university of groningen

faculty of science and engineering

Kontsevich (micro-)graph invariants in the geometry of Nambu-Poisson brackets: their nature and constraints

Master Project Mathematics

November 2024

Student: F.M. Schipper

First supervisor: dr. A.V. Kiselev

Second supervisor: prof. dr. M. Seri

Contents

Abstract

Kontsevich constructed a map between 'good' graph cocycles γ and infinitesimal deformations of Poisson bivectors on affine manifolds, that is, Poisson cocycles in the second Lichnerowicz–Poisson cohomology. We call the infinitesimal deformation $Q_\gamma(\mathcal{P})$ trivial if there exists a vector field $\vec{X}$ such that $Q_\gamma(\mathcal{P}) = [\![\mathcal{P}, \vec{X}]\!]$. For the class of Nambu-determinant Poisson brackets we establish that the known trivializing vector fields (also created from graphs) $\vec{X}^{\gamma_3}_{2D}$, $\vec{X}^{\gamma_3}_{3D}$, $\vec{X}^{\gamma_3}_{4D}$ and $\vec{X}^{\gamma_5}_{2D}$ are unique up to Hamiltonian vector fields. Moreover, we discuss the non-uniqueness of the choice of graphs to represent multivectors, and ideas that stem from this observation. Finally, we present a conjecture on the general form of the trivializing vector fields of $Q_{\gamma_3}(\mathcal{P})$ for all finite dimensions, where $\mathcal{P}$ is a Nambu-determinant Poisson bracket.

Acknowledgments

First of all the completion of my thesis would have not been possible without the support of Sander, especially with regards to running memory intensive computations on my laptop. I am grateful for my supervisor Arthemy for his thorough feedback and assistance during this project, and Mollie for being a fantastic sparring partner throughout this project. I would also like to extend my gratitude to Ricardo Buring for his support in the practical usage of gcaops, and providing rapid software updates to mitigate some technical issues I encountered. Finally, I would like to thank the Center for Information Technology of the University of Groningen for their support and and for providing access to the Hábrók high performance computing cluster, without which many of these computations would have been impossible.

1. Introduction

This thesis is focused on understanding specific infinitesimal deformations of Poisson structures $\mathcal{P}$ on affine finite-dimensional Poisson manifolds $(M^d_{\text{aff}}, \{\cdot,\cdot\}_{\mathcal{P}})$. These deformations are created from nontrivial wheel-graph cocycles γ in the Kontsevich graph complex **Gra** via a morphism taking as argument some wheel-graph cocycle $\gamma \in \mathbf{Gra}$ and returning a deformation $Q_\gamma(\mathcal{P})$ of the Poisson structure $\mathcal{P}$. Though there are infinitely many of such wheel-graph cocycles, we focus on two of them in this text, γ_3 and γ_5. These are the wheel-graph cocycles on the smallest number of vertices. We simplify the problem further by only considering affine finite-dimensional real space $\mathbb{R}^d_{\text{aff}}$, and by only considering Poisson structures of the Nambu-determinant class whenever the dimension of our space $d \geq 3$.

The thesis is structured as follows. We start with a short overview of the problems that were the primary focus of this research and how they were studied in Section 2. This overview is not meant to be understood completely the first time it is read, as it relies on concepts introduced in later sections. However, we think it will be helpful to (at least superficially) understand the goal of the thesis while reading the mathematical background. We recommend that after the introductory sections 3, 4 and 5, the reader returns to this problem explanation. Section 3 focuses on the necessary theory of Poisson geometry and supermanifolds. Section 4 focuses on the Kontsevich graph complex **Gra** and the language of micro-graphs specific for Nambu-determinant Poisson structures. Finally, Section 5 focuses on the necessary theory of deformation theory, and explains some of the history behind the topic.

Then, we recommend reading the papers in Appendix A, coauthored with Mollie S. Jagoe Brown and our supervisor Arthemy V. Kiselev. These papers detail a large amount of the research over the past year.

There is a final Section, Section 6, containing some more results, observations and conjectures.

There are in total three appendices. Appendix A contains the three papers, Appendix B contains code (and is referred to in the papers of Appendix A and in Section 6) and Appendix C contains a list of relevant graphs related to γ_3 (mostly relevant for the papers in Appendix A).

In summary, the recommended reading order is: Section 2, Section 3, Section 4, Section 5, Section 2, Appendix A (together with Appendix B and Appendix C) and Section 6 (together with Appendix B).

1.1. Practical matters

There are a few practical matters to address. The first is that many of the results follow from (large) computations. The code that was used to obtain all results mentioned in the thesis, or the attached papers, can be found attached in Appendix B, as well as in [1]. Proofs and statements referencing to code will clearly mention in which script it can be found. For the papers attached in Appendix A, the references in the proofs are maybe not so clear. The corresponding code is the first three scripts attached in Appendix B, also entitled by the name of the paper "Kontsevich graphs act on Nambu-Poisson brackets, III. Uniqueness aspects" and the section it corresponds to. If the reader would like to run the code themselves, be warned that some scripts require a lot of computing power[1] and that access to a high performance computation cluster (like Hábrók at the University of Groningen) might be necessary.

The second practical matter has to do with the graphs that are used. In the papers attached in Appendix A, we almost exclusively worked with graph encodings rather than graphs due to volume constraints. For completeness sake, all the graphs referenced in the papers can be found in Appendix C. In contrast, as there are no volume constraints on this thesis, the new graphs used in Section 6 are directly added into the text as they appear. Moreover, the graphs that appear as we work with the cocycle γ_3 are all denoted by Γ, whereas the graphs that appear when working with the cocycle γ_5 are all denoted by Δ to create as little confusion as possible.

When we speak about $2D$, $3D$ and $4D$, we always mean $\mathbb{R}^2_{\text{aff}}$, $\mathbb{R}^3_{\text{aff}}$ and $\mathbb{R}^4_{\text{aff}}$, unless otherwise indicated.

[1] The computations relating to γ_3 in dimensions 2 and 3, and the computations related to γ_5 in dimension 2 can be run on most laptops. The computations relating to γ_3 in dimension 4 could be carried out on a device with 32 GB RAM and 16 CPU cores, though it takes approximately a day. All scripts relating to $\vec{X}^{\gamma_5}_{3D}$ need at (the very) least 64 GB of RAM.

2. Problem explanation

We consider finite-dimensional affine Poisson manifolds $(M^d, \{\cdot,\cdot\}_{\mathcal{P}})$. Next, we take some nontrivial cocycle γ in the Kontsevich graph complex **Gra**, and compute the corresponding deformation term $Q_\gamma(\mathcal{P}) \in \mathfrak{X}^2(M^d, \{\cdot,\cdot\}_{\mathcal{P}})$. The first and foremost question is whether or not the deformation is trivial, that is, does there exist some vector field $\vec{X}^\gamma$ such that

$$Q_\gamma(\mathcal{P}) = [\![\mathcal{P}, \vec{X}^\gamma]\!], \tag{1}$$

where $[\![\cdot,\cdot]\!]$ is the Schouten bracket. As of now, there is no algorithmic way to find a trivializing vector field for any Poisson bracket in any finite dimension, nor is there a reason to believe that such a trivializing vector field must necessarily exist at all.

Let us now explain the problems we looked into for this thesis. First, to reduce the size of the problem, rather than searching over all vector fields, we only consider vector fields that are obtained from graphs. Moreover, rather than considering all Poisson structures on all finite-dimensional manifolds, we focused on Poisson structures[2] on $\mathbb{R}^2_{\text{aff}}$ and Poisson structures of the Nambu-determinant class on $\mathbb{R}^3_{\text{aff}}$ and $\mathbb{R}^4_{\text{aff}}$. Over the past year, together with Mollie S. Jagoe Brown and our supervisor Arthemy V. Kiselev, we looked into

- trivializing $Q_{\gamma_3}(\mathcal{P})$ on $\mathbb{R}^4_{\text{aff}}$ for Nambu-determinant Poisson brackets;
- the uniqueness of the trivializing vector fields $\vec{X}^{\gamma_3}_{2D}$, $\vec{X}^{\gamma_3}_{3D}$, $\vec{X}^{\gamma_3}_{4D}$ and $\vec{X}^{\gamma_5}_{2D}$;
- observing properties of the objects we are working with and finding explanations for these properties;
- trivializing $Q_{\gamma_5}(\mathcal{P})$ on $\mathbb{R}^3_{\text{aff}}$ for Nambu-determinant Poisson brackets.

In Appendix A, there are 3 papers attached, coauthored with Mollie S. Jagoe Brown and Arthemy V. Kiselev.

- The first paper "Kontsevich graphs act on Nambu–Poisson brackets, I. New identities for Jacobian determinants" [2] mainly focuses on observed properties.
- The second paper "Kontsevich graphs act on Nambu–Poisson brackets, II. The tetrahedral flow is a coboundary in 4D" [3] focuses on (finding) the trivializing vector field $\vec{X}^{4D}_{\gamma_3}$.
- The third paper "Kontsevich graphs act on Nambu–Poisson brackets, III. Uniqueness aspects" [4] focuses on the uniqueness of the trivializing vector fields $\vec{X}^{\gamma_3}_{2D}$, $\vec{X}^{\gamma_3}_{3D}$ and $\vec{X}^{\gamma_3}_{4D}$.

[2] On $\mathbb{R}^2_{\text{aff}}$ we do not have different classes of Poisson structures, and every Poisson structure is of the form $\mathcal{P} = \varrho(\boldsymbol{x})\frac{\partial}{\partial x} \wedge \frac{\partial}{\partial y}$.

There is a final section, Section 6, that discusses other results that are not mentioned (in detail) in the the aforementioned papers.

2.1. Problem approach

The problem approach is as follows. We are working with multivectors on affine finite-dimensional real space $\mathbb{R}^d_{\text{aff}}$, endowed with a Poisson structure $\mathcal{P}$. Via an isomorphism of Gerstenhaber algebras, rather than considering multivectors on $\mathbb{R}^d_{\text{aff}}$, we instead consider superfunctions on affine real supermanifolds $\mathbb{R}^{d|d}_{\text{aff}}$ of superdimension (d, d). This isomorphism sends functions on $\mathbb{R}^d_{\text{aff}}$ to superfunctions of degree 0 on $\mathbb{R}^{d|d}_{\text{aff}}$, vector fields on $\mathbb{R}^d_{\text{aff}}$ to superfunctions of degree 1 on $\mathbb{R}^{d|d}_{\text{aff}}$ etc. Now, the problem of finding a vector field $\vec{X}^{\gamma_3}_d$ by solving

$$Q_{\gamma_3}(\mathcal{P}) = [\![\mathcal{P}, \vec{X}^{\gamma_3}_d]\!] \tag{2}$$

is reduced to finding a superfunction of degree 1 solving Equation (2) where both bivectors $Q_{\gamma_3}(\mathcal{P})$ and $\mathcal{P}$ are now written as superfunctions of degree 2. The main reason for doing this is rooted in the fact that on $\mathbb{R}^{d|d}_{\text{aff}}$ the Schouten bracket takes an explicit form, and thus, computing with it is much easier than before.

For these computations of superfunctions, we use the SageMath code library gcaops[5] created by R. Buring. When the problem is solved on the superspace $\mathbb{R}^{d|d}_{\text{aff}}$, we can use the isomorphism of Gerstenhaber algebras again to move back to the space of multivectors over $\mathbb{R}^d_{\text{aff}}$.

Something that was already briefly touched upon in the introduction is that some of the code takes a long time to execute and requires significant computational resources. Both each 'next' cocycle γ_{i+2} contributes to larger computations compared to the cocycle γ_i, where $i = 3, 5, 7, ...$, and each dimension also adds to larger computations. For example, let us consider the number of non-isomorphic Hamiltonian graphs in dimension 2, 3 and 4, corresponding to the cocycles γ_3, γ_5 and γ_7.

	$\mathbb{R}^2_{\text{aff}}$	$\mathbb{R}^3_{\text{aff}}$	$\mathbb{R}^4_{\text{aff}}$
γ_3	1	6	21
γ_5	55	6548	141571
γ_7	6874	??	??

Table 1. How many non-isomorphic Hamiltonian graphs are there?

This small example illustrates that the size of the problem grows very rapidly, to the extend that we cannot compute how many non-isomorphic Hamiltonian graphs there are for γ_7 on $\mathbb{R}^3_{\text{aff}}$ and $\mathbb{R}^4$ on a machine with 32 GB of RAM.

3. Mathematical background

We start this chapter with a general overview of Poisson geometry and the theory of supermanifolds that we will use. When we speak about manifolds, we will always mean real, smooth, finite-dimensional manifolds, denoted by M^d. Here, d is the dimension of the manifold.

3.1. Poisson geometry

Most mathematicians are familiar with the definition of a Poisson manifold as follows.

Definition 1 (Smooth Poisson manifold)**.** A smooth Poisson manifold is a smooth manifold M^d equipped with a bracket $\{\cdot,\cdot\}\colon C^\infty(M^d)\times C^\infty(M^d)\to C^\infty(M^d)$ such that the bracket has the following properties:

(1) Skew-symmetry: $\{f,g\}=-\{g,f\}$,
(2) $\mathbb{R}$-bilinearity: $\{af+bg,h\}=a\{f,h\}+b\{g,h\}$, $\{f,cg+dh\}=c\{f,g\}+d\{f,h\}$,
(3) Jacobi identity: $\{\{f,g\},h\}+\{\{g,h\},f\}+\{\{h,f\},g\}=0$,
(4) Leibniz rule: $\{f,gh\}=\{f,g\}h+g\{f,h\}$.

We call the bracket $\{\cdot,\cdot\}$ the *Poisson bracket*.

Notation 1. We denote a smooth Poisson manifold by $(M^d,\{\cdot,\cdot\})$.

However, we will use a second, equivalent definition for the Poisson bracket $\{\cdot,\cdot\}$, first used by Lichnerowicz in [6]. Before we can introduce this definition of the Poisson bracket, we need to introduce some more theory. Recall that a *k-form* (or, *differential k-form*) $\omega\in\Gamma(\Lambda^k(T^*M^d))$ is a smooth section of the vector space of alternating covariant k-tensors on a manifold M^d. Dually to this, we define a *k-vector* (or, *k-vector field*) $A\in\Gamma(\bigwedge^k(TM^d))$ to be a smooth section of the vector space of alternating *contravariant* k-tensors on the manifold.

Notation 2. We denote the space of k-forms and k-vectors on a manifold M^d by $\Omega^k(M^d)$ and $\mathfrak{X}^k(M^d)$ respectively.

Remark 1. Note that 1-vectors are just vector fields.

Moreover, similarly to how we identify[3] a k-form ω with a $C^\infty(M^d)$-multilinear map of degree k on the space of smooth vector fields $\mathfrak{X}^1(M^d)$, that is,

$$\omega:\mathfrak{X}(M^d)\times\mathfrak{X}(M^d)\times\ldots\times\mathfrak{X}(M^d)\to C^\infty(M^d),$$

[3]More details can be found in most books on differential geometry, for example [7].

we can identify a k-vector A with a $C^\infty(M^d)$-multilinear map of degree k on the space of smooth 1-forms (or, *covectors*) $\Omega(M^d)$,

$$A : \Omega^1(M^d) \times \Omega^1(M^d) ... \times \Omega^1(M^d) \to C^\infty(M^d).$$

Remark 2. For $A \in \mathfrak{X}^k(M^d)$, we say that the grade of A (or, *grading of* A), denoted by $|A|$, is k. While defining this might seem unnecessary, there is good reason for this. When we consider more generally the space of multivectors[4] $\bigoplus_{k=0}^d \mathfrak{X}^k(M^d)$ (and not the space of k-vectors!), we can create elements of the form $\frac{\partial}{\partial x^{i_1}} \wedge \frac{\partial}{\partial x^{i_2}} \wedge \frac{\partial}{\partial x^{i_3}} + \frac{\partial}{\partial x^{j_1}} \wedge \frac{\partial}{\partial x^{j_2}}$. Clearly, this is neither a 2-vector nor a 3-vector. In case we have an element $A \in \bigoplus_{k=0}^d \mathfrak{X}^k(M^d)$ such that $A \in \mathfrak{X}^k(M^d)$ for some k, we say that A is *homogeneous*.[5]

Remark 3. The duality between k-forms and k-vectors is as follows [8, 9]. Let $\omega = \omega_1 \wedge \omega_2 \wedge ... \wedge \omega_k \in \Omega^k(M^d)$ and $A = A_1 \wedge A_2 \wedge ... \wedge A_k \in \mathfrak{X}^k(M^d)$. We consider a map $\Omega^k(M^d) \times \mathfrak{X}^k(M^d) \to \mathbb{R}$ via

$$(\omega_1 \wedge \omega_2 \wedge ... \wedge \omega_k) \otimes (A_1 \wedge A_2 \wedge ... \wedge A_k) \mapsto \sum_{\sigma \in S_k} (-1)^\sigma \prod_{i=1}^k \omega_i(A_{\sigma(i)}).$$

Note that we are actually just computing a determinant.

Example 1. Let us consider the duality pairing of a 1-form and a 1-vector. Take $\mathbb{R}^d$ with global coordinates $x^1, x^2, ..., x^d$, and let $\omega = dx^1 \in \Omega(\mathbb{R}^d)$. Moreover, let us consider some vector field $\vec{X} \in \mathfrak{X}(\mathbb{R}^d)$ given by $\vec{X} = f_1(\boldsymbol{x})\frac{\partial}{\partial x^1} + f_2(\boldsymbol{x})\frac{\partial}{\partial x^2} + ... + f_d(\boldsymbol{x})\frac{\partial}{\partial x^d}$, where $f_i(\boldsymbol{x}) \in C^\infty(\mathbb{R}^d)$. Then,

$$\begin{aligned}\omega \otimes \vec{X} &= dx^1 \otimes \left(\sum_{j=1}^d f_j(\boldsymbol{x}) \frac{\partial}{\partial x^j} \right) = \sum_{j=1}^d f_j(\boldsymbol{x})(dx^1) \otimes \frac{\partial}{\partial x^j} = \sum_{j=1}^d f_j(\boldsymbol{x}) dx^1 \left(\frac{\partial}{\partial x^j} \right) \\ &= \sum_{j=1}^d f_j(\boldsymbol{x}) \delta_{1j} = f_1(\boldsymbol{x}).\end{aligned}$$

Definition 2 (Schouten bracket)**.** The *Schouten bracket* $[\![\cdot, \cdot]\!] : \mathfrak{X}^k(M^d) \times \mathfrak{X}^m(M^d) \to \mathfrak{X}^{k+m-1}$ on a manifold M^d is the unique $\mathbb{R}$-bilinear operation satisfying the following:

- For $f, g \in C^\infty(M^d)$, $[\![f, g]\!] = 0$,
- For $\vec{X} \in \mathfrak{X}(M^d)$, $f \in C^\infty(M^d)$, the bracket is the Lie derivative:

$$[\![\vec{X}, f]\!] = \mathcal{L}_{\vec{X}}(f) = \vec{X}(f),$$

- For $\vec{X}, \vec{Y} \in \mathfrak{X}(M^d)$, $[\![\vec{X}, \vec{Y}]\!]$ is the Lie bracket, that is, $[\![\vec{X}, \vec{Y}]\!] = \vec{X}\vec{Y} - \vec{Y}\vec{X}$,

[4]Note that, just as with k-forms, we cannot have k-vectors on a d-dimensional manifold for $k > d$.

[5]Note that A is allowed to be a linear combination of different k-vectors.

- (Shifted-graded skew-symmetry) For homogeneous elements $A \in \mathfrak{X}^{|A|}(M^d)$ and $B \in \mathfrak{X}^{|B|}(M^d)$,
$$[\![A, B]\!] = -(-1)^{(|A|-1)(|B|-1)}[\![B, A]\!],$$
- (Shifted-graded Leibniz rule) For homogeneous elements $A \in \mathfrak{X}^{|A|}(M^d)$, $B \in \mathfrak{X}^{|B|}(M^d)$ and $C \in \mathfrak{X}^{|C|}(M^d)$,
$$[\![A, BC]\!] = [\![A, B]\!]C + (-1)^{(|A|-1)|B|}B[\![A, C]\!],$$
- (Shifted-graded Jacobi identity) For homogeneous elements $A \in \mathfrak{X}^{|A|}(M^d)$, $B \in \mathfrak{X}^{|B|}(M^d)$ and $C \in \mathfrak{X}^{|C|}(M^d)$,
$$[\![A, [\![B, C]\!]]\!] = [\![[\![A, B]\!], C]\!] + (-1)^{(|A|-1)(|B|-1)}[\![B, [\![A, C]\!]]\!].$$

Remark 4. The Schouten bracket is the natural generalization of the Lie bracket for vector fields to arbitrary multivectors, and coincides with the Lie bracket when restricted to vector fields and functions. Note that the properties of shifted-graded skew-symmetry and shifted-graded Jacobi identity restricted to vector fields, which have grading 1, become the usual properties of skew-symmetry and Jacobi-identity.

Definition 3 (Poisson structure)**.** A *Poisson structure* (also called a *Poisson bracket* or a *Poisson bivector*) $\mathcal{P} \in \mathfrak{X}^2(M^d)$ on a manifold M^d is a bivector such that
$$\tfrac{1}{2}[\![\mathcal{P}, \mathcal{P}]\!] = 0.$$

Remark 5. For some Poisson bivector $\mathcal{P}$ and $f, g \in C^\infty(M^d)$,
$$\{f, g\}_\mathcal{P} = \mathcal{P}(f, g) = -[\![f, [\![\mathcal{P}, g]\!]]\!].$$
Note that $[\![\mathcal{P}, g]\!] \in \mathfrak{X}(M^d)$, that is, it is a vector field. Let us call this vector field $\vec{X}_g$. Using Definition 2, we note that in fact,
$$\{f, g\}_\mathcal{P} = -[\![f, \vec{X}_g]\!] = [\![\vec{X}_g, f]\!] = \vec{X}_g(f).$$

Notation 3. We will use $\{\cdot, \cdot\}_\mathcal{P}$ and $\mathcal{P}$ interchangeably to refer to the Poisson structure. Moreover, if it is clear from context that a manifold M^d is equipped with a Poisson structure, we will at times write M^d rather than $(M^d, \{\cdot, \cdot\}_\mathcal{P})$ for readability of equations.

Definition 4 (Interior product)**.** The *interior product with a* 1*-form* α is defined by $i_\alpha \colon \mathfrak{X}^k(M^d) \to \mathfrak{X}^{k-1}(M^d)$, $A \mapsto i_\alpha A$ via
$$i_\alpha A(\alpha_1, ..., \alpha_{k-1}) = A(\alpha, \alpha_1, ..., \alpha_{k-1}).$$

Definition 5 (Nondegenerate Poisson bracket)**.** We call a Poisson bracket $\mathcal{P}$ on a manifold M^d *nondegenerate* if the map
$$\begin{aligned} \mathcal{P}^\# \colon T^*M^d &\to TM^d, \\ \alpha &\mapsto i_\alpha \mathcal{P}, \end{aligned}$$
is a vector bundle isomorphism.

In other words, $\mathcal{P}$ is nondegenerate if $\operatorname{rank} \mathcal{P}_x = \dim M^d = d$ for all $x \in M^d$ [10].

Lemma 1. The homomorphism $\mathrm{d}_\mathcal{P} = [\![\mathcal{P}, \cdot\,]\!] \colon \mathfrak{X}^k(M^d) \to \mathfrak{X}^{k+1}(M^d)$, where $\mathcal{P}$ is a Poisson bivector, is a differential. It is called the *Lichnerowicz–Poisson differential*.

Proof. We only need to show that $\mathrm{d}_\mathcal{P}^2 = [\![\mathcal{P}, [\![\mathcal{P}, \cdot\,]\!]]\!] = 0$. Using the Jacobi identity, we see that for any homogeneous multivector $A \in \mathfrak{X}^{|A|}(M^d)$,

$$[\![[\![\mathcal{P}, \mathcal{P}]\!], A]\!] = [\![\mathcal{P}, [\![\mathcal{P}, A]\!]]\!] - (-)^{(|\mathcal{P}|-1)(|\mathcal{P}|-1)}[\![\mathcal{P}, [\![\mathcal{P}, A]\!]]\!] = 2[\![\mathcal{P}, [\![\mathcal{P}, A]\!]]\!].$$

where we used the fact that $\mathcal{P}$ bivector, and thus $|\mathcal{P}| = 2$. Moreover, by Definition 3, the lefthand side of the equation equals 0. We conclude that $[\![\mathcal{P}, [\![\mathcal{P}, A]\!]]\!] = 0$ for all homogeneous multivectors A, and we extend this to arbitrary multivectors with the bilinearity of the Schouten bracket. This shows that $\mathrm{d}_\mathcal{P}$ is a differential on the space of multivectors. □

With this differential, we can define a cohomology theory [6] on the space of multivectors of a Poisson manifold $(M^d, \mathcal{P})$. The cochain complex is given directly below.

$$0 \longrightarrow \mathbb{R} \hookrightarrow C^\infty(M^d) \xrightarrow{(\mathrm{d}_\mathcal{P})_1} \mathfrak{X}(M^d) \xrightarrow{(\mathrm{d}_\mathcal{P})_2} \mathfrak{X}^2(M^d) \xrightarrow{(\mathrm{d}_\mathcal{P})_3} \dots \xrightarrow{(\mathrm{d}_\mathcal{P})_d} \mathfrak{X}^d(M^d) \xrightarrow{(\mathrm{d}_\mathcal{P})_{d+1}} 0 \quad (\star)$$

Remark 6. Of course, the maps $(\mathrm{d}_\mathcal{P})_i$ are all given by $[\![\mathcal{P}, \cdot]\!]$. The extra subscript is added here for readability of the next definition.

Definition 6 (Poisson cohomology)**.** The nth Poisson cohomology of the complex $(\star)$ is given by $H^n_\mathcal{P}(M^d) = \ker(\mathrm{d}_\mathcal{P})_{n+1} / \operatorname{im}(\mathrm{d}_\mathcal{P})_n$

Definition 7 (Casimir function)**.** Let $(M^d, \{\cdot,\cdot\}_\mathcal{P})$ be a Poisson manifold. We call $a \in C^\infty(M^d)$ a *Casimir* when $\{a, f\} = 0$ for all $f \in C^\infty(M^d)$, that is, a Poisson commutes with all other functions.

Definition 8 (Hamiltonian functions and vector fields)**.** We call $H \in C^\infty(M^d)$ *Hamiltonian functions* (or *Hamiltonians*). We call a vector field *Hamiltonian* if it is in the image of the Lichnerowicz–Poisson differential, that is $\vec{X}_H = \mathrm{d}_\mathcal{P}(H) = [\![\mathcal{P}, H]\!] \in \mathfrak{X}^1(M^d)$, for some Hamiltonian H.

Definition 9 (Poisson vector fields)**.** We call $\vec{X} \in \mathfrak{X}^1(\mathbb{R}^d)$ a *Poisson vector field* if $[\![\mathcal{P}, \vec{X}]\!] = 0$.

Remark 7. Note that $H^0_\mathcal{P}$ consists of functions $g \in C^\infty(M^d)$ such that $[\![P, g]\!] = 0$. Recalling Remark 5, we note that this means that $\vec{X}_g$ is the zero vector field, i.e.,

$$\{f, g\}_\mathcal{P} = \vec{X}_g(f) = 0$$

for all $f \in C^\infty(M^d)$. We conclude that $H^0_\mathcal{P}$ is spanned exactly by the Casimir functions on M^d.

Remark 8. Note that we can rephrase the definition of the first Poisson cohomology $H^1_\mathcal{P}(M)$ to be precisely the Poisson vector fields modulo Hamiltonian vector fields.

Definition 10 (Symplectic leaves)**.** Let us consider $\boldsymbol{x}, \boldsymbol{y} \in (M^d, \{\cdot,\cdot\}_{\mathcal{P}})$. We create an equivalence relation[6] in the following way. We say that $\boldsymbol{x} \sim \boldsymbol{y}$ if we can move from $\boldsymbol{x}$ to $\boldsymbol{y}$ by following flows of Hamiltonian vector fields. The equivalence classes of $(M^d, \{\cdot,\cdot\}_{\mathcal{P}})/\sim$ created this way are the *symplectic leaves* of the manifold.

Theorem 2. *(*[6, p. 259]*) Consider some Poisson manifold $(M^d, \{\cdot,\cdot\}_{\mathcal{P}})$. Suppose that $\mathcal{P}$ is nondegenerate. Then, the Poisson cohomology is isomorphic to the de Rham cohomology on M^d.*

Remark 9. We will be working with (very) degenerate Poisson bivectors on affine real finite-dimensional space $\mathbb{R}^d_{\text{aff}}$, and as such, we cannot use Theorem 2 to determine that the cohomology disappears, nor do we have a reason to expect that the cohomology *would* disappear. However, as it turns out (see [4]), under some assumptions, the first Poisson cohomology for these degenerate structures *is* trivial, that is, all Poisson vector fields are Hamiltonian vector fields.

Let us now introduce the class of Poisson structures we are interested in for this thesis, the Nambu-determinant Poisson structures.

Definition 11 (Nambu-determinant Poisson bracket)**.** A *Nambu-determinant Poisson bracket* on $\mathbb{R}^d$ is a Poisson bracket of the form

$$\mathcal{P}(f,g) = \{f,g\}_{\mathcal{P}} = \varrho(\boldsymbol{x}) \det \begin{bmatrix} f_{x^1} & g_{x^1} & a^1_{x^1} & \dots & a^{d-2}_{x^1} \\ f_{x^2} & g_{x^2} & a^1_{x^2} & \dots & a^{d-2}_{x^2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ f_{x^d} & g_{x^d} & a^1_{x^d} & \dots & a^{d-2}_{x^d} \end{bmatrix},$$

where $\varrho, f, g, a^i \in C^\infty(\mathbb{R}^d)$, and where we use the notation $f_{x^i} := \frac{\partial}{\partial x^i}(f)$.

Remark 10. Note that the a^i are Casimir functions; they Poisson commute with everything as the determinant of a matrix with two identical columns is always 0. Moreover, (not necessarily linear!) combinations of the a^i may be Casimirs. This observation also directly tells us that the Nambu-determinant Poisson structures are automatically degenerate (if $d \geq 3$), as rank $\mathcal{P}_{\boldsymbol{x}} \leq 2$ for all $\boldsymbol{x} \in \mathbb{R}^d$.

Remark 11. Note that on $\mathbb{R}^2$ with global coordinates x, y, the Nambu-determinant Poisson bracket reduces to

$$\{f,g\}_{\mathcal{P}} = \varrho(\boldsymbol{x}) \cdot \det \begin{bmatrix} f_x & g_x \\ f_y & g_y \end{bmatrix} = \varrho(\boldsymbol{x}) \cdot \frac{\partial}{\partial x} \wedge \frac{\partial}{\partial y}(f,g).$$

Example 2. Let us describe the basic structure of the symplectic leaves of a Poisson manifold equipped with a Nambu-determinant Poisson bracket. We claim that Casimirs must be constant on the symplectic leaves. Indeed, we have

$$\vec{X}_f(a) = -\{a,f\}_{\mathcal{P}} = 0,$$

[6]Note that the properties of transitivity, reflexivity and symmetry are readily checked.

that is, along the flow of Hamiltonian vector fields, the Casimir a must be constant. From this, we can directly infer that the symplectic leaves must be contained in the intersections of level sets of the Casimir functions. In fact, it can be shown that the symplectic leaves $\mathcal{S}$ of a Poisson manifold are exactly the connected immersed submanifolds S where for all $S \in \mathcal{S}$ and all $\boldsymbol{x} \in S$, the tangent space at $\boldsymbol{x}$ is spanned precisely by Hamiltonian vector fields [10].

3.2. Superalgebras and supercalculus

Definition 12 (Algebra)**.** An *algebra* A over a ring R is an R-module equipped with a bilinear product $\times : A \times A \to A$.

Definition 13 (Superalgebra)**.** A *superalgebra* A is a $\mathbb{Z}_2$-graded algebra. In other words, every homogeneous element $a \in A$ is graded by 0 or 1 (equivalently, is 'even' or 'odd'). We denote the grade of a by $|a|$.

Remark 12. Let A be a superalgebra, and $a, b \in A$ be homogeneous elements of this superalgebra with grading $|a|$, $|b|$. Then, their product ab has grading $|ab| = |a||b|$. In particular, this means that multiplying two even or two odd elements both give an even element, whereas multiplying one even and one odd element gives an odd element.

With said grading, we can decompose (the underlying module of) the algebra into two parts; $A = A_0 \oplus A_1$, where A_0 contains the even elements of the algebra while A_1 contains the odd elements. With this in mind, we can rephrase the above remark. For $a \in A_i$, $b \in A_j$, $ab \in A_{i+j}$, where $i, j, i + j \in \mathbb{Z}_2$.

Example 3. Consider some d-dimensional smooth manifold M^d with global even coordinates $x^1, .., x^d$, so that $x^i x^j = x^j x^i$. We can define d odd dual coordinates $\xi_1, ..., \xi_d$ along the fibers of T^*M^d. These odd coordinates satisfy $\xi_i \xi_j = -\xi_j \xi_i$.

Definition 14 (Supercommutative algebra)**.** A *supercommutative algebra* is a superalgebra such that for homogeneous elements a and b, we have $ab = (-1)^{|a||b|} ba$.

Remark 13. We have the following consequence of supercommutativity. For odd elements, we find

$$\xi_i^2 = \xi_i \xi_i = -\xi_i \xi_i = -\xi_i^2,$$

or, in other words, $\xi_i^2 = 0$.

The notion of a superalgebra and the corresponding splitting into an even and odd part can be very useful. As an example, supercommutative algebras are used to provide a mathematical framework for supersymmetry. In this context, we let even elements correspond to bosons, and odd elements correspond to fermions. This way, the elements of the algebra explicitly model that bosonic elements commute, while fermionic elements anticommute. For more details, see [11].

Definition 15 (Lie superalgebra)**.** A *Lie superalgebra* is a superalgebra endowed with a Lie superbracket. The Lie superbracket satisfies:

- (Super skew-symmetry) For homogeneous elements a and b, we have
$$[a,b] = -(-1)^{|a||b|}[b,a],$$
- (Super Jacobi identity) For homogeneous elements a, b, and c, we have
$$(-1)^{|a||c|}[a,[b,c]] + (-1)^{|b||a|}[b,[c,a]] + (-1)^{|c||b|}[c,[a,b]] = 0.$$

Mathematically, there is no reason to just consider $\mathbb{Z}_2$-gradings. We can also define $\mathbb{Z}$-gradings.[7]

Definition 16 ($\mathbb{Z}$-graded algebra)**.** A *$\mathbb{Z}$-graded Lie algebra* is a Lie algebra over a ring R where the underlying R-module M can be written as $M = \bigoplus_{\alpha\in\mathbb{Z}} M^\alpha$ such that the Lie bracket satisfies

- $[M^\alpha, M^\beta] \subset M^{\alpha+\beta}$,
- (Graded skew-symmetry) For homogeneous elements $a \in M^\alpha$ and $b \in M^\beta$, we have
$$[a,b] = -(-1)^{\alpha\beta}[b,a]$$
- (Graded Jacobi identity) For homogeneous elements $a \in M^\alpha$, $b \in M^\beta$ and $c \in M^\gamma$, we have
$$(-1)^{\alpha\beta}[a,[b,c]] + (-1)^{\beta\alpha}[b,[c,a]] + (-1)^{\gamma\beta}[c,[a,b]] = 0.$$

Definition 17 ($\mathbb{Z}$-graded Lie superalgebra)**.** A $\mathbb{Z}$-graded Lie superalgebra is a $\mathbb{Z}$-graded Lie algebra, where the homogeneous elements of the algebra also have an odd or even grading.

Example 4. Consider the situation as in Example 3. Let us take the standard $\mathbb{Z}$-grading on this space, where
$$|x^{i_1}x^{i_2}...x^{i_k}| = 0, \qquad |\xi_{j_1}\xi_{j_2}...\xi_{j_\ell}| = \ell,$$
and
$$|x^{i_1}x^{i_2}...x^{i_n}\xi_{j_1}\xi_{j_2}...\xi_{j_m}| = m.$$
Then, a homogeneous element of the form $x^{i_1}x^{i_2}x^{i_3}$ is even (or, has grading 0 in the $\mathbb{Z}_2$-grading), and has grading 0 in the $\mathbb{Z}$-grading. An element $\xi_{j_1}\xi_{j_2}\xi_{j_3}$ is odd (or, has grading 1 in the $\mathbb{Z}_2$-grading) and has grading 3 in the $\mathbb{Z}$-grading. The element $x^{i_1}\xi_{j_1}\xi_{j_2}$ is even (or, has grading 0 in the $\mathbb{Z}_2$ grading) and has grading 2 in the $\mathbb{Z}$-grading.

[7] In fact, the notion of grading is more general than just $\mathbb{Z}_2$- or $\mathbb{Z}$-gradings, and we can consider gradings indexed by monoids, see [12, Chapter 16.6]. However, for our purposes, we will only be interested in $\mathbb{Z}_2$- and $\mathbb{Z}$-gradings.

Warning: When we consider an element $\xi_{j_1}\xi_{j_2}...\xi_{j_k}$, it is *not* always an odd element; it is odd when k is odd, and even when k is even, recall Remark 12.

We will be working over a Gerstenhaber algebra, which combines the notion of supercommutativity with a graded Lie superalgebra.

Definition 18 (Gerstenhaber algeba)**.** A *Gerstenhaber algebra* is a graded supercommutative algebra with a graded Lie bracket $[\cdot,\cdot]$ of degree -1. Explicitly, the algebra has a $\mathbb{Z}$-grading, called the *grade* (written $|a|$ for a homogeneous element a) and satisfies the following:

- (Product is associative) For all elements a, b and c, we have $(ab)c = a(bc)$,
- (Product is supercommutative) For homogeneous elements a and b, we have
$$ab = (-1)^{|a||b|}ba,$$
- (Product has degree 0) For homogeneous elements a and b, we have $|ab| = |a| + |b|$,
- (Lie bracket has degree -1) For homogeneous elements a and b, we have
$$|[a,b]| = |a| + |b| - 1,$$
- (Shifted-graded Leibniz rule) For homogeneous elements a and b, we have
$$[a,bc] = [a,b]c + (-1)^{(|a|-1)|b|}b[a,c],$$
- (Shifted-graded skew-symmetry of the Lie bracket) For homogeneous elements a and b, we have
$$[a,b] = -(-1)^{(|a|-1)(|b|-1)}[b,a],$$
- (Shifted-graded Jacobi identity) For homogeneous elements a and b, we have
$$[a,[b,c]] = [[a,b],c] + (-1)^{(|a|-1)(|b|-1)}[b,[a,c]].$$

The explicit Gerstenhaber algebra we are interested in is the space of multivectors over an affine real Poisson manifold $(\mathbb{R}^d_{\text{aff}}, \{\cdot,\cdot\}_{\mathcal{P}})$ of finite dimension[8] d. The graded Lie bracket in this case is the Schouten bracket (see Definition 2). Moreover, the ring over which this Gerstenhaber algebra is defined is the ring of smooth functions on our manifold $C^\infty(\mathbb{R}^d_{\text{aff}})$.[9]

Remark 14. A Gerstenhaber algebra is *not* a (graded) Poisson superalgebra. In the case of Poisson superalgebras, the Lie bracket that is considered has degree 0, while in the Gerstenhaber algebra case, the bracket must have degree -1.

[8]Practically, we will only work with $d = 2, 3, 4$.

[9]For practical use in computer calculations, we will instead use the ring of differential polynomials (up to some chosen order, dictated explicitly by the graphs we are working with).

We can identify the space of multivectors $\bigoplus_{k=0}^{d}\mathfrak{X}^k(M^d)$ with functions on the affine real supermanifold $\mathbb{R}^{d|d}_{\text{aff}}$ of superdimension[10] (d,d). Let us denote the even coordinates on $\mathbb{R}^{d|d}$ by $x^1,x^2,...,x^d$, while we denote the odd coordinates by $\xi_1,\xi_2,...,\xi_d$. The identification between these Gerstenhaber algebras is via the identity map on the even coordinates, and the mapping sending a k-vector $\frac{\partial}{\partial x^{i_1}}\wedge\frac{\partial}{\partial x^{i_2}}\wedge...\wedge\frac{\partial}{\partial x^{i_k}}$ to a superfunction[11] $\xi_{i_1}\xi_{i_2}...\xi_{i_k}$ of *degree* k in $\mathbb{R}^{d|d}$ (see Example 3). In this sense, superfunctions of degree 1 are identified with vector fields, superfunctions of degree 2 are identified with bivectors etc. Note that in light of Remark 13, this identification is an isomorphism.

Remark 15. While we have a nice definition of the Schouten bracket (Definition 2) in the sense that it is very general, it is difficult to explicitly compute with it. Because of this, rather than considering the space of multivectors $\bigoplus_{k=0}^{d}\mathfrak{X}^k(\mathbb{R}^d_{\text{aff}})$ directly, we instead use the identification with $\mathbb{R}^{d|d}$ as described above. On this supermanifold, we *can* find explicit expressions of the Schouten bracket (see Definition 19 below). With this identification in mind, we will often call superfunctions of k odd coordinates k-vector fields and we will use $\mathfrak{X}^k(\mathbb{R}^{d|d})$ to denote the superfunctions on $\mathbb{R}^{d|d}$ of degree k.

Definition 19 (Schouten bracket on a real supermanifold)**.** The *Schouten bracket* on a real supermanifold $\mathbb{R}^{d|d}$ $[\![\cdot,\cdot]\!]:\mathfrak{X}^k(\mathbb{R}^{d|d})\times\mathfrak{X}^\ell(\mathbb{R}^{d|d})\to\mathfrak{X}^{k+\ell-1}(\mathbb{R}^{d|d})$ is defined by

$$[\![A,B]\!]=A\frac{\overleftarrow{\partial}}{\partial\xi_i}\frac{\overrightarrow{\partial}}{\partial x^i}B-A\frac{\overleftarrow{\partial}}{\partial x^i}\frac{\overrightarrow{\partial}}{\partial\xi_i}B.$$

Example 5. Consider the supermanifold $\mathbb{R}^{d|d}$ with global even coordinates $x^1,...,x^d$ and global odd coordinates $\xi_1,...,\xi_d$. Let us take two vector fields $\vec{X}=X^a(\boldsymbol{x})\xi_a$ and $\vec{Y}=Y^b(\boldsymbol{x})\xi_b$. We compute

$$\begin{aligned}
[\![\vec{X},\vec{Y}]\!]&=(X^a(\boldsymbol{x})\xi_a)\frac{\overleftarrow{\partial}}{\partial\xi_i}\frac{\overrightarrow{\partial}}{\partial x^i}(Y^b(\boldsymbol{x})\xi_b)-(X^a(\boldsymbol{x})\xi_a)\frac{\overleftarrow{\partial}}{\partial x^i}\frac{\overrightarrow{\partial}}{\partial\xi_i}(Y^b(\boldsymbol{x})\xi_b)\\
&=X^a(\boldsymbol{x})\delta^i_a\frac{\partial Y^b}{\partial x^i}(\boldsymbol{x})\xi_b-\frac{\partial X^a}{\partial x^i}(\boldsymbol{x})\xi_aY^b(\boldsymbol{x})\delta^i_b\\
&=X^i(\boldsymbol{x})\frac{\partial Y^b}{\partial x^i}(\boldsymbol{x})\xi_b-\frac{\partial X^a}{\partial x^i}(\boldsymbol{x})\xi_aY^i(\boldsymbol{x})\\
&=(X^i(\boldsymbol{x})\frac{\partial Y^q}{\partial x^i}(\boldsymbol{x})-\frac{\partial X^q}{\partial x^i}(\boldsymbol{x})Y^i(\boldsymbol{x}))\xi_q,
\end{aligned}$$

where a, b and q range over $1,2,...,d$.

[10]The superdimension is simply a pair $(a,b)\in\mathbb{N}^2_{\geq 0}$, denoting the fact that we have a even coordinates and b odd coordinates. In particular, our usual $\mathbb{R}^d$ is artificially a supermanifold with superdimension $(d,0)$. Whenever we have only even coordinates (respectively only odd coordinates) we call the manifold *even* (respectively odd).

[11]Note that this superfunction is a polynomial in the odd coordinates.

Note that this second definition of the Schouten bracket on real supermanifolds satisfies the properties of Definition 2; these properties can be readily verified by explicitly writing the definition of the Schouten bracket for real supermanifolds.

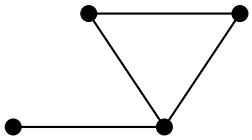
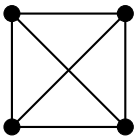
4. The Kontsevich graph complex

In this chapter, we introduce the Kontsevich graph complex and micro-graphs specific for Nambu-determinant Poisson structures, and explain how we can evaluate these graphs into multivectors.

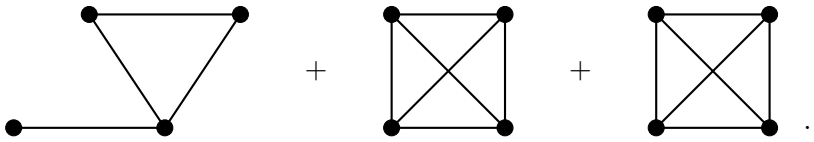
4.1. The Kontsevich graph complex **Gra**

Consider the (real) vector space of finite, unoriented, unlabeled graphs Γ such that Γ does not have loops (an edge from a vertex to itself), or double edges. As usual, we denote the vertices and edges of the graph Γ with $V(\Gamma)$ and $E(\Gamma)$ respectively. Moreover, we set $v(\Gamma) = |V(\Gamma)|$ and $e(\Gamma) = |E(\Gamma)|$. The addition of two graphs Γ_1 and Γ_2 in this vector space is the formal sum of the two graphs, $\Gamma_1 + \Gamma_2$, and scalar multiplication $a\Gamma$ is formally summing the graph Γ a times.

Example 6. For graphs

$\Gamma_1 =$  $\Gamma_2 =$  ,

we have

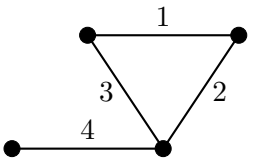
$\Gamma_1 + 2 \cdot \Gamma_2 =$  .

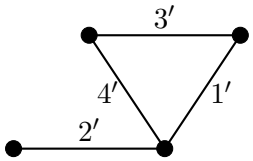
For each graph Γ, we order the edges $E^{\text{ord}}(\Gamma) = 1 \prec 2 \prec ... \prec e(\Gamma)$ and we denote this graph with ordered edges by $(\Gamma, E^{\text{ord}}(\Gamma))$. Let us consider two copies of the same graph Γ, and denote the two copies by Γ_1 and Γ_2. Moreover, let us assume that the edges of Γ_1 are ordered as $E^{\text{ord}}(\Gamma_1) = 1 \prec 2 \prec ... \prec e(\Gamma_2)$, while the edges of Γ_2 are ordered as $E^{\text{ord}}(\Gamma_2) = 1' \prec 2' \prec ... \prec e(\Gamma_2)'$. As Γ_1 and Γ_2 are topologically equivalent, they share the same edge set (and hence, have the same number of edges as well). Note that this means that the $E^{\text{ord}}(\Gamma_1)$ and $E^{\text{ord}}(\Gamma_2)$ must be related by a permutation σ on the edge ordering. We now define an equivalence relation via

$$(\Gamma_1, E^{\text{ord}}(\Gamma_1)) \sim (-1)^{\sigma}(\Gamma_2, E^{\text{ord}}(\Gamma_2)), \tag{3}$$

where σ is the permutation taking the edge ordering $E^{\text{ord}}(\Gamma_1)$ to the edge ordering $E^{\text{ord}}(\Gamma_2)$. When it is clear what the edge orderings are, we will use $\Gamma_1 \sim \pm\Gamma_2$ as well.

Example 7. Consider $(\Gamma_1, E^{\mathrm{ord}}(\Gamma_1))$ and $(\Gamma_2, E^{\mathrm{ord}}(\Gamma_2))$ as below.

$\Gamma_1 =$ 

$\Gamma_2 =$ 

Note that the permutation taking $E^{\mathrm{ord}}(\Gamma_1)$ to $E^{\mathrm{ord}}(\Gamma_2)$ is given by[12] $\sigma = (1342) = (42)(32)(12)$. The sign of the permutation is therefore -1. We conclude that $\Gamma_1 \sim -\Gamma_2$.

Definition 20 (Graph complex **Gra**)**.** Consider the vector space of finite, unoriented, unlabeled graphs, without loops or double edges, with edge ordering. We define **Gra** to be the quotient of this vector space modulo the equivalence relation $\sim$ of Equation (3).

Remark 16. Note that $[0] \in \mathbf{Gra}$ contains graphs satisfying that $\Gamma \sim -\Gamma$. This happens when the graph has a symmetry that induces an odd permutation σ. For example, take Γ_1 of Example 7. Note that we have a symmetry of the graph by flipping the edges 2 and 3. In other words $(\Gamma_1, E^{\mathrm{ord}}(\Gamma_1)) = (\Gamma_1, E^{\mathrm{ord}}(\Gamma_1)')$, where $E^{\mathrm{ord}}(\Gamma_1)' = 1 \prec 3 \prec 2 \prec 4$. The odd permutation this symmetry induces is $\sigma = (23)$. Hence, we see that $(\Gamma_1, E^{\mathrm{ord}}(\Gamma_1)) \sim -(\Gamma_1, \sigma(E^{\mathrm{ord}}(\Gamma_1))) \sim -(\Gamma_1, E^{\mathrm{ord}}(\Gamma_1)') = -(\Gamma_1, E^{\mathrm{ord}}(\Gamma_1))$, and we conclude that the graph belongs to $[0] \in \mathbf{Gra}$. We call such graphs *zero graphs*.

On **Gra**, we can define a bracket $[\cdot, \cdot]$ in the following way.

Definition 21 (Insertion)**.** Let $\Gamma_1, \Gamma_2 \in \mathbf{Gra}$. We define $\circ_i$ to be the (noncommutative!) operation

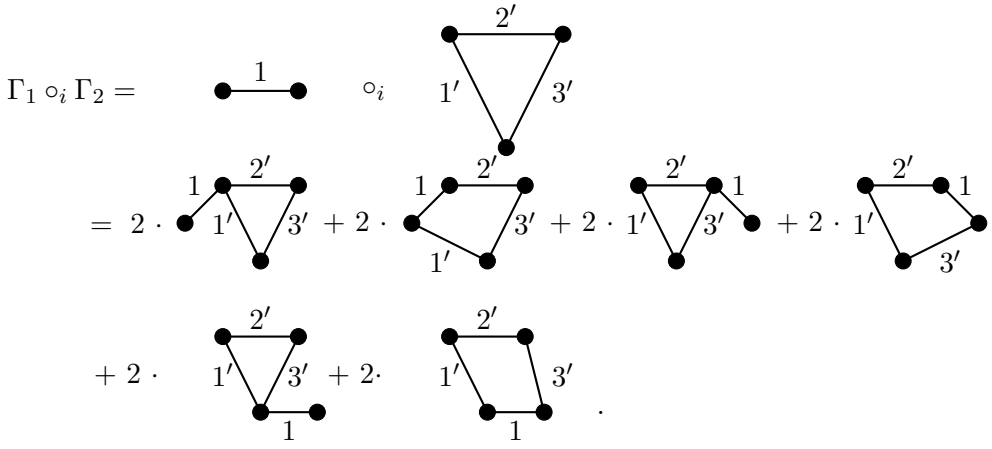
$$\Gamma_1 \circ_i \Gamma_2 = \sum_{v \in V(\Gamma_2)} (\Gamma_1 \to v \text{ in } \Gamma_2),$$

that is, we sum over all the vertices $v \in \Gamma_2$, insert the graph Γ_1 where the vertex v was, and redirect all the edges containing v to vertices in Γ_1 via Leibniz rules for each edge. Note that the result is a sum of graphs. For each such a graph, we have the edge ordering $E^{\mathrm{ord}}(\Gamma_1) \prec E^{\mathrm{ord}}(\Gamma_2)$.[13]

Example 8. Let us consider an example where Γ_1 is the graph on 2 vertices with 1 edge (also called the *stick graph*) ordered with 1, and where Γ_2 is a triangle, with the edges ordered by $1' \prec 2' \prec 3'$. Then, we have

[12] Here, we use the notation $(a_1 a_2 ... a_n)$ for permutations, meaning that a_1 is sent to a_2, a_2 to a_3 etc. The element a_n is sent to a_1.

[13] The redirected edges that contained v all had an order in Γ_2; we will keep them with the same order, even though the target vertex has changed.



