non trivial singularities on $t = 1$. Hence, canceling is necessary before substituting, but `SymPy`'s `cancel()` was too slow to be viable, and `limit()` didn't work either. Some clever techniques were employed to simplify this process.

First, in order to collect the coefficient of T^r, expanding the expression is necessary. However, there are many terms that are unnecessary to expand and in fact, overcomplicate the expression. So we only expand the terms that contain T. Next, we only want to cancel the terms that contain the factor $1-t$, so instead of directly canceling the expression, we first express it as $\sum_{i=0}^{n}\frac{c_i}{(1-t)^i}$. Starting with the coefficient of $\frac{1}{(1-t)^n}$, we iteratively divide c_i by $1-t$ exactly k times, i.e. until it has no remaining $1-t$ roots. Then, we add the remainder to c_{i-k}. If the expression can be evaluated at $t = 1$, each term can be divided at least once, which guarantees the algorithm to end. Once both expressions have been simplified to a polynomial in λ-powers of the $h^1(X)$ component of the curve and in $\mathbb{L}$, they are compared by subtracting the polynomials and ensuring the result is 0.

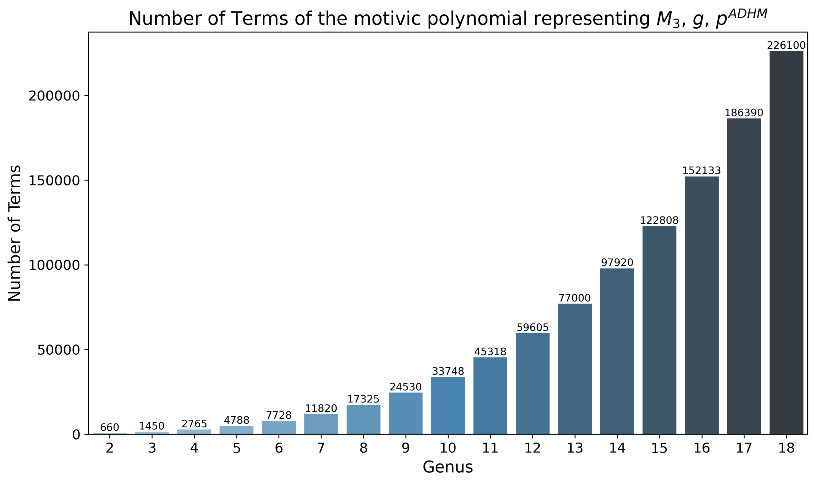


FIGURE 5. Size comparison between the polynomials generated for $M^{\mathrm{BB}}_{g,3,20} = M^{\mathrm{ADHM}}_{g,3,20}$ after simplification for different g, all taken with $r = 3$ and $p = 20$.

Thanks to these optimizations and some other clever tricks, we could verify Conjecture 5.1 up to genus 18 on the same machine the MATLAB algorithm was run on. We showed that both equations are equal when

- 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 $\hat{K}_0(\mathcal{CM}_{\mathbb{C}})$,*

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

To show the improvement between the MATLAB ad-hoc algorithm and the implementation in the `motives` package, Figure 4 shows a comparison between the time taken for both algorithms to complete the verification for different values of the genus. One can see that, with a logarithmic scale, the `motives` version appears to follow a linear growth, while the MATLAB algorithm has a greater complexity. The exact time it took for each computation to complete is displayed in Table 1. It is interesting to note that the time taken for genus 11 on the MATLAB algorithm, with a polynomial of 45318 terms, is greater than the time taken for genus 15 on the Python implementation, with a polynomial of 122808 terms, almost three times as many.

To put into perspective the increase in size of the polynomials, note that their degree is $r^2\deg(L)+1$ and they have $g+1$ variables. So although the final polynomials are significantly smaller, polynomials of degree 361 in 12 variables (which is the size of the biggest polynomial computed using the MATLAB implementation) have up to $\sim 3.4 \cdot 10^{24}$ monomials, while polynomials of degree 487 in 19 variables (which is the size of the biggest polynomial computed using the `motives` implementation) can have up to $\sim 3.1 \cdot 10^{36}$. The number of monomials of the motive for different g is represented in Figure 5.