The coefficients 2 appear because we create isomorphic graphs. We can do a sanity check by realising that we should create $3 \cdot 2^2 = 12$ graphs; we insert the stick graph in each of the 3 vertices of the triangle, and for each such insertion, we redirect 2 edges between 2 vertices, leading to 2^2 possibilities to redirect the edges.

Definition 22 (Bracket on **Gra**)**.** The bracket $[\cdot,\cdot] : \mathbf{Gra} \times \mathbf{Gra} \to \mathbf{Gra}$ acts on pairs of of graphs by

$$[\Gamma_1, \Gamma_2] = \Gamma_1 \circ_i \Gamma_2 - (-)^{e(\Gamma_1)\cdot e(\Gamma_2)} \Gamma_2 \circ_i \Gamma_1.$$

We extend this bracket linearly to generic elements of **Gra**.

This bracket is actually a graded Lie bracket (see [13, 14]) and moreover, we have an associated differential d, defined by

$$\mathsf{d}(\cdot) = [\bullet\!\!-\!\!\bullet,\ \cdot\ ].$$

With this differential, we get a differential graded Lie algebra.[14] Note that this differential takes a graph with $v(\Gamma)$ vertices and $e(\Gamma)$ edges, and returns a formal sum of graphs on $v(\Gamma)+1$ vertices and $e(\Gamma)+1$ edges.

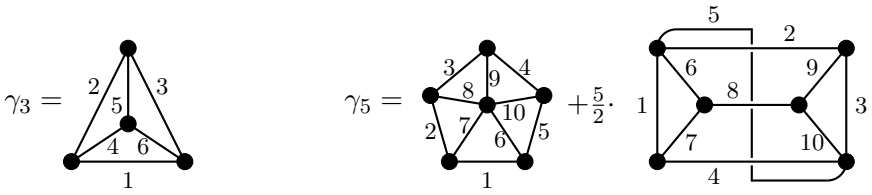
Definition 23 (Cocycles and coboundaries)**.** Elements $\sum_i a_i\Gamma_i \in \mathbf{Gra}$ such that $\mathsf{d}(\sum_i a_i\Gamma_i) = [\bullet\!\!-\!\!\bullet, \sum_i a_i\Gamma_i] = 0$ are *cocycles*. The elements $\sum_j a_j\Gamma_j \in \mathbf{Gra}$ such that there exists $\sum_k b_k\tilde{\Gamma}_k$ with the property $\sum_j a_j\Gamma_j = \mathsf{d}(\sum_k b_k\tilde{\Gamma}_k)$ are *coboundaries*.

We call a cocycle *nontrivial* if it is not a coboundary.

Example 9. There are infinitely many cocycles in **Gra**. In particular, Willwacher showed that there is an infinite sequence of (nontrivial) cocycles on n vertices and $2n-2$ edges [15]. All of these cocycles consist of a $(n-1)$-wheel, and possibly other graphs of n vertices

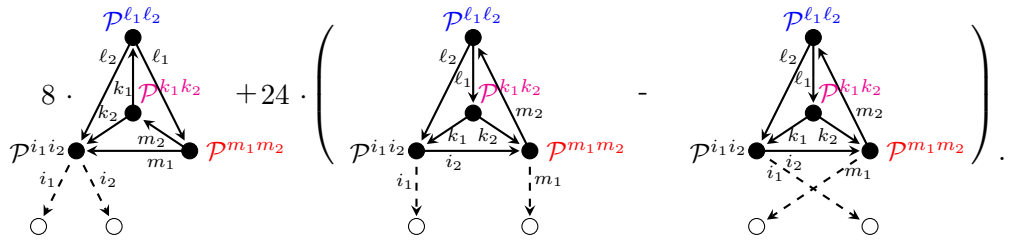
[14] The grading for a homogeneous element Γ is given by $e(\Gamma)$.

and $2n-2$ edges. The first of these wheel-graph cocyles (already mentioned by Kontsevich in [16]) is the tetrahedron γ_3 (represented only by the 3-wheel graph). The second such graph is γ_5 [17]; this representative consists of the 5-wheel and one other graph. The next wheel-graph cocycle (γ_7) already has 46 graphs in its representation [18].



Note that we can orient these graphs by directing the edges in all possible ways. In this directed graph complex, the graphs that were nontrivial cocycles remain nontrivial cocycles [16], [2]. We use these wheel-graph cocycles to create bivectors on affine Poisson manifolds via the *orientation morphism* $\vec{\mathrm{Or}}$.[15] Details of this morphism are discussed in [20]. Practically, we take a wheel-graph cocycle γ_{2i+1} in the unoriented graph complex, and orient the edges in all possible ways, as long as there is no vertex with more than two outgoing edges[16]. As the cocycles we are interested in are graphs on n vertices and $2n-2$ edges, we either have 1 vertex with no outgoing edges, or 2 vertices with only 1 outgoing edge. In the next step, we add two more edges so that at every vertex we have exactly 2 outgoing edges. These extra two edges are directed to *new* vertices. We call these new vertices *sinks*. The next step is to place a Poisson structure at each vertex in the (now oriented!) graphs. The incoming edges at each vertex correspond to derivations of the Poisson structure placed at this vertex. Now, we can create the corresponding bivector by writing down the derivations dictated by the graphs. Let us give an example of this procedure.

Example 10. Let us consider γ_3. Orienting this graph in all possible ways gives us



[15]This morphism is actually more general than how we use it here. In particular, as described below, we place a Poisson structure at each of the vertices of the cocycle. While we will be placing the *same* Poisson structure at every vertex, we are allowed to pick different Poisson structures for each vertex. In fact, depending on the graph cocycle, we can also add other multivectors at each vertex, see [19] for an example.

[16]This imitates the fact that Poisson structures are bivectors, and take only two functions as arguments, not more.

The coefficients 8 and 24 follow from simple combinatorial arguments. Moreover, note that as these graphs represent Poisson bivectors, they must be skew-symmetric in their arguments. The filled vertices show the original vertices of the graphs, while the unfilled vertices show where we can place 2 smooth functions as arguments of the bivectors. Clearly, the first graph is skew-symmetric in its arguments already, but taking either the second or the third graph on their own is not enough to satisfy skew-symmetry. Now, we place the Poisson bivector $\mathcal{P}$ at all the (original) vertices of the graph. With this, we can write down the explicit multivector depending on $\mathcal{P}$ associated to these graphs! Recalling that the directed edges indicate differentiation, we write for the first graph

$$\sum_{i_1,i_2}\left(\sum_{k_1,k_2,\ell_1,\ell_2,m_1,m_2}\frac{\partial^3\mathcal{P}^{i_1i_2}}{\partial x^{k_2}\partial x^{\ell_2}\partial x^{m_1}}\frac{\partial\mathcal{P}^{k_1k_2}}{\partial x^{m_2}}\frac{\partial\mathcal{P}^{\ell_1\ell_2}}{\partial x^{k_1}}\frac{\partial\mathcal{P}^{m_1m_2}}{\partial x^{\ell_1}}\right)\frac{\partial}{\partial x^{i_1}}\wedge\frac{\partial}{\partial x^{i_2}},$$

where the indices run over the coordinates. Doing the same for the second part of $\vec{\gamma}_3$ now gives us $\vec{\mathrm{Or}}(\gamma_3)(\mathcal{P})=Q_{\gamma_3}(\mathcal{P})\in\mathfrak{X}^2(\mathbb{R}^d_{\mathrm{aff}},\{\cdot,\cdot\}_{\mathcal{P}})$. Explicitly, we have

$$\begin{aligned}Q_{\gamma_3}(\mathcal{P}) = 8\ &\cdot\sum_{i_1,i_2}\left(\sum_{k_1,k_2,\ell_1,\ell_2,m_1,m_2}\frac{\partial^3\mathcal{P}^{i_1i_2}}{\partial x^{k_2}\partial x^{\ell_2}\partial x^{m_1}}\frac{\partial\mathcal{P}^{k_1k_2}}{\partial x^{m_2}}\frac{\partial\mathcal{P}^{\ell_1\ell_2}}{\partial x^{k_1}}\frac{\partial\mathcal{P}^{m_1m_2}}{\partial x^{\ell_1}}\right)\frac{\partial}{\partial x^{i_1}}\wedge\frac{\partial}{\partial x^{i_2}}\\ +\ 24\ &\cdot\sum_{i_1,m_1}\left(\sum_{i_2,k_1,k_2,\ell_1,\ell_2,m_2}\frac{\partial^2\mathcal{P}^{i_1i_2}}{\partial x^{k_2}\partial x^{\ell_2}}\frac{\partial\mathcal{P}^{k_1k_2}}{\partial x^{\ell_1}}\frac{\partial\mathcal{P}^{\ell_1\ell_2}}{\partial x^{m_2}}\frac{\partial^2\mathcal{P}^{m_1m_2}}{\partial x^{i_2}\partial x^{k_1}}\right)\frac{\partial}{\partial x^{i_1}}\wedge\frac{\partial}{\partial x^{m_1}}\\ -\ 24\ &\cdot\sum_{i_1,m_1}\left(\sum_{i_2,k_1,k_2,\ell_1,\ell_2,m_2}\frac{\partial^2\mathcal{P}^{i_1i_2}}{\partial x^{k_2}\partial x^{\ell_2}}\frac{\partial\mathcal{P}^{k_1k_2}}{\partial x^{\ell_1}}\frac{\partial\mathcal{P}^{\ell_1\ell_2}}{\partial x^{m_2}}\frac{\partial^2\mathcal{P}^{m_1m_2}}{\partial x^{i_2}\partial x^{k_1}}\right)\frac{\partial}{\partial x^{m_1}}\wedge\frac{\partial}{\partial x^{i_1}}.\end{aligned}$$

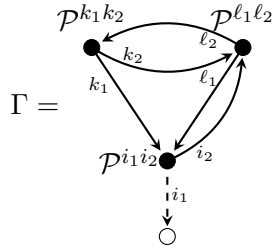
In other words, we actually have the first multivector 8 times, and the second multivector 48 times as bivectors are skew-symmetric in their arguments.

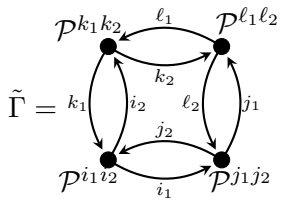
This procedure of taking an unoriented wheel-graph cocycle γ and some Poisson bivector $\mathcal{P}$ to create a new bivector[17] $\vec{\mathrm{Or}}(\gamma)(\mathcal{P})=Q_\gamma(\mathcal{P})$ is universal, that is, it works for all Poisson bivectors on affine finite-dimensional Poisson manifolds and with any wheel-graph cocycle. In particular, if we could show that a specific graph cocycle $\tilde{\gamma}$ is a graph coboundary (in either the undirected or directed graph complex), then this would directly imply that the bivector $Q_{\tilde{\gamma}}(\mathcal{P})$ is a coboundary in the Poisson cohomology defined in Definition 6 for all Poisson bivectors $\mathcal{P}$ on affine finite-dimensional Poisson manifolds [2, 16].

Note that we can use this construction to move from more general directed graphs, with n vertices with exactly 2 outgoing edges at each vertex and k sink vertices with no outgoing edges and precisely 1 incoming edge to k-vectors, depending on the Poisson structure $\mathcal{P}$. The multivectors obtained from graphs this way are exactly what we will restrict ourselves to

[17] Note that in light of Footnote 15, we should actually write $\vec{\mathrm{Or}}(\gamma)\left(\mathcal{P}^{\otimes^{v(\gamma)}}\right)$.

in the papers in Appendix A and Section 6.[18] We denote this evaluation of directed graphs into multivectors by ϕ. Two examples of the evaluation ϕ to multivectors on $(\mathbb{R}^d, \{\cdot,\cdot\}_{\mathcal{P}})$ are given directly below.

$\Gamma =$ 

$\tilde{\Gamma} =$ 

The corresponding multivectors are given below. The indices again run over the coordinates.

$$\phi(\Gamma) = \sum_{i_1,i_2,k_1,k_2,\ell_1,\ell_2} \frac{\partial^2 \mathcal{P}^{i_1 i_2}}{\partial x^{k_1} \partial x^{\ell_1}} \frac{\partial \mathcal{P}^{k_1 k_2}}{\partial x^{\ell_2}} \frac{\partial^2 \mathcal{P}^{\ell_1 \ell_2}}{\partial x^{k_2} \partial x^{i_2}} \frac{\partial}{\partial x^{i_1}} \in \mathfrak{X}(\mathbb{R}^d)$$

$$\phi(\tilde{\Gamma}) = \sum_{i_1,i_2,j_1,j_2,k_1,k_2,\ell_1,\ell_2} \frac{\partial^2 \mathcal{P}^{i_1 i_2}}{\partial\partial x^{k_1} \partial x^{j_2}} \frac{\partial^2 \mathcal{P}^{j_1 j_2}}{\partial x^{i_1} \partial x^{\ell_2}} \frac{\partial^2 \mathcal{P}^{k_1 k_2}}{\partial x^{\ell_1} \partial x^{i_2}} \frac{\partial^2 \mathcal{P}^{\ell_1 \ell_2}}{\partial x^{j_1} \partial x^{k_2}} \in C^\infty(\mathbb{R}^d)$$

4.2. Micro-graph calculus for Nambu-determinant Poisson brackets

Let us now consider what happens when we restrict our attention to the Nambu-determinant Poisson brackets over $\mathbb{R}^d_{\text{aff}}$. Recall from Definition 11 that we can write this bracket as

$$\{f,g\}_{\mathcal{P}} = \varrho(\boldsymbol{x}) \det \begin{bmatrix} f_{x_1} & g_{x_1} & a^1_{x_1} & \dots & a^{d-2}_{x_1} \\ f_{x_2} & g_{x_2} & a^1_{x_2} & \dots & a^{d-2}_{x_2} \\ \vdots & \vdots & \ddots & \vdots & \\ f_{x_d} & g_{x_d} & a^1_{x_d} & \dots & a^{d-2}_{x_d} \end{bmatrix}.$$

In fact, let us write this slightly differently again by writing out the definition of the determinant

$$\{f,g\}_{\mathcal{P}} = \varrho(\boldsymbol{x}) \sum_{i_1,i_2,\dots,i_d=1}^{d} \varepsilon^{i_1 i_2 \dots i_d} \frac{\partial a^1}{\partial x^{i_3}} \frac{\partial a^2}{\partial x^{i_4}} \dots \frac{\partial a^{d-2}}{\partial x^{i_d}} \frac{\partial}{\partial x^{i_1}} \wedge \frac{\partial}{\partial x^{i_2}}(f,g),$$

where $\varepsilon^{i_1 i_2 \dots i_d}$ is the *Levi-Civita symbol* defined as

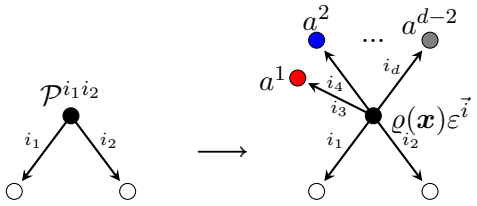
$$\varepsilon^{i_1 i_2 \dots i_d} = \begin{cases} (-1)^\sigma \text{ if there exists a permutation } \sigma \text{ such that } \sigma(12\dots d) = i_1 i_2 \dots i_d, \\ 0 \text{ otherwise.} \end{cases}$$

[18] Recall that the goal is to solve $Q_\gamma(\mathcal{P}) = [\![\mathcal{P}, \vec{X}^{\gamma_3}_d]\!]$. The bivector Q_γ is on $v(\gamma)$ copies of the Poisson structure. As $[\![\mathcal{P},\cdot]\!]$ adds an extra Poisson structure to whatever multivector it acts on, we see that the vector fields that can trivialize $Q_{\gamma(\mathcal{P})}$ must be on $v(\gamma) - 1$ copies of the Poisson structure.

In [21], the authors define dimension specific *Nambu-determinant micro-graphs* that mimic the structure of this particular class of Poisson brackets. Recall that for the universal (Kontsevich) graphs, when we move to multivectors, at each vertex the Poisson structure $\mathcal{P}$ is placed. In the case of Nambu-determinant micro-graphs, we replace this one vertex by a rooted tree of height one on $d-1$ vertices. The vertex content of the root of the tree is $\varrho(\boldsymbol{x})\varepsilon^{i_1 i_2 \dots i_d}$ and we call this vertex the *Levi-Civita vertex*, while the vertex content of the $d-2$ leaf vertices are precisely the Casimirs a^i and we call these vertices *Casimir vertices*. We will be coloring a^1 Casimir vertices with red and a^2 Casimirs with blue in what follows.

Remark 17. Recall that **Gra** does not contain graphs with double edges or loops. For these Nambu-determinant micro-graphs, we also do not allow double edges, but we do allow loops.[19]

The illustration below shows the building blocks corresponding to generic Poisson brackets, as well Nambu-determinant Poisson brackets.[20]



Remark 18. Note that switching the order of two edges corresponds to switching two rows in the determinant. As such, switching a pair of edges in the graph must be accompanied by a minus sign. A particular consequence of this is that the Poisson structures are skew-symmetric under interchanging two Casimirs.

Note that for Nambu-determinant Poisson brackets over $\mathbb{R}^2_{\text{aff}}$, we do not have any Casimirs, and thus also not any extra vertices. In fact, the micro-graphs coincide with the Kontsevich graphs for general Poisson structures.

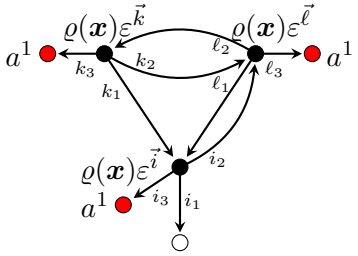
Remark 19. Casimir vertices in the micro-graphs do not have any outgoing edges as they are functions and not differential operators.

Remark 20. We call micro-graphs corresponding to multivectors over $n-$dimensional affine real space *n-dimensional micro-graphs* for simplicity.

Example 11. Let us consider the following three-dimensional micro-graph.

[19] In fact, we need graphs with loops to solve Equation (1), see Appendix C and [4, Claim 2].

[20] We write $\vec{i} = i_1 i_2 \dots i_d$ to have less crowded illustrations.



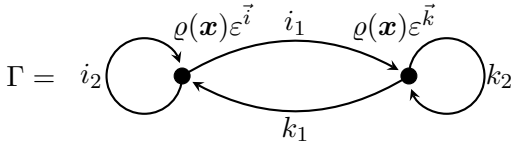
The vector field corresponding to this graph is

$$\sum_{\vec{i},\vec{k},\vec{\ell}} \varepsilon^{\vec{i}}\varepsilon^{\vec{k}}\varepsilon^{\vec{\ell}} \frac{\partial^2\varrho(\boldsymbol{x})}{\partial x^{k_1}\partial x^{\ell_1}} \frac{\partial\varrho(\boldsymbol{x})}{\partial x^{\ell_2}} \frac{\partial^2\varrho(\boldsymbol{x})}{\partial x^{i_2}\partial x^{k_2}} \frac{\partial a^1}{\partial x^{i_3}} \frac{\partial a^1}{\partial x^{k_3}} \frac{\partial a^1}{\partial x^{\ell_3}} \frac{\partial}{\partial x^{i_1}}.$$

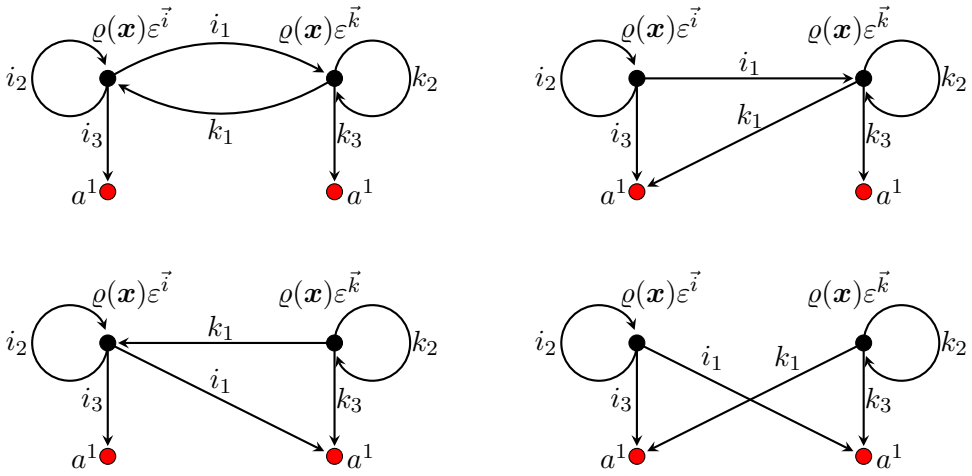
Let us now introduce some useful definitions relating two-dimensional micro-graphs with higher dimensional micro-graphs.

Definition 24 (d-Dimensional descendants)**.** The set of *d-dimensional descendants* $(\widehat{\Gamma})_d$ of a two-dimensional Kontsevich graph Γ is the collection of all the Nambu micro-graphs obtained from Γ by adding $d-2$ Casimir vertices at each Nambu-determinant Poisson structure and redirecting the two original outgoing edges at each Levi-Civita vertex via the Leibniz rule over all the vertices of the targeted Poisson structure(s).

Example 12. Let us consider the following two-dimensional micro-graph.

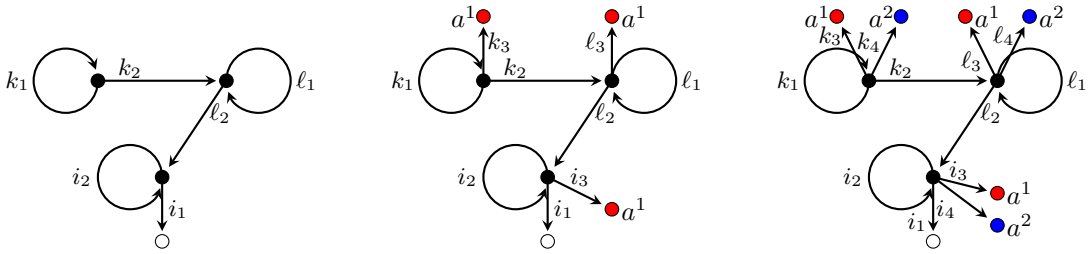


This graph has 4 three-dimensional descendants. The set $(\widehat{\Gamma})_{3D}$ contains exactly the following four graphs.



Definition 25 (Embedding)**.** The *embedding* of a micro-graph Γ_{dim} built from n Nambu-determinant Poisson structures into dimension $\dim + 1$ is the graph $\Gamma_{\text{dim}+1} = \text{emb}(\Gamma_{\text{dim}})$ such that to the Levi-Civita vertex of each Nambu-determinant Poisson structure, we add an extra Casimir vertex a^{d-1}. The original d outgoing edges of each Levi-Civita vertex keep their order, and the new edge is ordered last. The embedding can often be viewed as a specific type of descendant of a graph.

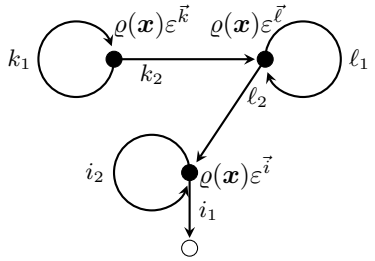
Example 13. Below is an example of a two-dimensional graph and its three- and four-dimensional embedding.

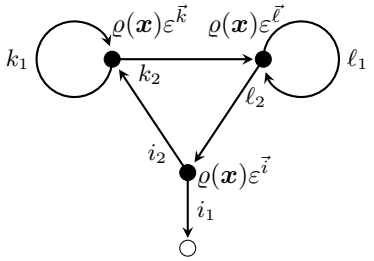


As it turns out, non-isomorphic (micro-)graphs may evaluate into the same multivector.

Definition 26. Two topologically non-isomorphic (micro-)graphs $\Gamma_1 \ncong \Gamma_2$ are called *synonyms* if $\phi(\Gamma_1) = c \cdot \phi(\Gamma_2)$ with $c \in \mathbb{R} \setminus \{0\}$, that is, the two (micro-)graphs provide the same multivector up to a nonzero constant.

Example 14. Consider the following two-dimensional micro-graphs. It is clear that they are topologically distinct.

$\Gamma_1 =$ 

$\Gamma_2 =$ 

Let us compute the corresponding vector fields over $\mathbb{R}^2_{\text{aff}}$, where we use $x^1 = x$ and $x^2 = y$. We see that for the first graph,

$$\begin{aligned}
\phi(\Gamma_1) = &\sum_{\vec{i},\vec{k},\vec{\ell}} \varepsilon^{\vec{i}}\varepsilon^{\vec{k}}\varepsilon^{\vec{\ell}} \frac{\partial^2\varrho(\boldsymbol{x})}{\partial x^{i_2}\partial x^{\ell_2}} \frac{\partial\varrho(\boldsymbol{x})}{\partial x^{k_1}} \frac{\partial^2\varrho(\boldsymbol{x})}{\partial x^{\ell_1}\partial x^{k_2}} \frac{\partial}{\partial x^{i_1}} \\
= \quad & \varepsilon^{xy}\varepsilon^{xy}\varepsilon^{xy} \frac{\partial^2\varrho(\boldsymbol{x})}{\partial y\partial y} \frac{\partial\varrho(\boldsymbol{x})}{\partial x} \frac{\varrho(\boldsymbol{x})}{\partial x\partial y} \frac{\partial}{\partial x} + \varepsilon^{yx}\varepsilon^{xy}\varepsilon^{xy} \frac{\partial^2\varrho(\boldsymbol{x})}{\partial x\partial y} \frac{\partial\varrho(\boldsymbol{x})}{\partial x} \frac{\varrho(\boldsymbol{x})}{\partial x\partial y} \frac{\partial}{\partial y} \\
& + \varepsilon^{xy}\varepsilon^{yx}\varepsilon^{xy} \frac{\partial^2\varrho(\boldsymbol{x})}{\partial y\partial y} \frac{\partial\varrho(\boldsymbol{x})}{\partial y} \frac{\varrho(\boldsymbol{x})}{\partial x\partial x} \frac{\partial}{\partial x} + \varepsilon^{xy}\varepsilon^{xy}\varepsilon^{yx} \frac{\partial^2\varrho(\boldsymbol{x})}{\partial y\partial x} \frac{\partial\varrho(\boldsymbol{x})}{\partial x} \frac{\varrho(\boldsymbol{x})}{\partial y\partial y} \frac{\partial}{\partial x} \\
& + \varepsilon^{yx}\varepsilon^{xy}\varepsilon^{xy} \frac{\partial^2\varrho(\boldsymbol{x})}{\partial x\partial y} \frac{\partial\varrho(\boldsymbol{x})}{\partial y} \frac{\varrho(\boldsymbol{x})}{\partial y\partial x} \frac{\partial}{\partial y} + \varepsilon^{yx}\varepsilon^{xy}\varepsilon^{yx} \frac{\partial^2\varrho(\boldsymbol{x})}{\partial x\partial x} \frac{\partial\varrho(\boldsymbol{x})}{\partial x} \frac{\varrho(\boldsymbol{x})}{\partial y\partial y} \frac{\partial}{\partial y} \\
& + \varepsilon^{xy}\varepsilon^{yx}\varepsilon^{yx} \frac{\partial^2\varrho(\boldsymbol{x})}{\partial y\partial x} \frac{\partial\varrho(\boldsymbol{x})}{\partial y} \frac{\varrho(\boldsymbol{x})}{\partial y\partial x} \frac{\partial}{\partial x} + \varepsilon^{yx}\varepsilon^{yx}\varepsilon^{yx} \frac{\partial^2\varrho(\boldsymbol{x})}{\partial x\partial x} \frac{\partial\varrho(\boldsymbol{x})}{\partial y} \frac{\varrho(\boldsymbol{x})}{\partial y\partial x} \frac{\partial}{\partial y} \\
= \quad & \left(\frac{\partial^2\varrho(\boldsymbol{x})}{\partial x\partial y} \frac{\partial^2\varrho(\boldsymbol{x})}{\partial x\partial y} \frac{\partial\varrho(\boldsymbol{x})}{\partial y} - \frac{\partial^2\varrho(\boldsymbol{x})}{\partial x\partial x} \frac{\partial^2\varrho(\boldsymbol{x})}{\partial y\partial y} \frac{\partial\varrho(\boldsymbol{x})}{\partial y} \right) \frac{\partial}{\partial x} \\
& + \left(\frac{\partial^2\varrho(\boldsymbol{x})}{\partial x\partial x} \frac{\partial^2\varrho(\boldsymbol{x})}{\partial y\partial y} \frac{\partial\varrho(\boldsymbol{x})}{\partial x} - \frac{\partial^2\varrho(\boldsymbol{x})}{\partial x\partial y} \frac{\partial^2\varrho(\boldsymbol{x})}{\partial x\partial y} \frac{\partial\varrho(\boldsymbol{x})}{\partial x} \right) \frac{\partial}{\partial y}.
\end{aligned}$$

A similar computation yields that also

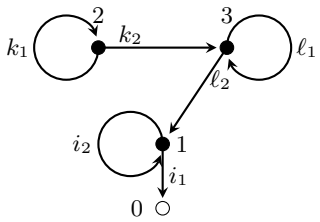
$$\begin{aligned}
\phi(\Gamma_2) = \quad & \left(\frac{\partial^2\varrho(\boldsymbol{x})}{\partial x\partial y} \frac{\partial^2\varrho(\boldsymbol{x})}{\partial x\partial y} \frac{\partial\varrho(\boldsymbol{x})}{\partial y} - \frac{\partial^2\varrho(\boldsymbol{x})}{\partial x\partial x} \frac{\partial^2\varrho(\boldsymbol{x})}{\partial y\partial y} \frac{\partial\varrho(\boldsymbol{x})}{\partial y} \right) \frac{\partial}{\partial x} \\
& + \left(\frac{\partial^2\varrho(\boldsymbol{x})}{\partial x\partial x} \frac{\partial^2\varrho(\boldsymbol{x})}{\partial y\partial y} \frac{\partial\varrho(\boldsymbol{x})}{\partial x} - \frac{\partial^2\varrho(\boldsymbol{x})}{\partial x\partial y} \frac{\partial^2\varrho(\boldsymbol{x})}{\partial x\partial y} \frac{\partial\varrho(\boldsymbol{x})}{\partial x} \right) \frac{\partial}{\partial y}.
\end{aligned}$$

Notation 4. To finish this section, let us briefly introduce *graph encodings*. While the graphs are already much easier to work with than the multivectors they evaluate into, we can create even shorter notation by considering encodings. For this to work properly, we need to state which vertices get which labels, and decide on the edge ordering. Practically, we will only be concerned with graphs on one sink and on no sinks, corresponding to vector fields and Hamiltonian functions respectively. In the case of graphs Γ on one sink, we always give the sink the label 0. Next, we label the Levi-Civita vertices by $1, 2, ..., v(\Gamma)$. When Γ is a three-dimensional graph, we label the corresponding a^1 Casimir vertices with

$v(\Gamma)+1, v(\Gamma)+2, ..., 2\cdot v(\Gamma)$. For four-dimensional graphs, we give the corresponding a^2 Casimir vertices the labels $2\cdot v(\Gamma)+1, 2\cdot v(\Gamma)+2, ..., 3\cdot v(\Gamma)$. In particular, the Levi-Civita vertex labeled by 1 has corresponding a^1 and a^2 Casimir vertices labeled by $v(\Gamma)+1$ and $2\cdot v(\Gamma)+1$ respectively.

For each of the Levi-Civita vertices, we now write down to which vertices (by their vertex label) the d outgoing edges go, respecting the order of the edges. This is a sequence of length $v(\Gamma)\cdot d$. We divide the edges outgoing from different Levi-Civita vertices by ; , while we divide edges belonging to the same Levi-Civita vertex by , .

Example 15. Let us consider an example. Take the graph from Example 13, and let us label the sink vertex by 0 and the Levi-Civita vertices by 1, 2, 3.



The corresponding encoding of this graph is now

$$[0,1;2,3;3,1].$$

The encodings of the three- and four-dimensional embeddings of this graph are respectively given by

$$[0,1,4;2,3,5;3,1,6], \qquad [0,1,4,7;2,3,5,8;3,1,6,9],$$

where we colored the a^1 Casimir vertex labels red and the a^2 Casimir vertex labels blue for convenience.

5. Deformation quantization

We can intuitively explain deformation theory as how much we can change a mathematical object without it losing its defining properties. In this chapter, we will introduce the rigorous theory of (infinitesimal) deformations.

Gerstenhaber introduced a rigorous theory of deformations for rings and algebras in 1964 [22]. This deformation theory shows parallels with the deformation theory of analytic structures, also known as Froelicher-Kodaira-Nijenhuis-Spencer theory.[21] In 1978, Lichnerowicz specified deformation theory further to deformations of Poisson structures [6]. In 1996 and 1997, Kontsevich introduced his graph complex **Gra** (see Section 4), and considered specific deformations of Poisson structures and associative algebras, namely the deformations obtained from cocycles in **Gra**. He showed that trivial cocycles in **Gra** corresponded to trivial deformations of the aforementioned structures. His assumption was that the nontrivial wheel-graph cocycles[22] would correspond to nontrivial deformations. Let us explain where infinitesimal deformations of Poisson brackets appear in Kontsevich's work.

5.1. From classical mechanics to quantum mechanics via deformation theory

In classical mechanics, the quantities of momentum and position commute, while in quantum mechanics, momentum and position do not commute. This noncommutativity is also the root of the *uncertainty principle*; while in classical mechanics momentum and position can both be observed at the same time, in quantum mechanics there will always be (some) uncertainty in what is observed. The above means that when physicists tried to move from a classical system to a quantum system, some commutative operations needed to become noncommutative. There are many ways of quantizing classical systems,[23] and deformation quantization uses the mathematical framework of deformation theory. The intuitive idea is the following. We start with a classical system, described via mathematical objects. We create a family of (quantum) systems, described by the *same* mathematical objects within a formal power series. This formal power series depends on a formal parameter[24] (say, $\hbar$). Moreover, we oblige that the family of systems reduces to the classical system we started with as the formal parameter approaches 0. In other words, the family must satisfy the *correspondence principle*. The above models the fact that many quantum systems can reduce to the same classical system. Schematically, we have a situation such as described

[21] The interested reader is referred to [23, 24, 25].

[22] In 2010, Thomas Willwacher showed the existence of infinitely many such nontrivial wheel-graph cocycles by relating them to the generators of the Grothendieck-Teichmüller Lie algebra [15].

[23] For example, other well-known quantizations methods are canonical quantization introduced by Dirac [26] and the path integral formulation by Feynman [27].

[24] In fact, we could let the family depend on more than one formal parameter, for example, when coupling multiple quantum systems [28].

below. Let us consider our classical system, depending on some mathematical object Q_0, with some defining property. Then, we take Q_n for $n \in \mathbb{N}$, which are all mathematical objects of the 'same type' as Q_0. Next, we consider the formal power series

$$Q_\hbar = Q_0 + \sum_{n=1}^{\infty} \hbar^n Q_n.$$

We require that at each value of $\hbar$, this formal sum has the same defining property as Q_0. Letting $\hbar \to 0$, we see that the above expression reduces to Q_0 again. When we consider deformations only up to order $o(\hbar)$ and not also the higher orders, we speak of *infinitesimal deformations*.

The prototypical example of a formal deformation is given by modelling a classical system with a commutative, associative product on a symplectic manifold M^{2d}. As we move to a quantum setting, multiplication loses the commutativity property, while keeping the associativity. The deformation is in this case of the associative product on M^{2d}. In other words, Q_0 is a commutative, associative product, and $Q_\hbar$ is an associative product. Note that the individual Q_n are *not* required to be associative. The product we create this way is called the *Moyal product* $\star$ (or, *star product*, or, *Weyl–Groenewold product*). In other words, for $f, g \in C^\infty(M^d)$, we have

$$f \star g = fg + \sum_{n=1}^{\infty} \hbar^n Q_n(f, g),$$

where the Q_n are bidifferential operators. Of course, these Moyal products are not unique. We create an equivalence between Moyal products in the following way. Consider some transformation sending $f \in C^\infty(M^{2d})$ to $f' = f + \sum_{n=1}^{\infty} \hbar^n D_n(f)$,[25] where D_i are differential operators. We can define another Moyal product $\star'$ via

$$f \star' g = (Id + \sum_{n=1}^{\infty} \hbar^n D_n)^{-1}(f' \star g').$$

If $\star$ and $\star'$ are related in this way, they are equivalent. We call the equivalence classes *gauge (equivalence) classes*, and we say that $\star'$ is in the *gauge class of* $\star$. A natural question that arises is if we can find ways to create 'new' star products.

In 1997, Kontsevich found a formula for the generalization of the Moyal product which works on arbitrary finite-dimensional Poisson manifolds $(M^d, \{\cdot,\cdot\}_\mathcal{P})$ rather than just symplectic manifolds [30]. Let $(A, \{\cdot,\cdot\}_\mathcal{P})$ be the Poisson algebra[26] of the Poisson manifold $(M^d, \{\cdot,\cdot\}_\mathcal{P})$. Then, the deformation quantization of this algebra is characterized by the following star

[25] The inverse of this mapping is well-defined, see [29].

[26] See Definition 30.

product $\star : A[[\hbar]] \times A[[\hbar]] \to A[[\hbar]]$[27]

$$a \star b = ab + \hbar\{a,b\}_{\mathcal{P}} + \sum_{i=2}^{\infty} \hbar^i B_i(a,b),$$

where the B_i are linear bidifferential operators. Now, we can deform this star product infinitesimally in the following way. We write

$$\mathcal{P} = \mathcal{P}_{\varepsilon=0} \to \dot{\mathcal{P}}_{\varepsilon=0} = \frac{d}{d\varepsilon}\mathcal{P}_\varepsilon = \mathcal{P}_{\varepsilon=0} + \varepsilon Q(\mathcal{P}) + o(\varepsilon),$$

where $Q(\mathcal{P})$ is a bivector, depending on the original Poisson bivector $\mathcal{P}$, called the *deformation term*[28] (in ε). The expression above is sometimes also called the *(infinitesimal) deformation of $\mathcal{P}$ by $Q(\mathcal{P})$*. The associated star product $\star_\varepsilon$ is now given by

$$a \star_\varepsilon b = ab + \hbar\{a,b\}_{\mathcal{P}_\varepsilon} + o(\hbar) = ab + \hbar\left[\{a,b\}_{\mathcal{P}_{\varepsilon=0}} + \varepsilon Q(\mathcal{P})(a,b) + o(\varepsilon)\right] + o(\hbar).$$

In his paper "Deformation quantization on Poisson manifolds" [16], Kontsevich showed that there exists a relation between the triviality of the 'new' star product $\star_\varepsilon$ and specific graphs in his graph complex. More precisely, Kontsevich showed that for a trivial cocycle γ in **Gra**,[29] the star product $\star_\varepsilon$ induced by the deformation term $Q_\gamma(\mathcal{P})$ is gauge equivalent to $\star$. With this in mind, the assumption was that for the nontrivial wheel-graph cocyles, $\star_\varepsilon$ *would* give a star product $\star_\varepsilon$ belonging to a different gauge class. However, this is actually not (always) the case! In all the examples tested so far, the nontrivial wheel-graph cocyles γ give trivial infinitesimal deformations. For γ_3, it has been shown that (infinitesimally) $Q_{\gamma_3}(\mathcal{P})$ is trivial, that is, there exists some vector field $\vec{X}$ such that $Q_{\gamma_3}(\mathcal{P}) = [\![\mathcal{P}, \vec{X}]\!]$, on $\mathbb{R}^2_{\text{aff}}$ for all Poisson bivectors [16, 31], and for the class of Nambu-determinant Poisson brackets, triviality has been established on $\mathbb{R}^3_{\text{aff}}$ [32, 21] and $\mathbb{R}^4_{\text{aff}}$ [3]. On $\mathbb{R}^2_{\text{aff}}$ and for all Poisson bivectors, triviality is also established for $Q_{\gamma_5}(\mathcal{P})$ [33]. Finally, there are some isolated examples that show the triviality of the deformation term for other classes[30] of Poisson brackets [19, 35, 33].

In fact, with knowing the trivializing vector field, one can explicitly construct the gauge transformation that produces the deformation of the star product, see [33, Chapter 9].

Some nice introductory texts related to deformation quantization are [36, 37].

---

[27] $A[[\hbar]]$ denotes the space of formal power series in $\hbar$ with coefficients in the algebra A. The rigourous definition can be found in Definition 27

[28] In the specific cases of $Q_{\gamma_3}(\mathcal{P})$ and $Q_{\gamma_5}(\mathcal{P})$, we also call these deformation terms the *tetrahedral flow* and the *pentagon (wheel) flow*, respectively.

[29] Recall that this means that $\mathsf{d}(\gamma) = 0$ and $\gamma = \mathsf{d}(\tilde{\gamma})$ for some $\tilde{\gamma}$.

[30] Specifically, the Poisson brackets checked in these isolated examples are R-matrix Poisson brackets and high polynomial degree Poisson brackets introduced by Vanhaecke in [34].

5.2. Deformations of associative algebras

We follow the theory of deformations as described by Gerstenhaber in [22]. We start with an easier example in the form of the deformation of associative algebras, and then move on to deformations of Poisson algebras. We will only discuss a small part of the theory that is of interest to us, the theory itself is (much) richer. The interested reader is invited to take a look at classical texts [22, 38, 39].

Let A be an associative algebra over some field $\mathbf{k}$. We denote the underlying vector space by V.

Definition 27 (Formal power series)**.** A *formal power series in one variable* t over the field $\mathbf{k}$ is a formal series

$$\sum_{n=1}^{\infty} a_n t^n = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + \dots ,$$

where t^n are formal powers of the variable t and $a_n \in \mathbf{k}$. Moreover, the space of formal power series over $\mathbf{k}$ forms a ring, denoted by $\mathbf{k}[[t]]$.

Remark 21. The formal power series ring $\mathbf{k}[[t]]$ can be seen as a generalization of the polynomial ring $\mathbf{k}[t]$, where elements are no longer obliged to have a finite number of terms.

In the same way that we extend the polynomial ring $\mathbf{k}[t]$ to its field of fractions, $\mathbf{k}(t)$, we also extend $\mathbf{k}[[t]]$ to its quotient power series field $\mathbf{k}((t))$. Let us write $K = \mathbf{k}((t))$, and consider the vector space $V_K = V \otimes_{\mathbf{k}} K$.[31] Note that any bilinear function $f\colon V \times V \to V$ can be extended by K-linearity to a bilinear function $F\colon V_K \times V_K \to V_K$.[32] A function F that is such an extension is said to be *defined over* $\mathbf{k}$.

Definition 28. (Formal deformation of an associative algebra) Consider an associative $\mathbf{k}$-algebra A, its power ring series $\mathbf{k}[[t]]$ and the associated quotient power series field $K = \mathbf{k}((t))$. A *formal deformation* of the algebra A is an algebra A_t whose underlying vector space is $V_K = V \otimes_{\mathbf{k}} K$. The multiplication F_t on A_t is associative and is expressible in the form

$$F_t(a,b) = F_0(a,b) + tF_1(a,b) + t^2F_2(a,b) + t^3F_3(a,b) + ..., \tag{4}$$

where each bilinear function $F_n\colon V_K \times V_K \to V_K$ is an extension of a bilinear function $f_n\colon V \times V \to V$, and f_0 is the original multiplication of the algebra A.

[31] Note that this vector space V_K is the vector space obtained from V by extending the coefficient domain from $\mathbf{k}$ to K [22].

[32] This extension works bilinearly by considering f on pairs of elements of the vector space V and collecting the correct amount of powers of t. As a simple example, we compute $F(a_0 + a_1 t, b_0 + b_1 t) = f(a_0, b_0) + t\left(f(a_1, b_0) + f(a_0, b_1)\right) + t^2 f(b_1, b_2)$.

Remark 22. We can see A_t as a generic element of a one-parameter family of deformations of the algebra A. Note that the choice of $F_t(a,b)$ is not (necessarily) unique, and that this choice leads to different algebras A_t, all (different) one parameter deformations of the algebra A.

Remark 23. The requirement that F_t be associative can be expressed in the following way. For each $a,b,c \in V_K$, and all $\nu \in \mathbb{N}_{\geq 0}$,

$$\sum_{\substack{\lambda+\mu=\nu,\\ \lambda,\mu\geq 0}} F_\lambda(F_\mu(a,b),c) - F_\lambda(a,F_\mu(b,c)) = 0.$$

Note that for $\nu = 0$, the above expression reduces to

$$F_0(F_0(a,b),c) - F_0(a,F_0(b,c)) = 0,$$

that is, it expresses that the multiplication on the original algebra A is associative as we assumed.

Remark 24. For an infinitesimal deformation, we only need that the required property (in this case, associativity) is satisfied up to order $o(t)$. In particular, in this context, we find that we require

$$F_0(F_1(a,b),c) - F_0(a,F_1(b,c)) + F_1(F_0(a,b),c) - F_1(a,F_0(b,c)) = 0,$$

or,

$$F_1(a,b)c - aF_1(b,c) + F_1(ab,c) - F_1(a,bc) = 0.$$

Note that with the above, we can define an infinitesimal deformation of A as follows.

Definition 29. (Infinitesimal deformation of an associative algebra) Consider some associative $\mathbf{k}$-algebra A, its power series ring $\mathbf{k}[[t]]$ and the associated quotient power series field $K = \mathbf{k}((t))$. An *infinitesimal deformation* of the algebra A is the function F_1 from Equation (4) seen as a function from $V \times V \to V$. The infinitesimal condition that needs to be satisfied is

$$F_1(a,b)c - aF_1(b,c) + F_1(ab,c) - F_1(a,bc) = 0,$$

for all elements $a,b,c \in V$.

Remark 25. In other words, for $a,b \in V$, instead of the product ab, we have a product defined by $F_t(a,b) = (F_0 + tF_1)(a,b) = ab + tF_1(a,b)$, which is a degree 1 polynomial in t.

5.3. Deformations of Poisson algebras

Let us consider a smooth Poisson manifold $(M^d, \{\cdot,\cdot\}_{\mathcal{P}})$.

Definition 30 (Poisson algebra)**.** The *Poisson algebra* corresponding to $(M^d, \{\cdot,\cdot\}_{\mathcal{P}})$ is the algebra of smooth functions $C^\infty(M^d, \{\cdot,\cdot\}_{\mathcal{P}})$ over $\mathbb{R}$. We denote the Poisson algebra by $(A, \{\cdot,\cdot\}_{\mathcal{P}})$.

Remark 26. The underlying vector space of the Poisson algebra is $C^\infty(M^d, \{\cdot,\cdot\}_\mathcal{P})$.

Similar to Definition 29, we can define infinitesimal deformations of a Poisson algebra.

Definition 31 (Infinitesimal deformation of a Poisson algebra)**.** Let $(A, \{\cdot,\cdot\}_\mathcal{P})$ be a Poisson algebra of some affine finite-dimensional Poisson manifold $(M^d, \{\cdot,\cdot\}_\mathcal{P})$. An *infinitesimal deformation* of the Poisson algebra is a function Q, seen as a function from $C^\infty(M^d) \times C^\infty(M^d) \to C^\infty(M^d)$. This function Q needs to satisfy

- $\mathbb{R}$-bilinearity,
- Skew-symmetry: $Q(a,b) = -Q(b,a)$,
- Leibniz rule: $Q(a,bc) = bQ(a,c) + Q(a,b)c$,

and the infinitesimal condition that needs to be satisfied is the Jacobi identity:

$$0 = \{Q(a,b),c\}_\mathcal{P} + \{Q(b,c),a\}_\mathcal{P} + \{Q(c,a),b\}_\mathcal{P} + Q(\{a,b\}_\mathcal{P},c) + Q(\{b,c\}_\mathcal{P},a) + Q(\{c,a\}_\mathcal{P},b).$$

Remark 27. Note that for the formal deformation of associative algebras, we required that at every order of t, associativity was preserved. In the case of Poisson algebras, we want that at every order of t, the properties of Poisson structures are preserved. These are precisely the $\mathbb{R}$-bilinearity, skew-symmetry, the Leibniz rule, and the Jacobi identity. For the Jacobi identity, at every order of t, we hence want that

$$\sum_{\substack{\lambda+\mu=\nu,\\ \lambda,\mu\geq 0}} F_\lambda(F_\mu(a,b),c) + F_\lambda(F_\mu(b,c),a) + F_\lambda(F_\mu(c,a),b) = 0. \tag{5}$$

Note that F_0 is just the Poisson bracket, while F_1 is the function Q. At $O(1)$, we see that Equation 5 just becomes the Jacobi identity of the Poisson bracket

$$\{\{a,b\}_\mathcal{P},c\}_\mathcal{P} + \{\{b,c\}_\mathcal{P},a\}_\mathcal{P} + \{\{c,a\}_\mathcal{P},b\}_\mathcal{P} = 0.$$

At $O(t)$, Equation 5 becomes exactly the infinitesimal condition in Definition 31.

Remark 28. Note that the intuitive way of thinking about deformations as 'changing the mathematical object without it losing its defining properties' can be seen very clearly here. Recall from Definition 3 that we define a Poisson structure $\mathcal{P}$ to be a bivector such that

$$[\![\mathcal{P},\mathcal{P}]\!] = 0.$$

When we infinitesimally deform our Poisson structure to $\mathcal{P} + \epsilon Q$, the requirements that this is still Poisson now comes down to

$$[\![\mathcal{P} + \epsilon Q, \mathcal{P} + \epsilon Q]\!] = 0.$$

At order $O(1)$, this above just tells us that $[\![\mathcal{P},\mathcal{P}]\!] = 0$, while at order $O(t)$, we see that we need

$$[\![\mathcal{P},Q]\!] = 0,$$

which is precisely what the infinitesimal condition in Definition 31 expresses [40].

6. Other results

This section contains an inventory of results that are not (or, not in full detail) presented in the three papers [2], [3] and [4]. We start with two subsections discussing the trivializing vector fields for $Q_{\gamma_5}(\mathcal{P})$ on $\mathbb{R}^2_{\text{aff}}$ (Section 6.1) and $\mathbb{R}^3_{\text{aff}}$ (Section 6.2). This is followed by an observation about the form of the sunflower graphs evaluating into the trivializing vector field $\vec{X}^{2D}_{\gamma_3}$ and a short discussion on whether this solution is truly special (Section 6.3). After this, we state a conjecture about the preservation of linear relations under embedding micro-graphs in higher dimensions (Section 6.4). We finish with a final observation relating to $\vec{X}^{2D}_{\gamma_3}$, which leads to a conjecture on a general strategy to find a vector field trivializing $Q_{\gamma_3}(\mathcal{P})$ for all finite dimensions for Nambu-determinant Poisson structures (Section 6.5).

6.1. The trivializing vector field $\vec{X}^{\gamma_5}_{2D}$

As detailed in [4] for $\vec{X}^{\gamma_3}_{2D}$, we can have multiple combinations of non-isomorphic graphs that evaluate into this trivializing vector field due to the appearance of synonyms. In [33, Chapter 8] a strategy for finding the trivializing vector field $\vec{X}^{\gamma_5}_{2D}$ is outlined, together with some graph representation. Let us detail two other graph representations of $\vec{X}^{\gamma_5}_{2D}$. The first graph representation of the trivializing vector field is achieved by 'brute force'. We created all the graphs on 5 vertices and 1 sink, with exactly two outgoing edges at each of the 5 vertices and one ingoing edge at the sink, see Appendix B.[33]

Lemma 3. There are 2225 non-isomorphic Kontsevich graphs on 5 vertices and one sink.

Proof. See Appendix B.[33] □

Claim 4. *The images of the 2225 non-isomorphic Kontsevich graphs of Lemma 3 under the evaluation ϕ from graphs to multivectors satisfy 2203 linear relations.*

Proof. See Appendix B.[33] □

Proposition 5. The trivializing vector field $\vec{X}^{\gamma_5}_{2D}$ for the pentagon wheel flow of Poisson bivectors over $\mathbb{R}^2$ is given by the following linear combination of 8 graphs evaluated into vector fields

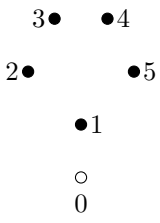
$$\begin{aligned}\vec{X}^{\gamma_5}_{2D} = &-10\cdot\phi(\Delta^{2D}_{13}) - 2\cdot\phi(\Delta^{2D}_{14}) - 2\cdot\phi(\Delta^{2D}_{15}) + 2\cdot\phi(\Delta^{2D}_{17}) + 4\cdot\phi(\Delta^{2D}_{18}) + 8\cdot\phi(\Delta^{2D}_{20})\\ &+ 4\cdot\phi(\Delta^{2D}_{21}) - 12\cdot\phi(\Delta^{2D}_{22}).\end{aligned}$$

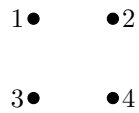
Proof. See Appendix B.[33] □

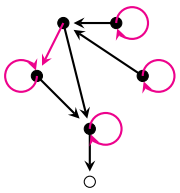
[33] Under the title: 'Finding $\vec{X}^{\gamma_5}_{2D}$ (Brute-force method)'.