`motives` simplicity and ease of use should also be remarked. All usage examples of the package have been short and intuitive. In fact, to perform the simplification and comparison of $M^{BB}_{g,r,p}$ and $M^{ADHM}_{g,r,p}$ one just needs to run the following Python code:

```
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 expect to extend the functionality of the library `motives` in future releases, such as by

- Implementing new λ-rings. Examples would be a sub-package for handling expressions in the K-theory of a variety, sub-packages oriented for the manipulation of symmetric polynomials, or λ-rings oriented to the study of purely combinatorial problems.
- Implementing equations for other useful geometric constructions, such as 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.
- Adding tools for the computation of E-polynomials, Poincaré polynomials and other invariants.
- Improved simplification and manipulation strategies for λ-rings, that would not necessarily reduce the motive to its basic expression as a polynomial of λ or Adams operations, but would cleverly leverage λ-ring properties.

Additionally, it is a natural next step to use `motives` to verify other open conjectures in the field in the same way it was applied to Mozgovoy's conjecture, like the conjectures on the geometry of Hilbert schemes described in [GMMR24].

Another interesting idea would be to use an artificial intelligence algorithm to optimize the comparison process (proving whether two equations are equal) used in the library. For example, some kind of reinforcement learning model could be applied to this problem, which would have some specific λ-ring transformations as its actions and a degree of similarity between the two expressions for its reward function.

Finally, we are working on the full proof of Mozgovoy's conjecture for rank 2 and 3, leveraging the insights gained from the library `motives` and using it to ensure correctness of every step of the proof.

References

[Alf22] David Alfaya. Simplification of λ-ring expressions in the Grothendieck ring of Chow motives. *Applicable Algebra in Engineering, Communication and Computing*, 33:599–628, 2022.

[AO24] David Alfaya and André Oliveira. Lie algebroid connections, twisted Higgs bundles and motives of moduli spaces. *Journal of Geometry and Physics*, 201:105195–1 – 105195–55, 2024.

[BD07] Kai Behrend and Ajneet Dhillon. On the motivic class of the stack of bundles. *Advances in Mathematics*, 212(2):617–644, 2007.

[Ber16] Daniel Bergh. Motivic classes of some classifying stacks. *Journal of the London Mathematical Society*, 93(1):219–243, 2016.

[CDP11] Wu-yen Chuang, Duiliu-Emanuel Diaconescu, and Guang Pan. Wallcrossing and cohomology of the moduli space of Hitchin pairs. *Commun. Number Theory Phys.*, 5(1):1–56, 2011.

[dB01] S. del Baño. On the chow motive of some moduli spaces. *Journal für die reine und angewandte Mathematik*, 2001(532):105–132, 2001.

[dB02] S. del Baño. On the motive of moduli spaces of rank two vector bundles over a curve. *Composition Mathematica*, 131:1–30, 2002.

[DY16] Ajneet Dhillon and Matthew B. Young. The motive of the classifying stack of the orthogonal group. *Michigan Math. J.*, 65(1):189–197, 2016.

[FNZ21] Carlos Florentino, Azizeh Nozad, and Alfonso Zamora. Serre polynomials of sln- and pgln-character varieties of free groups. *Journal of Geometry and Physics*, 161:104008, 2021.

[GL20] Tomás L. Gómez and Kyoung-Seog Lee. Motivic decompositions of moduli spaces of vector bundles on curves. *arXiv:2007.06067*, 2020.

[GMMR24] Michele Graffeo, Sergej Monavari, Riccardo Moschetti, and Andrea T. Ricolfi. The motive of the Hilbert scheme of points in all dimensions. *arXiv:2406.14321*, 2024.

[GP18] Ángel González-Prieto. Motivic theory of representation varieties via topological quantum field theories. *arXiv:1810.09714*, 2018.

[GPHS14] Oscar García-Prada, Jochen Heinloth, and Alexander Schmitt. On the motives of moduli of chains and Higgs bundles. *Journal of the European Mathematical Society*, 16:2617–2668, 2014.

[Gri19] Darij Grinberg. λ-rings: Definitions and basic properties, 2019. https://www.cip.ifi.lmu.de/ grinberg/algebra/lambda.pdf.

[Hei07] Franziska Heinloth. A note on functional equations for zeta functions with values in Chow motives. *Ann. Inst. Fourier (Grenoble)*, 57(6):1927–1945, 2007.

[Hum90] James E. Humphreys. *Reflection Groups and Coxeter Groups*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 1990.

[Kap00] M. Kapranov. The elliptic curve in the S-duality theory and Eisenstein series for Kac-Moody groups. *arXiv:math/0001005*, 2000.

[Knu73] Donald Knutson. *λ-Rings and the Representation Theory of the Symmetric Group*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1973.

[Lee18] Kyoung-Seog Lee. Remarks on motives of moduli spaces of rank 2 vector bundles on curves. *arXiv:1806.11101*, 2018.

[Man68] Y. Manin. Correspondences, motifs and monoidal transformations. *Matematicheskii Sbornik*, 77(119):475–507, 1968.

[Moz12] Sergey Mozgovoy. Solutions of the motivic ADHM recursion formula. *Int. Math. Res. Not. IMRN*, 2012(18):4218–4244, 2012.

[New78] P. E. Newstead. *Lectures on Introduction to Moduli Problems and Orbit Spaces*. Springer-Verlag, 1978.

[Sán14] Jonathan Sánchez. *Motives of moduli spaces of pairs and applications*. PhD thesis, Universidad Complutense, Madrid, 2014.

[San24] Daniel Sanchez. Motives meet sympy: studying λ-ring expressions in python. Scientific poster presented at the Red Temática de Geometría y Física (RTGF) annual meeting, November 25, 2024.

[SAP25] Daniel Sanchez, David Alfaya, and Jaime Pizarroso. Motives meet sympy: studying λ-ring expressions in python. *Electronic Research Archive*, 33(4):2118–2147, 2025.

[TV17] Mattia Talpo and Angelo Vistoli. The motivic class of the classifying stack of the special orthogonal group. *Bulletin of the London Mathematical Society*, 49(5):818–823, 2017.

D. Sanchez,
Institute for Research in Technology, ICAI School of Engineering, Comillas Pontifical University, Calle del Rey Francisco 4, 28015 Madrid, Spain
Email address: `dani.sanchez@alu.comillas.edu`