The graphs are given below.

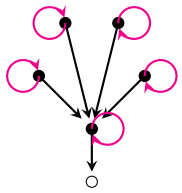
Notation 5. Rather than denoting the order of the edges by indices i_1, i_2,... , we denote the edge ordered first with the color black, while we denote the edge ordered second with the color magenta to make the pictures less crowded. The vertices are fixed in the plane with the following conventions. The left graph is for the graphs evaluating into vector fields, denoted by Δ_i^d, whereas the right graph is for graphs evaluating into Hamiltonians, denoted by $\Delta_{\mathcal{H}_i}^d$.



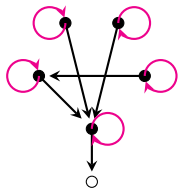


$\Delta_{13}^{2D} =$ 

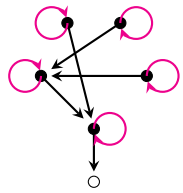
$[0,1;1,2;1,2;3,4;3,5]$

$\Delta_{14}^{2D} =$ 

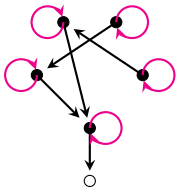
$[0,1;1,2;1,3;1,4;1,5]$

$\Delta_{15}^{2D} =$ 

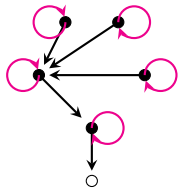
$[0,1;1,2;1,3;1,4;2,5]$

$\Delta_{17}^{2D} =$ 

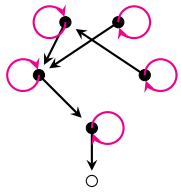
$[0,1;1,2;1,3;2,4;2,5]$

$\Delta_{18}^{2D} =$ 

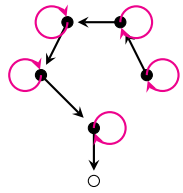
$[0,1;1,2;1,3;2,4;3,5]$

$\Delta_{20}^{2D} =$ 

$[0,1;1,2;2,3;2,4;2,5]$

$\Delta_{21}^{2D} =$ 

$[0,1;1,2;2,3;2,4;3,5]$

$\Delta_{22}^{2D} =$ 

$[0,1;1,2;2,3;3,4;4,5]$

Remark 29. These graphs to represent the trivializing vector field $\vec{X}_{2D}^{\gamma_5}$ are all of a very particular shape. This has to do with how the graphs (or rather, the graph encodings) are generated.[34] In other words, with the many synonyms among the 2225 non-isomorphic graphs on 5 vertices and 1 sink, there is a large bias, coming from the order in which

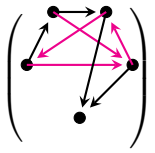
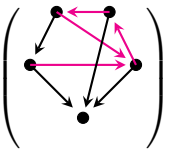
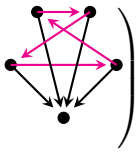
[34]Looking at the 8 graphs just above this, we see this very clearly. The very first graph Δ_1^{2D} is given by $[0,1;1,2;1,2;1,2;1,2]$, and the rest of the graphs are generated with increasing target vertices. See also Appendix B, under the title: 'Finding $\vec{X}_{2D}^{\gamma_5}$ (Brute-force method)'.

the 2225 graphs are generated, in how the 22 graphs are chosen to represent the linearly independent vector fields.

The second approach of finding graphs to represent the trivializing vector field is explained in [33, Chapter 8]. The method comes down to finding a function H such that the vector field[35] satisfies

$$\vec{X}_{2D}^{\gamma_5} = \frac{\partial H}{\partial y}\frac{\partial}{\partial x} - \frac{\partial H}{\partial x}\frac{\partial}{\partial y}.$$

In other words, we want to write $\vec{X}_{2D}^{\gamma_5}$ as a Hamiltonian[36] vector field. Note that this Hamiltonian H is on 5 copies of the Poisson structure. The next step is to find a linear combination of graphs on 5 vertices and no sink that evaluates into this Hamiltonian. Specifically, we create graphs on 5 vertices, with 4 vertices having 2 outgoing edges, and 1 vertex having no outgoing edges.[37] There are exactly 54 non-isomorphic graphs satisfying this. The graph representation of this Hamiltonian can be found below.

$H = -6 \cdot \phi$  $+ 2 \cdot \phi$  $- 2 \cdot \phi$  (6)

From these graphs, we can find a graph representation of $\vec{X}_{2D}^{\gamma_5}$ by adding two outgoing edges at the vertex with no outgoing edges. The first of these two edges will go to a (also newly added) sink, while the second edge will be redirected to each of the vertices in the graph via the Leibniz rule. Thus, in total, this representation is over 15 graphs.

Theorem 6. *The trivializing vector field $\vec{X}_{\gamma_5}^{2D}$ for the pentagon wheel flow of Poisson bivectors over $\mathbb{R}^2$ is given by the following linear combination of* 15 *graphs evaluated into vector fields*

$$\begin{aligned}\vec{X}_{2D}^{\gamma_5} = &-6\cdot\phi(\tilde{\Delta}_1^{2D}) - 6\cdot\phi(\tilde{\Delta}_2^{2D}) - 6\cdot\phi(\tilde{\Delta}_3^{2D}) - 6\cdot\phi(\tilde{\Delta}_4^{2D}) - 6\cdot\phi(\tilde{\Delta}_5^{2D})\\ &+2\cdot\phi(\tilde{\Delta}_6^{2D}) + 2\cdot\phi(\tilde{\Delta}_7^{2D}) + 2\cdot\phi(\tilde{\Delta}_8^{2D}) + 2\cdot\phi(\tilde{\Delta}_9^{2D}) + 2\cdot\phi(\tilde{\Delta}_{10}^{2D})\\ &-2\cdot\phi(\tilde{\Delta}_{11}^{2D}) - 2\cdot\phi(\tilde{\Delta}_{12}^{2D}) - 2\cdot\phi(\tilde{\Delta}_{13}^{2D}) - 2\cdot\phi(\tilde{\Delta}_{14}^{2D}) - 2\cdot\phi(\tilde{\Delta}_{15}^{2D}).\end{aligned}$$

Proof. See Appendix B.[38] □

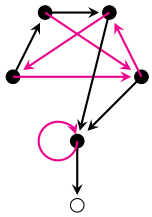
[35] Recall that in Section 2 we stated that we were looking for trivializing vector fields evaluated from graphs. Here, we first find the vector field, and then consider graphs representing it. The working idea is that the graphs found for the function H this way are 'special' in some sense, as will be discussed later.

[36] Here we mean Hamiltonian with respect to the standard symplectic structure, not Hamiltonian as defined in Definition 8.

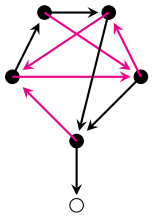
[37] This is slightly different compared to [33]. There, the graphs are created *just* with a maximum of two outgoing edges, which leads to more graphs for the representation of the trivializing vector field.

[38] Under the title: 'Finding $\vec{X}_{2D}^{\gamma_5}$ (Hamiltonian method)'.

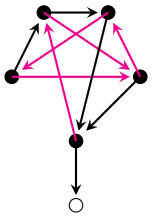
The graphs are given below.

$\tilde{\Delta}_1^{2D} =$ 

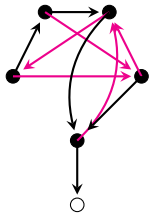
$[0,1;3,5;4,5;1,2;1,4]$

$\tilde{\Delta}_2^{2D} =$ 

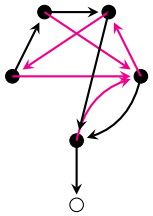
$[0,2;3,5;4,5;1,2;1,4]$

$\tilde{\Delta}_3^{2D} =$ 

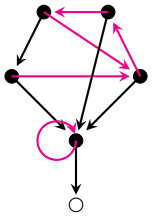
$[0,3;3,5;4,5;1,2;1,4]$

$\tilde{\Delta}_4^{2D} =$ 

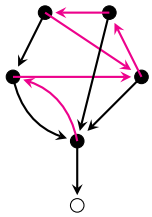
$[0,4;3,5;4,5;1,2;1,4]$

$\tilde{\Delta}_5^{2D} =$ 

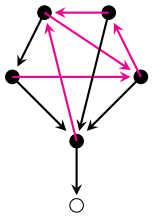
$[0,5;3,5;4,5;1,2;1,4]$

$\tilde{\Delta}_6^{2D} =$ 

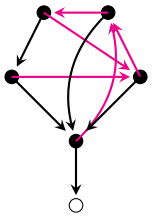
$[0,1;1,5;2,5;1,3;1,4]$

$\tilde{\Delta}_7^{2D} =$ 

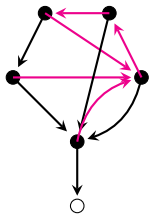
$[0,2;1,5;2,5;1,3;1,4]$

$\tilde{\Delta}_8^{2D} =$ 

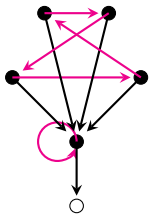
$[0,3;1,5;2,5;1,3;1,4]$

$\tilde{\Delta}_9^{2D} =$ 

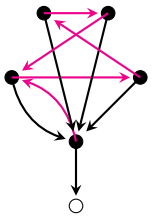
$[0,4;1,5;2,5;1,3;1,4]$

$\tilde{\Delta}_{10}^{2D} =$ 

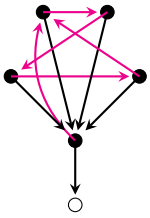
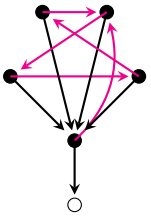
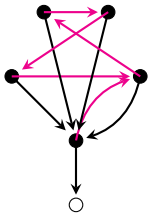
$[0,5;1,5;2,5;1,3;1,4]$

$\tilde{\Delta}_{11}^{2D} =$ 

$[0,1;1,5;1,4;1,2;1,3]$

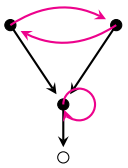
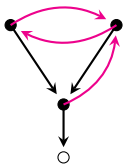
$\tilde{\Delta}_{12}^{2D} =$ 

$[0,2;1,5;1,4;1,2;1,3]$

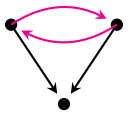
$\tilde{\Delta}^{2D}_{13} =$  $\tilde{\Delta}^{2D}_{14} =$  $\tilde{\Delta}^{2D}_{15} =$ 

$[0,3;1,5;1,4;1,2;1,3]$ $[0,4;1,5;1,4;1,2;1,3]$ $[0,5;1,5;1,4;1,2;1,3]$

Example 16. The graph representation found with the 'Hamiltonian' method for $\vec{X}^{\gamma_3}_{2D}$ is exactly the *sunflower solution*

$\vec{X}^{\gamma_3}_{2D} = 1 \cdot \phi\Big($  $\Big) + 2 \cdot \phi\Big($  $\Big)$

obtained from the graph



by letting one new edge go to the new sink, and directing the other edge over all the 3 original vertices. The trivializing vector fields $\vec{X}^{\gamma_3}_{3D}$ and $\vec{X}^{\gamma_3}_{4D}$ can be obtained by just considering descendants of this sunflower solution. One idea is that this same strategy might work for $\vec{X}^{\gamma_5}_{3D}$ as well, see Section 6.2.

Now that we have found graph representations of the trivializing vector field $\vec{X}^{\gamma_5}_{2D}$, we can again investigate whether or not this trivializing vector field is unique modulo Hamiltonian vector fields.

Proposition 7. There are 8 linearly independent vector fields $\vec{Z}^{2D}_1$, $\vec{Z}^{2D}_2$, $\vec{Z}^{2D}_3$, $\vec{Z}^{2D}_4$, $\vec{Z}^{2D}_5$, $\vec{Z}^{2D}_6$, $\vec{Z}^{2D}_7$ and $\vec{Z}^{2D}_8$ that span the solution space of the homogeneous equation,

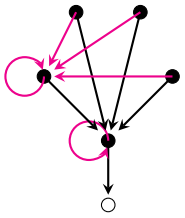
$$0 = [\![P, \vec{Z}]\!],$$

when restricting to vector fields evaluated from Kontsevich graphs on 5 vertices and 1 sink. Explicitly these vector fields are given by

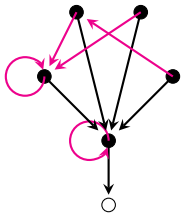
$$\begin{aligned}
\vec{Z}_1^{2D} &= 1\cdot\phi(\Delta_1^{2D}) - 2\cdot\phi(\Delta_3^{2D}),\\
\vec{Z}_2^{2D} &= 1\cdot\phi(\Delta_2^{2D}) - 1\cdot\phi(\Delta_3^{2D}) + 1\cdot\phi(\Delta_9^{2D}) + 1\cdot\phi(\Delta_{10}^{2D}) + 1\cdot\phi(\Delta_{11}^{2D}) - 3\cdot\phi(\Delta_{16}^{2D}),\\
\vec{Z}_3^{2D} &= 1\cdot\phi(\Delta_4^{2D}) + 2\phi(\Delta_9^{2D}) + 2\cdot\phi(\Delta_{11}^{2D}) - 1\cdot\phi(\Delta_{16}^{2D}) - 1\cdot\phi(\Delta_{19}^{2D}),\\
\vec{Z}_4^{2D} &= 1\cdot\phi(\Delta_5^{2D}) + 1\cdot\phi(\Delta_{10}^{2D}) - 2\cdot\phi(\Delta_{19}^{2D}),\\
\vec{Z}_5^{2D} &= 1\cdot\phi(\Delta_6^{2D}) - 1\cdot\phi(\Delta_{16}^{2D}),\\
\vec{Z}_6^{2D} &= 1\cdot\phi(\Delta_7^{2D}) + 1\cdot\phi(\Delta_{11}^{2D}) - 2\cdot\phi(\Delta_{16}^{2D}),\\
\vec{Z}_7^{2D} &= 1\cdot\phi(\Delta_8^{2D}) + 2\cdot\phi(\Delta_{11}^{2D}) - 1\cdot\phi(\Delta_{16}^{2D}),\\
\vec{Z}_8^{2D} &= 1\cdot\phi(\Delta_{12}^{2D}).
\end{aligned}$$

Proof. The proof can be found as (output of the) code in Appendix B.[39] □

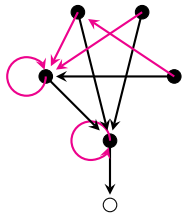
The graphs are given below.

$\Delta_1^{2D} =$ 

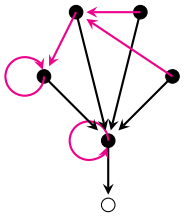
$[0,1;1,2;1,2;1,2;1,2]$

$\Delta_2^{2D} =$ 

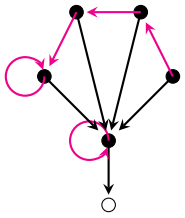
$[0,1;1,2;1,2;1,2;1,3]$

$\Delta_3^{2D} =$ 

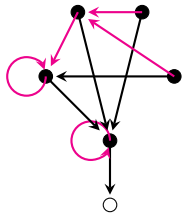
$[0,1;1,2;1,2;1,2;2,3]$

$\Delta_4^{2D} =$ 

$[0,1;1,2;1,2;1,3;1,3]$

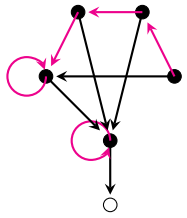
$\Delta_5^{2D} =$ 

$[0,1;1,2;1,2;1,3;1,4]$

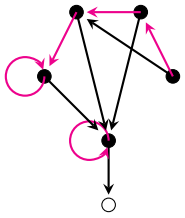
$\Delta_6^{2D} =$ 

$[0,1;1,2;1,2;1,3;2,3]$

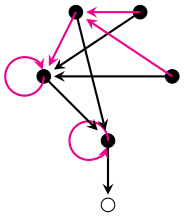
[39] Under the title: 'Finding $\vec{X}_{2D}^{\gamma_5}$ (Brute-force method)'.

$\Delta_7^{2D} =$ 

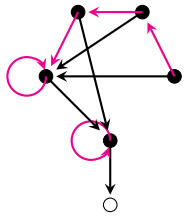
$[0,1;1,2;1,2;1,3;2,4]$

$\Delta_8^{2D} =$ 

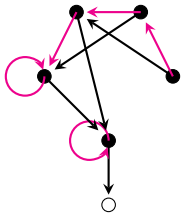
$[0,1;1,2;1,2;1,3;3,4]$

$\Delta_9^{2D} =$ 

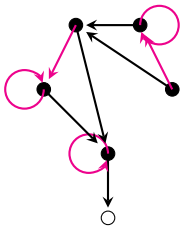
$[0,1;1,2;1,2;2,3;2,3]$

$\Delta_{10}^{2D} =$ 

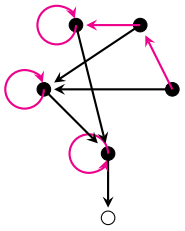
$[0,1;1,2;1,2;2,3;2,4]$

$\Delta_{11}^{2D} =$ 

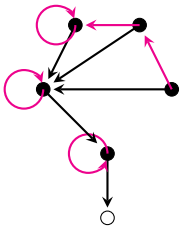
$[0,1;1,2;1,2;2,3;3,4]$

$\Delta_{12}^{2D} =$ 

$[0,1;1,2;1,2;3,4;3,4]$

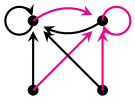
$\Delta_{16}^{2D} =$ 

$[0,1;1,2;1,3;2,3;2,4]$

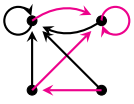
$\Delta_{19}^{2D} =$ 

$[0,1;1,2;2,3;2,3;2,4]$

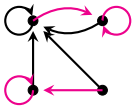
Lemma 8. There are 66 non-isomorphic Kontsevich graphs built on 4 vertices and no sink. Evaluating these graphs into Hamiltonian functions, we find 58 linear relations between these 66 functions. The following 8 graphs evaluate into linearly independent Hamiltonians.

$\Delta_{\mathcal{H}_1}^{2D} =$ 

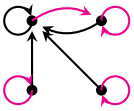
$[1,2;1,2;1,2;1,2]$

$\Delta_{\mathcal{H}_2}^{2D} =$ 

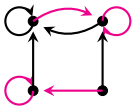
$[1,2;1,2;1,2;1,3]$

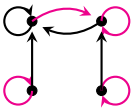
$\Delta_{\mathcal{H}_3}^{2D} =$ 

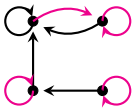
$[1,2;1,2;1,3;1,3]$

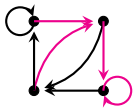
$\Delta_{\mathcal{H}_4}^{2D} =$ 

$[1,2;1,2;1,3;1,4]$

$\Delta^{2D}_{\mathcal{H}_5} =$  $[1,2;1,2;1,3;2,3]$

$\Delta^{2D}_{\mathcal{H}_6} =$  $[1,2;1,2;1,3;2,4]$

$\Delta^{2D}_{\mathcal{H}_7} =$  $[1,2;1,2;1,3;3,4]$

$\Delta^{2D}_{\mathcal{H}_8} =$  $[1,2;3,4;1,2;3,4]$

Proof. The proof can be found as (output of the) code in Appendix B.[40] □

Remark 30. The Hamiltonian $\Delta^{2D}_{\mathcal{H}_8}$ generated by the code is actually not the one we have written here, but rather the Hamiltonian encoded by $[1,2;1,2;3,4;3,4]$. This Hamiltonian is not connected, while we are only considering connected graphs. However, this is not an issue as there are many (connected) synonyms for this graph, one of which is the graph encoded by $[1,2;3,4;1,2;3,4]$.

Notation 6. We write $\mathcal{H}^{2D}_i = \phi(\Delta^{2D}_{\mathcal{H}_i})$.

Recall that when we restrict the cochain complex $(\star)$ to multivectors produced from graphs on affine finite-dimensional manifolds, we get a subcochain complex for Poisson cohomology [16],

$$0 \longrightarrow \mathbf{k} \hookrightarrow C^\infty_{\mathrm{gra}}(M^d_{\mathrm{aff}}) \xrightarrow{(\mathrm{d}_{\mathcal{P}})_1} \mathfrak{X}_{\mathrm{gra}}(M^d_{\mathrm{aff}}) \xrightarrow{(\mathrm{d}_{\mathcal{P}})_2} \mathfrak{X}^2_{\mathrm{gra}}(M^d_{\mathrm{aff}}) \xrightarrow{(\mathrm{d}_{\mathcal{P}})_3} \dots \xrightarrow{(\mathrm{d}_{\mathcal{P}})_d} \mathfrak{X}^d_{\mathrm{gra}}(M^d_{\mathrm{aff}}) \xrightarrow{(\mathrm{d}_{\mathcal{P}})_{d+1}} 0. \quad (*)$$

Theorem 9. *On $\mathbb{R}^2$, let $\mathcal{P} = \varrho \frac{\partial}{\partial x} \wedge \frac{\partial}{\partial y}$ be a (possibly degenerate) Poisson bivector. Consider the complex $(*)$ restricted to Hamiltonians on 4 copies of $\mathcal{P}$, vector fields on 5 copies of $\mathcal{P}$ and bivectors on 6 copies of $\mathcal{P}$. We establish that the corresponding homogeneous part of the Poisson-Lichnerowicz cohomology $H^1_{\mathrm{gra}}(\mathbb{R}^2_{\mathrm{aff}})$ is trivial.*

Proof. We write the vector fields $\vec{Z}^{2D}_i$ of Proposition 7 in terms of the Hamiltonian vector fields $\mathcal{H}^{2D}_1$, $\mathcal{H}^{2D}_2$, $\mathcal{H}^{2D}_3$, $\mathcal{H}^{2D}_4$, $\mathcal{H}^{2D}_5$, $\mathcal{H}^{2D}_6$, $\mathcal{H}^{2D}_7$ and $\mathcal{H}^{2D}_8$. Explicitly, we compute (see Appendix B,[41]

$$\begin{aligned}
\vec{Z}^{2D}_1 &= \tfrac{1}{2} \cdot \mathrm{d}_{\mathcal{P}}(\mathcal{H}^{2D}_1),\\
\vec{Z}^{2D}_2 &= 1 \cdot \mathrm{d}_{\mathcal{P}}(\mathcal{H}^{2D}_2) + \tfrac{1}{2} \cdot \mathrm{d}_{\mathcal{P}}(\mathcal{H}^{2D}_5) + \tfrac{1}{2} \cdot \mathrm{d}_{\mathcal{P}}(\mathcal{H}^{2D}_6),\\
\vec{Z}^{2D}_3 &= 1 \cdot \mathrm{d}_{\mathcal{P}}(\mathcal{H}^{2D}_3) + 1 \cdot \mathrm{d}_{\mathcal{P}}(\mathcal{H}^{2D}_7) + \tfrac{1}{8} \cdot \mathrm{d}_{\mathcal{P}}(\mathcal{H}^{2D}_8),\\
\vec{Z}^{2D}_4 &= 1 \cdot \mathrm{d}_{\mathcal{P}}(\mathcal{H}^{2D}_4),\\
\vec{Z}^{2D}_5 &= \tfrac{1}{2} \cdot \mathrm{d}_{\mathcal{P}}(\mathcal{H}^{2D}_5),\\
\vec{Z}^{2D}_6 &= \tfrac{1}{2} \cdot \mathrm{d}_{\mathcal{P}}(\mathcal{H}^{2D}_6),
\end{aligned}$$

[40] Under the title: 'Finding $\vec{X}^{\gamma_5}_{2D}$ (Brute-force method)'.
[41] Under the title: 'Finding $\vec{X}^{\gamma_5}_{2D}$ (Brute force method)'

$$\vec{Z}_7^{2D} = 1 \cdot \mathrm{d}_{\mathcal{P}}(\mathcal{H}_7^{2D}) + \tfrac{1}{8} \cdot \mathrm{d}_{\mathcal{P}}(\mathcal{H}_8^{2D}),$$
$$\vec{Z}_8^{2D} = \tfrac{1}{4} \cdot \mathrm{d}_{\mathcal{P}}(\mathcal{H}_8^{2D}).$$

□

Corollary 10. The trivializing vector field $\vec{X}_{2D}^{\gamma_5}$ of Theorem 6 is unique modulo Hamiltonian vector fields.

Remark 31. Recall from Definition 10 that the symplectic leaves are precisely the equivalence classes generated by the Hamiltonian vector fields. Moreover, we can add arbitrary Hamiltonian vector fields to the trivializing vector field. This means that the trivializing vector field tells us which symplectic leaves the flow can reach, and that we can move freely on these specific symplectic leaves.

6.2. The trivializing vector field $\vec{X}_{3D}^{\gamma_5}$

The next step is to see if we can find a trivializing vector field $\vec{X}_{3D}^{\gamma_5}$ on $\mathbb{R}^3_{\text{aff}}$. For γ_3, we could find a trivializing vector field over $\mathbb{R}^3_{\text{aff}}$ over the descendants of 5 out of 28 pairs of graphs representing the trivializing vector field $\vec{X}_{2D}^{\gamma_3}$, see [4, Table 1]. Moreover, for $\vec{X}_{4D}^{\gamma_3}$ we could find the trivializing vector field over the descendants of only 2 of the 28 pairs, see [4, Table 2]. One of the pairs over which we can find both trivializing vector field $\vec{X}_{3D}^{\gamma_3}$ and $\vec{X}_{3D}^{\gamma_4}$ is exactly the sunflower solution mentioned in Example 16. An immediate idea is to take the graphs of the trivializing vector fields of Proposition 5 and Theorem 6, look at their three-dimensional descendants, and try to find a trivializing vector over these. Note that we are missing a (possibly key) detail here. For γ_3, we had exactly one graph on 3 vertices where 2 vertices had exactly 2 outgoing edges, and we created our sunflower solution exactly from this single graph. Now, in contrast, for γ_5 we have 54 non-isomorphic graphs on 5 vertices with 4 vertices having exactly two outgoing edges. For the solution of Theorem 6, we 'only' use three of these 54 graphs, leading to the 15 graphs in the representation of Theorem 6. Then, we can look at the thee-dimensional descendants of these 15 graphs. But who is to say that the remaining 51 graphs that we found should not also be taken into consideration?

We checked whether we could find a trivializing vector field in the following three situations:

- Taking the three-dimensional descendants of the graphs appearing in Proposition 5;
- Taking the three-dimensional descendants of the graphs appearing in Theorem 6;
- Taking the three-dimensional descendants appearing from *all* 54 non-isomorphic graphs on 5 vertices with 4 vertices having exactly 2 outgoing edges (and adding two more edges and a sink as described in Section 6.1).

As it turned out, there was no trivializing vector field for any of the three situations described above. However, while writing this thesis and preparing nice, commented versions of the

code used, it became clear that there was a mistake in the code for finding the trivializing vector field $\vec{X}_{3D}^{\gamma_5}$. This mistake means that we would always (falsely) get an error as the system tries to find a trivializing vector field over the micro-graphs. In particular, it means that the corrected code (see Appendix B [42]) should be rerun to check for a trivializing vector field. All of the three scripts require at least 64 GB of RAM, and likely more.

Remark 32. After the supposed failing of finding a trivializing vector field over the relevant descendants, we also tried the 'brute force' method here (see Appendix B[43]), but unfortunately, the problem is very large. Creating all the 3-dimensional Nambu micro-graphs on 5 structures leads to a shocking 5811346 non-isomorphic graphs. With using University of Groningen's high performance computing cluster Hábrók, the code was canceled after a time limit of 5 days with using 898 GB RAM (up until the point it was canceled). This cancellation happened during evaluating the graphs into formulas. The remainder of the code requires many more (long and memory-heavy) computations. At the time of writing this thesis, the code *is* running on Hábrók with 4 TB of memory, 80 CPU cores and with a time limit of 10 days, but it will not finish before this thesis has been handed in.

6.3. The form of Γ_{11}^{2D} and Γ_{12}^{2D}.

There is always the possibility that there is no actual mechanism that dictates that $Q_d^{\gamma_3}$ must be trivializable, or that the sunflower solution is 'special'. In particular, note from Appendix C that over Γ_{11}^{2D} and Γ_{12}^{2D} we can create the largest amount of non-isomorphic descendants, as they contain the fewest loops[44] out of all the two-dimensional graphs. There is a possibility that as we go to higher dimensions, we will need vector fields coming from non-sunflower graphs, but that for the lower dimensional examples that we can compute, the topological differences in the graphs are not yet felt by the vector fields generated.

6.4. Preservation of linear relations under embeddings

Let us consider some linear combination of d-dimensional Nambu micro-graphs, that under the evaluation to multivectors satisfies

$$\sum_i a_i \phi(\Gamma_i) = 0,$$

where $a_i \in \mathbb{R}$. Then, we can consider the $(d+1)$-dimensional embeddings of these micro-graphs, $\mathrm{emb}(\Gamma_i)$. In all examples tried so far, we see that the linear relation over these

[42] Under the titles: 'Trivializing Q_{γ_5} in 3D (Descendants of the brute force 2D solution)', 'Trivializing Q_{γ_5} in 3D (Descendants of the 2D solution of the Hamiltonian method)' and 'Trivializing Q_{γ_5} in 3D (Descendants of all the 2D graphs Hamiltonian method)'.

[43] Under the title: 'Trivializing $Q_{\gamma_5}^{3D}$: The brute-force method'.

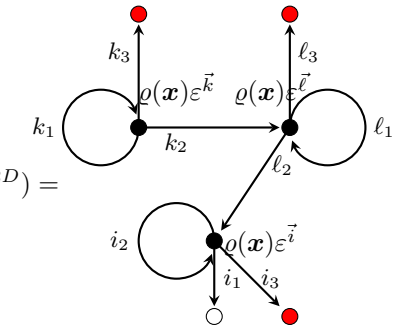
[44] Note that we cannot redirect a loop to a Casimir when we consider descendants, as this would lead to a double edge to the Casimir.

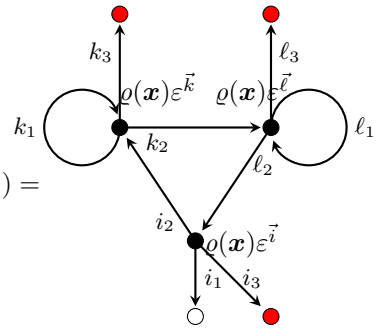
embeddings,

$$\sum_i a_i \phi(\mathrm{emb}(\Gamma_i)) = 0,$$

is preserved.

Example 17. Consider the synonyms in Example 14. Note that these graphs are actually Γ_1^{2D} and Γ_5^{2D} (see Appendix C). Their three-dimensional embeddings satisfy the same linear relation, that is, for the micro-graphs

$\mathrm{emb}(\Gamma_1^{2D}) =$ 

$\mathrm{emb}(\Gamma_5^{2D}) =$ 

we have

$$\phi(\mathrm{emb}(\Gamma_1^{2D})) - \phi(\mathrm{emb}(\Gamma_5^{2D})) = 0.$$

There are some other examples included in Appendix B.[45] Explicitly, those scripts show the preservation of the linear relations of the four-dimensional Hamiltonian micro-graphs for γ_3 embedded to five-dimensional micro-graphs, an example of a vanishing three-dimensional micro-graph being embedded to a vanishing four-dimensional micro-graph, and an example of a linear relation on 3 three-dimensional micro-graphs being preserved under embedding the graphs to four-dimensional micro-graphs.[46]

Conjecture 11. Let us consider some d-dimensional micro-graphs Γ_i such that under the evaluation to multivectors, they satisfy

$$\sum_i a_i \phi(\Gamma_i) = 0,$$

[45] Under the titles: 'Linear relations of the 4D Hamiltonians', 'Linear relations of the 4D Hamiltonians embedded to 5D', 'Vanishing graph in 3D', 'Vanishing graph in 3D embedded to 4D', 'Linear relation of 3 3D graphs' and 'Linear relation of 3 3D graphs embedded to 4D'.

[46] We checked more examples, but as the approach is the same, and the result does not change, we did not add all of the explicit examples here. Instead, we showed an example for a linear relation on 1 graph, 2 graphs, and 3 graphs.

where $a_i \in \mathbb{R}$. Then, under the embedding of the micro-graphs to dimension $d+1$, we have

$$\sum_i a_i \phi(\operatorname{emb} \Gamma_i) = 0.$$

Proving this is not as easy as one might expect. When we write out the multivectors of the embeddings, we will have terms that are a multiple of the original linear relation (when the indices of edges directed to the new Casimir all coincide), but we also get terms where the indices are mixed. Consider the graphs from Example 17. The vector fields they evaluate into are given by

$$\begin{aligned}
\phi(\operatorname{emb}(\Gamma_1^{2D})) &= \sum_{\vec{i},\vec{k},\vec{\ell}} \varepsilon^{\vec{i}}\varepsilon^{\vec{k}}\varepsilon^{\vec{\ell}} \frac{\partial^2 \varrho(\boldsymbol{x})}{\partial x^{i_2}\partial x^{\ell_2}} \frac{\partial \varrho(\boldsymbol{x})}{\partial x^{k_1}} \frac{\partial^2 \varrho(\boldsymbol{x})}{\partial x^{\ell_1}\partial x^{k_2}} \frac{\partial a^1}{\partial x^{i_3}} \frac{\partial a^1}{\partial x^{\ell_3}} \frac{\partial a^1}{\partial x^{k_3}} \frac{\partial}{\partial x^{i_1}} \\
\phi(\operatorname{emb}(\Gamma_5^{2D})) &= \sum_{\vec{i},\vec{k},\vec{\ell}} \varepsilon^{\vec{i}}\varepsilon^{\vec{k}}\varepsilon^{\vec{\ell}} \frac{\partial \varrho(\boldsymbol{x})}{\partial x^{\ell_2}} \frac{\partial^2 \varrho(\boldsymbol{x})}{\partial x^{i_2}\partial x^{k_1}} \frac{\partial^2 \varrho(\boldsymbol{x})}{\partial x^{\ell_1}\partial x^{k_2}} \frac{\partial a^1}{\partial x^{i_3}} \frac{\partial a^1}{\partial x^{\ell_3}} \frac{\partial a^1}{\partial x^{k_3}} \frac{\partial}{\partial x^{i_1}}.
\end{aligned}$$

Now, for the cases that $i_3 = k_3 = \ell_3$, we can write

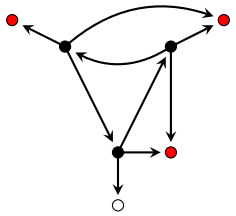
$$\begin{aligned}
&\sum_{\substack{\vec{i},\vec{k},\vec{\ell}, \\ i_3=k_3=\ell_3}} \varepsilon^{\vec{i}}\varepsilon^{\vec{k}}\varepsilon^{\vec{\ell}} \left(\frac{\partial a^1}{\partial x^{i_3}} \frac{\partial a^1}{\partial x^{\ell_3}} \frac{\partial a^1}{\partial x^{k_3}} \right) \frac{\partial^2 \varrho(\boldsymbol{x})}{\partial x^{i_2}\partial x^{\ell_2}} \frac{\partial \varrho(\boldsymbol{x})}{\partial x^{k_1}} \frac{\partial^2 \varrho(\boldsymbol{x})}{\partial x^{\ell_1}\partial x^{k_2}} \frac{\partial}{\partial x^{i_1}} \\
= &\sum_{\substack{\vec{i},\vec{k},\vec{\ell}, \\ i_3=k_3=\ell_3}} \varepsilon^{\vec{i}}\varepsilon^{\vec{k}}\varepsilon^{\vec{\ell}} \left(\frac{\partial a^1}{\partial x^{i_3}} \frac{\partial a^1}{\partial x^{\ell_3}} \frac{\partial a^1}{\partial x^{k_3}} \right) \frac{\partial \varrho(\boldsymbol{x})}{\partial x^{\ell_2}} \frac{\partial^2 \varrho(\boldsymbol{x})}{\partial x^{i_2}\partial x^{k_1}} \frac{\partial^2 \varrho(\boldsymbol{x})}{\partial x^{\ell_1}\partial x^{k_2}} \frac{\partial}{\partial x^{i_1}}
\end{aligned}$$

by invoking the linear relation in dimension 2. However, we when we consider mixed terms where i_3, k_3 and ℓ_3 do not coincide, this argument does not work.

6.5. γ_3: What does not work, issues and a conjecture

The eventual goal of the thesis was to find a general (possibly dimension specific) form for the trivializing vector fields evaluated from graphs. There were two main ideas (with some variations), that we will detail below.

Keeping the constants: With the two-dimensional sunflower solution (see Example 16), the first idea was to look at its three-dimensional descendants, and have the constants be dictated from the two-dimensional graph we created it from. As an example, consider the descendant below.



As it was obtained from Γ_{12}^{2D}, we would give it the constant 2, as

$$\vec{X}_{2D}^{\gamma_3} = 1 \cdot \phi(\Gamma_{11}) + 2 \cdot \phi(\Gamma_{12}).$$

There are two different ways of doing this, depending on how we count isomorphic micro-graphs after generating the descendants. Do we count these isomorphic micro-graphs only once, or as often as we create them? We tried out both cases, and in neither case the resulting vector field was trivializing $Q_{\gamma_3}^{3D}$.

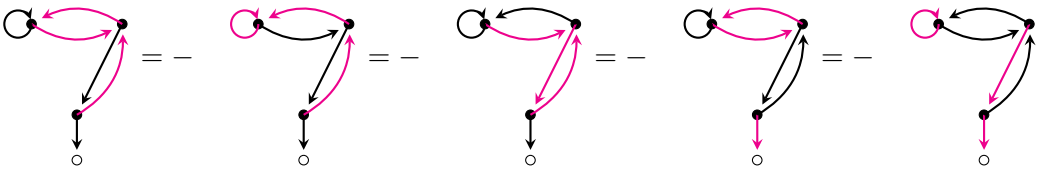
Summing all the non-isomorphic graphs: Recall that in two dimensions, we have 14 non-isomorphic graphs on 3 vertices and 1 sink that evaluate into three linearly independent vector fields [4, Claim 2]. Specifically, the linear relations are

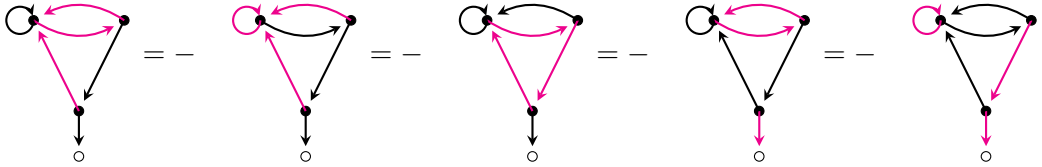
$$\begin{aligned}
&\phi(\Gamma_1^{2D}) = \phi(\Gamma_5^{2D}) = \phi(\Gamma_6^{2D}) = -\phi(\Gamma_7^{2D}) = \tfrac{1}{2}\phi(\Gamma_8^{2D}) = \phi(\Gamma_{12}^{2D}) = \phi(\Gamma_{13}^{2D}),\\
&\phi(\Gamma_2^{2D}) = \phi(\Gamma_4^{2D}) = -\phi(\Gamma_9^{2D}) = \phi(\Gamma_{11}^{2D}),\\
&\phi(\Gamma_3^{2D}) = \phi(\Gamma_{10}^{2D}) = \phi(\Gamma_{14}^{2D}).
\end{aligned}$$

Note that, since Γ_3^{2D}, Γ_{10}^{2D} and Γ_{14}^{2D} all evaluate into the Hamiltonian vector field, we can also write the trivializing vector field as

$$\begin{aligned}
\vec{X}_{2D}^{\gamma_3} =& \tfrac{1}{4}\left(4 \cdot \phi(\Gamma_{11}^{2D}) + 8 \cdot \phi(\Gamma_{12}^{2D})\right)\\
=& \tfrac{1}{4}\left(4 \cdot \phi(\Gamma_{11}^{2D}) + 8 \cdot \phi(\Gamma_{12}^{2D}) + 3 \cdot \phi(\Gamma_4^{2D})\right)\\
=& \tfrac{1}{4}\left(\phi(\Gamma_1^{2D}) + \phi(\Gamma_2^{2D}) + \phi(\Gamma_3^{2D}) + \phi(\Gamma_4^{2D}) + \phi(\Gamma_5^{2D}) + \phi(\Gamma_6^{2D}) - \phi(\Gamma_7^{2D}) + \phi(\Gamma_8^{2D})\right.\\
&\left. -\phi(\Gamma_9^{2D}) + \phi(\Gamma_{10}^{2D}) + \phi(\Gamma_{11}^{2D}) + \phi(\Gamma_{12}^{2D}) + \phi(\Gamma_{13}^{2D}) + \phi(\Gamma_{14}^{2D})\right).
\end{aligned}$$

Now, we are not only summing here and Γ_7^{2D} and Γ_9^{2D} appear with a minus sign. However, this might have to do with how we generated these graphs and decided on their edge ordering. Indeed, below we see that by simply switching around the order the edges of an odd amount of vertices gives us exactly what we want.

$\Gamma_7^{2D} =$ 

$\Gamma_9^{2D} =$  $= -$ $= -$ $= -$ $= -$

This observation leads to the following conjecture.

Conjecture 12. Up to appropriate signs dictated by the edge orderings of the graphs and up to some (possibly dimension related) constant, taking all non-isomorphic d-dimensional micro-graphs on 3 Nambu-determinant Poisson structures and summing the vector fields they evaluate into will give a trivializing vector field $\vec{X}_d^{\gamma_3}$ solving $Q_{\gamma_3}^d(\mathcal{P}) = [\![\mathcal{P}, \vec{X}_d^{\gamma_3}]\!]$.

An obstacle in checking if this works for $\vec{X}_{3D}^{\gamma_3}$ and $\vec{X}_{4D}^{\gamma_3}$ is that we do not know what the 'correct' edge orderings are as of right now. We could generate all the graphs Γ_i^d and check all the possibilities induced by the choice of signs. For all these choices of the signs, we would then sum over all of these graphs evaluated into vector fields, and we would check if the resulting vector field is proportional to $\vec{X}_d^{\gamma_3}$, that is,

$$\vec{X}_d^{\gamma_3} \overset{?}{\sim} \sum_i \pm\phi(\Gamma_i^d).$$

This task is not impossible, but also not particularly feasible due to the large amount of non-isomorphic micro-graphs appearing.

7. Conclusion

We have shown that the vector fields trivializing the infinitesimal deformations $Q_{\gamma_3}(\mathcal{P})$ and $Q_{\gamma_5}(\mathcal{P})$ on $\mathbb{R}^2_{\text{aff}}$ for all Poisson bivectors $\mathcal{P}$ are unique up to Hamiltonian vector fields. In a similar manner, we showed that the vector fields trivializing $Q_{\gamma_3}(\mathcal{P})$, restricted to Poisson bivectors of the Nambu-determinant class on $\mathbb{R}^3_{\text{aff}}$ and $\mathbb{R}^4_{\text{aff}}$ are unique up to Hamiltonian vector fields. While investigating the nature of the trivializing vector fields, we came across some unexpected and interesting observations, related to the phenomenon of synonyms and the preservation of linear relations under embeddings.

We also looked at trivializing $Q_{\gamma_5}(\mathcal{P})$ on $\mathbb{R}^3_{\text{aff}}$ for Nambu-determinant Poisson brackets. Unfortunately, due to an undetected mistake in the code, we did not find a trivializing vector field when considering descendants of two-dimensional graphs. However, the corrected code is attached in Appendix B, and should be rerun to check for a trivializing vector field over these descendants. The other approach, by taking all 5811346 non-isomorphic graphs on 5 Levi-Civita vertices, 5 corresponding Casimir vertices, and 1 sink requires more than 1 TB of RAM, and more than 5 days of running time on the high performance computer cluster Hábrók. As detailed in Section 6.2, the code is now running with with 4 TB of RAM, 80 CPU cores, and with a time limit of 10 days.

While researching the nature of the trivializing vector fields, we made some observations. The following two conjectures follow from these observations, and can be investigated in further research.

Conjecture. *Let us consider some d-dimensional micro-graphs Γ_i such that under the evaluation to multivectors, they satisfy*

$$\sum_i a_i \phi(\Gamma_i) = 0,$$

where $a_i \in \mathbb{R}$. Then, under the embedding of the micro-graphs to dimension $d+1$, we have

$$\sum_i a_i \phi(\operatorname{emb} \Gamma_i) = 0.$$

Conjecture. *Up to appropriate signs dictated by the edge orderings of the graphs and up to some (possibly dimension related) constant, taking all non-isomorphic $d-$dimensional micro-graphs on 3 Nambu-determinant Poisson structures and summing the vector fields they evaluate into will give a trivializing vector field $\vec{X}_d^{\gamma_3}$ solving $Q^d_{\gamma_3}(\mathcal{P}) = [\![\mathcal{P}, \vec{X}_d^{\gamma_3}]\!]$.*

A final remark is about the computational approach of the problem so far. For γ_3, the trivializing vector fields $\mathbb{R}^2$, $\mathbb{R}^3$ and $\mathbb{R}^4$ are relatively easy to compute, and we run intro trouble for $\mathbb{R}^5$. For γ_5, the trivializing vector field already becomes very difficult to compute for $\mathbb{R}^3$. It is not sustainable to rely on explicit computations for future research, and the focus should lie on the nature of the trivialization of the deformation term. In particular, it would be interesting to investigate if the triviality of $Q_{\gamma_3}(\mathcal{P})$ for non-Nambu-determinant

Poisson brackets $\mathcal{P}$ is preserved, or that it is specifically the class of Nambu-determinant Poisson brackets that imposes the triviality.

References

[1] Schipper F. thesisfloor; 2024. Last visisted: 2024-11-25. Available from: `https://github.com/floorschipper/thesisfloor`.

[2] Kiselev AV, Jagoe Brown MS, Schipper F. Kontsevich graphs act on Nambu–Poisson brackets, I. New identities for Jacobian determinants. Journal of Physics: Conference Series (accepted). 2024 *preprint: arXiv:240918875 [MathQA].*

[3] Jagoe Brown MS, Schipper F, Kiselev AV. Kontsevich graphs act on Nambu–Poisson brackets, II. The tetrahedral flow is a coboundary in 4D. Journal of Physics: Conference Series (accepted). 2024 *preprint: arXiv:240912555 [MathQA].*

[4] Schipper F, Jagoe Brown MS, Kiselev AV. Kontsevich graphs act on Nambu–Poisson brackets, III. Uniqueness aspects. Journal of Physics: Conference Series (accepted). 2024 *preprint arXiv:240915932 [MathQA].*

[5] Buring R. `gcaops`, commit:024818d; 2024. Last visisted: 2024-11-22. Available from: `https://github.com/rburing/gcaops`.

[6] Lichnerowicz A. Les variétés de Poisson et leurs algebres de Lie associées. Journal of différential geometry. 1977;12(2):253-300.

[7] Lee JM. Introduction to Smooth Manifolds. Springer New York; 2012.

[8] Conrad B. Math 396. Tensor algebras, tensor pairings, and duality; 2006. Available from: `https://math.stanford.edu/~conrad/diffgeomPage/handouts/tensor.pdf`.

[9] Nicolaescu LI. Lectures on the Geometry of Manifolds. World Scientific; 2020.

[10] Crainic M, Fernandes RL, Mărcuţ I. Lectures on Poisson geometry. vol. 217. American Mathematical Soc.; 2021.

[11] Berezin FA. Introduction to superanalysis. vol. 9. Springer Science & Business Media; 2013.

[12] Lang S. Algebra. vol. 211. Springer Science & Business Media; 2012.

[13] Rutten NJ, Kiselev AV; IOP Publishing. The defining properties of the Kontsevich unoriented graph complex. Journal of Physics: Conference Series. 2019;1194:012095 *arXiv:1811.10638 [math.CO].*

[14] Buring R, Kiselev AV, Rutten N; IOP Publishing. Infinitesimal deformations of Poisson bi-vectors using the Kontsevich graph calculus. Journal of Physics: Conference Series. 2018;965:012010 *arXiv:1710.02405 [math.CO].*

[15] Willwacher T. M. Kontsevich's graph complex and the Grothendieck–Teichmüller Lie algebra. Inventiones mathematicae. 2015;200(3):671-760 *arXiv:1009.1654 [math.QA].*

[16] Kontsevich M. Formality conjecture. Deformation theory and symplectic geometry. 1997;128:139-56.

[17] Buring R, Kiselev AV, Rutten NJ. Poisson brackets symmetry from the pentagon-wheel cocycle in the graph complex. Physics of Particles and Nuclei. 2018;49(5):924-8 *arXiv:1712.05259 [math-ph].*

[18] Buring R, Kiselev AV, Rutten NJ. The heptagon-wheel cocycle in the Kontsevich graph complex. Journal of Nonlinear Mathematical Physics. 2017;24(Suppl 1):157-73 *arXiv:1710.00658 [math.CO].*

[19] Buring R, Kiselev AV. Universal cocycles and the graph complex action on homogeneous Poisson brackets by diffeomorphisms. Physics of Particles and Nuclei Letters. 2020;17(5):707-13 *arXiv:1912.12664 [math.SG].*

[20] Buring R, Kiselev A. The orientation morphism: from graph cocycles to deformations of Poisson structures. Journal of Physics: Conference series. 2019;1194:012017 *arXiv:1811.07878 [math.CO].*

[21] Buring R, Kiselev AV. The tower of Kontsevich deformations for Nambu-Poisson structures on $\mathbb{R}^d$: Dimension-specific micro-graph calculus. SciPost Physics Proceedings. 2023;14:Paper 020 *arXiv:2212.08063 [math.CO].*

[22] Gerstenhaber M. On the deformation of rings and algebras. Annals of Mathematics. 1964;79:59-103.

[23] Kodaira K, Spencer DC. On Deformations of Complex Analytic Structures, I. Annals of Mathematics. 1958;67(2):328-401.

[24] Kodaira K, Spencer DC. On Deformations of Complex Analytic Structures, II. Annals of Mathematics. 1958;67(3):403-66.

[25] Kodaira K, Nirenberg L, Spencer DC. On the Existence of Deformations of Complex Analytic Structures. Annals of Mathematics. 1958;68(2):450-9.

[26] Dirac PAM. The principles of quantum mechanics. 27. Oxford university press; 1981.

[27] Feynman RP, Hibbs AR, Styer DF. Quantum mechanics and path integrals. Courier Corporation; 2010.

[28] Diosi L. Quantum dynamics with two Planck constants and the semiclassical limit. *preprint: arXiv:quant-ph/9503023*. 1995.

[29] Buring R, Kiselev A. The Expansion $\star$ mod $\bar{\mathrm{o}}$ $(\hbar^4)$ and Computer-Assisted Proof Schemes in the Kontsevich Deformation Quantization. Experimental Mathematics. 2022;31(3):701-54 *arXiv:1702.00681 [math.CO]*.

[30] Kontsevich M. Deformation quantization of Poisson manifolds. Letters in Mathematical Physics. 2003;66:157-216.

[31] Bouisaghouane A. The Kontsevich tetrahedral flow in 2D: a toy model. *preprint: arXiv:170206044 [MathDG]*. 2017.

[32] Buring R, Lipper D, Kiselev AV. The hidden symmetry of Kontsevich's graph flows on the spaces of Nambu-determinant Poisson brackets. Open Communications in Nonlinear Mathematical Physics. 2022;2:186-216 *2112.03897 [math.SG]*.

[33] Buring R. The action of Kontsevich's graph complex on Poisson structures and star products: an implementation. PhD thesis. 2022 October. Available from: `https://openscience.ub.uni-mainz.de/handle/20.500.12030/9292`.

[34] Vanhaecke P. Integrable systems in the realm of algebraic geometry. Springer Science & Business Media; 2001.

[35] Bouisaghouane A, Kiselev AV. Do the Kontsevich tetrahedral flows preserve or destroy the space of Poisson bi-vectors? Journal of Physics: Conference Series. 2017;804:012008 *arXiv:1609.06677 [Math.QA]*.

[36] Gutt S. Deformation quantization and group actions. Quantization, Geometry and Noncommutative Structures in Mathematics and Physics. 2017:17-73.

[37] Kiselev AV. Deformation quantisation, graph complex and Number Theory; 2020. Audiocourse (MasterMath, NL).

[38] Bayen F, Flato M, Fronsdal C, Lichnerowicz A, Sternheimer D. Deformation theory and quantization. I. Deformations of symplectic structures. Annals of Physics. 1978;111(1):61-110.

[39] Bayen F, Flato M, Fronsdal C, Lichnerowicz A, Sternheimer D. Deformation theory and quantization. II. Physical applications. Annals of Physics. 1978;111(1):111-51.

[40] Bouisaghouane A, Buring R, Kiselev A. The Kontsevich tetrahedral flow revisited. Journal of geometry and physics. 2017;119:272-85 *arXiv:1608.01710 [Math.QA]*.

Appendix A. Papers

Kontsevich graphs act on Nambu–Poisson brackets, I. New identities for Jacobian determinants

Arthemy V Kiselev, Mollie S Jagoe Brown and Floor Schipper

Bernoulli Institute for Mathematics, Computer Science and Artificial Intelligence, University of Groningen, P.O. Box 407, 9700 AK Groningen, The Netherlands

E-mail: a.v.kiselev@rug.nl, m.s.jagoe.brown@gmail.com, f.m.schipper@rug.nl

Abstract. Nambu-determinant brackets on $\mathbb{R}^d \ni \boldsymbol{x} = (x^1, \ldots, x^d)$, $\{f,g\}_d(\boldsymbol{x}) = \varrho(\boldsymbol{x}) \cdot \det\big(\partial(f,g,a_1,\ldots,a_{d-2})/\partial(x^1,\ldots,x^d)\big)$, with $a_i \in C^\infty(\mathbb{R}^d)$ and $\varrho \cdot \partial_{\boldsymbol{x}} \in \mathfrak{X}^d(\mathbb{R}^d)$, are a class of degenerate (rank $\leqslant 2$) Poisson structures with (non)linear coefficients, e.g., polynomials of arbitrarily high degree. With 'good' cocycles in the graph complex, Kontsevich associated universal – for all Poisson bi-vectors P on affine $\mathbb{R}^d_{\text{aff}}$ – elements $\dot{P} = Q^\gamma([P]) \in \mathrm{H}^2_P\big(\mathbb{R}^d_{\text{aff}}\big)$ in the Lichnerowicz–Poisson second cohomology groups; we note that known graph cocycles γ preserve the Nambu–Poisson class $\big\{P(\varrho,[\boldsymbol{a}])\big\}$, and we express, directly from γ, the evolution $\dot{\varrho}$, $\dot{\boldsymbol{a}}$ that induces $\dot{P}$.

Over all $d \geqslant 2$ at once, there is no 'universal' mechanism for the bi-vector cocycles Q^γ_d to be trivial, $Q^\gamma_d = [\![P, \vec{X}^\gamma_d([P])]\!]$, w.r.t. vector fields defined uniformly for all dimensions d by the same graph formula. While over $\mathbb{R}^2$, the graph flows $\dot{P} = Q^{\gamma_i}_{2D}\big(P(\varrho)\big)$ for $\gamma \in \{\gamma_3, \gamma_5, \gamma_7, \ldots\}$ are trivialised by vector fields $\vec{X}^{\gamma_i}_{2D} = (\mathrm{d}x \wedge \mathrm{d}y)^{-1}\mathrm{d}_{\mathrm{dR}}\big(\mathrm{Ham}^{\gamma_i}(P)\big)$ of peculiar shape, we detect that in $d \geqslant 3$, the 1-vectors from 2D, now with $P(\varrho,a_1,\ldots,a_{d-2})$ inside, do not solve the problems $Q^{\gamma_i}_{d\geqslant 3} = [\![P, \vec{X}^{\gamma_i}_{d\geqslant 3}\big(P(\varrho,[\boldsymbol{a}])\big)]\!]$, yet they do yield a good Ansatz where we find solutions $\vec{X}^{\gamma_i}_{d=3,4}\big(P(\varrho,[\boldsymbol{a}])\big)$. In the study of the step $d \mapsto d+1$, by adapting the Kontsevich graph calculus to the Nambu–Poisson class of brackets, we discover more identities for the Jacobian determinants within $P(\varrho,[\boldsymbol{a}])$, i.e. for multivector-valued $GL(d)$-invariants on $\mathbb{R}^d_{\text{aff}}$.

Introduction. This paper begins the series of three namesake texts which are devoted to deformations of Poisson brackets – by using Kontsevich's graph cocycles – within the class of Nambu-determinant Poisson structures on $\mathbb{R}^d$. In the subsequent article (II.), see [9], we establish the trivialisation of the tetrahedral γ_3-graph flow on the space of Nambu–Poisson brackets over $\mathbb{R}^4$, that is in dimension four (cf. [6] for $d = 3$ and [10, 1] for $d = 2$). The uniqueness of trivialising vector fields $\vec{X}^{\gamma_3}_d$, themselves encoded by (generalisations of) Kontsevich's graphs built of (Nambu–)Poisson bi-vectors, modulo Poisson vector fields with Hamiltonians also expressed in terms of graphs, is verified in the third article (III.), see [13].

Now, in this paper we summarise newly observed properties of the Nambu–Poisson brackets. We discover five classes of differential-polynomial identities which the Jacobian determinants (and the brackets derived from them) conjecturally satisfy; all these hypotheses are open problems about (an effective description of) multivector-valued $GL(d)$-invariants, $2 \leqslant d < \infty$, over affine spaces $\mathbb{R}^d$.

Definition 1. The Nambu-determinant Poisson bracket of $f, g \in C^\infty(\mathbb{R}^d)$ is expressed by the formula

$$\{f,g\}_d(\boldsymbol{x}) = \varrho(\boldsymbol{x}) \cdot \det\Big(\partial\big(f,g,a_1,\ldots,a_{d-2}\big)\big/\partial\big(x^1,\ldots,x^d\big)\Big), \qquad \boldsymbol{x} \in \mathbb{R}^d, \qquad (1)$$

where $a_i \in C^\infty(\mathbb{R}^d)$ are Casimirs, $\varrho(\boldsymbol{x}) \cdot \partial_{x^1} \wedge \ldots \wedge \partial_{x^d}$ is a d-vector, and (x^i) are global coordinates on $\mathbb{R}^d$, $d \geqslant 2$.

Remark 1. Kontsevich's construction of the graph cocycle action on the spaces of Poisson brackets is well defined over arbitrary finite-dimensional affine manifolds $M^{d<\infty}_{\text{aff}}$. In our present illustration of this concept and in our study of the action specifically upon the class of Nambu–Poisson brackets, we take $M := \mathbb{R}^d$ with natural global Cartesian coordinates, e.g., denoted by x, y, z, w on $\mathbb{R}^4$. Yet of course, the term 'Cartesian' serves here as the marker for an atlas of affine coordinate charts on M^d_{aff}, that is all the coordinate tuples which are obtained from a given one by using affine reparametrisations.

Remark 2. Over affine manifolds M^d_{aff}, the degree of polynomial functions is well defined; beyond scalar functions, this is also true in particular for the components $P^{ij}(\boldsymbol{x})$ of the Poisson tensor P (provided these components are polynomial in every chart of some cover for an orientable manifold M^d_{aff}). Therefore, Nambu's formula $P = [\![\ldots[\![\varrho\,\partial_{\boldsymbol{x}}, a_1]\!], \ldots a_{d-2}]\!]$ of Poisson structures $P(\varrho, [\boldsymbol{a}])$ on orientable M^d_{aff} offers us the brackets with coefficients of arbitrarily high polynomial degree, which is achieved by taking polynomial scalar functions a_i and taking the d-vector $\varrho(\boldsymbol{x}) \cdot \partial_{\boldsymbol{x}}$ (again, a tensor) with polynomial coefficient ϱ of suitable degrees.

Let us remember also that the symplectic leaves of the Nambu–Poisson structures $P(\varrho, [\boldsymbol{a}])$ are at most of dimension two. Indeed, the leaves are selected by intersecting the level sets of the $d-2$ Casimirs. (The Euler linear bracket on $\mathfrak{so}(3)^*$, in Cartesian coordinates described by $\{x,y\} = z$ and so on cyclically, foliates $\mathbb{R}^3$ by the concentric spheres $\{(x,y,z) | a = \frac{1}{2}(x^2+y^2+z^2) = \frac{1}{2}r^2 \geq 0\}$, providing a typical example: at $r = 0$, the zero-dimensional symplectic leaf amounts to the central point of all spheres.)

Remark 3. Not only does the binary bracket $\{\cdot,\cdot\}_d$ satisfy the Jacobi identity but also does the N-ary bracket,

$$\{f_1,\ldots,f_N\}_d(\boldsymbol{x}) = \varrho(\boldsymbol{x}) \cdot \det\Big(\partial\big(f_1,\ldots,f_N,a_{N-1},\ldots,a_{d-2}\big)\big/\partial\big(x^1,\ldots,x^d\big)\Big),$$

read off literally from Eq. (1) for $2 \leqslant N \leqslant d$, satisfy one of the many possible N-ary generalizations of the Jacobi identity, '*the adjoint action is a derivation of the bracket*' (see [12, 14]):

$$\begin{aligned}\{f_1,\ldots,f_{N-1},\{g_1,\ldots,g_N\}_d\}_d = \{\{f_1,\ldots,f_{N-1},g_1\}_d,g_2,\ldots,g_N\}_d + \\ \{g_1,\{f_1,\ldots,f_{N-1},g_2\}_d,g_3,\ldots,g_N\}_d + \cdots + \{g_1,\ldots,g_{N-1},\{f_1,\ldots,f_{N-1},g_N\}_d\}_d.\end{aligned}$$

Let us remember that at either $N = 2$ (Poisson case) or $N > 2$, the Jacobi identities are quadratic in the N-ary structure.

This paper is structured as follows. In §1 we recall from [10] Kontsevich's idea of acting – by suitable nontrivial graph cocycles γ – on the spaces of all Poisson bi-vectors P on affine manifolds of arbitrary finite dimension d. We note that for the wheel-cocycle generators γ_3, γ_5, γ_7, γ_9, ... of the Grothendieck–Teichmüller Lie algebra $\mathfrak{grt}$ (see [15] and [7]), the corresponding 2-cocycles $\dot{P} =_2 Q^{2\ell+1}(P) \in \ker[\![P,\cdot]\!]$ are *not* trivialised by

any vector fields $\vec{X}^{\gamma_{2\ell+1}}$ which would again be encoded by graphs and therefore, solve the trivialisation problems $Q^\gamma = [\![P, \vec{X}^\gamma]\!]$ universally over all dimensions $d \geqslant 2$.

In §2 we adapt Kontsevich's graph approach to the differential calculus of multivectors on $\mathbb{R}^d_{\text{aff}}$ of unspecified dimension d — now, to the dimension-specific classes of Nambu-determinant Poisson bi-vectors $P(\varrho, [\boldsymbol{a}])$ on affine $\mathbb{R}^{d\geqslant 3}$. We thus work with the *Nambu micro-graphs* (see [6]), in which the top-degree d-vector $\varrho \cdot \partial_{\boldsymbol{x}}$ is resolved against the Casimir(s) $\boldsymbol{a} = (a_1, \ldots, a_{d-2})$ in each subgraph that encodes a copy of the bi-vector $P(\varrho, [\boldsymbol{a}])$. We give examples of $(k \geqslant 0)$-vector Nambu micro-graphs which do not 'equal minus themselves thanks to an automorphism' but which nevertheless encode identically vanishing k-vectors. We observe that the $(d+1)$-dimensional *embedding* of a Nambu micro-graph which vanished in dimension d still vanishes in dimension $d+1$. Likewise, we see that for *synonyms*, i.e. for topologically nonisomorphic $(k \geqslant 0)$-vector Nambu micro-graphs which encode identically equal k-vectors on $\mathbb{R}^d$, their graph embeddings over dimension $d+1$ still are synonyms; the same is conjecturally true for longer nontrivial linear combinations of micro-graph formulas: if a formula $\phi(\sum_i c_i \Gamma_i) = 0 \in \mathfrak{X}^k\big(\mathbb{R}^d_{\text{aff}}\big)$ is a nontrivial relation and if $\Gamma \hookrightarrow \widehat{\Gamma}$ is the embedding of micro-graphs, then the formula $\phi(\sum_i c_i \widehat{\Gamma}_i) = 0 \in \mathfrak{X}^k\big(\mathbb{R}^{d+1}_{\text{aff}}\big)$ remains a valid relation. We seek to understand the mechanism of this preservation, under $d \mapsto d+1$, of relations for this class of multivector-valued $GL(d)$-invariants.

In the second part of this paper, starting in §3 we state the facts of trivialisation for Kontsevich's graph flows, $\dot{P}_{2D} = Q^\gamma_{2D}\big(P(\varrho)\big)$, over dimension $d = 2$ for the $(2\ell+1)$-wheel graph cocycles $\gamma \in \big\{\gamma_3, \gamma_5, \gamma_7\big\}$ and for the Lie bracket $[\gamma_3, \gamma_5]$. We notice that not only for the tetrahedron γ_3 but also for the larger graph cocycles, the affine spaces of trivialising vector fields $\vec{X}^\gamma_{2D}\big(P(\varrho)\big)$ that solve $Q^\gamma_{2D}\big(P(\varrho)\big) = [\![P, \vec{X}^\gamma_{2D}]\!]$ do contain a Hamiltonian vector field $\vec{X}^\gamma_{2D} = (\mathrm{d}x \wedge \mathrm{d}y)^{-1}\big(\mathrm{d}_{\mathrm{dR}}\mathrm{Ham}^\gamma\big(P(\varrho)\big)\big)$ given by the canonical symplectic structure $\omega_2 = \mathrm{d}x \wedge \mathrm{d}y$ on $\mathbb{R}^2$ and by Hamiltonians Ham^γ which, for every such graph cocycle γ, are encoded by graphs built of wedges. (In the subsequent papers [9, 13] we discover that in dimensions $d = 3$ and 4, solutions $\vec{X}^{\gamma_3}_{d=3,4}$ appear over the Ansatz of linear combinations of micro-graph *descendants* of the 1-vector graphs in such particular solution $\vec{X}^{\gamma_3}_{2D} = \omega_2^{-1}\big(\mathrm{d}_{\mathrm{dR}}\mathrm{Ham}^{\gamma_3}\big)$ — unlike for most of the other graph pairs that encode the solution in dimension two.)

Finally, in §4 we conjecture the formulas of velocities $\dot{\varrho}$ and $\dot{a}_i$, expressed directly in terms of the graph cocycles γ, that imply (by the Leibniz rule) the evolution $\dot{P} = Q^\gamma\big(P(\varrho, [\boldsymbol{a}])\big)$ of the Nambu–Poisson structures. By contrasting the antisymmetry of the Nambu-determinant brackets $P(\varrho, [\boldsymbol{a}])$ w.r.t. the flips $a_i \mapsto -a_i$ for all $d \geqslant 3$ and w.r.t. permutations of the set of Casimirs a_i for $d \geqslant 4$ against the symmetry of their γ-flows $Q^\gamma_d(P)$, we motivate the existence of trivialisation, $Q^{\gamma_{2\ell+1}}_d = [\![P, \vec{X}^{\gamma_{2\ell+1}}_d]\!]$, for the $(2\ell+1)$-wheel graph cocycles $\gamma_{2\ell+1}$ (and for their iterated commutators on even number of vertices, cf. [7, 15]).

1 Preliminaries: Kontsevich graph cocycles act on Poisson brackets

In the seminal paper [10], Kontsevich introduced the graph complex action on the spaces of multivectors on affine finite-dimensional manifolds. We recall that real vector spaces of undirected finite graphs with a global ordering of edges ($\mathrm{First} \wedge \ldots \wedge \mathrm{Last}$, quotient over the relations $\mathrm{Edge}_i \wedge \mathrm{Edge}_j = -\mathrm{Edge}_j \wedge \mathrm{Edge}_i$) are endowed with the structure of differential graded Lie (super-)algebra (dgLa), namely with the Lie bracket $[\cdot, \cdot]$ from

the graded commutator of graph insertion into vertices and with the vertex blow-up differential $\mathsf{d} = [\bullet\!\!-\!\!\bullet, \cdot]$; we refer to [7, 4] and references therein for all definitions and details. Graph cocycles on n vertices and $2n-2$ edges are of particular interest because it is in this vertex-edge bi-grading where graph cocycles γ can act – by non-identically zero shifts $P \mapsto P + \varepsilon Q^\gamma + \overline{o}(\varepsilon)$ and in a possibly nontrivial way, $Q^\gamma \neq [\![P, \vec{X}^\gamma]\!]$ – on the spaces of Poisson bi-vectors P over the affine manifold M^d_{aff} at hand. Willwacher established the existence of at least countably many such cocycles (see [15]): the $(2\ell+1)$-wheel graph cocycles $\gamma_{2\ell+1}$, $\ell \in \mathbb{N}$, stem from the generators of Drinfeld's Grothendieck-Teichmüller Lie algebra $\mathfrak{grt}$, so that their iterated commutators stay in the good vertex-edge bi-grading and remain non-trivial cocycles in the graph complex.

Example 1. The smallest nontrivial graph cocycle, on $n = 4$ vertices and $2n-2 = 6$ edges, is the 3-wheel itself: it is the tetrahedron γ_3 (the full graph on 4 vertices); it appeared already in [10]. The pentagon-wheel graph cocycle γ_5, consisting of two graphs on 6 vertices and 10 edges (see Table 1 on p. 5 below), was known to Kontsevich and to Willwacher; the cocycle γ_5 is described in [7]. The heptagon-wheel graph cocycle $[\gamma_7]$ was obtained in [7]; now, the space of graphs on 8 vertices and 14 edges is big enough to provide degree(s) of freedom in the cohomology class $[\gamma_7]$ due to the now-possible coboundaries $\mathsf{d}(\beta)$ from graphs β on 7 vertices and 13 edges; the shortest-known representative γ_7 of the nontrivial cohomology class $[\gamma_7]$ is a linear combination of 46 graphs. The next graph cocycle, in the vertex-edge bi-grading (9,16) immediately following (8,14) along the ray $(n, 2n-2)$, is represented by the commutator $[\gamma_3, \gamma_5]$; its encoding is worked out in [3]. At the ISQS28 conference (CVUT Prague, 1–5 July 2024), R. Buring reported a representative γ_9 of the 9-wheel graph cocycle on 10 vertices and 18 edges in each of its 13,723 terms. (As the vertex number grows, the $(n, 2n-2)$-homogeneous component of the graph space can contain not just one but many nontrivial graph cocycles which, modulo the coboundaries, are linearly independent.) Let us remember also that each of these good graph cocycles $\gamma_3, \ldots, \gamma_9$ was obtained 'anew', i.e. not – by following Willwacher's isomorphism – from the generators of the Lie algebra $\mathfrak{grt}$; the task of writing explicit formulas for the correspondence between $\mathfrak{grt}$ and representatives of the classes $[\gamma_{2\ell+1}]$, and of their iterated commutators, is a work in progress (M. Kontsevich, private communication).

Graphs γ with a global edge ordering $E(\gamma)$ are mapped to endomorphisms of the space of multivectors on M^d_{aff} by the edge orientation morphism (see [10] and [2, 4, 8]). Every directed edge $\vec{e}$, decorated with a summation index i_e which runs from 1 to d, denotes the derivation $\partial/\partial x^{i_e}$ of the (multi)vector contained in the arrowhead vertex; the local exterior ordering of the outgoing edges, $\vec{e}_1 \wedge \vec{e}_2 \wedge \ldots \wedge \vec{e}_k$, which thus expresses the skew-symmetry of the k-vector (in the arrowtail vertex) with respect to its arguments, is inherited at every vertex from the global ordering of edges in the initially taken graph γ, where $E(\gamma) = ... \wedge e_1 \wedge e_2 \wedge ... \wedge e_k \wedge ....$ In our present study of the graph complex action on Poisson brackets, it suffices to enlarge the graph γ by two sink vertices and to consider only those portraits of edge direction where the new graphs, with exactly one arrow directed to either of the sinks, are built entirely of wedges $\overset{i}{\leftarrow}\!\bullet\!\overset{j}{\rightarrow}$ (for the Poisson bi-vectors $P = (P^{ij})$ which are the building blocks).

Definition 2. Directed graphs built over $m \geqslant 0$ sinks from $n \geqslant 1$ wedges (with local ordering Left $\prec$ Right for the two outgoing arrows at every wedge top) are called the *Kontsevich (di)graphs*; note that 1-loops (*tadpoles*) are allowed, although the $(2\ell+1)$-wheel

graph cocycles (and their commutators) stemming from $\mathfrak{grt}$ always admit representatives $\gamma_{2\ell+1}$ (resp., $[\ldots[\gamma_{2\ell+1},\gamma_{2p+1}]\ldots]$) without 1-loops.

Example 2. By directing the four edges in the tetrahedron γ_3 in such a way that the vertices of γ_3 are the four wedge tops and the two excessive edges are sent to the two new sinks $\overline{0}$ and $\overline{1}$, we obtain – with multiplicities 8 and 24 – two topologically non-isomorphic pictures (see [2]): one is already skew over the sinks and the other, to give a bi-vector, is skew-symmetrised; this yiels $1+2=3$ Kontsevich graphs. Taken with their multiplicities $8:24=1:3$, they encode the bi-vector $Q^{\gamma_3}([P]) = \mathrm{O\vec{r}}(P\otimes P\otimes P\otimes P)$. Likewise, for the pentagon-wheel graph cocycle γ_5, we obtain the 91 bi-vectors realised by Kontsevich graphs ([8]), and so on (see [4, 3] and Table 1).

Table 1: The number of (un)directed graphs in the graph cocycles γ and Poisson cocycles Q^γ.

Cocycle γ:	γ_3	γ_5	$\gamma_7\in[\gamma_7]$	$[\gamma_3,\gamma_5]$	$\gamma_9\in[\gamma_9]$
# vertices:	4	6	8	9	10
# edges:	6	10	14	16	18
# graphs in γ:	1	2	46	68	13,723
# bi-vectors in Q^γ:	2	91	20,422	42,252	?
# directed graphs in Q^γ:	3	167	37,185	?	?

Lemma 1 (see [10] and [4, 15])**.** Whenever $\gamma\in\ker\mathsf{d}$ is a nontrivial graph cocycle over n vertices and $2n-2$ edges, and P is a Poisson bi-vector on an affine manifold M^d_{aff}, the bi-vector $Q^\gamma([P]) := \mathrm{O\vec{r}}(P^{\otimes^n})$ is a Poisson 2-cocycle: $Q^\gamma([P])\in\ker[\![P,\cdot]\!]$.

Claim 2. *Over all affine Poisson manifolds $(M^{d<\infty}_{\mathrm{aff}},P)$ at once, the $(2\ell+1)$-wheel graph cocycle deformations $\dot P = Q^{\gamma_{2\ell+1}}(P^{\otimes^{2\ell+2}})$ cannot be Poisson coboundaries 'universally' over $d\geqslant 2$ with respect to always the same linear combinations $X^{\gamma_{2\ell+1}}$ of Kontsevich 1-vector graphs built of $n=2\ell+1$ wedges. Specifically, there is no solution $\diamond$ – at the level of Formality graphs from* [11] *– to the equation*

$$Q^{\gamma_{2\ell+1}} - [\![P,\ \textit{any 1-vector graphs on } 2\ell+1 \textit{ wedges}\ ]\!] = \diamond\big(P,\tfrac{1}{2}[\![P,P]\!]\big),$$

where the right-hand side encodes bi-vectors that vanish by force of the Jacobi identity for the Poisson structure P.

Sketch of the proof. Tadpoles are neither produced nor destroyed by the differential calculus of graphs (when the Jacobiator is expanded by definition and when an arrow works over the vertices of a (sub)graph by the Leibniz rule, e.g., during the calculation of the Schouten bracket $[\![\cdot,\cdot]\!]$). Therefore, the linear problem of $\gamma_{2\ell+1}$-deformation's trivialisation at the level of Formality graphs is filtered by the number of tadpoles in a graph.

We recall that by construction, there are no tadpoles in the inhomogeneity, $\dot P = \mathrm{O\vec{r}}(\gamma_{2\ell+1})\big(P^{\otimes^{2\ell+2}}\big)$. To establish the absence of universal trivialisation, it suffices to inspect the 0th layer of the problem with Formality graphs without tadpoles; here, the obstruction is easily attained at all $\ell\in\mathbb{N}$. □

Let us remember that over every affine Poisson manifold $M^{d<\infty}_{\mathrm{aff}}$ of any finite dimension $d\geqslant 2$, each Kontsevich graph gives us a well-defined k-vector (that belongs – possibly,

after due antisymmetrisation – to the space $\mathfrak{X}^k(M^d_{\text{aff}})$); the formula of that k-vector behaves well under affine coordinate reparametrisations: the shifts are not felt at all, whereas the linear transformations from $GL(d<\infty)$ are absorbed by the reparametrised copies of the Poisson tensor in the vertices of Kontsevich graphs. Yet it does occur that topologically nonisomorphic Kontsevich graphs of equal arity (e.g., 1-vectors) and with equal number of vertices (hence of equal polynomial degree in the coefficients of $P=(P^{ij})$ or their derivatives) encode linearly dependent k-vector formulas in a given dimension $d<\infty$. That is, the projections of universally defined $GL(\infty)$-invariants (encoded by Kontsevich graphs) to $GL(d)$-invariants become constrained by linear relations.

Example 3 (see Claim 2 in [13])**.** The 14 admissible non-isomorphic 1-vector Kontsevich graphs built of 3 wedges over one sink evaluate, in dimension $d=2$, to only three linearly independent formulas of vector fields on $\mathbb{R}^2$.

In what follows, by evaluating Kontsevich *nonzero* graphs to the respective formulas in finite dimensions, we shall encounter (i) instant vanishings: $\phi(\Gamma)\equiv 0$ for $\Gamma\neq-\Gamma$, for a single graph Γ; (ii) *longer* linear relations that involve three or more graphs. By construction, these identities are dimension-dependent: besides, identities can be specific to the Nambu–Poisson class of bi-vectors over dimension $d<\infty$, that is not hold for arbitrary Poisson bi-vectors P.

2 Basic concept: Nambu micro-graphs over $\mathbb{R}^d_{\text{aff}}$

Definition 3. The *Nambu graph* over dimension d (here $3\leqslant d<\infty$) is the directed graph consisting of the source vertex (containing the d-vector coefficient $\varrho(\boldsymbol{x})\cdot\varepsilon^{i_1\cdots i_d}$) from which run d arrows (decorated with the summation indices $i_1,\ldots,i_d$); by convention, the 3rd, …, dth arrows head to the terminal vertices with the respective Casimirs $a_1,\ldots,a_{d-2}$, whereas the 1st and 2nd arrow, ordered Left $\prec$ Right as usual, encode the derivations of the arguments of the Nambu–Poisson bi-vector $P(\varrho,[\boldsymbol{a}])$ from Eq. (1). From the definition of the Levi-Civita symbol $\varepsilon^{i_1\cdots i_d}$ it follows that the d-tuple of outgoing arrows is wedge-ordered: a swap of any two arrows reverses the sign in front of the Nambu graph.

Nambu graphs, each realising a copy of Nambu–Poisson bracket (1), are the building blocks (i.e. subgraphs) in the *Nambu micro-graphs* over $m\geqslant 0$ sinks.[1]

Example 4. Let $d=3$; let 0,1,2 be the sinks, 3 and 4 be the Levi-Civita vertices, and 5,6 be the Casimir vertices. Then the digraph[2] $\Gamma_1=[0,1,5;2,5,6]$ is a Nambu micro-graph.

- Let $d=3$; let 0 and 1 be the sinks, 2 and 3 be the Levi-Civita vertices, and 4,5 be the Casimirs. Then the digraph $\Gamma_2=[0,1,4;3,4,5]$ is a Nambu micro-graph (with a 1-loop on vertex 3).
- Let $d=3$; let 0 and 1 be the sinks, 2 and 3 be the Levi-Civita vertices, and 4,5 be the Casimirs. Then the digraph $\Gamma_3=[0,1,4;2,4,5]$ is a Nambu micro-graph.
- Let $d=3$; let 1 and 2 be the Levi-Civita vertices and 3,4 be the Casimirs; then the digraph $\Gamma_4=[1,2,4;1,2,4]$ is *not* a Nambu micro-graph (because it is not built from the Nambu (sub)graphs: its vertices and edges are not organised into a union of whole copies of the Nambu–Poisson structure over $d=3$.

[1] We consider only finite Nambu micro-graphs; note also that 1-loops are allowed in Nambu micro-graphs.

[2] We list the target vertices of the ordered d-tuples of arrows issued from the Levi-Civita vertices, themselves ordered by a given vertex labelling.

Remark 4. Whenever the Poisson bracket at hand is Nambu (from Eq. (1)), linear combinations of Nambu micro-graphs can be obtained by magnifying the internal vertices of a Kontsevich graph under a microscope that resolves the elements ϱ against each of the Casimirs $a_1, \ldots, a_{d-2}$ in the Nambu–Poisson bi-vector. Every arrow which hit P in the Kontsevich graph now works over the elements of $P(\varrho, [\boldsymbol{a}])$ by the Leibniz rule. The Left $\prec$ Right ordering of the edge pairs from every wedge $\overset{L}{\leftarrow}\bullet\overset{R}{\rightarrow}$ for P is now inherited by the 1st and 2nd arrows in the d-tuple issued from the respective Levi-Civita vertex. However, not all Nambu micro-graphs are obtained by such Leibniz rule expansions (in particular, when some of the terms from these expansions are omitted – but not only then).

Definition 4. The *Kontsevich micro-graph* over dimension d is the (linear combination of) Nambu micro-graph(s) which is obtained from (a linear combinations of) Kontsevich's graphs by postulating the bi-vector P to be Nambu–Poisson, $P = P(\varrho, [\boldsymbol{a}])$ over $d \geqslant 3$, and then by working out all the Leibniz rules for each of the edges which acted on the vertices containing P in the originally taken Kontsevich graph(s).

Example 5. Let $d = 3$; let 0,1 be the sinks, 2 and 3 be the Levi-Civita vertices, and 4,5 be the Casimirs. Then the sum of digraphs $[0, 1, 4; 2, 3, 5] + [0, 1, 4; 4, 3, 5]$ is a Kontsevich micro-graph.

- But $\Gamma_3 \neq 0$ from Example 4 is *not* a Kontsevich micro-graph — because if it were, it would be obtained from a Kontsevich graph with a double edge; that Kontsevich graph would therefore be *zero*, i.e. equal to minus itself, whereas $\Gamma_3 \neq 0$ over $d = 3$.

Proposition 3. There exist nonzero but still vanishing (micro-)graphs.

Example 6. There are twelve vanishing Nambu micro-graphs (of them, three are zero and nine nonzero) within the set of 41 Nambu 1-vector micro-graphs, built of three Nambu (sub)graphs, which show up in the Kontsevich micro-graph expansion over $d = 3$ of the two 'sunflower' graphs Γ', Γ'', see Eq. (2) below, whose linear combination $X_{2D}^{\gamma_3}$ sufficed to trivialise the tetrahedral γ_3-flow on the space of (Poisson) bi-vectors in dimension two (cf. [10] and [1, 2, 5, 6], also [9, 13]); now over $d = 3$, these twelve (non)zero vanishing micro-graphs are listed in [9, Lemma 2].

- Again, among the 324 one-vector nambu micro-graphs which show up in the Kontsevich micro-graph expansion – now over $d = 4$ – of the 'sunflower' graphs, there are 54 vanishing micro-graphs (see the Appendix in [9]).
- Among the 21 Hamiltonians (i.e. 0-vector Nambu micro-graphs, without sinks) built of two Nambu structures over dimension $d = 4$, there is a unique nonzero vanishing graph $H_{d=4}^{\equiv 0} = [1, 2, 3, 5; 3, 4, 5, 6]$ (here 1,2 are the Levi-Civita vertices, 3 and 4 are the Casimirs a_1, and 5,6 are the Casimirs a_2, see [13, Lemma 16]). — In lower dimensions $d = 2, 3$, there are no vanishing Hamiltonians built of two (Nambu–)Poisson structures.

Definition 5. Consider a (micro-)graph Γ built from Nambu-Poisson bi-vector subgraphs over $\mathbb{R}^d$, with copies of $\varrho \cdot \varepsilon^{\vec{\imath}}$ and 'their own' Casimirs a_1, …, a_{d-2} in different vertices. Now over $\mathbb{R}^{d+1}$, let every Levi-Civita vertex $\varrho \cdot \varepsilon^{i_1 \cdots i_{d+1}}$ send a new arrow to a new terminal vertex (with 'Levi-Civita's own' new Casimir a_{d-1}) of in-degree $\equiv 1$; that is we *embed* $\Gamma \hookrightarrow \widehat{\Gamma}$ such that no Leibniz rules are reworked.

Note that in the resulting micro-graph $\widehat{\Gamma}$ with edges decorated by summation indices, the value $d + 1$ of the index on every new edge reproduces the formula of Γ times

$\big(\partial a_{d-1}/\partial x^{d+1}\big)^p$, with the power $p = \#\varrho$ in Γ — yet, in the course of summation, there appear cross-terms with $\partial a_{d-1}/\partial x^i$ with $i \leqslant d$.

Proposition 4. The only vanishing Hamiltonian $H^{\equiv 0}_{d=4}(P \otimes P) = 0$ over $\mathbb{R}^4$, when embedded into dimension five, remains vanishing: $\widehat{H}^{\equiv 0}_{d=5}(P \otimes P) = 0$.

- The embedding into dimension four remains vanishing for each of the twelve vanishing 1-vector descendants (in dimension three) of the two 'sunflower' graphs Γ', Γ'' from (2).

Definition 6 (cf. Definition 4 in §6 from [13])**.** Two topologically nonisomorphic graphs $\Gamma_1 \neq \Gamma_2$ are called *synonyms* if $\phi(\Gamma_1) = c \cdot \phi(\Gamma_2)$ with $c \in \mathbb{R} \setminus \{0\}$, that is, the two graphs provide the same multivector up to a nonzero constant.

Example 7. Over $d = 3$, consider the seven nonisomorphic 0-vector Nambu micro-graphs (i.e. Hamiltonians) built of two Nambu (sub)graphs. A pair and a triple of synonyms are displayed in [13, Eq. (4) and Lemma 11]; the remaining four formulas obtained from those seven graphs are linearly independent.

- Likewise, over $d = 4$, the 21 non-isomorphic Hamiltonians on two Nambu sub-graphs contain 8 pairs of synonyms (and one vanishing micro-graph): see [13, Eq. (5) and Lemma 16].

Proposition 5. For the seven and four synonyms of 1-vector graphs Γ' and Γ'' in the 'sunflower' $\mathfrak{X}^{\gamma_3}_{d=2} = \Gamma' + 2 \cdot \Gamma''$, the embedding of every linear relation $\Gamma'_\alpha = \Gamma'_\beta$ or $\Gamma''_r = \Gamma''_s$ (for their formulas in dimension two) into higher dimensions $d = 3$ and $d = 4$ remains a valid linear relation between the formulas of larger micro-graphs: $\widehat{\Gamma'_\alpha} = \widehat{\Gamma'_\beta}$ and $\widehat{\Gamma''_r} = \widehat{\Gamma''_s}$.

Open problem 1. Is it true that the embedding of Nambu micro-graphs always preserves linear relations between their respective formulas ?

3 If Kontsevich's flows over 2D are coboundaries, then which ones ?

Over dimension $d = 2$, every bi-vector $P = \varrho(x,y)\,\partial_x \wedge \partial_y$ is Poisson (in absence of nonzero tri-vectors $\frac{1}{2}[\![P,P]\!]$ for the left-hand side of the Jacobi identity). For the same reason, every bi-vector is a Poisson 2-cocycle. Yet the graph cocycle flows at hand are not obliged to be coboundaries because the Lichnerowicz-Poisson second cohomology does not vanish *a priori* over $d = 2$. Indeed, the structure P can degenerate on a locus inside M^2_{aff}, so that nontrivial Poisson cocycles start to exist.[3]

We recall from Claim 2 that no universal – at the level of Kontsevich graphs – trivialisation can be possible over $d \geqslant 2$ for the $\gamma_{2\ell+1}$-wheel graph flows $\dot{P} = \vec{\text{Or}}(\gamma_{2\ell+1})(P^{\otimes^{2\ell+2}})$. It is now all the more amazing that not only are these $\gamma_{2\ell+1}$-graph cocycle flows coboundaries over $d = 2$, i.e. $Q^{\gamma_{2\ell+1}}_{d=2} = [\![P(\varrho), \vec{X}^{\gamma_{2\ell+1}}_{d=2}]\!]$, but also there do exist particular solutions $X^{\gamma_i}_{d=2}$ that conjecturally provide linear combinations of (Kontsevich) micro-graphs over which solutions $\vec{X}^{\gamma_i}_{d\geqslant 2}$ appear in higher dimensions (e.g., for γ_3 and $d = 4$, see [9]).

Proposition 6 ([10, 1])**.** For the tetrahedron γ_3 on $n = 4$ vertices, the trivialising vector field $\vec{X}^{\gamma_3}_{d=2}(P \otimes P \otimes P)$ is unique modulo Hamiltonian vector fields with $H(P \otimes P)$ given by Kontsevich graphs. The formula of a particular representative $\vec{X}^{\gamma_3}_{d=2} \mod \vec{X}_{H(P\otimes P)}$ is

[3] For example, take $\varrho(x,y) := x^p y^q \cdot \varrho(x,y)$, where $p,q \gg 1$ and ϱ is smooth near the origin of $\mathbb{R}^2 \ni (x,y)$. Then every coboundary $[\![P, \vec{X}(x,y)]\!]$ also vanishes at (0,0) for all smooth vector fields $\vec{X}$ on $\mathbb{R}^2$, still there exist many bi-vectors $Q \in \mathfrak{X}^2(\mathbb{R}^2)$, hence $Q \in \ker[\![P,\cdot]\!]$, which do not vanish at (0,0), so these $Q \notin \text{im}[\![P,\cdot]\!]$ mark nontrivial Poisson 2-cocycles.

encoded by the 'sunflower' graph (see [2, App. F] and [6]),

$$X^{\gamma_3}_{d=2} = (0,1;1,3;1,2) + 2\cdot(0,2;1,3;1,2) = \Gamma' + 2\Gamma'' = \tag{2}$$  .

• The formula $\vec{X}^{\gamma_3}_{d=2} \in \mathfrak{X}^1(\mathbb{R}^2)$ of this 'sunflower' vector field is Hamiltonian (in the classical sense) with respect to the standard symplectic structure $\omega_2 = \mathrm{d}x\wedge\mathrm{d}y$ on $\mathbb{R}^2$: one can readily inspect that

$$\vec{X}^{\gamma_3}_{d=2} = \big(\mathrm{d}x\wedge\mathrm{d}y\big)^{-1}\big(\mathrm{d}_{\mathrm{deRham}}\mathrm{Ham}^{\gamma_3}(P\otimes P\otimes\varrho)\big),$$

where the formula of the Hamiltonian Ham^{γ_3} is encoded again by a graph, namely (1,3;1,2): it is built over three vertices 1,2,3 from two wedges (with tops 2 and 3); the vertex 1 is terminal, it contains ϱ.

• Under the mapping of $\mathrm{d}(\mathrm{Ham}^{\gamma_3})$ by $(\mathrm{d}x\wedge\mathrm{d}y)^{-1}$ to the vector field $\vec{X}^{\gamma_3}_{d=2}$, two edges are issued from that terminal vertex: one edge goes to the sink 0 (that is, to the argument of the 1-vector), while the other edge works by the Leibniz rule over all vertices of the Hamiltonian graph Ham^{γ_3}, whence at least one tadpole arises (specifically, in the graph Γ' of the 'sunflower').

Remark 5 ([6]). In any solution $\vec{X}^{\gamma_3}_{d=2}$, at least one tadpole is necessary.

For the graph cocycles $\gamma \in \big\{\gamma_5, \gamma_7, [\gamma_3,\gamma_5]\big\}$ beyond γ_3, the deformations $\dot{P} = \mathrm{O}\vec{\mathrm{r}}(\gamma)([P])$ of bi-vectors P over $\mathbb{R}^2$ are trivialised by vector fields $\vec{X}^{\gamma}_{d=2}([P])$, also encoded by Formality graphs; in each case, there is a particular solution $\vec{X}^{\gamma}_{d=2} = \omega_2^{-1}\big(\mathrm{d}_{\mathrm{dR}}(\mathrm{Ham}^{\gamma})\big)$ with the Hamiltonian encoded by graphs.

Proposition 7. For the pentagon-wheel graph cocycle γ_5 (see [7, 8]), the respective flow $\dot{P} = \mathrm{O}\vec{\mathrm{r}}(\gamma_5)(P^{\otimes^6}_{d=2}) =: Q^{\gamma_5}_{d=2}$ on the space of bi-vectors P over $\mathbb{R}^2$ is a coboundary, $Q^{\gamma_5}_{d=2} = [\![P, \vec{X}^{\gamma_5}_{d=2}]\!]$, with respect to the vector field $\vec{X}^{\gamma_5}_{d=2} = \omega_2^{-1}(\mathrm{d}_{\mathrm{dR}}(\mathrm{Ham}^{\gamma_5}))$ built of five wedges.

• The Hamiltonian Ham^{γ_5} is built of four wedges – their tops in the vertices 2,3,4,5 – and one terminal vertex 1 (containing $\varrho(x,y)$) in each of its three graphs; its encoding is

$$\mathrm{Ham}^{\gamma_5} = 6\cdot[3,5;4,5;1,2;1,2] - 2\cdot[3,5;4,5;1,2;1,4] - 2\cdot[1,5;1,4;1,2;1,3].$$

• The trivialising vector field $\vec{X}^{\gamma_5}_{d=2} = \phi(X^{\gamma_5}_{d=2})$ is encoded by $15 = 3\times 5$ graphs built of five wedges over one sink 0 in each term: the encodings for each of the three 5-tuples of 1-vector graphs are obtained by issuing the wedge from vertex 1, namely by sending its Left arrow to the sink 0 and by letting the Right edge (1,i) run over all the aerial vertices $i\in\{1,2,3,4,5\}$ (so that the edge (1,1) is the tadpole).

• This solution $\vec{X}^{\gamma_5}_{d=2}$ of the γ_5-flow trivialisation problem $Q^{\gamma_5}_{d=2} = [\![P, \vec{X}^{\gamma_5}_{d=2}]\!]$ over $d=2$ is unique modulo Poisson vector fields $\vec{X}_H = [\![P,H]\!]$ with Hamiltonians $H(P^{\otimes^4})$ encoded by Kontsevich graphs on four wedges.

Remark 6. For the heptagon-wheel graph cocycle γ_7 from [7], a solution $\vec{X}^{\gamma_7}_{d=2} = \omega_2^{-1}(\mathrm{d}_{\mathrm{dR}}\mathrm{Ham}^{\gamma_7})$ of the trivialisation problem $\mathrm{O}\vec{\mathrm{r}}(\gamma_7)(P^{\otimes^8}_{d=2}) = [\![P, \vec{X}^{\gamma_7}_{d=2}]\!]$ over $\mathbb{R}^2$ is known from [3, §6.4].

The graph commutator $[\gamma_3,\gamma_5]$, itself not a $(2\ell+1)$-wheel generator of $\mathfrak{grt}$, acts on bi-vectors P over $\mathbb{R}^2$ in a similar way: $\mathrm{O}\vec{\mathrm{r}}\big([\gamma_3,\gamma_5]\big)(P^{\otimes^9}_{d=2}) = [\![P, \vec{X}^{[\gamma_3,\gamma_5]}_{d=2}]\!]$ with $\vec{X}^{[\gamma_3,\gamma_5]}_{d=2} = \omega_2^{-1}(\mathrm{d}_{\mathrm{dR}}\mathrm{Ham}^{[\gamma_3,\gamma_5]})$ built of eight wedges.[4]

[4] R. Buring, private communication (14 May 2024).

However, in both the cases (for γ_7 and for $[\gamma_3, \gamma_5]$), the respective Hamiltonians, referred to the standard symplectic structure ω_2 on $\mathbb{R}^2$, were obtained at the level of homogeneous differential polynomials in ϱ, that is, not yet at the level of Formality graphs built only of wedges and one terminal vertex — in contrast with Propositions 6 and 7 where we make that graph realisation explicit.

Open problem 2. Is it true that in dimension $d = 2$, for each graph cocycle γ from the Grothendieck–Teichmüller Lie algebra $\mathfrak{grt}$ generated by the $(2\ell+1)$-wheel cocycles $\gamma_{2\ell+1}$, the γ-flow trivialisation problem always has a solution of the shape $\vec{X}^{\gamma}_{d=2} = \omega_2^{-1}\big(\mathrm{dHam}^{\gamma}\big)$, where, moreover, the directed graphs in the 0-vector Ham^{γ} are built of 2ℓ wedges (for copies of P) and one terminal vertex (with $\varrho(x, y)$)?

Remark 7. The 'sunflower' graph (2) is special: on its $(d = 3, 4)$-descendants, i.e. on the set of Kontsevich micro-graphs which appear from Kontsevich's two graphs in the 'sunflower', there exist a solution in dimension three and a solution in dimension four (see [9]). In the subsequent paper [13], by running over the synonyms of either graph in the 'sunflower' solution of the trivialisation problem at $d = 2$, we detect that this effect is not generic: over $(d \geqslant 3)$-descendants of the synonyms, solutions typically cease to exist.

4 Kontsevich graph flows of Nambu–Poisson brackets over $\mathbb{R}^{\geqslant 3}$

The Kontsevich graph cocycles γ on n vertices and $2n-2$ edges act on the space $\mathfrak{X}^2(\mathbb{R}^2)$ of bi-vectors over affine spaces $\mathbb{R}^d$ of any dimension $d \geqslant 2$; the graph flows $\dot{P} = \mathrm{O}\vec{\mathrm{r}}(\gamma)(P^{\otimes^n})$ preserve the subset of all *Poisson* bi-vectors P satisfying the Jacobi identity $\frac{1}{2}[\![P, P]\!] = 0$. Let us study whether in the set of *all* Poisson bi-vectors, Kontsevich's graph flows preserve the class $\{P(\varrho, [\boldsymbol{a}])\}$ of Nambu-determinant Poisson structures on $\mathbb{R}^d$.

Definition 7. The class of Nambu–Poisson brackets $P(\varrho, [\boldsymbol{a}])$ on $\mathbb{R}^d$, $d \geqslant 3$, is preserved by a flow $\frac{d}{d\varepsilon}(P) = Q([P])$ if there exist, for all $\varrho \cdot \partial_{\boldsymbol{x}} \in \mathfrak{X}^d(\mathbb{R}^d)$ and Casimirs $a_i \in C^{\infty}(\mathbb{R}^d)$ simultaneously, the evolution equations $\frac{d}{d\varepsilon}(\varrho) = R([\varrho], [\boldsymbol{a}])$ and $\frac{d}{d\varepsilon}(a_i) = A_i([\varrho], [\boldsymbol{a}])$ such that the evolution of Nambu bi-vector $P\big(\varrho, [\boldsymbol{a}]\big)$ along $Q([P])$ amounts to the Leibniz rule for $d/d\varepsilon$ acting on its components:

$$\tfrac{d}{d\varepsilon}\Big(P\big([\varrho], [\boldsymbol{a}]\big)\Big) = Q([P]) = P\big(\tfrac{d}{d\varepsilon}\varrho, [\boldsymbol{a}]\big) + \sum\nolimits_{i=1}^{d-2} P\big(\varrho, [a_1], \ldots, [\tfrac{d}{d\varepsilon}a_i], \ldots, [a_{d-2}]\big), \quad (3)$$

that is, the evolutions of $P(\varrho, [\boldsymbol{a}])$ and of its elements, ϱ and Casimirs a_i, match.

Example 8 ([5])**.** The γ_3-deformation restricts to the class of Nambu-determinant Poisson bi-vectors on (at least) $\mathbb{R}^3$ and $\mathbb{R}^4$. The same is true also for the graph cocycle γ_5 and its action on the Nambu–Poisson class of brackets (1) over $\mathbb{R}^3$.

Conjecture 8 (see [5])**.** Consider the Kontsevich γ-cocycle deformation $\dot{P} = \mathrm{O}\vec{\mathrm{r}}(\gamma)(P^{\otimes^n})$, where n is the number of vertices in each term of γ and $2n-2$ is the number of edges, and assume that this flow $\dot{P} = Q^{\gamma}([P])$ does restrict to the flow Q^{γ}_d on the class of Nambu–Poisson bi-vectors $P\big(\varrho, [\boldsymbol{a}]\big)$ over $\mathbb{R}^d$ for some $d \geqslant 3$. By definition, put (with reference of tuples of arguments to vertices of each term in the graph cocycle γ):

$$\tfrac{d}{d\varepsilon}a_i = \mathrm{O}\vec{\mathrm{r}}(\gamma)(a_i \otimes P^{\otimes^{n-1}}) + \mathrm{O}\vec{\mathrm{r}}(\gamma)(P \otimes a_i \otimes P^{\otimes^{n-2}}) + \ldots + \mathrm{O}\vec{\mathrm{r}}(\gamma)(P^{\otimes^{n-1}} \otimes a_i).$$

Then, the conjecture is that the fraction,

$$\tfrac{d}{d\varepsilon}\varrho = \frac{\Big(Q^\gamma_d([P]) - \sum_{i=1}^{d-2} P\big(\varrho, [a_1], \dots, [\tfrac{d}{d\varepsilon}a_i], \dots, [a_{d-2}]\big)\Big)(f,g)}{\det\Big(\partial\big(f, g, a_1, \dots, a_{d-2}\big)/\partial\big(x^1, \dots, x^d\big)\Big)},$$

is differential polynomial in ϱ and a_i, so that Leibniz rule (3) tautologically holds.

Example 9. The above conjecture is confirmed to be true for the tetrahedron graph cocycle γ_3 and dimensions $d = 3, 4$, and for the pentagon-wheel graph cocycle γ_5 and Nambu–Poisson structures over $d = 3$.

In the rest of this section we discuss the observed trivialisation of Kontsevich's graph cocycle flows $\dot P(\varrho, [\boldsymbol{a}]) = Q^\gamma_d([P])$ in the Lichnerowicz–Poisson second cohomology w.r.t. $[\![P(\varrho, [\boldsymbol{a}]), \cdot]\!]$, that is, we recall some evidence for the existence of vector field solutions $\vec X^\gamma_d([\varrho], [\boldsymbol{a}])$ for the equations $Q^\gamma_d([P]) = [\![P, \vec X^\gamma_d]\!]$. (The known solutions $\vec X^{\gamma_3}_{d=3,4}$ are encoded by the Nambu micro-graphs but not by Kontsevich micro-graphs, as they do not stem directly from the previously known solutions $\vec X^{\gamma_3}_{d=2}$, see papers [9, 13].)

Lemma 9. In any dimension $d \geqslant 3$, Nambu–Poisson bi-vectors in (1) are odd w.r.t. every Casimir a_i, namely $P(\varrho, \dots, [-a_i], \dots) = -P(\varrho, \dots, [a_i], \dots)$.

- For all $d \geqslant 4$, Nambu–Poisson bi-vectors in (1) are totally antisymmetric w.r.t. permutations $\sigma \in \mathbb{S}_{d-2}$ of the set of Casimirs $\boldsymbol{a}$: we have $P(\varrho, [\sigma(\boldsymbol{a})]) = (-)^\sigma \cdot P(\varrho, [\boldsymbol{a}])$.

Remark 8. In contrast with the above lemma, the graph cocycle generators $\gamma_{2\ell+1}$ of $\mathfrak{grt}$ consist of $(2\ell+1)$-wheels and other graphs on $2\ell+2$ vertices; this number is even, whence $Q^{\gamma_i}_{d\geqslant 3}(P(\varrho, \dots, [-a_i], \dots)) \equiv Q^{\gamma_i}_{d\geqslant 3}(P(\varrho, \dots, [a_i], \dots))$ and likewise, $Q^{\gamma_i}_{d\geqslant 4}(P(\varrho, \sigma(\boldsymbol{a}))) \equiv Q^{\gamma_i}_{d\geqslant 4}(P(\varrho, [\boldsymbol{a}]))$ for all $\sigma \in \mathbb{S}_{d-2}$. (The reasoning does not work for $[\gamma_3, \gamma_5]$ on 9 vertices and for other (iterated) commutators on an odd number of vertices.) This reveals that Kontsevich's graph cocycles γ_i, acting on Nambu–Poisson bi-vectors by infinitesimal deformations $\dot P = Q^{\gamma_i}_d([P])$, at once lose the structural property of these brackets.[5]

This loss of structural property of $P(\varrho, [\boldsymbol{a}])$ by $Q^{\gamma_i}_d([P])$ is an indirect but strong evidence that these graph cocycle flows are coboundaries over all $d \geqslant 3$ for each γ_i.

Example 10 (see [6, 9])**.** The tetrahedral flow $\dot P = Q^{\gamma_3}_d([P])$ is a Poisson coboundary for the class of Nambu–Poisson brackets (1) over $d = 3$ and $d = 4$.

Proposition 10. Along any vector field $\vec Y = -\vec X \in \mathfrak{X}^1(\mathbb{R}^{d\geqslant 3})$ on $\mathbb{R}^d$, the scalar functions a_i evolve as fast as $\dot a_i = (-\vec X)(a_i)$, and the evolution of $\varrho\cdot\partial_{\boldsymbol{x}} \in \mathfrak{X}^d(\mathbb{R}^d)$ is $\dot\varrho\,\partial_{\boldsymbol{x}} = [\![\varrho\partial_{\boldsymbol{x}}, \vec X]\!]$, which is standard. Now, the found vector fields $\vec X^{\gamma_3}_d$ trivialising the tetrahedral γ_3-flows of Nambu brackets (1) over $\mathbb{R}^3$ and $\mathbb{R}^4$ are such that

$$\tfrac{d}{d\varepsilon}a_i = \big(-\vec X^{\gamma_3}_d\big)(a_i) \qquad \text{and} \qquad \tfrac{d}{d\varepsilon}(\varrho)\,\partial_{\boldsymbol{x}} = [\![\varrho\,\partial_{\boldsymbol{x}}, \vec X^{\gamma_3}_d]\!]. \tag{4}$$

In other words, the evolution of Casimirs, obtained directly from the graph cocycle γ_3 (see Conjecture 8 and Example 9), and the evolution of d-vector $\varrho\,\partial_{\boldsymbol{x}}$, read from the γ_3-

[5] In fact, the Kontsevich deformation bi-vectors $Q^{\gamma_i}_d$ are well defined for the symplectic foliation of $\mathbb{R}^d$, no matter how it is described by the level sets of the Casimirs a_i (or of $-a_i$) or of their permutations, because the level sets of linear combinations $A\boldsymbol{a}$ define the same loci if $\det(A) \neq 0$.

deformation of the Nambu bi-vector $P\big(\varrho,[\boldsymbol{a}]\big)$, agree with the law of evolution of zero- and d-vectors along the vector field which trivialises the γ_3-flow.[6]

Conclusion. For the infinitesimal deformations $\dot{P}=Q([P])$ of Poisson bi-vectors P, the calculus of multivectors using Kontsevich and Nambu (micro-)graphs turns the PDE problem of deformations' (non)triviality in the Poisson cohomology into a problem from linear algebra. Yet the evaluation map ϕ, acting from (Nambu micro-)graphs Γ to poly-linear polydifferential operators and then, by antisymmetrisation, to multivectors $\phi(\mathrm{Alt}(\Gamma))\in\mathfrak{X}^k(M^d_{\mathrm{aff}})$, does have a nontrivial kernel, whence stem vanishing graphs, synonyms, and longer linear relations $\phi\big(\sum_i c_i\Gamma_i\big)=0$. The formulas which ϕ produces from (micro-)graphs are well defined w.r.t. affine changes $\mathbf{x}'(\boldsymbol{x})\rightleftarrows\boldsymbol{x}(\mathbf{x}')$ locally on M^d_{aff}. We pose the problem of effective description of multivector-valued invariants of the affine (essentially, only of $GL(d)$) group action on tensor fields over M^d_{aff}, so that the new kernel is as small as possible.

It remains unclear why the Nambu class $\big\{P(\varrho,[\boldsymbol{a}])\big\}$ is preserved by Kontsevich's graph cocycles γ, namely why the cocycles γ yield the genuine evolution of Casimirs $\boldsymbol{a}$ and why the evolution of $\varrho\,\partial_{\boldsymbol{x}}$ is then well defined from $Q^\gamma_d([P])$. The underlying mechanics of cross-terms cancellation looks similar to the noted preservation of identities $\phi\big(\sum_i c_i\Gamma_i\big)=0$ by the graph embeddings $\Gamma\hookrightarrow\widehat{\Gamma}$ into dimension $d+1$. By understanding the nature of (both) the mechanism(s), we shall gain deeper insight into the algebra and combinatorial topology of $GL(d)$-invariants.

We see that Nambu–Poisson brackets resist the Kontsevich graph action. We detect that the flows $\dot{P}=Q^\gamma_d([P(\varrho,[\boldsymbol{a}])])$ are Poisson coboundaries: $Q^\gamma_d=[\![P,\vec{X}^\gamma_d]\!]$, but the vector fields $\vec{X}^\gamma_{d\geqslant 3}$ are not obtained from $d=2$ by mere expansion of Leibniz rules. The choice of Nambu micro-graphs for a solution $\vec{X}^\gamma_d$ to appear is not yet codified; our preference of *the most natural* Ansatz for $\vec{X}^{\gamma_3}_{d=3,4}$ in [9] is intuitive.

Acknowledgements. The authors thank the organisers of the international conference on Integrable Systems & Quantum Symmetries (ISQS28) held on 1–5 July 2024 at CVUT Prague, Czech Republic, for stimulating discussions. The authors are grateful to R. Buring (INRIA Saclay, France) for the availability of gcaops software and support. The authors thank the Center for Information Technology of the University of Groningen for their support and for providing access to the Hábrók high performance computing cluster. The authors thank the University of Groningen for partial financial support.

A part of this research was done while AVK was visiting at the IHÉS, supported in part by the Nokia Fund. AVK thanks the IHÉS for hospitality, and thanks M. Kontsevich for helpful discussions and advice.

References

[1] Bouisaghouane A 2017 The Kontsevich tetrahedral flow in 2D: a toy model *arXiv preprint* arXiv:1702.06044 [math.DG]

[2] Bouisaghouane A, Buring R and Kiselev A V 2017 The Kontsevich tetrahedral flow revisited *J. Geom. Phys.* **19** 272–285

[3] Buring R 2022 The Kontsevich graph complex action on Poisson brackets and star-products: an implementation *PhD thesis* Johannes Gutenberg–Universität Mainz

[6] At $d=3$, equality (4) is inspected after the trivialising vector field $\vec{X}^{\gamma_3}_{d=3}$ is found by solving the equation $Q^{\gamma_3}_{d=3}([P])=[\![P,\vec{X}^{\gamma_3}_{d=3}([P])]\!]$. At $d=4$, equations (4) are solved for $\vec{X}^{\gamma_3}_{d=4}$, and then the equality $Q^{\gamma_3}_{d=4}([P])=[\![P,\vec{X}^{\gamma_3}_{d=4}([P])]\!]$ is confirmed. Likewise, for the pentagon-wheel cocycle γ_5, Eqs (4) over $\mathbb{R}^3$ are solved first.

[4] Buring R and Kiselev A V 2019 The orientation morphism: from graph cocycles to deformations of Poisson structures *J. Phys.: Conf. Ser.* **1194** 012017 (*Preprint* arXiv:1811.07878 [math.CO])

[5] Buring R, Kiselev A V and Lipper D 2022 The hidden symmetry of Kontsevich's graph flows on the spaces of Nambu-determinant Poisson brackets *Open Communications in Nonlinear Mathematical Physics* **2** Paper ocnmp:8844 186–216

[6] Buring R and Kiselev A V 2023 The tower of Kontsevich deformations for Nambu–Poisson structures on $\mathbb{R}^d$: Dimension-specific micro-graph calculus *SciPost Phys. Proc.* **14** Paper 020 1–11

[7] Buring R, Kiselev A V and Rutten N J 2017 The heptagon-wheel cocycle in the Kontsevich graph complex *J. Nonlin. Math. Phys.* **24** Suppl 1 Local & Nonlocal Symmetries in Mathematical Physics 157–173

[8] Buring R, Kiselev A V and Rutten N J 2018 Poisson brackets symmetry from the pentagon-wheel cocycle in the graph complex *Physics of Particles and Nuclei* **49**:5 Supersymmetry and Quantum Symmetries'2017 924–928

[9] Jagoe Brown M S, Schipper F and Kiselev A V 2024 Kontsevich graphs act on Nambu–Poisson brackets, II. The tetrahedral flow is a coboundary in 4D *arXiv preprint* arXiv:2409.12555 [math.QA]

[10] Kontsevich M 1997 Formality conjecture *Deformation theory and symplectic geometry (Ascona 1996)* 139–156

[11] Kontsevich M 2003 Deformation quantization of Poisson manifolds *Lett. Math. Phys.* **66**:3 157–216 (*Preprint* arXiv:q-alg/9709040)

[12] Nambu Y 1973 Generalized Hamiltonian dynamics *Phys. Rev.* **D7** 2405–2412

[13] Schipper F, Jagoe Brown M S and Kiselev A V 2024 Kontsevich graphs act on Nambu–Poisson brackets, III. Uniqueness aspects *arXiv preprint* arXiv:2409.15932 [math.QA]

[14] Takhtajan L 1994 On foundation of the generalized Nambu mechanics *Comm. Math. Phys.* **160**:2 295–315

[15] Willwacher T 2015 M. Kontsevich's graph complex and the Grothendieck–Teichmüller Lie algebra *Invent. Math.* **200**:3 671–760 (*Preprint* arXiv:1009.1654 [q-alg])

Kontsevich graphs act on Nambu–Poisson brackets, II. The tetrahedral flow is a coboundary in 4D

Mollie S Jagoe Brown, Floor Schipper and Arthemy V Kiselev

Bernoulli Institute for Mathematics, Computer Science and Artificial Intelligence, University of Groningen, P.O. Box 407, 9700 AK Groningen, The Netherlands

E-mail: m.s.jagoe.brown@gmail.com, f.m.schipper@rug.nl, a.v.kiselev@rug.nl

Abstract. Kontsevich constructed a map from suitable cocycles in the graph complex to infinitesimal deformations of Poisson bi-vector fields. Under the deformations, the bi-vector fields remain Poisson. We ask, are these deformations trivial, meaning, do they amount to a change of coordinates along a vector field? We examine this question for the tetrahedron, the smallest nontrivial suitable graph cocycle in the Kontsevich graph complex, and for the class of Nambu–Poisson brackets on $\mathbb{R}^d$.

Within Kontsevich's graph calculus, we use dimension-specific micro-graphs, in which each vertex represents an ingredient of the Nambu–Poisson bracket. For the tetrahedron, Kontsevich knew that the deformation is trivial for $d = 2$ (1996). In 2020, Buring and the third author found that the deformation is trivial for $d = 3$. Building on these discoveries, we now establish that the deformation is trivial for $d = 4$.

1 Introduction

Take an arbitrary Poisson geometry: energy is transformed into motion by a Poisson bracket. We ask: is this model isolated, or part of a larger family? To examine this, we deform the Poisson bracket. If the deformation simply amounts to a change of coordinates of the model, then the system is isolated with respect to that deformation. If the deformation causes a nontrivial change of the Poisson bracket, then we say that the system is one in a larger family of systems. The incoming energy has stayed the same, but the outgoing motion has changed, see Chapter 13 on Deformation Quantization in [1].

Kontsevich constructed a map from 'good graphs' γ in the Kontsevich graph complex, a differential graded Lie algebra of non-directed graphs, to bi-vector field flows $\dot{P} = Q^\gamma_d(P)$, which express the deformation of the Poisson bi-vector field P on $\mathbb{R}^d$ by γ, see [2]. The graphs γ are cocycles, meaning that for the differential $\mathsf{d} = [\bullet\!\!-\!\!\bullet, \cdot]$ in the Kontsevich graph complex, we have

$$\mathsf{d}(\gamma) = [\bullet\!\!-\!\!\bullet, \gamma] = 0.$$

The associated bi-vector field flow $\dot{P} = Q^\gamma_d(P)$ is a Poisson cocycle for the differential $\partial_P = [\![P, \cdot]\!]$, meaning

$$\partial_P\big(Q^\gamma_d(P)\big) = [\![P, Q^\gamma_d(P)]\!] = 0.$$

We enquire if $Q^\gamma_d(P)$ is a coboundary, which would mean there exists some trivialising vector field $\vec{X}^\gamma_d(P)$ such that $Q^\gamma_d(P) = [\![P, \vec{X}^\gamma_d(P)]\!]$, which implies that the deformation $\dot{P} = Q^\gamma_d(P)$ is trivial; the coordinates change along the vector field $\vec{X}^\gamma_d(P)$. We specifically deform the class of Nambu–Poisson brackets by the

simplest nontrivial graph in the Kontsevich graph complex, the tetrahedron γ_3. In this paper, we present the solution $\vec{X}^{\gamma_3}_{d=4}(P(\varrho,\boldsymbol{a}))$ found in dimension $d=4$ for Nambu–Poisson brackets $P(\varrho,\boldsymbol{a})$.

The authors recommend the following reading order of proceedings, by names of first authors: Kiselev [3], Jagoe Brown (this paper), and finally, Schipper [4].

This paper is structured in the following way. In section 2, we introduce the preliminaries necessary to approach the problem, then formulate it. In section 3, we examine the solution to the problem in dimensions two, three, and finally, dimension four, while presenting a series of simplifications which were crucial to obtain the new solution in dimension four. In section 4, we discuss the up-down behaviour of the problem from one dimension to another, and finally conclude.

2 Preliminaries

2.1 Basic concept

The theory behind this problem is due to Kontsevich, and is applicable to any class of Poisson bracket on an affine manifold, in any dimension. Recall that we can express any Poisson bracket in terms of a bi-vector field, in the following way:

$$\{f,g\} = P(f,g).$$

Deforming a Poisson bi-vector field P by a suitable graph cocycle γ in the Kontsevich graph complex is expressed as

$$\dot{P} = Q^{\gamma}(P),$$

where $Q^{\gamma}(P)$ is an infinitesimal symmetry built of as many copies of P as there are vertices in γ, see [6].

The setting of the problem is $\mathbb{R}^d$, with Cartesian coordinates given by $\mathbb{R}^d \ni \boldsymbol{x} = (x_1, x_2, ..., x_d)$. We deform the class of Nambu–Poisson brackets by the tetrahedron γ_3.

Definition 1 (Nambu–Poisson bracket)**.** The generalised Nambu-determinant Poisson bracket in dimension d for two smooth functions $f, g \in C^\infty(\mathbb{R}^d)$ is given as

$$\{f,g\}_d(\boldsymbol{x}) = \varrho(\boldsymbol{x}) \cdot \det \begin{pmatrix} f_{x_1} & g_{x_1} & a^1_{x_1} & a^2_{x_1} & \dots & a^{d-2}_{x_1} \\ f_{x_2} & g_{x_2} & a^1_{x_2} & a^2_{x_2} & \dots & a^{d-2}_{x_2} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ f_{x_d} & g_{x_d} & a^1_{x_d} & a^2_{x_d} & \dots & a^{d-2}_{x_d} \end{pmatrix} (\boldsymbol{x}),$$

where $a^1, ..., a^{d-2} \in C^\infty(\mathbb{R}^d)$ are Casimirs, which Poisson-commute with any function. The function ϱ is the inverse density, or the coefficient of a d-vector field.

Definition 2 (The tetrahedron γ_3)**.** The tetrahedron γ_3 is the smallest nontrivial suitable graph cocycle of the Kontsevich graph complex, a differential graded Lie algebra of non-directed graphs. By γ_3 being a cocycle, we mean that it satisfies $\mathsf{d}(\gamma_3) = 0$, where $\mathsf{d} = [\bullet\!\!-\!\!\bullet, \cdot]$ is the differential in the Kontsevich graph complex. The graph γ_3 is constructed on 4 vertices and 6 edges.

Definition 3 (The γ_3-flow)**.** The γ_3-flow $Q^{\gamma_3}(P)$ is a bi-vector field built with four copies of P. It is an infinitesimal symmetry of the Jacobi identity for P, and corresponds to the deformation of P by γ_3. It is obtained from γ_3 via the orientation morphism described in [6]; the formula is given in [7].

Problem formulation. To inspect whether the γ_3-flow is Poisson-trivial, we investigate if $Q^{\gamma_3}_d(P)$ is a 2-coboundary. This is equivalent to establishing the existence of a trivialising vector field $\vec{X}^{\gamma_3}_d(P)$ such that

$$\dot{P} = Q^{\gamma_3}_d(P) = [\![P, \vec{X}^{\gamma_3}_d(P)]\!], \tag{1}$$

where d is the dimension and $[\![\cdot,\cdot]\!]$ is the Schouten bracket[1]. We wish to solve equation (1) in $d=4$.

We used software package `gcaops`[2] (**G**raph **C**omplex **A**ction **O**n **P**oisson **S**tructures) for SageMath by Buring. With it, we input graph encodings, from which we obtained Formality graphs and then their formulas. We created a linear algebraic system and solved it for coefficients in the linear combination of graphs that encode $\vec{X}^{\gamma_3}_d(P)$. We used the High Performance Computing cluster at the University of Groningen, Hábrók. All code which gave results in this paper can be found as additional material attached to this paper and to [4].

[1] The Schouten bracket $[\![\cdot,\cdot]\!]$ is a unique extension of the commutator $[\cdot,\cdot]$ on the space of vector fields to the space of polyvector fields. By definition, the Schouten bracket coincides with the Lie bracket when evaluated on 1-vectors. When evaluated on p-vector X, q-vector Y and r-vector Z, the Schouten bracket satisfies the equations $[\![X,Y]\!] = -(-1)^{(p-1)(q-1)}[\![Y,X]\!]$ and $[\![X, Y\wedge Z]\!] = [\![X,Y]\!]\wedge Z + (-1)^{q(p-1)} Y \wedge [\![X,Z]\!]$, see [8].

[2] `https://github.com/rburing/gcaops`

2.2 Notation
We solve equation (1) on the level of formulas, using Kontsevich's graph calculus to write them. For this, we introduce the graph language created by Kontsevich, commonly used in deformation quantisation. Its main convenience is that formulas change with the dimension, but pictures of graphs do not change. We specifically use this graph language for graphs built of wedges $\overset{L}{\longleftarrow}\bullet\overset{R}{\longrightarrow}$, which are Poisson bi-vector fields. The directed edges are derivations which act on the content of vertices. To write the graph encodings up to and including dimension four, we use the following convention:

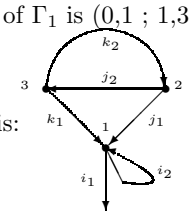
- 0 represents the sink,
- 1, 2, 3 represent Levi–Civita symbols,
- 4, 5, 6 represent Casimirs a^1,
- 7, 8, 9 represent Casimirs a^2.

We denote by ϕ the map of Formality graphs to their formulas obtained by Kontsevich's graph language.

Example 1. Let us take the following graphs Γ_1 and Γ_2.

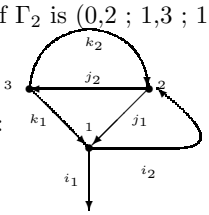
The encoding of Γ_1 is (0,1 ; 1,3 ; 1,2).

The graph Γ_1 is:



The encoding of Γ_2 is (0,2 ; 1,3 ; 1,2).

The graph Γ_2 is:



Let the dimension be two. The inert sums of the formulas for the graphs are as follows.

$$\phi(\Gamma_1) = \sum_{\substack{i_1,i_2,\\ j_1,j_2,\\ k_1,k_2=1}}^{d=2} \varepsilon^{i_1 i_2}\cdot\varepsilon^{j_1 j_2}\cdot\varepsilon^{k_1 k_2}\cdot\partial_{i_2 j_1 k_1}(\varrho)\cdot\partial_{k_2}(\varrho)\cdot\partial_{j_2}(\varrho)\cdot\partial_{i_1}(\ )$$

$$\phi(\Gamma_2) = \sum_{\substack{i_1,i_2,\\ j_1,j_2,\\ k_1,k_2=1}}^{d=2} \varepsilon^{i_1 i_2}\cdot\varepsilon^{j_1 j_2}\cdot\varepsilon^{k_1 k_2}\cdot\partial_{k_1 j_1}(\varrho)\cdot\partial_{i_2 k_2}(\varrho)\cdot\partial_{j_2}(\varrho)\cdot\partial_{i_1}(\ )$$

The sums are constructed by taking the product of the content of vertices, which contain ϱ. The arrows act on vertices as derivations[3]. The Levi–Civita symbol encodes the determinant in the Nambu–Poisson bracket, see Definition 1.

Definition 4 (The sunflower graph)**.** A linear combination of the above Kontsevich graphs (graphs built of wedges $\overset{L}{\longleftarrow}\bullet\overset{R}{\longrightarrow}$, see [3, 4, 6]) can be expressed as the sunflower graph

$$\text{sunflower } = \quad = 1\cdot\Gamma_1 + 2\cdot\Gamma_2.$$

The outer circle means that the outgoing arrow acts on the three vertices via the Leibniz rule. When the arrow acts on the upper two vertices, we obtain two isomorphic graphs, hence the coefficient 2 in the linear combination.

3 Vector fields trivialising the γ_3-flow in 2D, 3D, and now, in 4D

3.1 The trivialising vector field for γ_3-flow in 2D expressed by Kontsevich graphs
In 2D, any Poisson bracket is a Nambu–Poisson bracket because it is given by

$$\{f,g\}_{d=2}(x,y) = \varrho(x,y)\cdot\det\begin{pmatrix} f_x & g_x \\ f_y & g_y\end{pmatrix}(x,y), \text{ that is, } \quad \{f,g\}_{d=2}(x,y) = \sum_{i,j=1}^{d=2}\varepsilon^{ij}\cdot\varrho\cdot\partial_i(f)\cdot\partial_j(g),$$

for some ϱ, where i, j are indices, ε^{ij} is the Levi–Civita symbol, and the Cartesian coordinates are expressed as $x_1 = x$, $x_2 = y$.

[3] We denote ∂_i to mean the partial derivative with respect to x_i, represented by the arrow i; ∂_{ij} is the partial derivative with respect to x_i and x_j, so $\partial_{ij} = \partial_i\partial_j$.

Proposition 1 (Cf. [2], [8])**.** There exists a trivialising vector field in 2D for the γ_3-flow. It is given by the sunflower graph

$$\vec{X}^{\gamma_3}_{d=2}(P) = \phi(\text{sunflower}).$$

The sunflower gives a formula in 2D to solve equation (1), namely $\dot{P} = Q^{\gamma_3}_{d=2}(P) = [\![P, \vec{X}^{\gamma_3}_{d=2}(P)]\!]$. The affine space of solutions on graphs is of dimension 1; that is, the trivialising vector field $\vec{X}^{\gamma_3}_{d=2}(P)$ is unique up to 1-dimensional shifts, themselves encoded by Kontsevich graphs. A full descriptions of these shifts using graphs can be found in [4].

Remark 1. This solution, the sunflower, when extended to higher dimensions $d > 2$, is not a solution.

3.2 The trivialising vector field for γ_3-flow in 3D expressed by Nambu micro-graphs
In 3D, with Cartesian coordinates expressed as $x_1 = x$, $x_2 = y$, $x_3 = z$, the Nambu–Poisson bracket is given by

$$\{f, g\}_{d=3}(x, y, z) = \varrho(x, y, z) \cdot \det \begin{pmatrix} f_x & g_x & a^1_x \\ f_y & g_y & a^1_y \\ f_z & g_z & a^1_z \end{pmatrix} (x, y, z),$$

that is,

$$\{f, g\}_{d=3}(x, y, z) = \sum_{i,j,k=1}^{d=3} \varepsilon^{ijk} \cdot \varrho \cdot \partial_i(f) \cdot \partial_j(g) \cdot \partial_k(a^1),$$

for some ϱ, where i, j, k are indices and ε^{ijk} is the Levi–Civita symbol which encodes the determinant.

Definition 5 (Nambu micro-graph)**.** Nambu micro-graphs are built using the Nambu–Poisson bracket $P(\varrho, \boldsymbol{a})$ as subgraphs with ordered and directed edges. The vertex of the source of each $P(\varrho, \boldsymbol{a})$ in dimension d contains $\varepsilon^{i_1 \cdots i_d} \varrho$, with d-many outgoing edges. The first two edges act on the arguments of that bi-vector field subgraph, and the last $d-2$ edges go to the Casimirs $a^1, ..., a^{d-2}$.

Example 2. Replacing the wedges of the sunflower graph with Nambu building blocks of any dimension gives an example of a Nambu micro-graph.

Non-example 3. The encoding (0,1 ; 1,3,5 ; 2) is not a Nambu micro-graph. This can immediately be seen by the fact that the vertices do not have the same number of outgoing arrows.

Instead of searching for a trivialising vector field $\vec{X}^{\gamma_3}_{d=3}(\varrho, \boldsymbol{a})$ over all 1-vector micro-graphs built of 3 Levi–Civita symbols and 3 Casimirs a^1, we restrict our scope.

Simplification 1. Search for solutions $\vec{X}^{\gamma_3}_{d=3}(P(\varrho, \boldsymbol{a}))$ over the formulas given by Nambu micro-graphs.

Definition 6 (d-descendants)**.** The d-descendants of a given ($d' = 2$)-dimensional Kontsevich graph is the set of Nambu micro-graphs obtained in the following way. Take a ($d' = 2$)-dimensional Kontsevich graph. To each vertex, add $(d-2)$-many Casimirs by $(d-2)$-many outgoing edges. Extend the original incoming arrows to work via the Leibniz rule over the newly added Casimirs [4].

Example 4. We give the encodings of the 3D-descendants of the 2D sunflower. Recall that the sink is denoted by 0, and the Levi–Civita symbols by 1, 2, 3. In 3D, we have the three Casimirs a^1 denoted by 4, 5, 6. Then,

$$\text{sunflower } = \quad = 1 \cdot \Gamma_1 + 2 \cdot \Gamma_2 = 1 \cdot (0,1 \ ; \ 1,3 \ ; \ 1,2) + 2 \cdot (0,2 \ ; \ 1,3 \ ; \ 1,2)$$

gives

$$\text{3D-descendants } = \sum_{\substack{i_1,i_2 \in \{1,4\} \\ j \in \{3,6\} \\ k \in \{2,5\}}} (0,1,4 \ ; \ i_1,j,5 \ ; \ i_2,k,6) \ + \sum_{\substack{i_1,i_2 \in \{1,4\} \\ j \in \{3,6\} \\ k_1,k_2 \in \{2,5\}}} (0,k_1,4 \ ; \ i_1,j,5 \ ; \ i_2,k_2,6).$$

[4] The case of $d' > 2$ is more delicate. This example gives all 4D-descendants from a given 3D Nambu micro-graph. We take the 3D Nambu micro-graph given by (0,1,4 ; 1,3,5 ; 1,2,6). Its 4D-descendants are given by $\sum_{i_1,i_2 \in \{1,4,7\}, j \in \{3,6,9\}, k \in \{2,5,8\}} (0,1,4,7 \ ; \ i_1,j,5,8 \ ; \ i_2,k,6,9)$. It is left to the reader to produce the generalised definition of d-descendants from d'-dimension.

Lemma 2. There are 41 distinct micro-graphs in the 3D expansion of the 2D sunflower, their encodings are below. The first 10 come from Γ_1, the next 31 come from Γ_2. In bold are those whose Formality graphs Γ give formulas equal to zero $\phi(\Gamma) = 0$. A graph is *zero* when it has a symmetry under which it is skew.

1. (0,1,4 ; 1,3,5 ; 1,2,6)
2. (0,1,4 ; 4,3,5 ; 4,2,6)
3. (0,1,4 ; 1,6,5 ; 1,5,6)
4. (0,1,4 ; 4,6,5 ; 4,5,6)
5. (0,1,4 ; 1,3,5 ; 4,2,6)
6. (0,1,4 ; 1,6,5 ; 1,2,6)
7. (0,1,4 ; 1,6,5 ; 4,2,6)
8. (0,1,4 ; 4,6,5 ; 1,2,6)
9. (0,1,4 ; 4,6,5 ; 4,2,6)
10. **(0,1,4 ; 1,6,5 ; 4,5,6)** $\uparrow \Gamma_1$

---

11. (0,2,4 ; 1,3,5 ; 1,2,6) $\downarrow \Gamma_2$
12. (0,2,4 ; 4,3,5 ; 1,2,6)
13. **(0,2,4 ; 1,3,5 ; 4,2,6)**
14. (0,2,4 ; 4,3,5 ; 4,2,6)
15. (0,2,4 ; 1,6,5 ; 1,2,6)
16. (0,2,4 ; 4,6,5 ; 1,2,6)
17. (0,2,4 ; 1,6,5 ; 4,2,6)
18. (0,2,4 ; 4,6,5 ; 4,2,6)
19. (0,5,4 ; 1,3,5 ; 1,2,6)
20. **(0,5,4 ; 4,3,5 ; 1,2,6)**
21. **(0,5,4 ; 1,3,5 ; 4,2,6)**
22. (0,5,4 ; 4,3,5 ; 4,2,6)
23. (0,5,4 ; 1,6,5 ; 1,2,6)
24. **(0,5,4 ; 4,6,5 ; 1,2,6)**
25. **(0,5,4 ; 1,6,5 ; 4,2,6)**
26. (0,5,4 ; 4,6,5 ; 4,2,6)
27. (0,2,4 ; 1,3,5 ; 1,5,6)
28. (0,2,4 ; 4,3,5 ; 1,5,6)
29. **(0,2,4 ; 1,3,5 ; 4,5,6)**
30. (0,2,4 ; 4,3,5 ; 4,5,6)
31. (0,2,4 ; 1,6,5 ; 1,5,6)
32. **(0,2,4 ; 4,6,5 ; 1,5,6)**
33. **(0,2,4 ; 1,6,5 ; 4,5,6)**
34. (0,2,4 ; 4,6,5 ; 4,5,6)
35. (0,5,4 ; 1,3,5 ; 1,5,6)
36. (0,5,4 ; 4,3,5 ; 1,5,6)
37. **(0,5,4 ; 1,3,5 ; 4,5,6)**
38. **(0,5,4 ; 4,3,5 ; 4,5,6)** *zero*
39. (0,5,4 ; 1,6,5 ; 1,5,6)
40. (0,5,4 ; 4,6,5 ; 1,5,6)
41. **(0,5,4 ; 4,6,5 ; 4,5,6)** *zero*

Our next simplification is a lucky guess, in contrast with simplification 1.

Simplification 2. Search for the trivialising vector field $\vec{X}^{\gamma_3}_{d=3}(P)$ over 41 3D-descendants of the 2D sunflower, from the above Lemma.

Corollary 3. Simplifications 1 and 2 make the problem smaller:

$$366 \xrightarrow{\#1,\#2} 41,$$

while still allowing us to reach a solution. Here, 366 is the number of all 1-vector micro-graphs built of 3 Levi–Civita symbols and 3 Casimirs a^1; 41 is the number of 3D-descendants of the 2D sunflower.

Proposition 4. There exists a trivialising vector field $\vec{X}^{\gamma_3}_{d=3}(P) = \phi(X^{\gamma_3}_{d=3})$ in 3D. It is given over 10 3D-descendants of the 2D sunflower:

$$\begin{aligned} X^{\gamma_3}_{d=3} = {} & 8\cdot(0,1,4;1,3,5;1,2,6) + 24\cdot(0,1,4;1,6,5;4,2,6) + 8\cdot(0,1,4;4,3,5;4,2,6) \\ & + 24\cdot(0,1,4;4,6,5;4,2,6) + 12\cdot(0,1,4;4,6,5;4,5,6) + 16\cdot(0,2,4;1,3,5;1,2,6) \\ & + 16\cdot(0,2,4;1,3,5;1,5,6) + 12\cdot(0,2,4;1,6,5;1,5,6) + 16\cdot(0,2,4;4,3,5;1,5,6) + 24\cdot(0,5,4;1,3,5;1,2,6). \end{aligned}$$

This is a linear combination of Nambu micro-graphs which gives a formula in 3D to solve equation (1), namely $\dot{P} = Q^{\gamma_3}_{d=3}(P) = [\![P, \vec{X}^{\gamma_3}_{d=3}(P)]\!]$. The affine space of solutions on graphs is of dimension 3; that is, the trivialising vector field $\vec{X}^{\gamma_3}_{d=3}(P)$ is unique up to a 3-dimensional space of Poisson 1-cocycles X with $[\![P, X]\!] = 0$, where X is encoded by Nambu micro-graphs. A full description of these shifts using graphs can be found in [4].

We verified that the deformation of P by γ_3 is trivial in 3D: it amounts to a change of coordinates. The space of 3D-descendants from the 2D sunflower is sufficient to find a solution in 3D. We now ask:

Can we find a 4D solution over the space of 4D-descendants from the 3D trivialising vector field? (Answer: no! But from the 2D sunflower: yes!)

3.3 New result: the trivialising vector field for γ_3-flow in 4D

In 4D, with Cartesian coordinates expressed as $x_1 = x$, $x_2 = y$, $x_3 = z$, $x_4 = w$, the Nambu–Poisson bracket is given by

$$\{f,g\}_{d=4}(x,y,z,w) = \varrho(x,y,z,w)\cdot\det\begin{pmatrix} f_x & g_x & a^1_x & a^2_x \\ f_y & g_y & a^1_y & a^2_y \\ f_z & g_z & a^1_z & a^2_z \\ f_w & g_w & a^1_w & a^2_w \end{pmatrix}(x,y,z,w),$$

that is,

$$\{f,g\}_{d=4}(x,y,z,w) = \sum_{i,j,k,\ell=1}^{d=4} \varepsilon^{ijk\ell} \cdot \varrho \cdot \partial_i(f) \cdot \partial_j(g) \cdot \partial_k(a^1) \cdot \partial_\ell(a^2),$$

for some ϱ, where i,j,k,ℓ are indices and $\varepsilon^{ijk\ell}$ is the Levi–Civita symbol which encodes the determinant.

To tackle the full problem was non-viable, see Appendix 1 in [9] for details. The main issue is that the size of the problem increases with the dimension: $\phi(Q_d^{\gamma_3}(P))$ is 1 line in 2D, 2 pages in 3D, 3GB in 4D.

We begin to apply the simplifications that we have introduced so far.

- Simplification 1: only look over graphs built with the 4D Nambu building blocks $P(\varrho, a^1, a^2)$. The vertex of the source of the building blocks contains $\varepsilon^{ijk\ell}\varrho$.
- Simplification 2: only look over 4D-descendants of the 2D sunflower. These are 324 graphs, which give 123 linearly independent formulas; their encodings can be found in the Appendix.

⚠ There are two Casimirs a^1, a^2, yielding an extra property to take into account.

Lemma 5. The Nambu–Poisson bracket $P(\varrho, a^1, a^2)$ is skew-symmetric under the swap $a^1 \rightleftarrows a^2$:

$$P(\varrho, a^1, a^2) = -P(\varrho, a^2, a^1).$$

The γ_3-flow $Q_{d=4}^{\gamma_3}(P)$ is built of four copies of P, therefore $Q_{d=4}^{\gamma_3}(P)$ is symmetric under $a^1 \rightleftarrows a^2$; by swapping a^1 and a^2, we accumulate four minus signs:

$$\begin{aligned} Q_{d=4}^{\gamma_3}\Big(P(\varrho,a^2,a^1)\otimes P(\varrho,a^2,a^1)\otimes P(\varrho,a^2,a^1)\otimes P(\varrho,a^2,a^1)\Big) \\ = (-)^4 Q_{d=4}^{\gamma_3}\Big(P(\varrho,a^1,a^2)\otimes P(\varrho,a^1,a^2)\otimes P(\varrho,a^1,a^2)\otimes P(\varrho,a^1,a^2)\Big) \\ = Q_{d=4}^{\gamma_3}\Big(P(\varrho,a^1,a^2)\otimes P(\varrho,a^1,a^2)\otimes P(\varrho,a^1,a^2)\otimes P(\varrho,a^1,a^2)\Big). \end{aligned}$$

Therefore, to find a vector field $\vec{X}_{d=4}^{\gamma_3}(P)$ such that

$$\dot{P} = Q_{d=4}^{\gamma_3}(P) = [\![P, \vec{X}_{d=4}^{\gamma_3}(P)]\!],$$

we need to find $\vec{X}_{d=4}^{\gamma_3}(P)$ which is skew-symmetric under $a^1 \rightleftarrows a^2$. This can be seen by the fact that $\vec{X}_{d=4}^{\gamma_3}(P)$ is built of three copies of P, so accumulates three minus signs when swapping a^1 and a^2, therefore gives $(-)^3 = -$, therefore is skew-symmetric under $a^1 \rightleftarrows a^2$.

To take this into account, we use the 324 4D-descendants of the 2D sunflower obtained by simplifications 1 and 2, identify the 123 ones with linearly independent formulas, and skew-symmetrise them. That is, for each $4D$-descendant Γ of the 2D sunflower we construct a skew pair:

$$\text{skew pair} = \tfrac{1}{2}\Big(\phi\big(\Gamma(a^1,a^2)\big) - \phi\big(\Gamma(a^2,a^1)\big)\Big).$$

To construct skew pairs, we take the formula of the graph Γ with ordering of edges to Casimirs a^1, a^2 with $a^1 \prec a^2$, and subtract the formula of the graph Γ with ordering $a^2 \prec a^1$. We divide by 2 to preserve the coefficients. By construction, each skew pair is purely obtained at the level of formulas[5].

Lemma 6. There are 64 linearly independent skew pairs, see Appendix.

Simplification 3. Search over these 64 skew pairs for a trivialising vector field $\vec{X}_{d=4}^{\gamma_3}(P)$.

Corollary 7. The three simplifications 1, 2, and 3 reduced the size of our problem 300 times:

$$19\ 957 \xrightarrow{\#1,\#2} 324 \xrightarrow{\#3} 64.$$

Here, 19 957 is the number of all 1-vector micro-graphs built of 4 Levi–Civita symbols, 4 Casimirs a^1 and 4 Casimirs a^2; 324 is the number of 4D-descendants of the 2D sunflower; 64 is the number of skew pairs obtained from the 123 linearly independent formulas of the 324 4D-descendants.

Proposition 8. There exists a trivialising vector field $\vec{X}_{d=4}^{\gamma_3}(P) = \phi(X_{d=4}^{\gamma_3})$ in 4D. Searching over the 64 skew pairs on the High Performing Computing cluster Hábrók took 10 hours. It is given over 27 skew pairs of 1-vector Nambu micro-graphs:

[5] See `https://github.com/rburing`.

$$
\begin{aligned}
X^{\gamma_3}_{d=4} \;=\; & -8 \cdot \big((0,1,4,7\ ;\ 1,3,5,8\ ;\ 1,2,6,9) - (0,1,7,4\ ;\ 1,3,8,5\ ;\ 1,2,9,6)\big) \\
& + \ -48 \cdot \big((0,1,4,7\ ;\ 1,6,5,8\ ;\ 4,2,6,9) - (0,1,7,4\ ;\ 1,9,8,5\ ;\ 7,2,9,6)\big) \\
& + \ -16 \cdot \big((0,1,4,7\ ;\ 4,3,5,8\ ;\ 4,2,6,9) - (0,1,7,4\ ;\ 7,3,8,5\ ;\ 7,2,9,6)\big) \\
& + \ -48 \cdot \big((0,1,4,7\ ;\ 4,6,5,8\ ;\ 4,2,6,9) - (0,1,7,4\ ;\ 7,9,8,5\ ;\ 7,2,9,6)\big) \\
& + \ -48 \cdot \big((0,1,4,7\ ;\ 4,9,5,8\ ;\ 4,2,6,9) - (0,1,7,4\ ;\ 7,6,8,5\ ;\ 7,2,9,6)\big) \\
& + \ -16 \cdot \big((0,1,4,7\ ;\ 4,3,5,8\ ;\ 7,2,6,9) - (0,1,7,4\ ;\ 7,3,8,5\ ;\ 4,2,9,6)\big) \\
& + \ -48 \cdot \big((0,1,4,7\ ;\ 4,6,5,8\ ;\ 7,2,6,9) - (0,1,7,4\ ;\ 7,9,8,5\ ;\ 4,2,9,6)\big) \\
& + \ 12 \cdot \big((0,1,4,7\ ;\ 1,6,5,8\ ;\ 1,5,6,9) - (0,1,7,4\ ;\ 1,9,8,5\ ;\ 1,8,9,6)\big) \\
& + \ -24 \cdot \big((0,1,4,7\ ;\ 4,6,5,8\ ;\ 4,5,6,9) - (0,1,7,4\ ;\ 7,9,8,5\ ;\ 7,8,9,6)\big) \\
& + \ 24 \cdot \big((0,1,4,7\ ;\ 4,9,5,8\ ;\ 7,5,6,9) - (0,1,7,4\ ;\ 7,6,8,5\ ;\ 4,8,9,6)\big) \\
& + \ -24 \cdot \big((0,1,4,7\ ;\ 7,6,5,8\ ;\ 7,5,6,9) - (0,1,7,4\ ;\ 4,9,8,5\ ;\ 4,8,9,6)\big) \\
& + \ -16 \cdot \big((0,2,4,7\ ;\ 1,3,5,8\ ;\ 1,2,6,9) - (0,2,7,4\ ;\ 1,3,8,5\ ;\ 1,2,9,6)\big) \\
& + \ -32 \cdot \big((0,2,4,7\ ;\ 1,3,5,8\ ;\ 1,5,6,9) - (0,2,7,4\ ;\ 1,3,8,5\ ;\ 1,8,9,6)\big) \\
& + \ -12 \cdot \big((0,2,4,7\ ;\ 1,6,5,8\ ;\ 1,5,6,9) - (0,2,7,4\ ;\ 1,9,8,5\ ;\ 1,8,9,6)\big) \\
& + \ 48 \cdot \big((0,2,4,7\ ;\ 1,9,5,8\ ;\ 4,5,6,9) - (0,2,7,4\ ;\ 1,6,8,5\ ;\ 7,8,9,6)\big) \\
& + \ -48 \cdot \big((0,2,4,7\ ;\ 1,9,5,8\ ;\ 7,5,6,9) - (0,2,7,4\ ;\ 1,6,8,5\ ;\ 4,8,9,6)\big) \\
& + \ -32 \cdot \big((0,2,4,7\ ;\ 4,3,5,8\ ;\ 1,5,6,9) - (0,2,7,4\ ;\ 7,3,8,5\ ;\ 1,8,9,6)\big) \\
& + \ -32 \cdot \big((0,2,4,7\ ;\ 7,3,5,8\ ;\ 1,5,6,9) - (0,2,7,4\ ;\ 4,3,8,5\ ;\ 1,8,9,6)\big) \\
& + \ -48 \cdot \big((0,5,4,7\ ;\ 1,3,5,8\ ;\ 1,2,6,9) - (0,8,7,4\ ;\ 1,3,8,5\ ;\ 1,2,9,6)\big) \\
& + \ -48 \cdot \big((0,5,4,7\ ;\ 1,9,5,8\ ;\ 1,2,6,9) - (0,8,7,4\ ;\ 1,6,8,5\ ;\ 1,2,9,6)\big) \\
& + \ -96 \cdot \big((0,5,4,7\ ;\ 1,9,5,8\ ;\ 4,2,6,9) - (0,8,7,4\ ;\ 1,6,8,5\ ;\ 7,2,9,6)\big) \\
& + \ -48 \cdot \big((0,5,4,7\ ;\ 4,9,5,8\ ;\ 4,2,6,9) - (0,8,7,4\ ;\ 7,6,8,5\ ;\ 7,2,9,6)\big) \\
& + \ -48 \cdot \big((0,5,4,7\ ;\ 7,9,5,8\ ;\ 4,2,6,9) - (0,8,7,4\ ;\ 4,6,8,5\ ;\ 7,2,9,6)\big) \\
& + \ -48 \cdot \big((0,5,4,7\ ;\ 7,9,5,8\ ;\ 7,2,6,9) - (0,8,7,4\ ;\ 4,6,8,5\ ;\ 4,2,9,6)\big) \\
& + \ 24 \cdot \big((0,5,4,7\ ;\ 1,6,5,8\ ;\ 1,5,6,9) - (0,8,7,4\ ;\ 1,9,8,5\ ;\ 1,8,9,6)\big) \\
& + \ 48 \cdot \big((0,5,4,7\ ;\ 4,6,5,8\ ;\ 7,5,6,9) - (0,8,7,4\ ;\ 7,9,8,5\ ;\ 4,8,9,6)\big) \\
& + \ 48 \cdot \big((0,5,4,7\ ;\ 4,9,5,8\ ;\ 7,5,6,9) - (0,8,7,4\ ;\ 7,6,8,5\ ;\ 4,8,9,6)\big).
\end{aligned}
$$

This is a linear combination of skew pairs which gives a formula in 4D to solve equation (1), namely $\dot{P} = Q^{\gamma_3}_{d=4}(P) = [\![P, \vec{X}^{\gamma_3}_{d=4}(P)]\!]$. The affine space of solutions on graphs is of dimension 7; that is, the trivialising vector field $\vec{X}^{\gamma_3}_{d=4}(P)$ is unique up to a 7-dimensional space of Poisson 1-cocycles X with $[\![P, X]\!] = 0$, where X is encoded by Nambu micro-graphs. A full description of these shifts using graphs can be found in [4].

As far as we understand, nothing could have predicted this result. Without this series of simplifications (see Corollary 7), approaching the problem in dimension 4 was impossible in [9] two years ago.

4 Discussion

We summarise in which sense the graphs used in problems 2D, 3D, and 4D were different. In 2D, we found a solution over Kontsevich graphs; in 3D over Nambu micro-graphs; in 4D over skew pairs, obtained by skew-symmetrising formulas of Nambu micro-graphs. The problem has been solved at the level of formulas; the graphs provide a roadmap to find the few ones which have given solutions so far. Each dimension has presented a peculiarity related to properties of the class of Nambu–Poisson brackets. In 3D, we encoded the first instance where the Nambu–Poisson bracket is degenerate (of rank ≤ 2), that is, different than the maximal-rank Poisson bracket. In 4D, we skew-symmetrised the formulas of micro-graphs to account for solutions' anti-symmetry with respect to two Casimirs. Overall, we notice an interplay between expansions of graphs between dimensions, and the 'goodness' of their formulas. The behaviour of solutions in dimension d to dimensions $d-1$ and $d+1$ exhibits two curious properties.

Claim 9. *We can project a 4D solution down to a 3D solution by setting the last Casimir equal to the last coordinate: $a^2 = w$. Similarly, we can project a 3D solution down to a 2D solution by setting the Casimir equal to the last coordinate: $a^1 = z$. Formulas project down to previously found formulas:*

$$\phi\big(\vec{X}^{\gamma_3}_{d=4}(P)\big) \xrightarrow{a^2=w} \phi\big(\vec{X}^{\gamma_3}_{d=3}(P)\big) \xrightarrow{a^1=z} \phi\big(\vec{X}^{\gamma_3}_{d=2}(P)\big).$$

Claim 10. *There exist solutions in 3D and 4D over the descendants of a 2D solution, the sunflower. But descendants of known solutions in 3D do not give solutions in 4D.*

Comment. This would have been practical for reducing computing time. We have 41 3D graphs obtained from expanding the sunflower, and solutions in 3D over 10 such graphs. Moving up to a higher dimension, it would be ideal to search over descendants of a 3D solution, but we observe this is not possible.

Open problem 1. What is the formula of the trivialising vector field $\vec{X}^{\gamma_3}_d(P)$ in dimensions $d \geq 5$, if it exists at all?

Open problem 2. In which dimensions are deformations of Nambu–Poisson brackets by other 'good graphs' in the Kontsevich graph complex, such as $\gamma_5, \gamma_7, [\gamma_3, \gamma_5]$, (non)trivial?

The order of the problem also increases with the choice of 'good graph', for instance computing with γ_5 in dimension 3 is more costly than computing with γ_3 in dimension 4, see [3, 10].

5 Conclusion

We observed that in dimension 4, the Kontsevich γ_3-flow is trivial for the class of Nambu–Poisson brackets. In other words, the deformation of the class of Nambu–Poisson brackets by γ_3 in 4D amounts to a change of coordinates along a vector field $\vec{X}^{\gamma_3}_{d=4}(P)$. So far, in dimensions 2, 3 and 4, we have that the Nambu–Poisson system is isolated in that it withstands the deforming action of the Kontsevich graph γ_3. Achieving this result was only possible by using the series of simplifications introduced here.

Acknowledgements

The authors thank the organizers of the ISQS28 conference on 1–5 July 2024 in CVUT Prague for a dynamic and welcoming atmosphere; and the Center for Information Technology of the University of Groningen for access to the High Performance Computing cluster, Hábrók. The authors are especially grateful to R. Buring for computational support for his software package gcaops. The participation of M.S. Jagoe Brown and F. Schipper in the ISQS28 was supported by the Master's Research Project funds at the Bernoulli Institute, University of Groningen; that of A.V. Kiselev was supported by project 135110.

References

[1] Laurent-Gengoux C, Pichereau A and Vanhaecke P 2013 *Poisson Structures* (Springer)

[2] Kontsevich M 1997 Formality conjecture *Deformation theory and symplectic geometry (Ascona 1996)* 139–156

[3] Kiselev A V, Jagoe Brown M S and Schipper F 2024 Kontsevich graphs act on Nambu-–Poisson brackets, I. New identities for Jacobian determinants *arXiv preprint arXiv:2409.18875 [math.QA]*

[4] Schipper F, Jagoe Brown M S and Kiselev A V 2024 Kontsevich graphs act on Nambu-–Poisson brackets, III. Uniqueness aspects *arXiv preprint arXiv:2409.15932 [math.QA]*

[5] Jagoe Brown M S, Schipper F and Kiselev A V 2024 Kontsevich graphs act on Nambu-–Poisson brackets, II. The tetrahedral flow is a coboundary in 4D *arXiv preprint arXiv:2409.12555 [math.QA]*

[6] Kiselev A V and Buring R 2021 The Kontsevich graph orientation morphism revisited *Banach Center Publications* **123** *Homotopy algebras, deformation theory and quantization* 123–139

[7] Bouisaghouane A, Buring R and Kiselev A V 2017 The Kontsevich tetrahedral flow revisited *J. Geom. Phys.* **19** 272–285

[8] Bouisaghouane A 2017 The Kontsevich tetrahedral flow in 2D: a toy model *arXiv preprint, arXiv:1702.06044 [math.DG]*

[9] Buring R and Kiselev A V 2023 The tower of Kontsevich deformations for Nambu–Poisson structures on $\mathbb{R}^d$: Dimension-specific micro-graph calculus *SciPost Phys. Proc.* **14** Paper 020 1–11

[10] Buring R, Kiselev A V and Lipper D 2022 The hidden symmetry of Kontsevich's graph flows on the spaces of Nambu-determinant Poisson brackets Open Communications in Nonlinear Mathematical Physics **2** Paper ocnmp:8844 186–216

Appendix
There are 324 micro-graphs in the 4D expansion of the 2D sunflower, their encodings are given below. The first 81 come from Γ_1, the next 243 come from Γ_2. In bold are the 54 encodings whose Formality graphs Γ give formulas equal to zero $\phi(\Gamma) = 0$. A graph is zero when it has a symmetry under which it is skew.

1. (0, 1, 4, 7, 1, 3, 5, 8, 1, 2, 6, 9)
2. (0, 1, 4, 7, 1, 6, 5, 8, 1, 2, 6, 9)
3. (0, 1, 4, 7, 1, 9, 5, 8, 1, 2, 6, 9)
4. (0, 1, 4, 7, 1, 3, 5, 8, 4, 2, 6, 9)
5. (0, 1, 4, 7, 1, 6, 5, 8, 4, 2, 6, 9)
6. (0, 1, 4, 7, 1, 9, 5, 8, 4, 2, 6, 9)
7. (0, 1, 4, 7, 1, 3, 5, 8, 7, 2, 6, 9)
8. (0, 1, 4, 7, 1, 6, 5, 8, 7, 2, 6, 9)
9. (0, 1, 4, 7, 1, 9, 5, 8, 7, 2, 6, 9)
10. (0, 1, 4, 7, 4, 3, 5, 8, 1, 2, 6, 9)
11. (0, 1, 4, 7, 4, 6, 5, 8, 1, 2, 6, 9)
12. (0, 1, 4, 7, 4, 9, 5, 8, 1, 2, 6, 9)
13. (0, 1, 4, 7, 4, 3, 5, 8, 4, 2, 6, 9)
14. (0, 1, 4, 7, 4, 6, 5, 8, 4, 2, 6, 9)
15. (0, 1, 4, 7, 4, 9, 5, 8, 4, 2, 6, 9)
16. (0, 1, 4, 7, 4, 3, 5, 8, 7, 2, 6, 9)
17. (0, 1, 4, 7, 4, 6, 5, 8, 7, 2, 6, 9)
18. (0, 1, 4, 7, 4, 9, 5, 8, 7, 2, 6, 9)
19. (0, 1, 4, 7, 7, 3, 5, 8, 1, 2, 6, 9)
20. (0, 1, 4, 7, 7, 6, 5, 8, 1, 2, 6, 9)
21. (0, 1, 4, 7, 7, 9, 5, 8, 1, 2, 6, 9)
22. (0, 1, 4, 7, 7, 3, 5, 8, 4, 2, 6, 9)
23. (0, 1, 4, 7, 7, 6, 5, 8, 4, 2, 6, 9)
24. (0, 1, 4, 7, 7, 9, 5, 8, 4, 2, 6, 9)
25. (0, 1, 4, 7, 7, 3, 5, 8, 7, 2, 6, 9)
26. (0, 1, 4, 7, 7, 6, 5, 8, 7, 2, 6, 9)
27. (0, 1, 4, 7, 7, 9, 5, 8, 7, 2, 6, 9)
28. (0, 1, 4, 7, 1, 3, 5, 8, 1, 5, 6, 9)
29. (0, 1, 4, 7, 1, 6, 5, 8, 1, 5, 6, 9)
30. (0, 1, 4, 7, 1, 9, 5, 8, 1, 5, 6, 9)
31. (0, 1, 4, 7, 1, 3, 5, 8, 4, 5, 6, 9)
32. **(0, 1, 4, 7, 1, 6, 5, 8, 4, 5, 6, 9)**
33. (0, 1, 4, 7, 1, 9, 5, 8, 4, 5, 6, 9)
34. (0, 1, 4, 7, 1, 3, 5, 8, 7, 5, 6, 9)
35. (0, 1, 4, 7, 1, 6, 5, 8, 7, 5, 6, 9)
36. (0, 1, 4, 7, 1, 9, 5, 8, 7, 5, 6, 9)
37. (0, 1, 4, 7, 4, 3, 5, 8, 1, 5, 6, 9)
38. **(0, 1, 4, 7, 4, 6, 5, 8, 1, 5, 6, 9)**
39. (0, 1, 4, 7, 4, 9, 5, 8, 1, 5, 6, 9)
40. (0, 1, 4, 7, 4, 3, 5, 8, 4, 5, 6, 9)
41. (0, 1, 4, 7, 4, 6, 5, 8, 4, 5, 6, 9)
42. (0, 1, 4, 7, 4, 9, 5, 8, 4, 5, 6, 9)
43. (0, 1, 4, 7, 4, 3, 5, 8, 7, 5, 6, 9)
44. (0, 1, 4, 7, 4, 6, 5, 8, 7, 5, 6, 9)
45. (0, 1, 4, 7, 4, 9, 5, 8, 7, 5, 6, 9)
46. (0, 1, 4, 7, 7, 3, 5, 8, 1, 5, 6, 9)
47. (0, 1, 4, 7, 7, 6, 5, 8, 1, 5, 6, 9)
48. (0, 1, 4, 7, 7, 9, 5, 8, 1, 5, 6, 9)
49. (0, 1, 4, 7, 7, 3, 5, 8, 4, 5, 6, 9)
50. (0, 1, 4, 7, 7, 6, 5, 8, 4, 5, 6, 9)
51. (0, 1, 4, 7, 7, 9, 5, 8, 4, 5, 6, 9)
52. (0, 1, 4, 7, 7, 3, 5, 8, 7, 5, 6, 9)
53. (0, 1, 4, 7, 7, 6, 5, 8, 7, 5, 6, 9)
54. (0, 1, 4, 7, 7, 9, 5, 8, 7, 5, 6, 9)
55. (0, 1, 4, 7, 1, 3, 5, 8, 1, 8, 6, 9)
56. (0, 1, 4, 7, 1, 6, 5, 8, 1, 8, 6, 9)
57. (0, 1, 4, 7, 1, 9, 5, 8, 1, 8, 6, 9)
58. (0, 1, 4, 7, 1, 3, 5, 8, 4, 8, 6, 9)
59. (0, 1, 4, 7, 1, 6, 5, 8, 4, 8, 6, 9)
60. (0, 1, 4, 7, 1, 9, 5, 8, 4, 8, 6, 9)
61. (0, 1, 4, 7, 1, 3, 5, 8, 7, 8, 6, 9)
62. (0, 1, 4, 7, 1, 6, 5, 8, 7, 8, 6, 9)
63. **(0, 1, 4, 7, 1, 9, 5, 8, 7, 8, 6, 9)**
64. (0, 1, 4, 7, 4, 3, 5, 8, 1, 8, 6, 9)
65. (0, 1, 4, 7, 4, 6, 5, 8, 1, 8, 6, 9)
66. (0, 1, 4, 7, 4, 9, 5, 8, 1, 8, 6, 9)
67. (0, 1, 4, 7, 4, 3, 5, 8, 4, 8, 6, 9)
68. (0, 1, 4, 7, 4, 6, 5, 8, 4, 8, 6, 9)
69. (0, 1, 4, 7, 4, 9, 5, 8, 4, 8, 6, 9)
70. (0, 1, 4, 7, 4, 3, 5, 8, 7, 8, 6, 9)
71. (0, 1, 4, 7, 4, 6, 5, 8, 7, 8, 6, 9)
72. (0, 1, 4, 7, 4, 9, 5, 8, 7, 8, 6, 9)
73. (0, 1, 4, 7, 7, 3, 5, 8, 1, 8, 6, 9)
74. (0, 1, 4, 7, 7, 6, 5, 8, 1, 8, 6, 9)
75. **(0, 1, 4, 7, 7, 9, 5, 8, 1, 8, 6, 9)**
76. (0, 1, 4, 7, 7, 3, 5, 8, 4, 8, 6, 9)
77. (0, 1, 4, 7, 7, 6, 5, 8, 4, 8, 6, 9)
78. (0, 1, 4, 7, 7, 9, 5, 8, 4, 8, 6, 9)
79. (0, 1, 4, 7, 7, 3, 5, 8, 7, 8, 6, 9)
80. (0, 1, 4, 7, 7, 6, 5, 8, 7, 8, 6, 9)
81. (0, 1, 4, 7, 7, 9, 5, 8, 7, 8, 6, 9) $\uparrow \Gamma_1$

---

82. (0, 2, 4, 7, 1, 3, 5, 8, 1, 2, 6, 9) $\downarrow \Gamma_2$
83. (0, 2, 4, 7, 1, 6, 5, 8, 1, 2, 6, 9)
84. (0, 2, 4, 7, 1, 9, 5, 8, 1, 2, 6, 9)
85. **(0, 2, 4, 7, 1, 3, 5, 8, 4, 2, 6, 9)**
86. (0, 2, 4, 7, 1, 6, 5, 8, 4, 2, 6, 9)
87. (0, 2, 4, 7, 1, 9, 5, 8, 4, 2, 6, 9)
88. **(0, 2, 4, 7, 1, 3, 5, 8, 7, 2, 6, 9)**
89. (0, 2, 4, 7, 1, 6, 5, 8, 7, 2, 6, 9)
90. (0, 2, 4, 7, 1, 9, 5, 8, 7, 2, 6, 9)
91. (0, 2, 4, 7, 4, 3, 5, 8, 1, 2, 6, 9)
92. (0, 2, 4, 7, 4, 6, 5, 8, 1, 2, 6, 9)
93. (0, 2, 4, 7, 4, 9, 5, 8, 1, 2, 6, 9)
94. (0, 2, 4, 7, 4, 3, 5, 8, 4, 2, 6, 9)
95. (0, 2, 4, 7, 4, 6, 5, 8, 4, 2, 6, 9)
96. (0, 2, 4, 7, 4, 9, 5, 8, 4, 2, 6, 9)
97. (0, 2, 4, 7, 4, 3, 5, 8, 7, 2, 6, 9)
98. (0, 2, 4, 7, 4, 6, 5, 8, 7, 2, 6, 9)
99. (0, 2, 4, 7, 4, 9, 5, 8, 7, 2, 6, 9)
100. (0, 2, 4, 7, 7, 3, 5, 8, 1, 2, 6, 9)
101. (0, 2, 4, 7, 7, 6, 5, 8, 1, 2, 6, 9)
102. (0, 2, 4, 7, 7, 9, 5, 8, 1, 2, 6, 9)
103. (0, 2, 4, 7, 7, 3, 5, 8, 4, 2, 6, 9)
104. (0, 2, 4, 7, 7, 6, 5, 8, 4, 2, 6, 9)
105. (0, 2, 4, 7, 7, 9, 5, 8, 4, 2, 6, 9)
106. (0, 2, 4, 7, 7, 3, 5, 8, 7, 2, 6, 9)
107. (0, 2, 4, 7, 7, 6, 5, 8, 7, 2, 6, 9)
108. (0, 2, 4, 7, 7, 9, 5, 8, 7, 2, 6, 9)
109. (0, 2, 4, 7, 1, 3, 5, 8, 1, 5, 6, 9)
110. (0, 2, 4, 7, 1, 6, 5, 8, 1, 5, 6, 9)
111. (0, 2, 4, 7, 1, 9, 5, 8, 1, 5, 6, 9)
112. **(0, 2, 4, 7, 1, 3, 5, 8, 4, 5, 6, 9)**
113. **(0, 2, 4, 7, 1, 6, 5, 8, 4, 5, 6, 9)**
114. (0, 2, 4, 7, 1, 9, 5, 8, 4, 5, 6, 9)
115. **(0, 2, 4, 7, 1, 3, 5, 8, 7, 5, 6, 9)**
116. (0, 2, 4, 7, 1, 6, 5, 8, 7, 5, 6, 9)
117. (0, 2, 4, 7, 1, 9, 5, 8, 7, 5, 6, 9)
118. (0, 2, 4, 7, 4, 3, 5, 8, 1, 5, 6, 9)
119. **(0, 2, 4, 7, 4, 6, 5, 8, 1, 5, 6, 9)**
120. (0, 2, 4, 7, 4, 9, 5, 8, 1, 5, 6, 9)
121. (0, 2, 4, 7, 4, 3, 5, 8, 4, 5, 6, 9)
122. (0, 2, 4, 7, 4, 6, 5, 8, 4, 5, 6, 9)
123. (0, 2, 4, 7, 4, 9, 5, 8, 4, 5, 6, 9)
124. (0, 2, 4, 7, 4, 3, 5, 8, 7, 5, 6, 9)
125. (0, 2, 4, 7, 4, 6, 5, 8, 7, 5, 6, 9)
126. (0, 2, 4, 7, 4, 9, 5, 8, 7, 5, 6, 9)
127. (0, 2, 4, 7, 7, 3, 5, 8, 1, 5, 6, 9)
128. (0, 2, 4, 7, 7, 6, 5, 8, 1, 5, 6, 9)
129. (0, 2, 4, 7, 7, 9, 5, 8, 1, 5, 6, 9)
130. (0, 2, 4, 7, 7, 3, 5, 8, 4, 5, 6, 9)
131. (0, 2, 4, 7, 7, 6, 5, 8, 4, 5, 6, 9)

132. (0, 2, 4, 7, 7, 9, 5, 8, 4, 5, 6, 9)
133. (0, 2, 4, 7, 7, 3, 5, 8, 7, 5, 6, 9)
134. (0, 2, 4, 7, 7, 6, 5, 8, 7, 5, 6, 9)
135. (0, 2, 4, 7, 7, 9, 5, 8, 7, 5, 6, 9)
136. (0, 2, 4, 7, 1, 3, 5, 8, 1, 8, 6, 9)
137. (0, 2, 4, 7, 1, 6, 5, 8, 1, 8, 6, 9)
138. (0, 2, 4, 7, 1, 9, 5, 8, 1, 8, 6, 9)
139. **(0, 2, 4, 7, 1, 3, 5, 8, 4, 8, 6, 9)**
140. (0, 2, 4, 7, 1, 6, 5, 8, 4, 8, 6, 9)
141. (0, 2, 4, 7, 1, 9, 5, 8, 4, 8, 6, 9)
142. **(0, 2, 4, 7, 1, 3, 5, 8, 7, 8, 6, 9)**
143. (0, 2, 4, 7, 1, 6, 5, 8, 7, 8, 6, 9)
144. **(0, 2, 4, 7, 1, 9, 5, 8, 7, 8, 6, 9)**
145. (0, 2, 4, 7, 4, 3, 5, 8, 1, 8, 6, 9)
146. (0, 2, 4, 7, 4, 6, 5, 8, 1, 8, 6, 9)
147. (0, 2, 4, 7, 4, 9, 5, 8, 1, 8, 6, 9)
148. (0, 2, 4, 7, 4, 3, 5, 8, 4, 8, 6, 9)
149. (0, 2, 4, 7, 4, 6, 5, 8, 4, 8, 6, 9)
150. (0, 2, 4, 7, 4, 9, 5, 8, 4, 8, 6, 9)
151. (0, 2, 4, 7, 4, 3, 5, 8, 7, 8, 6, 9)
152. (0, 2, 4, 7, 4, 6, 5, 8, 7, 8, 6, 9)
153. (0, 2, 4, 7, 4, 9, 5, 8, 7, 8, 6, 9)
154. (0, 2, 4, 7, 7, 3, 5, 8, 1, 8, 6, 9)
155. (0, 2, 4, 7, 7, 6, 5, 8, 1, 8, 6, 9)
156. **(0, 2, 4, 7, 7, 9, 5, 8, 1, 8, 6, 9)**
157. (0, 2, 4, 7, 7, 3, 5, 8, 4, 8, 6, 9)
158. (0, 2, 4, 7, 7, 6, 5, 8, 4, 8, 6, 9)
159. (0, 2, 4, 7, 7, 9, 5, 8, 4, 8, 6, 9)
160. (0, 2, 4, 7, 7, 3, 5, 8, 7, 8, 6, 9)
161. (0, 2, 4, 7, 7, 6, 5, 8, 7, 8, 6, 9)
162. (0, 2, 4, 7, 7, 9, 5, 8, 7, 8, 6, 9)
163. (0, 5, 4, 7, 1, 3, 5, 8, 1, 2, 6, 9)
164. (0, 5, 4, 7, 1, 6, 5, 8, 1, 2, 6, 9)
165. (0, 5, 4, 7, 1, 9, 5, 8, 1, 2, 6, 9)
166. **(0, 5, 4, 7, 1, 3, 5, 8, 4, 2, 6, 9)**
167. **(0, 5, 4, 7, 1, 6, 5, 8, 4, 2, 6, 9)**
168. (0, 5, 4, 7, 1, 9, 5, 8, 4, 2, 6, 9)
169. **(0, 5, 4, 7, 1, 3, 5, 8, 7, 2, 6, 9)**
170. (0, 5, 4, 7, 1, 6, 5, 8, 7, 2, 6, 9)
171. (0, 5, 4, 7, 1, 9, 5, 8, 7, 2, 6, 9)
172. **(0, 5, 4, 7, 4, 3, 5, 8, 1, 2, 6, 9)**
173. **(0, 5, 4, 7, 4, 6, 5, 8, 1, 2, 6, 9)**
174. **(0, 5, 4, 7, 4, 9, 5, 8, 1, 2, 6, 9)**
175. (0, 5, 4, 7, 4, 3, 5, 8, 4, 2, 6, 9)
176. (0, 5, 4, 7, 4, 6, 5, 8, 4, 2, 6, 9)
177. (0, 5, 4, 7, 4, 9, 5, 8, 4, 2, 6, 9)
178. (0, 5, 4, 7, 4, 3, 5, 8, 7, 2, 6, 9)
179. (0, 5, 4, 7, 4, 6, 5, 8, 7, 2, 6, 9)
180. (0, 5, 4, 7, 4, 9, 5, 8, 7, 2, 6, 9)
181. **(0, 5, 4, 7, 7, 3, 5, 8, 1, 2, 6, 9)**
182. **(0, 5, 4, 7, 7, 6, 5, 8, 1, 2, 6, 9)**
183. **(0, 5, 4, 7, 7, 9, 5, 8, 1, 2, 6, 9)**
184. (0, 5, 4, 7, 7, 3, 5, 8, 4, 2, 6, 9)
185. (0, 5, 4, 7, 7, 6, 5, 8, 4, 2, 6, 9)
186. (0, 5, 4, 7, 7, 9, 5, 8, 4, 2, 6, 9)
187. (0, 5, 4, 7, 7, 3, 5, 8, 7, 2, 6, 9)
188. (0, 5, 4, 7, 7, 6, 5, 8, 7, 2, 6, 9)
189. (0, 5, 4, 7, 7, 9, 5, 8, 7, 2, 6, 9)
190. (0, 5, 4, 7, 1, 3, 5, 8, 1, 5, 6, 9)
191. (0, 5, 4, 7, 1, 6, 5, 8, 1, 5, 6, 9)
192. (0, 5, 4, 7, 1, 9, 5, 8, 1, 5, 6, 9)
193. **(0, 5, 4, 7, 1, 3, 5, 8, 4, 5, 6, 9)**
194. (0, 5, 4, 7, 1, 6, 5, 8, 4, 5, 6, 9)
195. (0, 5, 4, 7, 1, 9, 5, 8, 4, 5, 6, 9)
196. **(0, 5, 4, 7, 1, 3, 5, 8, 7, 5, 6, 9)**
197. (0, 5, 4, 7, 1, 6, 5, 8, 7, 5, 6, 9)
198. (0, 5, 4, 7, 1, 9, 5, 8, 7, 5, 6, 9)
199. (0, 5, 4, 7, 4, 3, 5, 8, 1, 5, 6, 9)
200. (0, 5, 4, 7, 4, 6, 5, 8, 1, 5, 6, 9)
201. (0, 5, 4, 7, 4, 9, 5, 8, 1, 5, 6, 9)
202. **(0, 5, 4, 7, 4, 3, 5, 8, 4, 5, 6, 9)**
203. **(0, 5, 4, 7, 4, 6, 5, 8, 4, 5, 6, 9)**
204. **(0, 5, 4, 7, 4, 9, 5, 8, 4, 5, 6, 9)**
205. (0, 5, 4, 7, 4, 3, 5, 8, 7, 5, 6, 9)
206. (0, 5, 4, 7, 4, 6, 5, 8, 7, 5, 6, 9)
207. (0, 5, 4, 7, 4, 9, 5, 8, 7, 5, 6, 9)
208. (0, 5, 4, 7, 7, 3, 5, 8, 1, 5, 6, 9)
209. (0, 5, 4, 7, 7, 6, 5, 8, 1, 5, 6, 9)
210. (0, 5, 4, 7, 7, 9, 5, 8, 1, 5, 6, 9)
211. (0, 5, 4, 7, 7, 3, 5, 8, 4, 5, 6, 9)
212. (0, 5, 4, 7, 7, 6, 5, 8, 4, 5, 6, 9)
213. (0, 5, 4, 7, 7, 9, 5, 8, 4, 5, 6, 9)
214. (0, 5, 4, 7, 7, 3, 5, 8, 7, 5, 6, 9)
215. (0, 5, 4, 7, 7, 6, 5, 8, 7, 5, 6, 9)
216. (0, 5, 4, 7, 7, 9, 5, 8, 7, 5, 6, 9)
217. (0, 5, 4, 7, 1, 3, 5, 8, 1, 8, 6, 9)
218. (0, 5, 4, 7, 1, 6, 5, 8, 1, 8, 6, 9)
219. (0, 5, 4, 7, 1, 9, 5, 8, 1, 8, 6, 9)
220. **(0, 5, 4, 7, 1, 3, 5, 8, 4, 8, 6, 9)**
221. (0, 5, 4, 7, 1, 6, 5, 8, 4, 8, 6, 9)
222. (0, 5, 4, 7, 1, 9, 5, 8, 4, 8, 6, 9)
223. **(0, 5, 4, 7, 1, 3, 5, 8, 7, 8, 6, 9)**
224. (0, 5, 4, 7, 1, 6, 5, 8, 7, 8, 6, 9)
225. (0, 5, 4, 7, 1, 9, 5, 8, 7, 8, 6, 9)
226. (0, 5, 4, 7, 4, 3, 5, 8, 1, 8, 6, 9)
227. (0, 5, 4, 7, 4, 6, 5, 8, 1, 8, 6, 9)
228. (0, 5, 4, 7, 4, 9, 5, 8, 1, 8, 6, 9)
229. (0, 5, 4, 7, 4, 3, 5, 8, 4, 8, 6, 9)
230. (0, 5, 4, 7, 4, 6, 5, 8, 4, 8, 6, 9)
231. (0, 5, 4, 7, 4, 9, 5, 8, 4, 8, 6, 9)
232. (0, 5, 4, 7, 4, 3, 5, 8, 7, 8, 6, 9)
233. **(0, 5, 4, 7, 4, 6, 5, 8, 7, 8, 6, 9)**
234. **(0, 5, 4, 7, 4, 9, 5, 8, 7, 8, 6, 9)**
235. (0, 5, 4, 7, 7, 3, 5, 8, 1, 8, 6, 9)
236. (0, 5, 4, 7, 7, 6, 5, 8, 1, 8, 6, 9)
237. (0, 5, 4, 7, 7, 9, 5, 8, 1, 8, 6, 9)
238. (0, 5, 4, 7, 7, 3, 5, 8, 4, 8, 6, 9)
239. **(0, 5, 4, 7, 7, 6, 5, 8, 4, 8, 6, 9)**
240. **(0, 5, 4, 7, 7, 9, 5, 8, 4, 8, 6, 9)**
241. (0, 5, 4, 7, 7, 3, 5, 8, 7, 8, 6, 9)
242. (0, 5, 4, 7, 7, 6, 5, 8, 7, 8, 6, 9)
243. (0, 5, 4, 7, 7, 9, 5, 8, 7, 8, 6, 9)
244. (0, 8, 4, 7, 1, 3, 5, 8, 1, 2, 6, 9)
245. (0, 8, 4, 7, 1, 6, 5, 8, 1, 2, 6, 9)
246. (0, 8, 4, 7, 1, 9, 5, 8, 1, 2, 6, 9)
247. **(0, 8, 4, 7, 1, 3, 5, 8, 4, 2, 6, 9)**
248. (0, 8, 4, 7, 1, 6, 5, 8, 4, 2, 6, 9)
249. (0, 8, 4, 7, 1, 9, 5, 8, 4, 2, 6, 9)
250. **(0, 8, 4, 7, 1, 3, 5, 8, 7, 2, 6, 9)**
251. (0, 8, 4, 7, 1, 6, 5, 8, 7, 2, 6, 9)
252. **(0, 8, 4, 7, 1, 9, 5, 8, 7, 2, 6, 9)**
253. **(0, 8, 4, 7, 4, 3, 5, 8, 1, 2, 6, 9)**
254. **(0, 8, 4, 7, 4, 6, 5, 8, 1, 2, 6, 9)**
255. **(0, 8, 4, 7, 4, 9, 5, 8, 1, 2, 6, 9)**
256. (0, 8, 4, 7, 4, 3, 5, 8, 4, 2, 6, 9)
257. (0, 8, 4, 7, 4, 6, 5, 8, 4, 2, 6, 9)
258. (0, 8, 4, 7, 4, 9, 5, 8, 4, 2, 6, 9)
259. (0, 8, 4, 7, 4, 3, 5, 8, 7, 2, 6, 9)
260. (0, 8, 4, 7, 4, 6, 5, 8, 7, 2, 6, 9)
261. (0, 8, 4, 7, 4, 9, 5, 8, 7, 2, 6, 9)
262. **(0, 8, 4, 7, 7, 3, 5, 8, 1, 2, 6, 9)**
263. **(0, 8, 4, 7, 7, 6, 5, 8, 1, 2, 6, 9)**
264. **(0, 8, 4, 7, 7, 9, 5, 8, 1, 2, 6, 9)**
265. (0, 8, 4, 7, 7, 3, 5, 8, 4, 2, 6, 9)
266. (0, 8, 4, 7, 7, 6, 5, 8, 4, 2, 6, 9)
267. (0, 8, 4, 7, 7, 9, 5, 8, 4, 2, 6, 9)
268. (0, 8, 4, 7, 7, 3, 5, 8, 7, 2, 6, 9)
269. (0, 8, 4, 7, 7, 6, 5, 8, 7, 2, 6, 9)
270. (0, 8, 4, 7, 7, 9, 5, 8, 7, 2, 6, 9)
271. (0, 8, 4, 7, 1, 3, 5, 8, 1, 5, 6, 9)
272. (0, 8, 4, 7, 1, 6, 5, 8, 1, 5, 6, 9)
273. (0, 8, 4, 7, 1, 9, 5, 8, 1, 5, 6, 9)
274. **(0, 8, 4, 7, 1, 3, 5, 8, 4, 5, 6, 9)**
275. (0, 8, 4, 7, 1, 6, 5, 8, 4, 5, 6, 9)

276. (0, 8, 4, 7, 1, 9, 5, 8, 4, 5, 6, 9)
277. **(0, 8, 4, 7, 1, 3, 5, 8, 7, 5, 6, 9)**
278. (0, 8, 4, 7, 1, 6, 5, 8, 7, 5, 6, 9)
279. (0, 8, 4, 7, 1, 9, 5, 8, 7, 5, 6, 9)
280. (0, 8, 4, 7, 4, 3, 5, 8, 1, 5, 6, 9)
281. (0, 8, 4, 7, 4, 6, 5, 8, 1, 5, 6, 9)
282. (0, 8, 4, 7, 4, 9, 5, 8, 1, 5, 6, 9)
283. (0, 8, 4, 7, 4, 3, 5, 8, 4, 5, 6, 9)
284. (0, 8, 4, 7, 4, 6, 5, 8, 4, 5, 6, 9)
285. (0, 8, 4, 7, 4, 9, 5, 8, 4, 5, 6, 9)
286. (0, 8, 4, 7, 4, 3, 5, 8, 7, 5, 6, 9)
287. **(0, 8, 4, 7, 4, 6, 5, 8, 7, 5, 6, 9)**
288. **(0, 8, 4, 7, 4, 9, 5, 8, 7, 5, 6, 9)**
289. (0, 8, 4, 7, 7, 3, 5, 8, 1, 5, 6, 9)
290. (0, 8, 4, 7, 7, 6, 5, 8, 1, 5, 6, 9)
291. (0, 8, 4, 7, 7, 9, 5, 8, 1, 5, 6, 9)
292. (0, 8, 4, 7, 7, 3, 5, 8, 4, 5, 6, 9)
293. **(0, 8, 4, 7, 7, 6, 5, 8, 4, 5, 6, 9)**
294. **(0, 8, 4, 7, 7, 9, 5, 8, 4, 5, 6, 9)**
295. (0, 8, 4, 7, 7, 3, 5, 8, 7, 5, 6, 9)
296. (0, 8, 4, 7, 7, 6, 5, 8, 7, 5, 6, 9)
297. (0, 8, 4, 7, 7, 9, 5, 8, 7, 5, 6, 9)
298. (0, 8, 4, 7, 1, 3, 5, 8, 1, 8, 6, 9)
299. (0, 8, 4, 7, 1, 6, 5, 8, 1, 8, 6, 9)
300. (0, 8, 4, 7, 1, 9, 5, 8, 1, 8, 6, 9)
301. **(0, 8, 4, 7, 1, 3, 5, 8, 4, 8, 6, 9)**
302. (0, 8, 4, 7, 1, 6, 5, 8, 4, 8, 6, 9)
303. (0, 8, 4, 7, 1, 9, 5, 8, 4, 8, 6, 9)
304. **(0, 8, 4, 7, 1, 3, 5, 8, 7, 8, 6, 9)**
305. (0, 8, 4, 7, 1, 6, 5, 8, 7, 8, 6, 9)
306. (0, 8, 4, 7, 1, 9, 5, 8, 7, 8, 6, 9)
307. (0, 8, 4, 7, 4, 3, 5, 8, 1, 8, 6, 9)
308. (0, 8, 4, 7, 4, 6, 5, 8, 1, 8, 6, 9)
309. (0, 8, 4, 7, 4, 9, 5, 8, 1, 8, 6, 9)
310. (0, 8, 4, 7, 4, 3, 5, 8, 4, 8, 6, 9)
311. (0, 8, 4, 7, 4, 6, 5, 8, 4, 8, 6, 9)
312. (0, 8, 4, 7, 4, 9, 5, 8, 4, 8, 6, 9)
313. (0, 8, 4, 7, 4, 3, 5, 8, 7, 8, 6, 9)
314. (0, 8, 4, 7, 4, 6, 5, 8, 7, 8, 6, 9)
315. (0, 8, 4, 7, 4, 9, 5, 8, 7, 8, 6, 9)
316. (0, 8, 4, 7, 7, 3, 5, 8, 1, 8, 6, 9)
317. (0, 8, 4, 7, 7, 6, 5, 8, 1, 8, 6, 9)
318. (0, 8, 4, 7, 7, 9, 5, 8, 1, 8, 6, 9)
319. (0, 8, 4, 7, 7, 3, 5, 8, 4, 8, 6, 9)
320. (0, 8, 4, 7, 7, 6, 5, 8, 4, 8, 6, 9)
321. (0, 8, 4, 7, 7, 9, 5, 8, 4, 8, 6, 9)
322. **(0, 8, 4, 7, 7, 3, 5, 8, 7, 8, 6, 9)**
323. **(0, 8, 4, 7, 7, 6, 5, 8, 7, 8, 6, 9)**
324. **(0, 8, 4, 7, 7, 9, 5, 8, 7, 8, 6, 9)**

The following are the index numbers of the 123 encodings in the 4D expansion of the 2D sunflower which provide Formality graphs Γ whose formulas $\phi(\Gamma)$ are linearly independent. Note that indices start from 1. In bold are the indices of the 64 4D sunflower encodings which provide Formality graphs Γ whose skew pairs $\frac{1}{2}\Big(\phi\big(\Gamma(a^1,a^2)\big) - \phi\big(\Gamma(a^2,a^1)\big)\Big)$ are linearly independent.

1, **2**, 3, **4**, **5**, **6**, 7, 8, 9, **11**, **12**, **13**, **14**, **15**, **16**, **17**, **18**, 20, 21, 23, 24, 25, 26, 27, **29**, **30**, **33**, **35**, **36**, 39, **41**, **42**, **44**, **45**, 48, **51**, **53**, 54, 57, 60, 69, 72, 81, **82**, **83**, 84, **94**, **95**, **96**, **97**, **98**, 103, 105, 106, 107, 108, **109**, **110**, **111**, **114**, **117**, **118**, 120, **127**, 129, 136, 138, 145, 154, **163**, **165**, **168**, 171, **175**, **177**, 180, **186**, **187**, **188**, **189**, **190**, **191**, **192**, 195, **197**, **198**, **206**, **207**, **208**, 210, **211**, **213**, **214**, **215**, **216**, **217**, **218**, 219, 224, 236, **241**, **242**, 243, 244, 245, 256, 268, 269, 271, 283, 285, 296, 298, 299, 300, 302, 303, 307, 310, 311, 312, 313, 314.

For clarity, below are the indices of the 64 encodings in the 4D expansion of the 2D sunflower which provide Formality graphs Γ whose skew pairs $\frac{1}{2}\Big(\phi\big(\Gamma(a^1,a^2)\big) - \phi\big(\Gamma(a^2,a^1)\big)\Big)$ are linearly independent. Note that indices start from 1.

1, 2, 4, 5, 6, 11, 12, 13, 14, 15, 16, 17, 18, 29, 30, 33, 35, 36, 41, 42, 44, 45, 51, 53, 82, 83, 94, 95, 96, 97, 98, 109, 110, 111, 114, 117, 118, 127, 163, 165, 168, 175, 177, 186, 187, 188, 189, 190, 191, 192, 197, 198, 206, 207, 208, 211, 213, 214, 215, 216, 217, 218, 241, 242.

Kontsevich graphs act on Nambu–Poisson brackets, III. Uniqueness aspects

F M Schipper, M S Jagoe Brown and A V Kiselev

Bernoulli Institute for Mathematics, Computer Science and Artificial Intelligence, University of Groningen, P.O. Box 407, 9700 AK Groningen, The Netherlands

E-mail: f.m.schipper@rug.nl, a.v.kiselev@rug.nl

Abstract. Kontsevich constructed a map between 'good' graph cocycles γ and infinitesimal deformations of Poisson bivectors on affine manifolds, that is, Poisson cocycles in the second Lichnerowicz–Poisson cohomology. For the tetrahedral graph cocycle γ_3 and for the class of Nambu-determinant Poisson bivectors P over $\mathbb{R}^2$, $\mathbb{R}^3$ and $\mathbb{R}^4$, we know the fact of trivialization, $\dot{P} = [\![P, \vec{X}^{\gamma_3}_{\mathrm{dim}}]\!]$, by using dimension-dependent vector fields $\vec{X}^{\gamma_3}_{\mathrm{dim}}$ expressed by Kontsevich (micro-)graphs. We establish that these trivializing vector fields $\vec{X}^{\gamma_3}_{\mathrm{dim}}$ are unique modulo Hamiltonian vector fields $\vec{X}_H = \mathrm{d}_P(H) = [\![P, H]\!]$, where d_P is the Lichnerowicz–Poisson differential and where the Hamiltonians H are also represented by Kontsevich (micro-)graphs. However, we find that the choice of Kontsevich (micro-)graphs to represent the aforementioned multivectors is not unique.

1 Introduction

In 1977, Lichnerowicz introduced a cohomology theory for Poisson manifolds [1]. In this theory, the differential is given by $\mathrm{d}_P = [\![P, \cdot\,]\!]$, where the bracket $[\![\cdot\,,\,\cdot\,]\!]$ is the Schouten bracket and P is a Poisson bivector. The corresponding cochain complex is given by

$$0 \longrightarrow \mathbb{R} \hookrightarrow C^\infty(M^d) \xrightarrow{\mathrm{d}_P} \mathfrak{X}(M^d) \xrightarrow{\mathrm{d}_P} \mathfrak{X}^2(M^d) \xrightarrow{\mathrm{d}_P} \ldots \xrightarrow{\mathrm{d}_P} \mathfrak{X}^d(M^d) \longrightarrow 0. \qquad (\star)$$

In 1996, Kontsevich related 'good' graph cocycles γ in his graph complex GC to infinitesimal deformations of Poisson bivectors $\dot{P} = Q^\gamma(P) \in \mathfrak{X}^2(M^d_{\mathrm{aff}})$ (which belong to the kernel of the Poisson differential d_P) on an affine Poisson manifold M^d_{aff} [2]. The smallest good graph cocycle γ is the tetrahedral graph cocycle γ_3. We investigate whether the corresponding tetrahedral flow $Q^{\gamma_3} = \mathsf{Or}(\gamma_3)(P)$ is trivial, i.e., whether in addition to being a cocycle, it is also a coboundary. While an immediate thought is to study the trivialization problem on the level of graphs, it is shown that there cannot exist a universal trivializing solution on the level of directed graphs [2, 3]. Instead, we use the morphism ϕ to pass from graphs to multivectors [4], and study the trivialization problem $\dot{P} = Q^{\gamma_3}(P) = [\![P, \vec{X}^{\gamma_3}]\!]$ on the level of multivectors that can be represented by graphs. Let us denote by $\mathfrak{X}^k_{\mathrm{gra}}(M^d_{\mathrm{aff}})$ the space of those k-vectors on an affine manifold M^d_{aff} which are obtained from graphs. When we restrict the cochain complex $(\star)$ to these spaces $\mathfrak{X}^k(M^d_{\mathrm{aff}})$, we get a subcochain complex for Poisson cohomology [2]

$$0 \longrightarrow \mathbb{R} \hookrightarrow C^\infty_{\mathrm{gra}}(M^d_{\mathrm{aff}}) \xrightarrow{\mathrm{d}_P} \mathfrak{X}_{\mathrm{gra}}(M^d_{\mathrm{aff}}) \xrightarrow{\mathrm{d}_P} \mathfrak{X}^2_{\mathrm{gra}}(M^d_{\mathrm{aff}}) \xrightarrow{\mathrm{d}_P} \ldots \xrightarrow{\mathrm{d}_P} \mathfrak{X}^d_{\mathrm{gra}}(M^d_{\mathrm{aff}}) \longrightarrow 0. \qquad (*)$$

Additionally, we restrict to the class of Nambu-determinant Poisson brackets [5], and we use dimension specific Kontsevich (micro-)graphs for this class, as introduced in [6]. This text is a continuation of [4].

This text is structured as follows. In section 2 we introduce some notions and phrase the problem. Then we state the main results: the trivializing vector fields of the tetrahedral graph flow over $\mathbb{R}^d$ ($d \leqslant 4$) are unique modulo Hamiltonian vector fields, see section 3 for $\mathbb{R}^2$, section 4 for $\mathbb{R}^3$ and section 5 for $\mathbb{R}^4$. In section 6 we discuss the non-uniqueness of graphs chosen to represent specific multivectors. (All proofs presented in this text are direct calculations.[1])

2 The trivializing vector fields modulo Hamiltonian vector fields: preliminaries

We consider vector fields $\vec{Y} \in \mathfrak{X}_{\text{gra}}(\mathbb{R}^d)$ solving the homogeneous equation $\mathrm{d}_P(\vec{Y}) = 0$, and we show explicitly that these vector fields are expressed by a (linear combination of) Hamiltonian vector field(s) that we compute in advance. The proof structure is the same for each dimension $d = 2, 3, 4$.

Recall that to solve the trivialization problem for the tetrahedral flow, we must find a vector field $\vec{X}^{\gamma_3}_{\text{dim}}$ satisfying the nonhomogeneous linear algebraic equation

$$\dot{P} = Q^{\gamma_3}_{\text{dim}}(P) = [\![P, \vec{X}^{\gamma_3}_{\text{dim}}]\!]. \tag{1}$$

As usual, solutions $\vec{Y}_{\text{dim}}$ to the homogeneous equation,

$$[\![P, \vec{Y}_{\text{dim}}]\!] = 0 \in \mathfrak{X}^2_{\text{gra}}(\mathbb{R}^d), \tag{2}$$

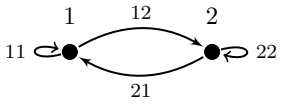
give us all the solutions to equation (1) via $\vec{X}^{\gamma_3}_{\text{dim}} + \vec{Y}_{\text{dim}}$. In the following sections, we show that the vector fields $\vec{Y}_{\text{dim}}$ solving the homogeneous system (2) are Hamiltonian vector fields.

Definition 1. We call $H \in C^\infty_{\text{gra}}(\mathbb{R}^d)$ *Hamiltonians*. Moreover, we call a vector field $\vec{X}_H \in \mathfrak{X}_{\text{gra}}(\mathbb{R}^d)$ *Hamiltonian* if it is in the image of the Lichnerowicz–Poisson differential $\mathrm{d}_P = [\![P, \cdot\,]\!]$, that is, $\vec{X}_H = \mathrm{d}_P(H) = [\![P, H]\!]$, for some Hamiltonian H.

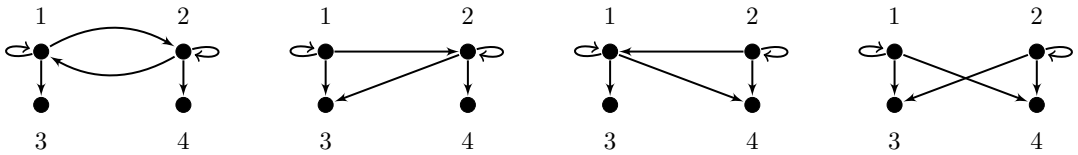
Notation 1. We will denote a directed edge $(i, j) \in E(\Gamma)$ of a graph Γ, where $i, j \in V(\Gamma)$, by the shorthand notation ij.

Definition 2. The set of *d-dimensional descendants* $(\widehat{\Gamma})_d$ of a two-dimensional Kontsevich graph Γ is the collection of all the Nambu micro-graphs obtained from Γ by adding $d-2$ Casimir vertices at each Nambu-determinant Poisson structure and redirecting the two original outgoing edges at each Levi-Civita vertex via the Leibniz rule over all the vertices of the targeted Poisson structure(s).

Example 1. Consider the two-dimensional Kontsevich graph encoded[2] by $\Gamma = [1, 2; 1, 2]$. This is a graph on Levi-Civita vertices 1 and 2, with directed and ordered edges $11 = (1, 1) \prec (1, 2) = 12$, $21 = (2, 1) \prec (2, 2) = 22$.



The three-dimensional Nambu micro-graph descendants of this Kontsevich graph are given by



[1]The SageMath package gcaops (https://github.com/rburing/gcaops) is used to convert graphs to multivectors, and to solve linear algebraic systems for coefficients of graphs. All the code used for these calculations is attached.

[2]Edges are issued only from the Levi-Civita vertices. The outgoing edges corresponding to each Levi-Civita vertex are separated by the semicolon ; and encoded by the label of the target vertex. As an extra example, the encoding $[1, 2, 3; 2, 3, 4]$ has two Levi-Civita vertices $1, 2$, as well as two Casimir vertices 3 and 4, and six directed edges $11 \prec 12 \prec 13$, $22 \prec 23 \prec 24$. See also Example 2.

where 3 (respectively 4) is the Casimir vertex added to Levi-Civita vertex 1 (respectively 2). The corresponding encodings are given by

$$(\widehat{\Gamma})_{3D} = \big\{[1,2,3;1,2,4],\quad [1,2,3;3,2,4],\quad [1,4,3;1,2,4],\quad [1,4,3;3,2,4]\big\}.$$

Definition 3. The *embedding* of a Kontsevich (micro-)graph $\Gamma_{\dim}$ built from n Nambu-determinant Poisson structures into dimension $\dim+1$ is the graph $\Gamma_{\dim+1} = \mathrm{emb}(\Gamma_{\dim})$ such that to the Levi-Civita vertex of each Nambu-determinant Poisson structure, we add an extra Casimir vertex a^{d-1}. The original d outgoing edges of each Levi-Civita vertex keep their order, and the new edge is ordered last. The embedding can often be viewed as a specific type of descendant of a graph.

Example 2. Consider again the two-dimensional Kontsevich graph encoded by $[1,2;1,2]$. The embedding into three dimensions is encoded by $[1,2,3;1,2,4]$, where $3,4$ are the new Casimir vertices. The edges are ordered $11 \prec 12 \prec \mathbf{13}$, $21 \prec 22 \prec \mathbf{24}$, where $\mathbf{13}$, $\mathbf{24}$ are the new edges.

3 The trivializing vector fields modulo Hamiltonian vector fields: $\vec{X}_{2D}^{\gamma_3}$

The fact of trivialization of the tetrahedral flow of Poisson bivectors over $\mathbb{R}^2$ has been known since 1996 by M. Kontsevich [2] (see also [7]).

Lemma 1. There are 14 nonisomorphic Kontsevich graphs on three Levi-Civita vertices and one sink. Explicitly, these 14 graphs are given by the following encodings.[3]

$$\begin{array}{lllll}
\Gamma_1^{2D} = [0,1;2,3;1,3] & \Gamma_2^{2D} = [0,1;1,2;1,3] & \Gamma_3^{2D} = [0,3;2,3;2,3] & \Gamma_4^{2D} = [0,3;2,3;1,3] & \Gamma_5^{2D} = [0,2;2,3;1,3] \\
\Gamma_6^{2D} = [0,3;1,2;1,3] & \Gamma_7^{2D} = [0,3;2,3;1,2] & \Gamma_8^{2D} = [0,3;1,2;1,2] & \Gamma_9^{2D} = [0,2;2,3;1,2] & \Gamma_{10}^{2D} = [0,2;1,2;1,2] \\
\Gamma_{11}^{2D} = [0,1;1,3;1,2] & \Gamma_{12}^{2D} = [0,3;1,3;1,2] & \Gamma_{13}^{2D} = [0,1;1,3;2,3] & \Gamma_{14}^{2D} = [0,1;1,3;1,3] &
\end{array}$$

Claim 2 (See the attached code)**.** *The images of the* 14 *nonisomorphic Kontsevich graphs* $\Gamma_1^{2D},...,\Gamma_{14}^{2D}$ *under the morhpism* ϕ *from graphs to multivectors satisfy the following linear relations:*

$$\begin{aligned}
&\phi(\Gamma_1^{2D}) = \phi(\Gamma_5^{2D}) = \phi(\Gamma_6^{2D}) = -\phi(\Gamma_7^{2D}) = \tfrac{1}{2}\phi(\Gamma_8^{2D}) = \phi(\Gamma_{12}^{2D}) = \phi(\Gamma_{13}^{2D}), \\
&\phi(\Gamma_2^{2D}) = \phi(\Gamma_4^{2D}) = -\phi(\Gamma_9^{2D}) = \phi(\Gamma_{11}^{2D}), \qquad \phi(\Gamma_3^{2D}) = \phi(\Gamma_{10}^{2D}) = \phi(\Gamma_{14}^{2D}).
\end{aligned} \tag{3}$$

This means that in dimension two, when restricting ourselves to the formulas which the 14 graphs are evaluated into, we get only three linearly independent vector fields $\phi(\Gamma_{11}^{2D})$, $\phi(\Gamma_{12}^{2D})$ and $\phi(\Gamma_3^{2D})$: over the first two vector fields we find a unique solution $\vec{X}_{2D}^{\gamma_3}$ to equation (1), while the third vector field solves the homogeneous system (2).

Proposition 3 ([7, Proposition 1])**.** The trivializing vector field $\vec{X}_{2D}^{\gamma_3}$ for the tetrahedral flow of Poisson bivectors $P = \varrho(x,y)\,\partial_x \wedge \partial_y$ with Cartesian coordinates, up to a normalization constant $\frac{1}{8}$, is given by

$$\begin{aligned}
\vec{X}_{2D}^{\gamma_3} &= 1\cdot\phi(\Gamma_{11}^{2D}) + 2\cdot\phi(\Gamma_{12}^{2D}) \\
&= (-2\varrho_y(\varrho_{xy})^2 + 2\varrho_y\varrho_{xx}\varrho_{yy} + (\varrho_y)^2\varrho_{xxy} - 2\varrho_x\varrho_y\varrho_{xyy} + (\varrho_x)^2\varrho_{yyy})\xi_x \\
&\quad + (2\varrho_x(\varrho_{xy})^2 - 2\varrho_x\varrho_{xx}\varrho_{yy} - (\varrho_y^2)\varrho_{xxx} + 2\varrho_x\varrho_y\varrho_{xxy} - (\varrho_x)^2\varrho_{xyy})\xi_y.
\end{aligned}$$

Proposition 4. On $\mathbb{R}^2$, there is a unique vector field represented by Kontsevich graphs (modulo nonzero constant multiples) $\vec{Y}^{2D}$ such that $[\![P,\vec{Y}^{2D}]\!] = 0$.

Proof. See the attached code for the computation yielding precisely one vector field solving equation (2). Explicitly, the vector field is given by

$$\phi(\Gamma_3^{2D}) = (\varrho\varrho_{yy}\varrho_{xxy} - 2\varrho\varrho_{xy}\varrho_{xyy} + \varrho\varrho_{xx}\varrho_{yyy})\xi_x + (-\varrho\varrho_{yy}\varrho_{xxx} + 2\varrho\varrho_{xy}\varrho_{xxy} - \varrho\varrho_{xx}\varrho_{xyy})\xi_y.$$

Another direct computation yields that $\mathrm{d}_P(\phi(\Gamma_3^{2D})) = [\![P,\phi(\Gamma_3^{2D})]\!] = 0$. □

Let us examine the degree of freedom coming from $\phi(\Gamma_3^{2D})$. Consider the Hamiltonians we can create from Kontsevich graphs on $\mathbb{R}^2$. Since the vector fields at hand contain three copies of the Poisson structure as vertices, and the Poisson–Lichnerowicz differential $\mathrm{d}_P = [\![P,\ \cdot\ ]\!]$ adds another Poisson structure, we conclude that our Hamiltonian(s) must contain two Poisson structures.

[3] Here, 0 is the sink vertex, the Levi-Civita vertices are given by 1, 2, 3.

Lemma 5. There is only one way to create a Kontsevich graph on two Levi-Civita vertices 1, 2 and no sink. The encoding for this Hamiltonian graph is given by $\Gamma^{2D}_{H_1} = [1, 2; 1, 2]$.

Notation 2. Put $H_i^{\dim} = \phi(\Gamma^{\dim}_{H_i})$; the same notation is used for dimensions three and four.

Theorem 6. *On $\mathbb{R}^2$, let $P = \varrho(x, y)\ \partial_x \wedge \partial_y$ be a (possibly degenerate) Poisson bivector. Consider the complex $(*)$ restricted to Hamiltonians on 2 copies of P, vector fields on 3 copies of P and bivectors on 4 copies of P. We establish that the corresponding homogeneous part of the first Poisson-Lichnerowicz cohomology $H^1_{\mathrm{gra}}(\mathbb{R}^2)$ is trivial.*

Proof (see the attached code). We write the vector field $\vec{Y}^{2D}$ of Proposition 4 in terms of the Hamiltonian vector field $\mathrm{d}_P(H_1^{2D})$. By a direct calculation, we establish that

$$2 \cdot \vec{Y}^{2D} = 2 \cdot \phi(\Gamma_3^{2D}) = \mathrm{d}_P(H_1^{2D}),$$

that is, the degree of freedom is provided by the Hamiltonian shift. It follows immediately that the corresponding homogeneous part of $H^1_{\mathrm{gra}}(\mathbb{R}^2) = \ker \mathrm{d}_P \,/\mathrm{im}\ \mathrm{d}_P$ is trivial. □

Corollary 7. The trivializing vector field $\vec{X}^{\gamma_3}_{2D}$ of Proposition 3 is unique modulo Hamiltonian vector fields.

4 The trivializing vector fields modulo Hamiltonian vector fields: $\vec{X}^{\gamma_3}_{3D}$

In dimension three, triviality of the tetrahedral graph cocycle was established in [5,6]. Interestingly, we can also find a trivializing vector field over just the descendants $(\widehat{\Gamma}^{2D}_{11})_{3D}$ and $(\widehat{\Gamma}^{2D}_{12})_{3D}$.

Lemma 8 ([4, Lemma 3])**.** The set $(\widehat{\Gamma}^{2D}_{11})_{3D} \cup (\widehat{\Gamma}^{2D}_{12})_{3D}$ contains 41 non-isomorphic Nambu micro-graphs.

Proposition 9. For the tetrahedral flow of Poisson bivectors over $\mathbb{R}^3$, the trivializing vector field $\vec{X}^{\gamma_3}_{3D}$, restricted to vector fields corresponding to the descendants $(\widehat{\Gamma}^{2D}_{11})_{3D} \cup (\widehat{\Gamma}^{2D}_{12})_{3D}$ of the graphs $\Gamma^{2D}_{11}, \Gamma^{2D}_{12}$ of Lemma 1, is given by

$$\begin{aligned}\vec{X}^{\gamma_3}_{3D} = \ & 8 \cdot \phi(\Gamma^{3D}_1) + 24 \cdot \phi(\Gamma^{3D}_4) + 8 \cdot \phi(\Gamma^{3D}_7) + 24 \cdot \phi(\Gamma^{3D}_8) + 12 \cdot \phi(\Gamma^{3D}_{16}) + 16 \cdot \phi(\Gamma^{3D}_{17}) + 16 \cdot \phi(\Gamma^{3D}_{25}) \\ & + 12 \cdot \phi(\Gamma^{3D}_{26}) + 16 \cdot \phi(\Gamma^{3D}_{29}) + 24 \cdot \phi(\Gamma^{3D}_{33}).\end{aligned}$$

Proof. See [4] and the attached code. □

The corresponding encodings[4] of the 1-vector graphs appearing in $\vec{X}^{\gamma_3}_{3D}$ are these:

$\Gamma^{3D}_1 = [0, 1, 4; 1, 3, 5; 1, 2, 6]$ $\Gamma^{3D}_4 = [0, 1, 4; 1, 6, 5; 4, 2, 6]$ $\Gamma^{3D}_7 = [0, 1, 4; 4, 3, 5; 4, 2, 6]$ $\Gamma^{3D}_8 = [0, 1, 4; 4, 6, 5; 4, 2, 6]$
$\Gamma^{3D}_{16} = [0, 1, 4; 4, 6, 5; 4, 5, 6]$ $\Gamma^{3D}_{17} = [0, 2, 4; 1, 3, 5; 1, 2, 6]$ $\Gamma^{3D}_{25} = [0, 2, 4; 1, 3, 5; 1, 5, 6]$ $\Gamma^{3D}_{26} = [0, 2, 4; 1, 6, 5; 1, 5, 6]$
$\Gamma^{3D}_{29} = [0, 2, 4; 4, 3, 5; 1, 5, 6]$ $\Gamma^{3D}_{33} = [0, 5, 4; 1, 3, 5; 1, 2, 6]$.

Proposition 10. There are three linearly independent vector fields $\vec{Y}^{3D}_1$, $\vec{Y}^{3D}_2$ and $\vec{Y}^{3D}_3$ that span the solution space of $[\![P, \vec{X}^{3D}]\!] = 0$ when restricting to solution over linear combinations of the descendants $(\widehat{\Gamma}^{2D}_{11})_{3D} \cup (\widehat{\Gamma}^{2D}_{12})_{3D}$.

Proof. See the attached code for the computation yielding precisely three vector fields solving equation (2). Explicitly, these vector fields are (with their encodings found directly below)

$$\begin{aligned}\vec{Y}^{3D}_1 &= 1 \cdot \phi(\Gamma^{3D}_2) + 1 \cdot \phi(\Gamma^{3D}_{18}) + 1 \cdot \phi(\Gamma^{3D}_{34}) + 1 \cdot \phi(\Gamma^{3D}_{41}) \\ \vec{Y}^{3D}_2 &= 1 \cdot \phi(\Gamma^{3D}_4) + \tfrac{1}{2} \cdot \phi(\Gamma^{3D}_{31}) + \tfrac{1}{2} \cdot \phi(\Gamma^{3D}_{45}) \\ \vec{Y}^{3D}_3 &= 1 \cdot \phi(\Gamma^{3D}_{10}) + 1 \cdot \phi(\Gamma^{3D}_{31}) + 2 \cdot \phi(\Gamma^{3D}_{34}) + 2 \cdot \phi(\Gamma^{3D}_{42}) + 1 \cdot \phi(\Gamma^{3D}_{45}).\end{aligned}$$

□

[4]Here, 0 is the sink, we have Levi-Civita vertices 1, 2 and 3 with the respective Casimir vertices 4, 5 and 6.

The corresponding encodings[5] are as follows:

$\Gamma_{2}^{3D} = [0,1,4;1,6,5;1,2,6]$ $\Gamma_{4}^{3D} = [0,1,4;1,6,5;4,2,6]$ $\Gamma_{10}^{3D} = [0,1,4;1,6,5;1,5,6]$
$\Gamma_{18}^{3D} = [0,2,4;1,6,5;1,2,6]$ $\Gamma_{31}^{3D} = [0,2,4;4,3,5;4,5,6]$ $\Gamma_{34}^{3D} = [0,5,4;1,6,5;1,2,6]$
$\Gamma_{41}^{3D} = [0,5,4;1,3,5;1,5,6]$ $\Gamma_{42}^{3D} = [0,5,4;1,6,5;1,5,6]$ $\Gamma_{45}^{3D} = [0,5,4;4,3,5;1,5,6]$

Again, we examine these degrees of freedom.

Lemma 11. There are seven nonisomorphic Nambu micro-graphs on two Levi-Civita vertices 1, 2, two corresponding Casimir vertices 3, 4 and no sink.

The encodings for these seven Hamiltonians are given directly below.

$\Gamma_{H_1}^{3D} = [2,3,4;1,3,4]$ $\Gamma_{H_2}^{3D} = [2,3,4;2,3,4]$ $\Gamma_{H_3}^{3D} = [2,3,4;1,2,4]$ $\Gamma_{H_4}^{3D} = [1,3,4;1,2,4]$
$\Gamma_{H_5}^{3D} = [1,3,4;2,3,4]$ $\Gamma_{H_6}^{3D} = [1,2,3;1,2,4]$ $\Gamma_{H_7}^{3D} = [1,2,3;2,3,4]$

We detect the following relations (they are explicitly verified in the attached code):

$$H_1^{3D} = H_5^{3D}, \qquad H_3^{3D} = -H_4^{3D} = -H_7^{3D}. \tag{4}$$

Remark 1. The graph $\Gamma_{H_6}^{3D}$ is precisely the three-dimensional embedding of the two-dimensional Hamiltonian $\Gamma_{H_1}^{2D}$ from Lemma 5. Since we are working only over $(\widehat{\Gamma}_{11}^{2D})_{3D}$ and $(\widehat{\Gamma}_{12}^{2D})_{3D}$, there is no fourth linearly independent vector field $\vec{Y}_{3D}^{4}$ created from linear combinations of vector fields evaluated from $(\widehat{\Gamma}_{11}^{2D})_{3D}$ and $(\widehat{\Gamma}_{12}^{2D})_{3D}$ in dimension three satisfying $[\![P, \vec{Y}_{3D}^{4}]\!] = 0$. When we run the same code over *all* three-dimensional Nambu micro-graphs (with three Levi-Civita vertices, three corresponding Casimir vertices and one sink), we *do* get this fourth vector field $\vec{Y}_{3D}^{4}$ which nontrivially depends on $\mathrm{d}_P(H_6^{3D}) = [\![P, H_6^{3D}]\!]$.

Theorem 12. *On $\mathbb{R}^3$, let P be a (degenerate) Nambu-determinant Poisson bivector. Consider the complex $(*)$ restricted to Hamiltonians on 2 copies of P, vector fields on 3 copies of P, bivectors on 4 copies of P and trivectors on 5 copies of P. We establish that the corresponding homogeneous part of the first Poisson–Lichnerowicz cohomology $H^1_{\mathrm{gra}}(\mathbb{R}^3)$ is trivial.*

Proof (see the attached code). We write each of the three vector fields $\vec{Y}_i^{3D}$ of Proposition 10 in terms of the Hamiltonian vector fields $\mathrm{d}_P(H_1^{3D})$, $\mathrm{d}_P(H_2^{3D})$ and $\mathrm{d}_P(H_3^{3D})$. We compute

$$\vec{Y}_1^{3D} = 1 \cdot \mathrm{d}_P(H_3^{3D}), \qquad \vec{Y}_2^{3D} = \tfrac{1}{4} \cdot \mathrm{d}_P(H_1^{3D}), \qquad \vec{Y}_3^{3D} = \tfrac{1}{2} \cdot \mathrm{d}_P(H_1^{3D}) - 1 \cdot \mathrm{d}_P(H_2^{3D}),$$

that is, the degrees of freedom are provided by the Hamiltonian shifts. It follows immediately that the corresponding homogeneous part of $H^1_{\mathrm{gra}}(\mathbb{R}^3) = \ker \mathrm{d}_P \,/\mathrm{im}\ \mathrm{d}_P$ is trivial. □

Corollary 13. The trivializing vector field $\vec{X}_{3D}^{\gamma_3}$ over the descendants $(\widehat{\Gamma}_{11}^{2D})_{3D}$ and $(\widehat{\Gamma}_{12}^{2D})_{3D}$ of Proposition 9 is unique modulo Hamiltonian vector fields.

5 The trivializing vector fields modulo Hamiltonian vector fields: $\vec{X}_{4D}^{\gamma_3}$

In dimension four, triviality of the tetrahedral graph cocycle is established in [4]. The trivializing vector field is found again over the descendants of the two-dimensional solution from Proposition 3, but in addition, we request that the vector field is skew-symmetric with respect to the two Casimirs a^1 and a^2, see [4].

Remark 2. As we are working with Nambu-determinant Poisson brackets, we see that the Poisson structure itself is skew-symmetric with respect to the Casimirs a^1 and a^2. By requesting that our Hamiltonians H are symmetric, we ensure that the Hamiltonian vector fields $\mathrm{d}_P(H)$ we consider are skew-symmetric with respect to a^1 and a^2.

Notation 3. Let us denote by $\phi^-(\Gamma_{4D})$ (respectively $\phi^+(\Gamma_{4D})$) the skew-symmetrized (respectively symmetrized) multivector obtained[6] from the graph Γ_{4D} by swapping the Casimirs a^1 and a^2. We write $(H_i^{4D})^+$ for the symmetrized Hamiltonian function represented by the graph $\Gamma_{H_i}^{4D}$.

[5] Here, 0 is the sink, we have Levi-Civita vertices 1, 2 and 3 with the respective Casimir vertices 4, 5 and 6.

[6] Consider as an example the four-dimensional graph $\Gamma(a^1, a^2)$ with encoding $[0,1,4,7;1,3,6,9;1,5,8,9]$ where 0 is the sink, $1,2,3$ are Levi-Civita vertices with corresponding a^1 Casimir vertices $4,5,6$ and a^2 Casimir vertices $7,8,9$. We simply swap the pairs of Casimirs vertices belonging to each Poisson structure to obtain $\Gamma(a^2, a^1) = [0,1,7,4;1,3,9,6;1,8,5,6]$. Then, $\phi^- = \frac{1}{2}(\phi(\Gamma(a^1,a^2)) - \phi(\Gamma(a^2,a^1)))$.

Remark 3. In the following, (skew-)symmetry is *always* with respect to the Casimirs a^1 and a^2.

Proposition 14 ([4, Proposition 8])**.** Over $\mathbb{R}^4$, there exists a skew solution $(\vec{X}^{\gamma_3}_{4D})^-$ solving equation (1); this trivializing vector field consists of 27 skew-symmetrized vector fields obtained from the descendants $(\widehat{\Gamma}^{2D}_{11})_{4D} \cup (\widehat{\Gamma}^{2D}_{12})_{4D}$.

Proposition 15. There are seven linearly independent vector fields $\vec{Y}^{4D}_1$, $\vec{Y}^{4D}_2$, $\vec{Y}^{4D}_3$, $\vec{Y}^{4D}_4$, $\vec{Y}^{4D}_5$, $\vec{Y}^{4D}_6$ and $\vec{Y}^{4D}_7$ that span the solution space of $[\![P, \vec{X}^{4D}]\!] = 0$ when restricting to solutions over linear combinations of the skew-symmetrized descendants $(\widehat{\Gamma}^{2D}_{11})_{4D} \cup (\widehat{\Gamma}^{2D}_{12})_{4D}$.

Proof. See the attached code for the computation yielding precisely seven skew-symmetric vector fields solving equation (2). Explicitly, these vector fields are

$$\begin{aligned}
\vec{Y}^1_{4D} &= 1\cdot\phi^-(\Gamma^{4D}_2) - \tfrac{1}{2}\cdot\phi^-(\Gamma^{4D}_9) + 1\cdot\phi^-(\Gamma^{4D}_{26}) + \tfrac{1}{2}\cdot\phi^-(\Gamma^{4D}_{33}) + 1\cdot\phi^-(\Gamma^{4D}_{35}) - 1\cdot\phi^-(\Gamma^{4D}_{36}) + 1\cdot\phi^-(\Gamma^{4D}_{40})\\
&\quad - 1\cdot\phi^-(\Gamma^{4D}_{41}) + \tfrac{1}{2}\cdot\phi^-(\Gamma^{4D}_{42}) + 1\cdot\phi^-(\Gamma^{4D}_{48}) + 1\cdot\phi^-(\Gamma^{4D}_{61})\\
\vec{Y}^2_{4D} &= 1\cdot\phi^-(\Gamma^{4D}_4) + \tfrac{1}{2}\cdot\phi^-(\Gamma^{4D}_9) - 1\cdot\phi^-(\Gamma^{4D}_{35}) + 1\cdot\phi^-(\Gamma^{4D}_{36}) + 1\cdot\phi^-(\Gamma^{4D}_{41}) - \tfrac{1}{2}\cdot\phi^-(\Gamma^{4D}_{42})\\
\vec{Y}^3_{4D} &= 1\cdot\phi^-(\Gamma^{4D}_{10}) - 1\cdot\phi^-(\Gamma^{4D}_{16}) + 1\cdot\phi^-(\Gamma^{4D}_{18}) + 1\cdot\phi^-(\Gamma^{4D}_{20}) - \tfrac{1}{2}\cdot\phi^-(\Gamma^{4D}_{24}) - 1\cdot\phi^-(\Gamma^{4D}_{31}) - 1\cdot\phi^-(\Gamma^{4D}_{34})\\
&\quad - 1\cdot\phi^-(\Gamma^{4D}_{35}) + 1\cdot\phi^-(\Gamma^{4D}_{36}) + 2\cdot\phi^-(\Gamma^{4D}_{40}) + 1\cdot\phi^-(\Gamma^{4D}_{41}) - 1\cdot\phi^-(\Gamma^{4D}_{43}) - 1\cdot\phi^-(\Gamma^{4D}_{45}) + 1\cdot\phi^-(\Gamma^{4D}_{46})\\
&\quad - 1\cdot\phi^-(\Gamma^{4D}_{47}) - \tfrac{1}{2}\cdot\phi^-(\Gamma^{4D}_{54}) - 1\cdot\phi^-(\Gamma^{4D}_{61}) + 1\cdot\phi^-(\Gamma^{4D}_{63}) - \tfrac{1}{2}\cdot\phi^-(\Gamma^{4D}_{64})\\
\vec{Y}^4_{4D} &= 1\cdot\phi^-(\Gamma^{4D}_{12}) + 1\cdot\phi^-(\Gamma^{4D}_{16}) - 1\cdot\phi^-(\Gamma^{4D}_{18}) - 1\cdot\phi^-(\Gamma^{4D}_{20}) + \tfrac{1}{2}\cdot\phi^-(\Gamma^{4D}_{24}) + 1\cdot\phi^-(\Gamma^{4D}_{31}) + 1\cdot\phi^-(\Gamma^{4D}_{35})\\
&\quad - 1\cdot\phi^-(\Gamma^{4D}_{36}) + 1\cdot\phi^-(\Gamma^{4D}_{43}) - 1\cdot\phi^-(\Gamma^{4D}_{46}) + 1\cdot\phi^-(\Gamma^{4D}_{47}) + \tfrac{1}{2}\cdot\phi^-(\Gamma^{4D}_{54}) - 1\cdot\phi^-(\Gamma^{4D}_{63}) + \tfrac{1}{2}\cdot\phi^-(\Gamma^{4D}_{64})\\
\vec{Y}^5_{4D} &= 1\cdot\phi^-(\Gamma^{4D}_{14}) + 4\cdot\phi^-(\Gamma^{4D}_{16}) - 4\cdot\phi^-(\Gamma^{4D}_{18}) + 4\cdot\phi^-(\Gamma^{4D}_{31}) + 1\cdot\phi^-(\Gamma^{4D}_{33}) + 2\cdot\phi^-(\Gamma^{4D}_{49}) + 4\cdot\phi^-(\Gamma^{4D}_{62})\\
\vec{Y}^6_{4D} &= 1\cdot\phi^-(\Gamma^{4D}_{15}) + 1\cdot\phi^-(\Gamma^{4D}_{34}) + 2\cdot\phi^-(\Gamma^{4D}_{50}) + 2\cdot\phi^-(\Gamma^{4D}_{62})\\
\vec{Y}^7_{4D} &= 1\cdot\phi^-(\Gamma^{4D}_{22}) - 2\cdot\phi^-(\Gamma^{4D}_{44}) + 1\cdot\phi^-(\Gamma^{4D}_{54}) + 1\cdot\phi^-(\Gamma^{4D}_{64}).
\end{aligned}$$

□

The corresponding encodings[7] are:

$$\begin{aligned}
&\Gamma^{4D}_2 = [0,1,4,7;1,6,5,8;1,2,6,9] && \Gamma^{4D}_4 = [0,1,4,7;1,6,5,8;4,2,6,9] && \Gamma^{4D}_9 = [0,1,4,7;4,6,5,8;4,2,6,9]\\
&\Gamma^{4D}_{10} = [0,1,4,7;4,9,5,8;4,2,6,9] && \Gamma^{4D}_{12} = [0,1,4,7;4,6,5,8;7,2,6,9] && \Gamma^{4D}_{14} = [0,1,4,7;1,6,5,8;1,5,6,9]\\
&\Gamma^{4D}_{15} = [0,1,4,7;1,9,5,8;1,5,6,9] && \Gamma^{4D}_{16} = [0,1,4,7;1,9,5,8;4,5,6,9] && \Gamma^{4D}_{18} = [0,1,4,7;1,9,5,8;7,5,6,9]\\
&\Gamma^{4D}_{20} = [0,1,4,7;4,9,5,8;4,5,6,9] && \Gamma^{4D}_{22} = [0,1,4,7;4,9,5,8;7,5,6,9] && \Gamma^{4D}_{24} = [0,1,4,7;7,6,5,8;7,5,6,9]\\
&\Gamma^{4D}_{26} = [0,2,4,7;1,6,5,8;1,2,6,9] && \Gamma^{4D}_{31} = [0,2,4,7;4,6,5,8;7,2,6,9] && \Gamma^{4D}_{33} = [0,2,4,7;1,6,5,8;1,5,6,9]\\
&\Gamma^{4D}_{34} = [0,2,4,7;1,9,5,8;1,5,6,9] && \Gamma^{4D}_{35} = [0,2,4,7;1,9,5,8;4,5,6,9] && \Gamma^{4D}_{36} = [0,2,4,7;1,9,5,8;7,5,6,9]\\
&\Gamma^{4D}_{40} = [0,5,4,7;1,9,5,8;1,2,6,9] && \Gamma^{4D}_{41} = [0,5,4,7;1,9,5,8;4,2,6,9] && \Gamma^{4D}_{42} = [0,5,4,7;4,3,5,8;4,2,6,9]\\
&\Gamma^{4D}_{43} = [0,5,4,7;4,9,5,8;4,2,6,9] && \Gamma^{4D}_{44} = [0,5,4,7;7,9,5,8;4,2,6,9] && \Gamma^{4D}_{45} = [0,5,4,7;7,3,5,8;7,2,6,9]\\
&\Gamma^{4D}_{45} = [0,5,4,7;7,3,5,8;7,2,6,9] && \Gamma^{4D}_{46} = [0,5,4,7;7,6,5,8;7,2,6,9] && \Gamma^{4D}_{47} = [0,2,4,7;4,3,5,8;1,5,6,9]\\
&\Gamma^{4D}_{48} = [0,5,4,7;1,3,5,8;1,5,6,9] && \Gamma^{4D}_{49} = [0,5,4,7;1,6,5,8;1,5,6,9] && \Gamma^{4D}_{50} = [0,5,4,7;1,9,5,8;1,5,6,9]\\
&\Gamma^{4D}_{54} = [0,5,4,7;4,9,5,8;7,5,6,9] && \Gamma^{4D}_{61} = [0,5,4,7;1,3,5,8;1,8,6,9] && \Gamma^{4D}_{62} = [0,5,4,7;1,6,5,8;1,8,6,9]\\
&\Gamma^{4D}_{63} = [0,5,4,7;7,3,5,8;7,8,6,9] && \Gamma^{4D}_{64} = [0,5,4,7;7,6,5,8;7,8,6,9].
\end{aligned}$$

Lemma 16. There are 21 nonisomorphic Nambu micro-graphs on two Levi-Civita vertices 1, 2, with two corresponding a^1 Casimir vertices 3, 4 and two corresponding a^2 Casimir vertices 5, 6 and no sink. The encodings for these 21 Hamiltonians are given below.

$$\begin{aligned}
&\Gamma^{4D}_{H_1} = [1,2,3,5;1,2,4,6] && \Gamma^{4D}_{H_2} = [1,2,3,5;2,3,4,6] && \Gamma^{4D}_{H_3} = [1,2,3,5;2,4,5,6] && \Gamma^{4D}_{H_4} = [1,3,4,5;2,3,4,6]\\
&\Gamma^{4D}_{H_5} = [1,3,4,5;2,4,5,6] && \Gamma^{4D}_{H_6} = [1,3,5,6;2,4,5,6] && \Gamma^{4D}_{H_7} = [1,2,3,5;1,3,4,6] && \Gamma^{4D}_{H_8} = [1,2,3,5;1,4,5,6]\\
&\Gamma^{4D}_{H_9} = [1,2,3,5;3,4,5,6] && \Gamma^{4D}_{H_{10}} = [1,3,4,5;1,3,4,6] && \Gamma^{4D}_{H_{11}} = [1,3,5,6;1,3,4,6] && \Gamma^{4D}_{H_{12}} = [1,3,4,5;1,4,5,6]\\
&\Gamma^{4D}_{H_{13}} = [1,3,5,6;1,4,5,6] && \Gamma^{4D}_{H_{14}} = [1,3,4,5;3,4,5,6] && \Gamma^{4D}_{H_{15}} = [1,3,5,6;3,4,5,6] && \Gamma^{4D}_{H_{16}} = [2,3,4,5;1,3,4,6]\\
&\Gamma^{4D}_{H_{17}} = [2,3,5,6;1,4,5,6] && \Gamma^{4D}_{H_{18}} = [2,3,4,5;1,4,5,6] && \Gamma^{4D}_{H_{19}} = [2,3,4,5;3,4,5,6] && \Gamma^{4D}_{H_{20}} = [2,3,5,6;3,4,5,6]\\
&\Gamma^{4D}_{H_{21}} = [3,4,5,6;3,4,5,6]
\end{aligned}$$

[7]Here, 0 is the sink, we have Levi-Civita vertices 1, 2, 3 whereas 4, 5, 6 (respectively 7, 8, 9) are the corresponding a^1 Casimir vertices (respectively a^2 Casimir vertices).

We detect the following relations (see the attached code).

$$H_2^{4D} = -H_7^{4D} \qquad H_4^{4D} = H_{16}^{4D} \qquad H_6^{4D} = H_{17}^{4D} \qquad H_{11}^{4D} = H_{12}^{4D} \qquad H_{15}^{4D} = H_{20}^{4D} \tag{5}$$
$$H_3^{4D} = -H_8^{4D} \qquad H_5^{4D} = H_{18}^{4D} \qquad H_9^{4D} = 0 \qquad H_{14}^{4D} = H_{19}^{4D}$$

Note that these Hamiltonians are not yet symmetric under a^1 and a^2. After symmetrizing, we find a maximal linearly independent set consisting of only $(H_1^{4D})^+$, $(H_2^{4D})^+$, $(H_4^{4D})^+$, $(H_5^{4D})^+$, $(H_{10}^{4D})^+$, $(H_{11}^{4D})^+$, $(H_{14}^{4D})^+$ and $(H_{21}^{4D})^+$, see the attached code.

Remark 4. Again, $\Gamma_{H_1}^{4D}$ is precisely the four-dimensional embedding of the two-dimensional Hamiltonian H_1^{2D}. As we are only working over $(\widehat{\Gamma}_{11}^{2D})_{4D} \cup (\widehat{\Gamma}_{12}^{2D})_{4D}$, none of the vector fields $\vec{Y}_i^{4D}$ will dependent on this Hamiltonian.

Theorem 17. *On $\mathbb{R}^4$, let P be a (degenerate) Nambu-determinant Poisson bivector. Consider the complex $(*)$ restricted to symmetric Hamiltonians on 2 copies of P, skew-symmetric vector fields on 3 copies of P, symmetric bivectors on 4 copies of P, etc. We establish that the corresponding homogeneous part of the first Poisson–Lichnerowicz cohomology $H^1_{\mathrm{gra}}(\mathbb{R}^4)$ is trivial.*

Proof (see the attached code). We write each of the seven skew-symmetric vector fields $\vec{Y}_i^{4D}$ of Proposition 15 in terms of the skew-symmetric Hamiltonian vector fields $\mathrm{d}_P((H_2^{4D})^+)$, $\mathrm{d}_P((H_4^{4D})^+)$, $\mathrm{d}_P((H_5^{4D})^+)$, $\mathrm{d}_P((H_{10}^{4D})^+)$, $\mathrm{d}_P((H_{11}^{4D})^+)$, $\mathrm{d}_P((H_{14}^{4D})^+)$ and $\mathrm{d}_P((H_{21}^{4D})^+)$. We compute

$$\vec{Y}^1_{4D} = 1 \cdot \mathrm{d}_P((H_2^{4D})^+) + \tfrac{1}{4} \cdot \mathrm{d}_P((H_4^{4D})^+) \qquad \vec{Y}^5_{4D} = 1 \cdot \mathrm{d}_P((H_{10}^{4D})^+)$$
$$\vec{Y}^2_{4D} = -\tfrac{1}{4} \cdot \mathrm{d}_P((H_4^{4D})^+) \qquad \vec{Y}^6_{4D} = -1 \cdot \mathrm{d}_P((H_{11}^{4D})^+)$$
$$\vec{Y}^3_{4D} = \tfrac{1}{2} \cdot \mathrm{d}_P((H_5^{4D})^+) - \tfrac{1}{2} \cdot \mathrm{d}_P((H_{14}^{4D})^+) - \tfrac{1}{16} \cdot \mathrm{d}_P((H_{21}^{4D})^+) \qquad \vec{Y}^7_{4D} = \tfrac{1}{8} \cdot \mathrm{d}_P((H_{21}^{4D})^+)$$
$$\vec{Y}^4_{4D} = \tfrac{1}{2} \cdot \mathrm{d}_P((H_{14}^{4D})^+) + \tfrac{1}{16} \cdot \mathrm{d}_P((H_{21}^{4D})^+)$$

that is, the degrees of freedom are provided by the Hamiltonian shifts. It follows immediately that the corresponding homogeneous part of $H^1_{\mathrm{gra}}(\mathbb{R}^4) = \ker \mathrm{d}_P / \mathrm{im}\ \mathrm{d}_P$ is trivial. □

Corollary 18. The trivializing skew-symmetric vector field $(\vec{X}_{4D}^{\gamma_3})^-$ over skew-symmetrized vector fields obtained from the descendants $(\widehat{\Gamma}_{11}^{2D})_{4D}$ and $(\widehat{\Gamma}_{12}^{2D})_{4D}$ of Proposition 14 is unique modulo skew-symmetric Hamiltonian vector fields.

6 Non-uniqueness of graphs

Definition 4. Two topologically nonisomorphic graphs $\Gamma_1 \ncong \Gamma_2$ are called *synonyms* if $\phi(\Gamma_1) = c \cdot \phi(\Gamma_2)$ with $c \in \mathbb{R} \setminus \{0\}$, that is, the two graphs provide the same multivector up to a nonzero constant.

We have already seen many synonyms, for example within the three and four dimensional Hamiltonians (equations (4), (5)), and the two-dimensional vector fields[8] (equation (3)). We do not yet understand these synonyms. Two graphs that evaluate to the same multivector in one dimension might not exhibit the same properties in a higher dimension. One of the most clear examples of this is the behaviour of pairs of graphs that give the two-dimensional solution $\vec{X}_{2D}^{\gamma_3}$. Using the relations of equation (3), we can create 28 pairs in two dimensions such that each pair solves the trivialization problem. But, when we move to dimension three we detect that there exists a solution over the descendants of only 5 of these 28 two-dimensional pairs (see the attached code), see table 1.

Table 1: Does a trivializing vector field exist over the three-dimensional descendants of the trivializing pair $(\Gamma_i^{2D})_{3D}, (\Gamma_j^{2D})_{3D}$ where $i \in \{2, 4, 9, 11\}$ and $j \in \{1, 5, 6, 7, 8, 12, 13\}$?

	$(\widehat{\Gamma}_1^{2D})_{3D}$	$(\widehat{\Gamma}_5^{2D})_{3D}$	$(\widehat{\Gamma}_6^{2D})_{3D}$	$(\widehat{\Gamma}_7^{2D})_{3D}$	$(\widehat{\Gamma}_8^{2D})_{3D}$	$(\widehat{\Gamma}_{12}^{2D})_{3D}$	$(\widehat{\Gamma}_{13}^{2D})_{3D}$
$(\widehat{\Gamma}_2^{2D})_{3D}$	No	No	No	No	**Yes**	**Yes**	No
$(\widehat{\Gamma}_4^{2D})_{3D}$	No	No	No	No	No	No	No
$(\widehat{\Gamma}_9^{2D})_{3D}$	No	No	No	No	No	No	No
$(\widehat{\Gamma}_{11}^{2D})_{3D}$	No	No	No	**Yes**	**Yes**	**Yes**	No

[8] We also have synonyms of vector fields in dimensions 3 and 4, see the attached code, but no explicit examples can be given in this text due to volume constraints.

Similarly, we can take these 5 'yes'-pairs over which we find a solution in dimension 3, and consider their four-dimensional descendants. In this case, we can only find a solution over the descendants of two of these pairs, see table 2 (see the attached code).

Table 2: Does a trivializing vector field exist over the four-dimensional descendants of the trivializing pair $(\Gamma_i^{2D})_{4D}, (\Gamma_j^{2D})_{4D}$ where $i \in \{2, 4, 9, 11\}$ and $j \in \{1, 5, 6, 7, 8, 12, 13\}$?

	$(\widehat{\Gamma}_1^{2D})_{4D}$	$(\widehat{\Gamma}_5^{2D})_{4D}$	$(\widehat{\Gamma}_6^{2D})_{4D}$	$(\widehat{\Gamma}_7^{2D})_{4D}$	$(\widehat{\Gamma}_8^{2D})_{4D}$	$(\widehat{\Gamma}_{12}^{2D})_{4D}$	$(\widehat{\Gamma}_{13}^{2D})_{4D}$
$(\widehat{\Gamma}_2^{2D})_{4D}$	No	No	No	No	**No**	**Yes**	No
$(\widehat{\Gamma}_4^{2D})_{4D}$	No	No	No	No	No	No	No
$(\widehat{\Gamma}_9^{2D})_{4D}$	No	No	No	No	No	No	No
$(\widehat{\Gamma}_{11}^{2D})_{4D}$	No	No	No	**No**	**No**	**Yes**	No

7 Conclusion

The appearance of synonyms in the trivialization problem makes it difficult to detect patterns in the graphs that show up in the solution (provided both a solution and a pattern exist at all!) and adds an extra barrier in guessing what graphs may appear in the trivializing vector field for a particular dimension.

Remark 5. We cannot compute in dimension $d \geqslant 5$ because of the time complexity of the system that needs to be solved. Moreover, there is no guarantee that we can find *any* solution over the five-dimensional descendants of the pairs Γ_2^{2D}, Γ_{12}^{2D} and Γ_{11}^{2D}, Γ_{12}^{2D}, see table 2. Indeed, there is no reason for us to consider *just* the pairs of graphs. We might need to consider linear combinations of more than two graphs as they can still solve the trivialization problem.

Acknowledgements

The authors are grateful to the organizers of the International conference on Integrable Systems and Quantum Symmetries (ISQS 28) for an opportunity to present and discuss new results. The authors thank the Center for Information Technology of the University of Groningen for their support and for providing access to the Hábrók high performance computing cluster. The authors thank the University of Groningen for partial financial support. Lastly, the authors thank R. Buring for the gcaops software and his instructions on how to work with it.

References

[1] Lichnerowicz A 1977 Les variétés de Poisson et leurs algebres de Lie associées *Journal of différential geometry* **12** 253–300

[2] Kontsevich M 1997 Formality conjecture *Deformation theory and symplectic geometry* **128** 139–156

[3] Kiselev A V, Jagoe Brown M S and Schipper F 2024 Kontsevich graphs act on Nambu–Poisson brackets, I. New identities for Jacobian determinants *arXiv preprint arXiv:2409 [Math.QA] (in preparation)*

[4] Jagoe Brown M S, Schipper F and Kiselev A V 2024 Kontsevich graphs act on Nambu–Poisson brackets, II. The tetrahedral flow is a coboundary in 4D *arXiv preprint arXiv:2409.12555 [Math.QA]*

[5] Buring R, Lipper D and Kiselev A V 2022 The hidden symmetry of Kontsevich's graph flows on the spaces of Nambu-determinant Poisson brackets *Open Communications in Nonlinear Mathematical Physics* **2** 186–216

[6] Buring R and Kiselev A V 2023 The tower of Kontsevich deformations for Nambu–Poisson structures on $\mathbb{R}^d$: Dimension-specific micro-graph calculus *SciPost Physics Proceedings* **14** 020

[7] Bouisaghouane A 2017 The Kontsevich tetrahedral flow in 2D: a toy model *arXiv preprint arXiv:1702.06044*

Appendix B. Code

Kontsevich graphs act on Nambu–Poisson brackets, III. Uniqueness aspects (Section 3)

November 22, 2024

[1]:
```
# 17-09-2024 The code below is a combination of already existing code by R. Buring, and adjustments made by F. Schipper
# (particularly, the biggest adjustments are checking that the basis element(s) of the cocycle space are Hamiltonian,
# commentary and printing of output). It is supplementary material accompanying proceedings written for the ISQS28 (Integrable
#Systems and Quantum Symmetries) conference (see also: arXiv:2409.15932).

# We import the following (see https://github.com/rburing/gcaops) to be able to run the code.
from gcaops.graph.formality_graph import FormalityGraph
from gcaops.algebra.differential_polynomial_ring import DifferentialPolynomialRing
from gcaops.algebra.superfunction_algebra import SuperfunctionAlgebra
from gcaops.graph.undirected_graph_complex import UndirectedGraphComplex
from gcaops.graph.directed_graph_complex import DirectedGraphComplex

# These are the encodings of the 14 Kontsevich graphs on 3 vertices and 1 sink in 2D.
two_d_graphs="(0,1;2,3;1,3)+(0,1;1,2;1,3)+(0,3;2,3;2,3)+(0,3;2,3;1,3)+(0,2;2,3;1,3)+(0,3;1,2;1,3)+(0,3;2,3;1,2)+(0,3;1,2;1,2)+(0,2;2,3;1,2)+(0,2;1,2;1,2)+(0,1;1,3;1,2)+(0,3;1,3;1,2)+(0,1;1,3;2,3)+(0,1;1,3;1,3)"

# To move from the encodings of the graphs to actual graphs, we use the next function. The function splits the
# encoding by vertex via ;, and then the target vertices by ,. A graph is returned on 3 vertices, 1 sink, and with edges
# (origin vertex, target vertex). As an example, the first encoding (0,1;2,3;1,3) correspond to a graph with 1 sink (vertex 0),
# and 3 regular vertices (vertices 1,2,3), with 6 edges (1,0), (1,1), (2,2), (2,3), (3,1), (3,3).
def encoding_to_graph(encoding):
    targets = [tuple(int(v) for v in t.split(',')) for t in encoding[1:-1].split(";")]
    edges = sum([[(k+1,v) for v in t] for (k,t) in enumerate(targets)], [])
```

```
    return FormalityGraph(1, 3, edges)

# Split the encodings and compute the corresponding graphs
encodings = two_d_graphs.split("+")
graphs = [encoding_to_graph(e) for e in encodings]
print('We have', len(graphs), 'graphs.\n')

# Create the differential polynomial ring. We are working in 2D, so we only have␣
 ↪even coordinates x,y and the corresponding
# odd coordinates xi[0] and xi[1]. rho is exactly the rho in a 2D Poisson␣
 ↪bracket (P= rho dx dy). Finally,
# max_differential_orders tells the programme how many times rho can be␣
 ↪differentiated. The maximum is stipulated by the graphs,
# as we cannot have double edges. Thus, the maximum in degree of each vertex is␣
 ↪3. As we will be looking at [[P,X]], we add
# an extra +1 to the differential orders since taking the Schouten bracket with␣
 ↪P introduces an extra derivative.
D2=DifferentialPolynomialRing(QQ,('rho', ), ('x','y'),␣
 ↪max_differential_orders=[3+1])
rho, =D2.fibre_variables()
x,y= D2.base_variables()
even_coords=[x,y]

S2.<xi0,xi1>=SuperfunctionAlgebra(D2, D2.base_variables())
xi=S2.gens()
odd_coords=xi

# We now compute the vector fields corresponding to the graphs. E is the Euler␣
 ↪vector field in the sink (vertex 0), and
# epsilon is the Levi-Civita tensor. Note that we have a Levi-Civita tensor at␣
 ↪each of the vertices 1, 2, 3. We first compute
# the sign of each term appearing in the formulas, and then compute the␣
 ↪differential polynomial.
X_vector_fields=[]
E=x*xi[0]+y*xi[1]
epsilon = xi[0]*xi[1]
import itertools
for g in graphs:
    term = S2.zero()
    for index_choice in itertools.product(itertools.permutations(range(2)),␣
 ↪repeat=3):
        sign = epsilon[index_choice[0]] * epsilon[index_choice[1]]*␣
 ↪epsilon[index_choice[2]]
        vertex_content = [E, S2(rho), S2(rho), S2(rho)]
        for ((source, target), index) in zip(g.edges(), sum(map(list,␣
 ↪index_choice), [])):
```

```
            vertex_content[target] = vertex_content[target].
 ↪diff(even_coords[index])
        term += sign * prod(vertex_content)
    X_vector_fields.append(term)

# We check how many (if any) graphs evaluate to 0.
zeros=X_vector_fields.count(0)
print('There are', zeros, 'graphs that evaluate to 0 under the morhpism from␣
 ↪graphs to multivectors.\n')

# In case the are graphs that evaluate to 0, the line below shows which graphs␣
 ↪do so (note that the counting starts from 1!).
#[k+1 for (k,X) in enumerate (X_vector_fields) if X==0]

# In 2D on only 3 vertices and 1 sink, the multivectors are pretty small and can␣
 ↪be printed easily.
print('Here are the vector fields the graphs are evaluated into:')
for i in range(len(X_vector_fields)):
    print('graph', i+1,':', X_vector_fields[i])
print()

# The next part is to find out linear relations of the vector fields we just␣
 ↪computed. We look at the monomials that appear
# in each xi[0] and xi[1] parts of the vector field, and store them in␣
 ↪X_monomial_basis.
X_monomial_basis = [set([]) for i in range(2)]
for i in range(2):
    for X in X_vector_fields:
        X_monomial_basis[i]|=set(X[i].monomials())
X_monomial_basis=[list(b) for b in X_monomial_basis]
X_monomial_index= [{m:k for k,m in enumerate(b)} for b in X_monomial_basis]
X_monomial_count= sum(len(b) for b in X_monomial_basis); X_monomial_count

# Next, we use this monomial basis to create a matrix that identifies each␣
 ↪vector field by the monomials that appear in it.
X_evaluation_matrix= matrix(QQ, X_monomial_count, len(X_vector_fields),␣
 ↪sparse=True)
for i in range(len(X_vector_fields)):
    v=vector(QQ, X_monomial_count, sparse=True)
    index_shift=0
    for j in range(2):
        f=X_vector_fields[i][j]
        for coeff, monomial in zip(f.coefficients(), f.monomials()):
            monomial_index=X_monomial_index[j][monomial]
            v[index_shift+monomial_index]=coeff
        index_shift+=len(X_monomial_basis[j])
```

```
    X_evaluation_matrix.set_column(i,v)

# We can now detect linear relation by computing the nullity of this matrix
nullity = X_evaluation_matrix.right_nullity()
print('The vector fields have', nullity, 'linear relations among themselves.')
print('Claim 2: These relations are expressed by \n', X_evaluation_matrix.
 ↪right_kernel(), '\n')

# We now compute the tetrahedral flow. First, we create the Poisson bivector P␣
 ↪and check that it satisfies [[P,P]]=0
P= rho*epsilon
if P.bracket(P)!=0:
    print('P is not a Poisson bivector. \n')

# Introduce the graph complex, and find the tetrahedron as a graph on 4 vertices␣
 ↪and 6 edges in the cohomology. This
# tetrahedron is unoriented, so we orient it. The next step is to make sure that␣
 ↪the graph is built of wedges. This means
# that there cannot be more than 2 outgoing edges at each vertex. This gives 2␣
 ↪graphs; one where 3 vertices have 2 outgoing
# edges, and one where 2 vertices have 2 outgoing edges and 2 vertices have 1␣
 ↪outgoing edge.
# tetrahedron_oriented_filtered.show() gives a drawing of these graphs.
# Finally, the bivector corresponding to the graph is computed.

GC=UndirectedGraphComplex(QQ, implementation='vector', sparse=True)
tetrahedron= GC.cohomology_basis(4,6)[0]
dGC=DirectedGraphComplex(QQ, implementation='vector')
tetrahedron_oriented= dGC(tetrahedron)
tetrahedron_oriented_filtered= tetrahedron_oriented.filter(max_out_degree=2)
# tetrahedron_oriented_filtered.show()
tetrahedron_operation= S2.graph_operation(tetrahedron_oriented_filtered)
Q_tetra= tetrahedron_operation(P, P, P, P) /8
print('The tetrahedral flow in 2D is', Q_tetra, '\n')

# Now that we have the tetrahedral flow, we see if we can create a vectorfield X␣
 ↪from our previously computed
# graphs-to-vector fields such that [[P,X]]= Q_tetra. We first look which␣
 ↪bivectors X_bivectors the vector fields become
# after taking the Schouten bracket with P.
X_bivectors=[]
for X in X_vector_fields:
    X_bivectors.append(P.bracket(X))

zero_bivectors = X_bivectors.count(0)
```

```
print('There are', zero_bivectors, 'vector fields in X_vector_fields that␣
 ↪evaluate to 0 bivectors after taking the Schouten bracket with P.')
print('These vector fields that evaluate to 0 are obtained from the graphs:')
for (k,X) in enumerate(X_bivectors):
    if X==0:
        print('graph', k+1,)
print()

# Now, we extract the monomials appearing in these bivectors (as well as in␣
 ↪Q_tetra).
Q_monomial_basis={}
from itertools import combinations
for i,j in combinations(range(2),2):
    Q_monomial_basis[i,j]=set(Q_tetra[i,j].monomials())
    for P_X in X_bivectors:
        Q_monomial_basis[i,j]|= set(P_X[i,j].monomials())

Q_monomial_basis={idx: list(b) for idx, b in Q_monomial_basis.items()}
Q_monomial_index= {idx:{m:k for k,m in enumerate(b)} for idx, b in␣
 ↪Q_monomial_basis.items()}
Q_monomial_count=sum(len(b) for b in Q_monomial_basis.values());

# We create the vector representation of Q_tetra in terms of the monomials.
Q_tetra_vector= vector(QQ, Q_monomial_count, sparse=True)
index_shift=0
for i,j in Q_monomial_basis:
    for coeff, monomial in Q_tetra[i,j]:
        monomial_index= Q_monomial_index[i,j][monomial]
        Q_tetra_vector[monomial_index+index_shift]=coeff
    index_shift+=len(Q_monomial_basis[i,j])

# We create the matrix that represents the X_bivectors in terms of the monomials.
X_bivector_evaluation_matrix= matrix(QQ, Q_monomial_count, len(X_bivectors),␣
 ↪sparse=True)
for k in range(len(X_bivectors)):
    P_X=X_bivectors[k]
    v=vector(QQ,Q_monomial_count, sparse=True)
    index_shift=0
    for i,j in Q_monomial_basis:
        for coeff, monomial in P_X[i,j]:
            monomial_index=Q_monomial_index[i,j][monomial]
            v[monomial_index+index_shift]=coeff
        index_shift+=len(Q_monomial_basis[i,j])
    X_bivector_evaluation_matrix.set_column(k,v)

# We solve the linear system.
X_solution=X_bivector_evaluation_matrix.solve_right(Q_tetra_vector)
```

```
print('Proposition 3: The solution on vector fields is given by', X_solution,'␣
 ↪\n')
# So if we take 2*graph 1 +1*graph 2, evaluate this to a vector field X, then␣
 ↪[[P, X]]= Q_tetra.
# Note that because of the linear relations, this solution is not unique on the␣
 ↪level of graphs.

# Finally, we will check the homogeneous system. We look at the kernel of the␣
 ↪X_bivector_evaluation_matrix and filter out the
# nullity of the X_evaluation matrix
X_cocycle_space= X_bivector_evaluation_matrix.right_kernel().
 ↪quotient(X_evaluation_matrix.right_kernel())
X_cocycles=[X_cocycle_space.lift(v) for v in X_cocycle_space.basis()]; X_cocycles
if all(X_bivector_evaluation_matrix*X_cocycle==0 for X_cocycle in X_cocycles)!=␣
 ↪True:
    print('There is an error in the cocycle space.')
print('The space of solutions to the homogeneous system has dimension',␣
 ↪X_cocycle_space.dimension(), )
print ('This shift is given by (Proposition 4) ', X_cocycles, 'and the␣
 ↪corresponding vector field is', X_vector_fields[2], '\n')

# We check that this shift is induced by the Hamiltonian vector field
hamiltonian="(0,1;0,1)"

# We create a new encoding to graph definition, as this is a graph on 2 vertices␣
 ↪and no sink.
def hamiltonian_encoding_to_graph(encoding):
    targets = [tuple(int(v) for v in t.split(',')) for t in encoding[1:-1].
 ↪split(";")]
    edges = sum([[(k,v) for v in t] for (k,t) in enumerate(targets)], [])
    return FormalityGraph(0, 2, edges)

hamiltonian_graph= encoding_to_graph(hamiltonian)

# This computation is also slightly different; we do not have the Euler vector␣
 ↪field since we do not have a sink, and as
# we have only 2 copies of the Poisson bivector, the index_choice for the sign␣
 ↪is only on 2 repeats as well. Moreover, there
# are only 2 vertices to give vertex_content to.
epsilon= xi[0]*xi[1]
import itertools
hamiltonian_formula=S2.zero()
for index_choice in itertools.product(itertools.permutations(range(2)),repeat=2):
    vertex_content = [S2(rho), S2(rho)]
    sign = epsilon[index_choice[0]] * epsilon[index_choice[1]]
```

```
    for ((source, target), index) in zip(hamiltonian_graph.edges(),␣
 ↪sum(map(list,index_choice), [])):
        vertex_content[target] = diff(vertex_content[target],even_coords[index])
    hamiltonian_formula += sign * prod(vertex_content)

hamiltonian_vector_field=P.bracket(hamiltonian_formula)
print('The Hamiltonian vector field is given by', hamiltonian_vector_field, '\n')

if hamiltonian_vector_field== 2*X_vector_fields[2]:
    print('Theorem 6: The shift in the cocycle space is given by 1/2 the␣
 ↪Hamiltonian vector field.')
```

```
We have 14 graphs.

There are 0 graphs that evaluate to 0 under the morhpism from graphs to
multivectors.

Here are the vector fields the graphs are evaluated into:
graph 1 : (-rho_y*rho_xy^2 + rho_y*rho_xx*rho_yy)*xi0 + (rho_x*rho_xy^2 -
rho_x*rho_xx*rho_yy)*xi1
graph 2 : (rho_y^2*rho_xxy - 2*rho_x*rho_y*rho_xyy + rho_x^2*rho_yyy)*xi0 +
(-rho_y^2*rho_xxx + 2*rho_x*rho_y*rho_xxy - rho_x^2*rho_xyy)*xi1
graph 3 : (rho*rho_yy*rho_xxy - 2*rho*rho_xy*rho_xyy + rho*rho_xx*rho_yyy)*xi0 +
(-rho*rho_yy*rho_xxx + 2*rho*rho_xy*rho_xxy - rho*rho_xx*rho_xyy)*xi1
graph 4 : (rho_y^2*rho_xxy - 2*rho_x*rho_y*rho_xyy + rho_x^2*rho_yyy)*xi0 +
(-rho_y^2*rho_xxx + 2*rho_x*rho_y*rho_xxy - rho_x^2*rho_xyy)*xi1
graph 5 : (-rho_y*rho_xy^2 + rho_y*rho_xx*rho_yy)*xi0 + (rho_x*rho_xy^2 -
rho_x*rho_xx*rho_yy)*xi1
graph 6 : (-rho_y*rho_xy^2 + rho_y*rho_xx*rho_yy)*xi0 + (rho_x*rho_xy^2 -
rho_x*rho_xx*rho_yy)*xi1
graph 7 : (rho_y*rho_xy^2 - rho_y*rho_xx*rho_yy)*xi0 + (-rho_x*rho_xy^2 +
rho_x*rho_xx*rho_yy)*xi1
graph 8 : (-2*rho_y*rho_xy^2 + 2*rho_y*rho_xx*rho_yy)*xi0 + (2*rho_x*rho_xy^2 -
2*rho_x*rho_xx*rho_yy)*xi1
graph 9 : (-rho_y^2*rho_xxy + 2*rho_x*rho_y*rho_xyy - rho_x^2*rho_yyy)*xi0 +
(rho_y^2*rho_xxx - 2*rho_x*rho_y*rho_xxy + rho_x^2*rho_xyy)*xi1
graph 10 : (rho*rho_yy*rho_xxy - 2*rho*rho_xy*rho_xyy + rho*rho_xx*rho_yyy)*xi0
+ (-rho*rho_yy*rho_xxx + 2*rho*rho_xy*rho_xxy - rho*rho_xx*rho_xyy)*xi1
graph 11 : (rho_y^2*rho_xxy - 2*rho_x*rho_y*rho_xyy + rho_x^2*rho_yyy)*xi0 +
(-rho_y^2*rho_xxx + 2*rho_x*rho_y*rho_xxy - rho_x^2*rho_xyy)*xi1
graph 12 : (-rho_y*rho_xy^2 + rho_y*rho_xx*rho_yy)*xi0 + (rho_x*rho_xy^2 -
rho_x*rho_xx*rho_yy)*xi1
graph 13 : (-rho_y*rho_xy^2 + rho_y*rho_xx*rho_yy)*xi0 + (rho_x*rho_xy^2 -
rho_x*rho_xx*rho_yy)*xi1
graph 14 : (rho*rho_yy*rho_xxy - 2*rho*rho_xy*rho_xyy + rho*rho_xx*rho_yyy)*xi0
+ (-rho*rho_yy*rho_xxx + 2*rho*rho_xy*rho_xxy - rho*rho_xx*rho_xyy)*xi1

The vector fields have 11 linear relations among themselves.
```

```
Claim 2: These relations are expressed by
 Vector space of degree 14 and dimension 11 over Rational Field
Basis matrix:
[ 1  0  0  0  0  0  0  0  0  0  0  0 -1  0]
[ 0  1  0  0  0  0  0  0  0  0 -1  0  0  0]
[ 0  0  1  0  0  0  0  0  0  0  0  0  0 -1]
[ 0  0  0  1  0  0  0  0  0  0 -1  0  0  0]
[ 0  0  0  0  1  0  0  0  0  0  0  0 -1  0]
[ 0  0  0  0  0  1  0  0  0  0  0  0 -1  0]
[ 0  0  0  0  0  0  1  0  0  0  0  0  1  0]
[ 0  0  0  0  0  0  0  1  0  0  0  0 -2  0]
[ 0  0  0  0  0  0  0  0  1  0  1  0  0  0]
[ 0  0  0  0  0  0  0  0  0  1  0  0  0 -1]
[ 0  0  0  0  0  0  0  0  0  0  0  1 -1  0]

The tetrahedral flow in 2D is (rho_y^3*rho_xxx - 3*rho_x*rho_y^2*rho_xxy +
3*rho_x^2*rho_y*rho_xyy - rho_x^3*rho_yyy)*xi0*xi1

There are 3 vector fields in X_vector_fields that evaluate to 0 bivectors after
taking the Schouten bracket with P.
These vector fields that evaluate to 0 are obtained from the graphs:
graph 3
graph 10
graph 14

Proposition 3: The solution on vector fields is given by (2, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0)

The space of solutions to the homogeneous system has dimension 1
This shift is given by (Proposition 4)  [(0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0)] and the corresponding vector field is (rho*rho_yy*rho_xxy -
2*rho*rho_xy*rho_xyy + rho*rho_xx*rho_yyy)*xi0 + (-rho*rho_yy*rho_xxx +
2*rho*rho_xy*rho_xxy - rho*rho_xx*rho_xyy)*xi1

The Hamiltonian vector field is given by (2*rho*rho_yy*rho_xxy -
4*rho*rho_xy*rho_xyy + 2*rho*rho_xx*rho_yyy)*xi0 + (-2*rho*rho_yy*rho_xxx +
4*rho*rho_xy*rho_xxy - 2*rho*rho_xx*rho_xyy)*xi1

Theorem 6: The shift in the cocycle space is given by 1/2 the Hamiltonian vector
field.
```

Kontsevich graphs act on Nambu–Poison brackets, III. Uniqueness aspects (section 4)

November 22, 2024

[1]:
```
# 17-09-2024 The code below is a combination of already existing code by R. Buring, and adjustments made by F. Schipper
# (particularly, the biggest adjustments are checking that the basis element(s) of the cocycle space are Hamiltonian,
# commentary and printing of output). It is supplementary material accompanying proceedings written for the ISQS28 (Integrable
#Systems and Quantum Symmetries) conference (see also: arxiv link??).

# We import the following (see https://github.com/rburing/gcaops) to be able to run the code.
from gcaops.graph.formality_graph import FormalityGraph
from gcaops.algebra.differential_polynomial_ring import DifferentialPolynomialRing
from gcaops.algebra.superfunction_algebra import SuperfunctionAlgebra
from gcaops.graph.undirected_graph_complex import UndirectedGraphComplex
from gcaops.graph.directed_graph_complex import DirectedGraphComplex

# We start by generating the encodings for the descendants of Gamma_{11}^2D and Gamma_{12}^2D
import itertools

X_graph_encodings = []
for (i1,j1,j2,k) in itertools.product([2,5], [1,4], [1,4], [3,6]):
    X_graph_encodings.append((0,1,4,j1,k,5,j2,i1,6))

for (i1,i2,j1,j2,k) in itertools.product([2,5], [2,5], [1,4], [1,4], [3,6]):
    X_graph_encodings.append((0,i1,4,j1,k,5,j2,i2,6))

# To move from the encodings of the graphs to the actual graphs, we use the next function. The definition splits the
# encoding by vertex via ;, and then the targets by ,. A graph is returned on 3 Levi-Civita vertices, 3 corresponding Casimir
# vertices, 1 sink, and with edges (origin vertex, target vertex).

def encoding_to_graph(encoding):
```

```
    targets = [encoding[0:3], encoding[3:6], encoding[6:9]]
    edges = sum([[(k+1,v) for v in t] for (k,t) in enumerate(targets)], [])
    return FormalityGraph(1, 6, edges)

X_graphs = [encoding_to_graph(e) for e in X_graph_encodings]
print('We have', len(X_graphs), 'graphs.\n')

# Below, we check the how many graphs that were created via the encodings are␣
 ↪(non)isomorphic. It does so in the following way:
# for each formality graph in X_graphs, it computes the edges of the canonical␣
 ↪form (canonical in the sense that isomorphic
# graphs return the same canonical form) of said formality graph. If a graph␣
 ↪with these edges already appears in X_graphs_iso,
# the new (isomorphic) graph is not added to the list. If the graph does not yet␣
 ↪appear, it is added to the list. The list is
# indexed by h, NOT by an index from 0,...,40. For this, unmute the line␣
 ↪X_graphs_iso=list(X_graphs_iso.values())
X_graphs_iso={}
for g in X_graphs:
    h=tuple(g.canonical_form().edges())
    if not h in X_graphs_iso:
        X_graphs_iso[h]=g
X_graphs_iso=list(X_graphs_iso.values())
print ('Lemma 8: There are', len(X_graphs_iso), 'nonisomorphic graphs.\n')

# Create the differential polynomial ring. We are working in 3D, so we only have␣
 ↪even coordinates x,y,z and the corresponding
# odd coordinates xi[0], xi[1] and xi[2]. rho is exactly the rho in a 3D␣
 ↪Nambu-determinant Poisson bracket,
# (P(f,g)= rho d(f,g,a)/d(x,y,z)), and a is the Casimir.  Finally,␣
 ↪max_differential_orders  tells the programme how many times
# rho and a can be  differentiated. The maximum is stipulated by the graphs, as␣
 ↪we cannot have double edges. Thus, the
# maximum in degree of each rho vertex is 3. As we will be looking at [[P,X]],␣
 ↪we add an extra +1 to the differential orders
# since taking the Schouten bracket with P introduces an extra derivative.
D3 = DifferentialPolynomialRing(QQ, ('rho','a'), ('x','y','z'),␣
 ↪max_differential_orders=[3+1,3+1])
rho, a = D3.fibre_variables()
x,y,z = D3.base_variables()
even_coords = [x,y,z]

S3.<xi0,xi1,xi2> = SuperfunctionAlgebra(D3, D3.base_variables())
xi = S3.gens()
odd_coords = xi
```

```
# We now compute the vector fields corresponding to the graphs. E is the Euler␣
 ↪vector field in the sink (vertex 0), and
# epsilon is the Levi-Civita tensor. Note that we have a Levi-Civita tensor at␣
 ↪each of the vertices 1, 2, 3. We first compute
# the sign of each term appearing in the formulas, and then compute the␣
 ↪differential polynomial.
X_vector_fields=[]
E = x*xi[0] + y*xi[1] + z*xi[2]
epsilon = xi[0]*xi[1]*xi[2]
for g in X_graphs:
    term = S3.zero()
    for index_choice in itertools.product(itertools.permutations(range(3)),␣
 ↪repeat=3):
        sign = epsilon[index_choice[0]] * epsilon[index_choice[1]] *␣
 ↪epsilon[index_choice[2]]
        vertex_content = [E, S3(rho), S3(rho), S3(rho), S3(a), S3(a), S3(a)]
        for ((source, target), index) in zip(g.edges(), sum(map(list,␣
 ↪index_choice), [])):
            vertex_content[target] = diff(vertex_content[target],␣
 ↪even_coords[index])
        term += sign * prod(vertex_content)
    X_vector_fields.append(term)

# We check how many (if any) graphs evaluate to 0.
zeros=X_vector_fields.count(0)
print('There are', zeros, 'graphs that evaluate to 0 under the morhpism from␣
 ↪graphs to multivectors.')

# In case the are graphs that evaluate to 0, the line below shows which graphs␣
 ↪do so (note that the counting starts from 1!).
print('These graphs are:')
for (k,X) in enumerate(X_vector_fields):
    if X==0:
        print('graph', k+1,)
print()

# The next part is to find out linear relations of the vector fields we just␣
 ↪computed. We look at the monomials that appear
# in each xi[0], xi[1] and xi[2] parts of the vector field, and store them in␣
 ↪X_monomial_basis.
X_monomial_basis = [set([]) for i in range(3)]
for i in range(3):
    for X in X_vector_fields:
        X_monomial_basis[i] |= set(X[i].monomials())
X_monomial_basis = [list(b) for b in X_monomial_basis]
X_monomial_count = sum(len(b) for b in X_monomial_basis)
```

```
# Next, we use this monomial basis to create a matrix that identifies each␣
 ↪vector field by the monomials that appear in it.
X_evaluation_matrix = matrix(QQ, X_monomial_count, len(X_vector_fields))
for i in range(len(X_vector_fields)):
    v = vector(QQ, X_monomial_count)
    index_shift = 0
    for j in range(3):
        f = X_vector_fields[i][j]
        for coeff, monomial in zip(f.coefficients(), f.monomials()):
            monomial_index = X_monomial_basis[j].index(monomial)
            v[index_shift + monomial_index] = coeff
        index_shift += len(X_monomial_basis[j])
    X_evaluation_matrix.set_column(i, v)

# We can now detect linear relation by computing the nullity of this matrix␣
 ↪(Note that we already know of 7 linear relations
# as 7 of the 48 graphs are counted double via an isomorphism.
nullity = X_evaluation_matrix.right_nullity()
print('The vector fields have', nullity, 'linear relations among themselves.\n')

# Unmute the next line if you want to explicitly see the 28 linear relations␣
 ↪among the vector fields
#print('These relations are expressed by \n', X_evaluation_matrix.right_kernel().
 ↪basis(), '\n')

# We now compute the tetrahedral flow. First, we create the Poisson bivector P␣
 ↪and check that it satisfies [[P,P]]=0.
P= (rho*epsilon).bracket(a)
if P.bracket(P)!=0:
    print('P is not a Poisson bivector. \n')

# Introduce the graph complex, and find the tetrahedron as a graph on 4 Poisson␣
 ↪vertices and 6 edges in the cohomology. This
# tetrahedron is unoriented, so we orient it. The next step is to make sure that␣
 ↪the graph is built of wedges. This means
# that there cannot be more than 2 outgoing edges at each vertex. This gives 2␣
 ↪graphs; one where 3 vertices have 2 outgoing
# edges, and one where 2 vertices have 2 outgoing edges and 2 vertices have 1␣
 ↪outgoing edge.
# tetrahedron_oriented_filtered.show() gives a drawing of these graphs.
# Finally, the bivector corresponding to the graph is computed.
GC=UndirectedGraphComplex(QQ, implementation='vector', sparse=True)
tetrahedron= GC.cohomology_basis(4,6)[0]
dGC=DirectedGraphComplex(QQ, implementation='vector')
tetrahedron_oriented= dGC(tetrahedron)
```

```
tetrahedron_oriented_filtered= tetrahedron_oriented.filter(max_out_degree=2)
#tetrahedron_oriented_filtered.show()
tetrahedron_operation= S3.graph_operation(tetrahedron_oriented_filtered)
Q_tetra= tetrahedron_operation(P, P, P, P)
# print('The tetrahedral flow in 3D is', Q_tetra, '\n')

# Now that we have the tetrahedral flow, we see if we can create a vectorfield X␣
 ↪from our previously computed
# graphs-to-vectorfields such that [[P,X]]= Q_tetra. We first look which␣
 ↪bivectors X_bivectors the vector fields become
# after taking the Schouten bracket with P.
X_bivectors=[]
for X in X_vector_fields:
    X_bivectors.append(P.bracket(X))

# There are no nonzero (new) vector fields evaluated from 1 graph such that␣
 ↪[[P,X_{graph}]]=0.
zero_bivectors = X_bivectors.count(0)
print('There are', zero_bivectors, 'vector fields in X_vector_fields that␣
 ↪evaluate to 0 bivectors after taking the Schouten bracket with P .')
print('These vector fields that evaluate to 0 are obtained from the graphs:')
for (k,X) in enumerate(X_bivectors):
    if X==0:
        print('graph', k+1,)
print()

# Now, we extract the monomials appearing in these bivectors (as well as in␣
 ↪Q_tetra).
Q_monomial_basis={}
from itertools import combinations
for i,j in combinations(range(2),2):
    Q_monomial_basis[i,j]=set(Q_tetra[i,j].monomials())
    for P_X in X_bivectors:
        Q_monomial_basis[i,j]|= set(P_X[i,j].monomials())

Q_monomial_basis={idx: list(b) for idx, b in Q_monomial_basis.items()}
Q_monomial_index= {idx:{m:k for k,m in enumerate(b)} for idx, b in␣
 ↪Q_monomial_basis.items()}
Q_monomial_count=sum(len(b) for b in Q_monomial_basis.values());

# We create the vector representation of Q_tetra in terms of the monomials.
Q_tetra_vector= vector(QQ, Q_monomial_count, sparse=True)
index_shift=0
for i,j in Q_monomial_basis:
    for coeff, monomial in Q_tetra[i,j]:
        monomial_index= Q_monomial_index[i,j][monomial]
```

```
        Q_tetra_vector[monomial_index+index_shift]=coeff
    index_shift+=len(Q_monomial_basis[i,j])

# We create the matrix that represents the X_bivectors in terms of the monomials.
X_bivector_evaluation_matrix= matrix(QQ, Q_monomial_count, len(X_bivectors),␣
↪sparse=True)
for k in range(len(X_bivectors)):
    P_X=X_bivectors[k]
    v=vector(QQ,Q_monomial_count, sparse=True)
    index_shift=0
    for i,j in Q_monomial_basis:
        for coeff, monomial in P_X[i,j]:
            monomial_index=Q_monomial_index[i,j][monomial]
            v[monomial_index+index_shift]=coeff
        index_shift+=len(Q_monomial_basis[i,j])
    X_bivector_evaluation_matrix.set_column(k,v)

# We soule the linear system.
X_solution=X_bivector_evaluation_matrix.solve_right(Q_tetra_vector)
print('Proposition 9: The solution on vector fields is given by', X_solution,'␣
↪\n')

# Finally, we will check the homogeneous system. We look at the kernel of the␣
↪X_bivector_evaluation_matrix and filter out the
# nullity of the X_evaluation matrix.
X_cocycle_space= X_bivector_evaluation_matrix.right_kernel().
↪quotient(X_evaluation_matrix.right_kernel())
X_cocycles=[X_cocycle_space.lift(v) for v in X_cocycle_space.basis()]
if all(X_bivector_evaluation_matrix*X_cocycle==0 for X_cocycle in X_cocycles)!=␣
↪True:
    print('There is an error in the cocycle space.')
print('The space of solutions to the homogeneous system has dimension',␣
↪X_cocycle_space.dimension(), )
print ('Proposition 10: These shifts are given by', X_cocycles, '\n')

# Let us write the vector fields as vector fields X such that [[P,X]]=0, rather␣
↪than just the combination of graphs.
shifts_formulas = [sum(X_cocycle[j]*X_vector_fields[j] for j in␣
↪range(len(X_vector_fields))) for X_cocycle in X_cocycles]

# We check that the shifts above is induced by the Hamiltonian vector field.
hamiltonian_encodings= [(1,2,3,0,2,3), (1,2,3,1,2,3), (1,2,3,0,1,3),␣
↪(0,2,3,0,1,3), (0,2,3,1,2,3), (0,1,2,0,1,3), (0,1,2,1,2,3)]

# We create a new encoding to graph definition, as the Hamiltonians are graphs␣
↪on 2 Levi-Civita vertices, 2 Casimir vertices
```

```
# and no sink.
def hamiltonian_encoding_to_graph(encoding):
    targets = [encoding[0:3], encoding[3:6]]
    edges = sum([[(k+1,v) for v in t] for (k,t) in enumerate(targets)], [])
    return FormalityGraph(0, 4, edges)
hamiltonian_graphs = [encoding_to_graph(e) for e in hamiltonian_encodings]

# This computation is also slightly different; we do not have the Euler vector␣
 ↪field since we do not have a sink, and as
# we have only 2 copies of the Poisson bivector, the index_choice for the sign␣
 ↪is only on 2 repeats as well. Moreover, there
# are only 2 Levi-Civita vertices and 2 Casimir vertices to give vertex_content␣
 ↪to.
hamiltonian_formulas = []
epsilon = xi[0]*xi[1]*xi[2] # Levi-Civita tensor
for h in hamiltonian_graphs:
    term = S3.zero()
    for index_choice in itertools.product(itertools.permutations(range(3)),␣
 ↪repeat=2):
        # TODO: Check that this way of evaluating is correct.
        sign = epsilon[index_choice[0]] * epsilon[index_choice[1]]
        vertex_content = [S3(rho), S3(rho), S3(a), S3(a)]
        for ((source, target), index) in zip(h.edges(), sum(map(list,␣
 ↪index_choice), [])):
            vertex_content[target] = diff(vertex_content[target],␣
 ↪even_coords[index])
        term += sign * prod(vertex_content)
    hamiltonian_formulas.append(term)

# We create the vector fields [[P,H]] from the Hamiltonians H.
hamiltonian_vector_fields=[]
for hamiltonian in hamiltonian_formulas:
    hamiltonian_vector_fields.append(P.bracket(hamiltonian))

# Create the basis of monomials.
hamiltonian_monomial_basis = {}
for j in range(len(shifts_formulas)):
    for i in range(3):
        hamiltonian_monomial_basis[i] = set(shifts_formulas[j][i].monomials())
        for formula in hamiltonian_vector_fields:
            hamiltonian_monomial_basis[i] |= set(formula[i].monomials())

hamiltonian_monomial_basis = {idx: list(b) for idx, b in␣
 ↪hamiltonian_monomial_basis.items()}
hamiltonian_monomial_index = {idx: {m : k for k, m in enumerate(b)} for idx, b␣
 ↪in hamiltonian_monomial_basis.items()}
```

```
hamiltonian_monomial_count = sum(len(b) for b in hamiltonian_monomial_basis.
 ↪values())

# Now, we write the shifts in terms of a vector of the monomials.
def shift_formula_to_vector(shift_formula):
        shift_vector = vector(QQ,hamiltonian_monomial_count, sparse=True)
        index_offset = 0
        for i in hamiltonian_monomial_basis:
            for coeff, monomial in shift_formula[i]:
                monomial_index = hamiltonian_monomial_index[i][monomial]
                shift_vector[monomial_index + index_offset] = coeff
            index_offset += len(hamiltonian_monomial_basis[i])
        return shift_vector

shifts_vectors = [shift_formula_to_vector(shift_formula) for shift_formula in␣
 ↪shifts_formulas]

# Create the evaluation matrix for the Hamiltonian vector fields.
hamiltonian_evaluation_matrix =␣
 ↪matrix(QQ,hamiltonian_monomial_count,len(hamiltonian_vector_fields),sparse=True)
for k in range(len(hamiltonian_vector_fields)):
    formula = hamiltonian_vector_fields[k]
    v = vector(QQ, hamiltonian_monomial_count, sparse=True)
    index_shift = 0
    for i in hamiltonian_monomial_basis:
        for coeff, monomial in formula[i]:
            monomial_index = hamiltonian_monomial_index[i][monomial]
            v[monomial_index +index_shift] = coeff
        index_shift += len(hamiltonian_monomial_basis[i])
    hamiltonian_evaluation_matrix.set_column(k, v)

nullity=hamiltonian_evaluation_matrix.right_nullity()
kernel_basis= hamiltonian_evaluation_matrix.right_kernel().basis()
print('The nullity among the Hamiltonian vector fields is ', nullity, '')
# Note that the below is on the level of vector fields, not formulas.␣
 ↪Unfortunately, the evaluation_matrix procedure only
# works from vector fields onward, and not for 0-vectors (formulas). You can␣
 ↪explictly check that the linear relations ARE
# already preserved on the level of formulas by simply evaluating␣
 ↪hamiltonian_formulas[0]==hamiltonian_formulas[4] etc.
print('Explicitly, these are the linear combinations that evaluate to 0:',␣
 ↪kernel_basis, '\n')

# We can now solve the shifts! Note that if (at least one) shift is NOT␣
 ↪Hamiltonian, the code will break here as no solution
# can be found.
```

```
shifts_solutions = [hamiltonian_evaluation_matrix.solve_right(shift_vector) for␣
↪shift_vector in shifts_vectors]

# Write the solutions above from combinations of graphs to their vector field␣
↪forms to check validity.
solution_formulas = [sum(shift_solution[i]*hamiltonian_vector_fields[i] for i in␣
↪range(len(shift_solution))) for shift_solution in shifts_solutions]

if any(P.bracket(solution_formula) != 0 for solution_formula in␣
↪solution_formulas) or solution_formulas != shifts_formulas:
    print('There is an error in computing the shifts.')
else:
    print('Theorem 12: The shifts in the cocycle space are Hamiltonian.')

for k, shift_solution in enumerate(shifts_solutions):
        print('The shift #', k+1, 'is the following linear combination of␣
↪Hamiltonian vector fields:', shift_solution)
```

```
We have 48 graphs.

Lemma 8: There are 41 nonisomorphic graphs.

There are 13 graphs that evaluate to 0 under the morhpism from graphs to
multivectors.
These graphs are:
graph 12
graph 14
graph 19
graph 27
graph 28
graph 30
graph 35
graph 36
graph 37
graph 38
graph 43
graph 47
graph 48

The vector fields have 28 linear relations among themselves.

There are 13 vector fields in X_vector_fields that evaluate to 0 bivectors after
taking the Schouten bracket with P .
These vector fields that evaluate to 0 are obtained from the graphs:
graph 12
graph 14
graph 19
```

```
graph 27
graph 28
graph 30
graph 35
graph 36
graph 37
graph 38
graph 43
graph 47
graph 48

Proposition 9: The solution on vector fields is given by (8, 0, 0, 24, 0, 0, 8,
24, 0, 0, 0, 0, 0, 0, 0, 12, 16, 0, 0, 0, 0, 0, 0, 0, 16, 12, 0, 0, 16, 0, 0, 0,
24, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)

The space of solutions to the homogeneous system has dimension 3
Proposition 10: These shifts are given by [(0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 0, 0), (0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1/2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1/2, 0, 0, 0), (0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 1, 0, 0,
0)]

The nullity among the Hamiltonian vector fields is  3
Explicitly, these are the linear combinations that evaluate to 0: [
(1, 0, 0, 0, -1, 0, 0),
(0, 0, 1, 0, 0, 0, 1),
(0, 0, 0, 1, 0, 0, -1)
]

Theorem 12: The shifts in the cocycle space are Hamiltonian.
The shift # 1 is the following linear combination of Hamiltonian vector fields:
(0, 0, 1, 0, 0, 0, 0)
The shift # 2 is the following linear combination of Hamiltonian vector fields:
(1/4, 0, 0, 0, 0, 0, 0)
The shift # 3 is the following linear combination of Hamiltonian vector fields:
(1/2, -1, 0, 0, 0, 0, 0)
```

Kontsevich graphs act on Nambu–Poisson brackets, III. Uniqueness aspects (Section 5)

November 22, 2024

[1]:
```
# 18-09-2024 The code below is a combination of already existing code by R. Buring, and adjustments made by F. Schipper
# (particularly, the biggest adjustments are checking that the basis element(s) of the cocycle space are Hamiltonian,
# commentary, printing of output and rewriting some code to be able to run it parallelized). It is supplementary material
# accompanying proceedings written for the ISQS28 (Integrable Systems and Quantum Symmetries) conference (see also: arxiv link??).

# Extra packages that are useful
# for Pool, note that the # of processors throughout this is set to 24. Depending on your machine, you want to change this
# amount. A larger problem in probably the amount of memory you have available. You can split up parts of the code, and run
# smaller amounts, while saving necessary vector fields, lists of encodings etc on disc (eg via the pickle library). Once you
# need the saved information, load it back in. On a 32GB RAM machine, the code ran after splitting it into 6 smaller pieces
from multiprocessing import Pool
import itertools

# We import the following (see https://github.com/rburing/gcaops) to be able to run the code.
from gcaops.graph.formality_graph import FormalityGraph
from gcaops.algebra.differential_polynomial_ring import DifferentialPolynomialRing
from gcaops.algebra.superfunction_algebra import SuperfunctionAlgebra
from gcaops.graph.undirected_graph_complex import UndirectedGraphComplex
from gcaops.graph.directed_graph_complex import DirectedGraphComplex

# We start by generating the encodings for the descendants of Gamma_11^2D and Gamma_12^2D
X_graph_encodings = []
for (i2,j1,j2,k) in itertools.product([2,5,8],[1,4,7],[1,4,7],[3,6,9]):
    X_graph_encodings.append((0,1,4,7,j1,k,5,8,j2,i2,6,9))
```

```
for (i1,i2,j1,j2,k) in itertools.product([2,5,8],[2,5,8], [1,4,7], [1,4,7],␣
 ↪[3,6,9]):
    X_graph_encodings.append((0,i1,4,7,j1,k,5,8,j2,i2,6,9))

# To move from the encodings of the graphs to actual graphs, we use the next␣
 ↪function. The function splits the
# encoding by vertex via ;, and then the targets by ,. A graph is returned on 3␣
 ↪Levi-Civita vertices, 3 corresponding a^1
# Casimir vertices, 3 corresponding a^2 Casimir vertices, 1 sink, and with edges␣
 ↪(original vertex, target vertex).
def encoding_to_graph(encoding):
    targets = [encoding[0:4], encoding[4:8], encoding[8:12]]
    edges = sum([[(k+1,v) for v in t] for (k,t) in enumerate(targets)], [])
    return FormalityGraph(1, 9, edges)

# The computation below still goes reasonably fast as there's only 324 graphs.␣
 ↪When working with significantly more graphs
# (for instance with gamma_5 rather than gamma_3) you will want to parallelize␣
 ↪these computations as well.
X_graphs = [encoding_to_graph(e) for e in X_graph_encodings]
print('We have', len(X_graphs), 'graphs.\n')

# Below, we check the how many graphs that were created via the encodings are␣
 ↪(non)isomorphic. It does so in the following way:
# for each formality graph in X_graphs, it computes the edges of the canonical␣
 ↪form (canonical in the sense that isomorphic
# graphs return the same canonical form) of said formality graph. If a graph␣
 ↪with these edges already appears in X_graphs_iso,
# the new (isomorphic) graph is not added to the list. If the graph does not yet␣
 ↪appear, it is added to the list. The list is
# indexed by h, NOT by an index from 0,...,40. For this, unmute the line␣
 ↪X_graphs_iso=list(X_graphs_iso.values())
#X_graphs_iso={}
#for g in X_graphs:
#    h=tuple(g.canonical_form().edges())
#    if not h in X_graphs_iso:
#        X_graphs_iso[h]=g
#X_graphs_iso=list(X_graphs_iso.values())
#print ('There are', len(X_graphs_iso), 'nonisomorphic graphs.\n')

# Create the differential polynomial ring. We are working in 4D, so we have even␣
 ↪coordinates x,y,z,w and the corresponding
# odd coordinates xi[0], xi[1], xi[2] and xi[3]. rho is exactly the rho in a 4D␣
 ↪Nambu-determinant Poisson bracket,
```

```
# (P(f,g)= rho d(f,g,a^1,a^2)/d(x,y,z,w)), and a^1, a^2 are the Casimirs. ␣
 ↪Finally, max_differential_orders  tells the
# programme how many times rho and a^1/a^2 can be  differentiated. The maximum␣
 ↪is stipulated by the graphs (we cannot have
# double edges). Thus, the maximum in degree of each rho vertex is 4. As we will␣
 ↪be looking at [[P,X]], we add an extra +1 to
# the differential orders since taking the Schouten bracket with P introduces an␣
 ↪extra derivative.
D4 = DifferentialPolynomialRing(QQ, ('rho','a1', 'a2'), ('x','y','z', 'w'),␣
 ↪max_differential_orders=[3+1,4+1,4+1])
rho, a1, a2 = D4.fibre_variables()
x,y,z,w = D4.base_variables()
even_coords = [x,y,z,w]

S4.<xi0,xi1,xi2,xi3> = SuperfunctionAlgebra(D4, D4.base_variables())
xi = S4.gens()
odd_coords = xi

# We now compute the vector fields corresponding to the graphs. E is the Euler␣
 ↪vector field in the sink (vertex 0), and
# epsilon is the Levi-Civita tensor. Note that we have a Levi-Civita tensor at␣
 ↪each of the vertices 1, 2, 3. We first compute
# the sign of each term appearing in the formulas, and then compute the␣
 ↪differential polynomial. Note that compared to 2D and
# 3D, this is now a definition rather than a for loop; this is to be able to␣
 ↪parallelize the computations.
epsilon = xi[0]*xi[1]*xi[2]*xi[3]
E = x*xi[0] + y*xi[1] + z*xi[2] + w*xi[3]
def evaluate_graph(g):
    result = S4.zero()
    for index_choice in itertools.product(itertools.permutations(range(4)),␣
 ↪repeat=3):
        sign = epsilon[index_choice[0]] * epsilon[index_choice[1]] *␣
 ↪epsilon[index_choice[2]]
        vertex_content = [E, S4(rho), S4(rho), S4(rho), S4(a1), S4(a1), S4(a1),␣
 ↪S4(a2), S4(a2), S4(a2)]
        for ((source, target), index) in zip(g.edges(), sum(map(list,␣
 ↪index_choice), [])):
            vertex_content[target] = vertex_content[target].
 ↪derivative(even_coords[index])
        result += sign * prod(vertex_content)
    return result

# We compute the formulas.
X_vector_fields = []
with Pool(processes=24) as pool:
```

```
    X_vector_fields = list(pool.imap(evaluate_graph, X_graphs))

# We check how many (if any) graphs evaluate to 0.
zeros=X_vector_fields.count(0)
print('There are', zeros, 'graphs that evaluate to 0 under the morhpism from␣
 ↪graphs to multivectors.')

# In case the are graphs that evaluate to 0, the line below shows which graphs␣
 ↪do so (note that the counting starts from 1!).
print('These graphs are:')
for (k,X) in enumerate(X_vector_fields):
    if X==0:
        print('graph', k+1,)
print()

# The next part is to find out linear relations of the vector fields we just␣
 ↪computed. We look at the monomials that appear
# in each xi[0], xi[1], xi[2] and xi[3] part of the vector fields, and store␣
 ↪them in X_monomial_basis.
X_monomial_basis = [set([]) for i in range(4)]
for i in range(4):
    for X in X_vector_fields:
        X_monomial_basis[i] |= set(X[i].monomials())
X_monomial_basis = [list(b) for b in X_monomial_basis]
X_monomial_count = sum(len(b) for b in X_monomial_basis)

# Next, we use this monomial basis to create a matrix that identifies each␣
 ↪vector field by the monomials that appear in it.
X_monomial_to_index = [{monomial : idx for (idx,monomial) in␣
 ↪enumerate(X_monomial_basis[j])} for j in range(4)]
X_evaluation_matrix = matrix(QQ, X_monomial_count, len(X_vector_fields))
for i in range(len(X_vector_fields)):
    v = vector(QQ, X_monomial_count)
    index_shift = 0
    for j in range(4):
        f = X_vector_fields[i][j]
        for coeff, monomial in zip(f.coefficients(), f.monomials()):
            monomial_index = X_monomial_to_index[j][monomial]
            v[index_shift + monomial_index] = coeff
        index_shift += len(X_monomial_basis[j])
    X_evaluation_matrix.set_column(i, v)

# We can now detect linear relations by computing the nullity of this matrix.
nullity = X_evaluation_matrix.right_nullity()
print('The vector fields have', nullity, 'linear relations among themselves.\n')
```

```
# Unmute the next line if you want to explicitly see the 28 linear relations␣
 ↪among the vector fields.
#print('These relations are expressed by \n', X_evaluation_matrix.right_kernel().
 ↪basis(), '\n')

# We save a list of linearly independent vector fields.
pivots = X_evaluation_matrix.pivots()
print(' A maximal subset of linearly independent graphs is given by:',␣
 ↪list(pivots),'\n')

# Next, let us create the swapped encodings in order to be able skew-symmetrize␣
 ↪the vector fields (skew-symmetrizing is with
# respect to a^1 and a^2).
swapped_X_graph_encodings= []
for i in range(len(X_graph_encodings)):
    new_encoding=[]
    for j in range(12):
        if X_graph_encodings[i][j]==4:
             new_encoding.append(7)
        elif X_graph_encodings[i][j]==7:
            new_encoding.append(4)
        elif X_graph_encodings[i][j]==5:
            new_encoding.append(8)
        elif X_graph_encodings[i][j]==8:
            new_encoding.append(5)
        elif X_graph_encodings[i][j]==6:
            new_encoding.append(9)
        elif X_graph_encodings[i][j]==9:
            new_encoding.append(6)
        else:
            new_encoding.append(X_graph_encodings[i][j])
    swapped_X_graph_encodings.append(new_encoding)

# Now, we compute the formulas corresponding to the swapped encodings.
swapped_graphs = [encoding_to_graph(e) for e in swapped_X_graph_encodings]

# We find the corresponding vector fields.
swapped_vector_fields = []
with Pool(processes=24) as pool:
    swapped_vector_fields = list(pool.imap(evaluate_graph, swapped_graphs))

# We create skew vector fields from the linearly independent non-skewed vector␣
 ↪fields.
skew_vector_fields=[]
for i in pivots:
    skew_vector_fields.append(1/2*(X_vector_fields[i]-swapped_vector_fields[i]))
```

```
# We also create a new list storing the corresponding encodings (only the␣
 ↪original encodings, not also the swapped encodings).
independent_encodings=[]
for i in pivots:
    independent_encodings.append(X_graph_encodings[i])

# While we got rid of linear dependencies in X_vector_fields, after skewing the␣
 ↪vector fields, new dependencies show up. We
# again get rid of these in the exact same manner.
X_skew_monomial_basis = [set([]) for i in range(4)]
for i in range(4):
    for X in skew_vector_fields:
        X_skew_monomial_basis[i] |= set(X[i].monomials())
X_skew_monomial_basis = [list(b) for b in X_skew_monomial_basis]
X_skew_monomial_index = [{m : k for k, m in enumerate(b)} for b in␣
 ↪X_skew_monomial_basis]

X_skew_monomial_count = sum(len(b) for b in X_skew_monomial_basis)

X_skew_monomial_to_index = [{monomial : idx for (idx,monomial) in␣
 ↪enumerate(X_skew_monomial_basis[j])} for j in range(4)]
X_skew_evaluation_matrix = matrix(QQ, X_skew_monomial_count,␣
 ↪len(skew_vector_fields), sparse=True)
for i in range(len(skew_vector_fields)):
    v = vector(QQ, X_skew_monomial_count, sparse=True)
    index_shift = 0
    for j in range(4):
        f = skew_vector_fields[i][j]
        for coeff, monomial in zip(f.coefficients(), f.monomials()):
            monomial_index = X_skew_monomial_to_index[j][monomial]
            v[index_shift + monomial_index] = coeff
        index_shift += len(X_skew_monomial_basis[j])
    X_skew_evaluation_matrix.set_column(i, v)

print('We have', X_skew_evaluation_matrix.rank(), 'linearly independent skewed␣
 ↪vector fields obtained from the graphs. \n')

# We collect the linearly independent skewed vector fields and the corresponding␣
 ↪encodings
pivots_2 = X_skew_evaluation_matrix.pivots()
print('A maximal subset of linearly independent skewed vector fields is given by␣
 ↪graphs', list(pivots_2),'\n', )
independent_encodings_skew = [independent_encodings[k] for k in pivots_2]
skew_vector_fields_independent = [skew_vector_fields[k] for k in pivots_2]
```

```
# We now compute the tetrahedral flow. First, we create the Poisson bivector P
 ↪and check that it satisfies [[P,P]]=0.
P= (rho*epsilon).bracket(a1).bracket(a2)
if P.bracket(P)!=0:
    print('P is not a Poisson bivector. \n')

# Introduce the graph complex, and find the tetrahedron as a graph on 4 Poisson
 ↪vertices and 6 edges in the cohomology. This
# tetrahedron is unoriented, so we orient it. The next step is to make sure that
 ↪the graph is built of wedges. This means
# that there cannot be more than 2 outgoing edges at each vertex. This gives 2
 ↪graphs; one where 3 vertices have 2 outgoing
# edges, and one where 2 vertices have 2 outgoing edges and 2 vertices have 1
 ↪outgoing edge.
# tetrahedron_oriented_filtered.show() gives a drawing of these graphs.
# Finally, the bivector corresponding to the graph is computed.
GC=UndirectedGraphComplex(QQ, implementation='vector', sparse=True)
tetrahedron= GC.cohomology_basis(4,6)[0]
dGC=DirectedGraphComplex(QQ, implementation='vector')
tetrahedron_oriented= dGC(tetrahedron)
tetrahedron_oriented_filtered= tetrahedron_oriented.filter(max_out_degree=2)
#tetrahedron_oriented_filtered.show()
tetrahedron_operation= S4.graph_operation(tetrahedron_oriented_filtered)
Q_tetra= tetrahedron_operation(P, P, P, P)
# !!! You don't want to print Q_tetra, it is very large
# print('The tetrahedral flow in 4D is', Q_tetra, '\n')

print('Q_tetra is computed.\n')

# Instead of solving the (very large) linear system directly, we use another
 ↪method in 4D. This method is explained
# in https://arxiv.org/abs/2112.03897. Essentially, we solve 2 (somewhat)
 ↪smaller linear systems.
def casimir_flow(f):
    return 4*tetrahedron_operation(P,P,P,f)

a = [S4(a1), S4(a2)]
adot = [casimir_flow(a[0]), casimir_flow(a[1])]

X_a_multivectors = [[vector_field.bracket(casimir) for vector_field in
 ↪skew_vector_fields_independent] for casimir in a]

X_a_basis = [set(flow_of_casimir[()].monomials()) for flow_of_casimir in adot]
for k in range(len(a)):
    for X_a_multivector in X_a_multivectors[k]:
        X_a_basis[k] |= set(X_a_multivector[()].monomials())
```

```
X_a_basis = [list(B) for B in X_a_basis]

X_a_monomial_index = [{m : k for k, m in enumerate(B)} for B in X_a_basis]
X_a_evaluation_matrix = [matrix(QQ, len(B), len(skew_vector_fields_independent),␣
 ↪sparse=True) for B in X_a_basis]
for i in range(len(a)):
    for j in range(len(skew_vector_fields_independent)):
        v = vector(QQ, len(X_a_basis[i]), sparse=True)
        multivector = X_a_multivectors[i][j][()]
        for coeff, monomial in zip(multivector.coefficients(), multivector.
 ↪monomials()):
            monomial_index = X_a_monomial_index[i][monomial]
            v[monomial_index] = coeff
        X_a_evaluation_matrix[i].set_column(j, v)

adot_vector = [vector(QQ, len(B)) for B in X_a_basis]
for i in range(len(a)):
    f = adot[i][()]
    for coeff, monomial in zip(f.coefficients(), f.monomials()):
        monomial_index = X_a_monomial_index[i][monomial]
        adot_vector[i][monomial_index] = coeff

P0 = (rho*epsilon).bracket(adot[0]).bracket(a2)
P1 = (rho*epsilon).bracket(a1).bracket(adot[1])
Q_remainder = Q_tetra - P0 - P1
P_without_rho = epsilon.bracket(a1).bracket(a2)
rhodot = Q_remainder[0,1] // P_without_rho[0,1]

X_rho_multivectors = [vector_field.bracket(rho*epsilon) for vector_field in␣
 ↪skew_vector_fields_independent]

X_rho_basis = set(rhodot.monomials())
for X_rho_multivector in X_rho_multivectors:
    X_rho_basis |= set(X_rho_multivector[0,1,2,3].monomials())
X_rho_basis = list(X_rho_basis)

X_rho_monomial_index = {m : k for k, m in enumerate(X_rho_basis)}
X_rho_evaluation_matrix = matrix(QQ, len(X_rho_basis),␣
 ↪len(skew_vector_fields_independent), sparse=True)
for j in range(len(skew_vector_fields_independent)):
    f = X_rho_multivectors[j][0,1,2,3]
    v = vector(QQ, len(X_rho_basis), sparse=True)
    for coeff, monomial in zip(f.coefficients(), f.monomials()):
        monomial_index = X_rho_monomial_index[monomial]
        v[monomial_index] = coeff
    X_rho_evaluation_matrix.set_column(j, v)
```

```
rhodot_vector = vector(QQ, len(X_rho_basis), sparse=True)
for coeff, monomial in zip(rhodot.coefficients(), rhodot.monomials()):
    monomial_index = X_rho_monomial_index[monomial]
    rhodot_vector[monomial_index] = coeff

big_matrix = X_a_evaluation_matrix[0].stack(X_a_evaluation_matrix[1]).
 ↪stack(X_rho_evaluation_matrix)
big_vector = vector(list(-adot_vector[0]) + list(-adot_vector[1]) +␣
 ↪list(-rhodot_vector))
X_solution_vector = big_matrix.solve_right(big_vector)
print('Proposition 14: There exist a solution over skewed vector fields from␣
 ↪graphs. It is given by', X_solution_vector, '\n')

# Now, we create the evaluation matrix of the skewed independent vector fields;␣
 ↪we need this in order to find a basis
# for the cocycle space.
X_skew_ind_monomial_basis = [set([]) for i in range(4)]
for i in range(4):
    for vector_field in skew_vector_fields_independent:
        X_skew_ind_monomial_basis[i] |= set(vector_field[i].monomials())
X_skew_ind_monomial_basis = [list(b) for b in X_skew_ind_monomial_basis]
X_skew_ind_monomial_index = [{m : k for k, m in enumerate(b)} for b in␣
 ↪X_skew_ind_monomial_basis]

X_skew_ind_monomial_count = sum(len(b) for b in X_skew_ind_monomial_basis)

X_skew_ind_evaluation_matrix = matrix(QQ, X_skew_ind_monomial_count,␣
 ↪len(skew_vector_fields_independent), sparse=True)
for i in range(len(skew_vector_fields_independent)):
    v = vector(QQ, X_skew_ind_monomial_count, sparse=True)
    index_shift = 0
    for j in range(4):
        vector_field = skew_vector_fields_independent[i][j]
        for coeff, monomial in zip(vector_field.coefficients(), vector_field.
 ↪monomials()):
            monomial_index = X_skew_ind_monomial_index[j][monomial]
            v[index_shift + monomial_index] = coeff
        index_shift += len(X_skew_ind_monomial_basis[j])
    X_skew_ind_evaluation_matrix.set_column(i, v)

# We compute a basis for the cocycle space.
X_cocycle_space = big_matrix.right_kernel().
 ↪quotient(X_skew_ind_evaluation_matrix.right_kernel())
X_cocycles=[X_cocycle_space.lift(v) for v in X_cocycle_space.basis()]
print('Proposition 15: The cocycle space has dimension',X_cocycle_space.
 ↪dimension(), '. The shifts are given by', X_cocycles, '\n')
```

```
# We compute the vector fields corresponding to the computed basis elements of␣
 ↪the cocycle space.
shifts_formulas = [sum(X_cocycle[j]*skew_vector_fields_independent[j] for j in␣
 ↪range(len(skew_vector_fields_independent))) for X_cocycle in X_cocycles]

# Now, we start the procedure of checking whether the shifts appearing in the␣
 ↪cocycle space are Hamiltonian.
hamiltonian_encodings= [(0, 1, 2, 4, 0, 1, 3, 5), (0, 1, 2, 4, 1, 2, 3, 5), (0,␣
 ↪1, 2, 4, 1, 3, 4, 5), (0, 2, 3, 4, 1, 2, 3, 5), (0, 2, 3, 4, 1, 3, 4, 5), (0,␣
 ↪2, 4, 5, 1, 3, 4, 5), (0, 1, 2, 4, 0, 2, 3, 5), (0, 1, 2, 4, 0, 3, 4, 5), (0,␣
 ↪1, 2, 4, 2, 3, 4, 5), (0, 2, 3, 4, 0, 2, 3, 5), (0, 2, 4, 5, 0, 2, 3, 5), (0,␣
 ↪2, 3, 4, 0, 3, 4, 5), (0, 2, 4, 5, 0, 3, 4, 5), (0, 2, 3, 4, 2, 3, 4, 5), (0,␣
 ↪2, 4, 5, 2, 3, 4, 5), (1, 2, 3, 4, 0, 2, 3, 5), (1, 2, 4, 5, 0, 3, 4, 5), (1,␣
 ↪2, 3, 4, 0, 3, 4, 5), (1, 2, 3, 4, 2, 3, 4, 5), (1, 2, 4, 5, 2, 3, 4, 5), (2,␣
 ↪3, 4, 5, 2, 3, 4, 5)]

# We find the graphs corresponding to the encodings.
def hamiltonian_encoding_to_graph(encoding):
    targets = [encoding[0:4], encoding[4:8]]
    edges = sum([[(k+1,v) for v in t] for (k,t) in enumerate(targets)], [])
    return FormalityGraph(0, 6, edges)

hamiltonian_graphs = [encoding_to_graph(e) for e in hamiltonian_encodings]

# Since our vector fields are skew with respect to the Casimirs a1 and a2, we␣
 ↪require that the Hamiltonians are symmetric with
# respect to a1 and a2; in this way, as the Poisson bivector P itself is skew␣
 ↪(wrt a1 and a2), we guarantee that the
# Hamiltonian vector  fields are also skew symmetric wrt a1 and a2.

hamiltonian_encodings_interchanged= [(0, 1, 4, 2, 0, 1, 5, 3), (0, 1, 4, 2, 1,␣
 ↪4, 5, 3), (0, 1, 4, 2, 1, 5, 2, 3), (0, 4, 5, 2, 1, 4, 5, 3), (0, 4, 5, 2, 1,␣
 ↪5, 2, 3), (0, 4, 2, 3, 1, 5, 2, 3), (0, 1, 4, 2, 0, 4, 5, 3), (0, 1, 4, 2, 0,␣
 ↪5, 2, 3), (0, 1, 4, 2, 4, 5, 2, 3), (0, 4, 5, 2, 0, 4, 5, 3), (0, 4, 2, 3, 0,␣
 ↪4, 5, 3), (0, 4, 5, 2, 0, 5, 2, 3), (0, 4, 2, 3, 0, 5, 2, 3), (0, 4, 5, 2, 4,␣
 ↪5, 2, 3), (0, 4, 2, 3, 4, 5, 2, 3), (1, 4, 5, 2, 0, 4, 5, 3), (1, 4, 2, 3, 0,␣
 ↪5, 2, 3), (1, 4, 5, 2, 0, 5, 2, 3), (1, 4, 5, 2, 4, 5, 2, 3), (1, 4, 2, 3, 4,␣
 ↪5, 2, 3), (4, 5, 2, 3, 4, 5, 2, 3)]

hamiltonian_graphs_interchanged = [encoding_to_graph(e) for e in␣
 ↪hamiltonian_encodings_interchanged]

# We create a new function to evaluate the Hamiltonian graphs to Hamiltonian␣
 ↪formulas.
def evaluate_ham(g):
```

```
    result = S4.zero()
    for index_choice in itertools.product(itertools.permutations(range(4)),␣
 ↪repeat=2):
        sign = epsilon[index_choice[0]] * epsilon[index_choice[1]]
        vertex_content = [S4(rho), S4(rho), S4(a1), S4(a1), S4(a2), S4(a2)]
        for ((source, target), index) in zip(g.edges(), sum(map(list,␣
 ↪index_choice), [])):
            vertex_content[target] = vertex_content[target].
 ↪derivative(even_coords[index])
        result += sign * prod(vertex_content)
    return result

# We compute the Hamiltonian formulas.
hamiltonian_formulas = []
with Pool(processes=24) as pool:
    hamiltonian_formulas = list(pool.imap(evaluate_ham, hamiltonian_graphs))

# We compute the Hamiltonian formulas with a1 and a2 swapped.
hamiltonian_formulas_interchanged = []
with Pool(processes=24) as pool:
    hamiltonian_formulas_interchanged = list(pool.imap(evaluate_ham,␣
 ↪hamiltonian_graphs_interchanged))

# We obtain the symmetrized Hamiltonian formulas.
hamiltonian_formulas_symm=[]
for i in range(len(hamiltonian_formulas)):
    hamiltonian_formulas_symm.append(1/
 ↪2*(hamiltonian_formulas[i]+hamiltonian_formulas_interchanged[i]))

# We check if we have symmetrized Hamiltonian formulas that evaluate to 0.
print('We have', hamiltonian_formulas_symm.count(0), 'Hamiltonian graph that␣
 ↪evaluates to 0.')
print('The Hamiltonian graph evaluated to 0 is:\n')
for (k,ham) in enumerate(hamiltonian_formulas_symm):
    if ham==0:
        print('Hamiltonian', k+1,'\n')

#From the symmetrized Hamiltonians we compute the skew-symmetrized Hamiltonian␣
 ↪vector fields.
hamiltonian_vector_fields_skew=[]
for formula in hamiltonian_formulas_symm:
    hamiltonian_vector_fields_skew.append(P.bracket(formula))

# We create the Hamiltonian monomial basis to be able to express the basis of␣
 ↪the cocycle space in terms of the monomials
# appearing in the skewed Hamiltonian vector fields.
```

```
hamiltonian_monomial_basis = {}
for j in range(len(shifts_formulas)):
    for i in range(4):
        hamiltonian_monomial_basis[i] = set(shifts_formulas[j][i].monomials())
        for formula in hamiltonian_vector_fields_skew:
            hamiltonian_monomial_basis[i] |= set(formula[i].monomials())

hamiltonian_monomial_basis = {idx: list(b) for idx, b in␣
↪hamiltonian_monomial_basis.items()}
hamiltonian_monomial_index = {idx: {m : k for k, m in enumerate(b)} for idx, b␣
↪in hamiltonian_monomial_basis.items()}
hamiltonian_monomial_count = sum(len(b) for b in hamiltonian_monomial_basis.
↪values())

def shift_formula_to_vector(shift_formula):
        shift_vector = vector(QQ,hamiltonian_monomial_count, sparse=True)
        index_offset = 0
        for i in hamiltonian_monomial_basis:
            for coeff, monomial in shift_formula[i]:
                monomial_index = hamiltonian_monomial_index[i][monomial]
                shift_vector[monomial_index + index_offset] = coeff
            index_offset += len(hamiltonian_monomial_basis[i])
        return shift_vector

shifts_vectors = [shift_formula_to_vector(shift_formula) for shift_formula in␣
↪shifts_formulas]

# Let us collect which monomials appear in which skewed Hamiltonians vector␣
↪fields, and store this in an evaluation matrix.
hamiltonian_evaluation_matrix =␣
↪matrix(QQ,hamiltonian_monomial_count,len(hamiltonian_vector_fields_skew),sparse=True)
for k in range(len(hamiltonian_vector_fields_skew)):
    vector_field = hamiltonian_vector_fields_skew[k]
    v = vector(QQ, hamiltonian_monomial_count, sparse=True)
    index_shift = 0
    for i in hamiltonian_monomial_basis:
        for coeff, monomial in vector_field[i]:
            monomial_index = hamiltonian_monomial_index[i][monomial]
            v[monomial_index +index_shift] = coeff
        index_shift += len(hamiltonian_monomial_basis[i])
    hamiltonian_evaluation_matrix.set_column(k, v)

# Let us check the nullity of the skewed Hamiltonian vector fields among␣
↪themselves.
nullity=hamiltonian_evaluation_matrix.right_nullity()
kernel_basis= hamiltonian_evaluation_matrix.right_kernel().basis()
```

```
print('The nullity among the skewed Hamiltonian vector fields is ', nullity, '.
 ↪\n')
# Note that the below is on the level of vector fields, not formulas.␣
 ↪Unfortunately, the evaluation_matrix procedure only
# works from vector fields onward, and not for 0-vectors (formulas). You can␣
 ↪explictly check that the linear relations ARE
# already preserved on the level of formulas by simply evaluating␣
 ↪hamiltonian_formulas[x]==hamiltonian_formulas[y] etc.
print('Explicitly, these are the linear combinations that evaluate to 0:',␣
 ↪kernel_basis, '\n')

# We can now solve the shifts! Note that if (at least one) shift is NOT␣
 ↪Hamiltonian, the code will break here as no solution
# can be found.
shifts_solutions = [hamiltonian_evaluation_matrix.solve_right(shift_vector) for␣
 ↪shift_vector in shifts_vectors]


# Write the solutions above from combinations of graphs to their vector field␣
 ↪forms to check validity.
solution_formulas = [sum(shift_solution[i]*hamiltonian_vector_fields_skew[i] for␣
 ↪i in range(len(shift_solution))) for shift_solution in shifts_solutions]


# We check that the solutions we found in terms of the skewed Hamiltonian vector␣
 ↪fields are truly correct; we check
# explicitly whether they agree on the previously computed formulass for the␣
 ↪basis elements of the cocycle space, and
# that they ARE elements of the cocycle space (i.e. they evaluate to 0 under␣
 ↪[[P, ]]).
if any(P.bracket(solution_formula) != 0 for solution_formula in␣
 ↪solution_formulas) or solution_formulas != shifts_formulas:
    print('There is an error in computing the shifts.')
else:
    print('Theorem 17: The shifts in the cocycle space are Hamiltonian.')

for k, shift_solution in enumerate(shifts_solutions):
        print('The shift #', k+1, 'is the following linear combination of␣
 ↪Hamiltonian vector fields:', shift_solution)
```

```
We have 324 graphs.

There are 54 graphs that evaluate to 0 under the morhpism from graphs to
multivectors.
These graphs are:
graph 32
graph 38
```

```
graph 63
graph 75
graph 85
graph 88
graph 112
graph 113
graph 115
graph 119
graph 139
graph 142
graph 144
graph 156
graph 166
graph 167
graph 169
graph 172
graph 173
graph 174
graph 181
graph 182
graph 183
graph 193
graph 196
graph 202
graph 203
graph 204
graph 220
graph 223
graph 233
graph 234
graph 239
graph 240
graph 247
graph 250
graph 252
graph 253
graph 254
graph 255
graph 262
graph 263
graph 264
graph 274
graph 277
graph 287
graph 288
graph 293
graph 294
graph 301
```

```
graph 304
graph 322
graph 323
graph 324

The vector fields have 201 linear relations among themselves.

 A maximal subset of linearly independent graphs is given by: [0, 1, 2, 3, 4, 5,
6, 7, 8, 10, 11, 12, 13, 14, 15, 16, 17, 19, 20, 22, 23, 24, 25, 26, 28, 29, 32,
34, 35, 38, 40, 41, 43, 44, 47, 50, 52, 53, 56, 59, 68, 71, 80, 81, 82, 83, 93,
94, 95, 96, 97, 102, 104, 105, 106, 107, 108, 109, 110, 113, 116, 117, 119, 126,
128, 135, 137, 144, 153, 162, 164, 167, 170, 174, 176, 179, 185, 186, 187, 188,
189, 190, 191, 194, 196, 197, 205, 206, 207, 209, 210, 212, 213, 214, 215, 216,
217, 218, 223, 235, 240, 241, 242, 243, 244, 255, 267, 268, 270, 282, 284, 295,
297, 298, 299, 301, 302, 306, 309, 310, 311, 312, 313]

We have 64 linearly independent skewed vector fields obtained from the graphs.

A maximal subset of linearly independent skewed vector fields is given by graphs
[0, 1, 3, 4, 5, 9, 10, 11, 12, 13, 14, 15, 16, 24, 25, 26, 27, 28, 30, 31, 32,
33, 35, 36, 43, 44, 46, 47, 48, 49, 50, 56, 57, 58, 59, 60, 61, 63, 69, 70, 71,
73, 74, 76, 77, 78, 79, 80, 81, 82, 84, 85, 86, 87, 88, 90, 91, 92, 93, 94, 95,
96, 100, 101]

Q_tetra is computed.

Proposition 14: There exist a solution over skewed vector fields from graphs. It
is given by (-8, 0, 0, -48, 0, 0, 0, -16, -48, -48, -16, -48, 0, 12, 0, 0, 0, 0,
-24, 0, 0, 24, 0, -24, -16, 0, 0, 0, 0, 0, 0, -32, -12, 0, 48, -48, -32, -32,
-48, -48, -96, 0, -48, -48, 0, 0, -48, 0, 24, 0, 0, 0, 48, 48, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0)

Proposition 15: The cocycle space has dimension 7 . The shifts are given by [(0,
1, 0, 0, 0, 0, 0, 0, -1/2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
0, 0, 0, 0, 0, 1/2, 0, 1, -1, 0, 0, 0, 1, -1, 1/2, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0), (0, 0, 0, 1, 0, 0, 0, 0, 1/2, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 1, 0, 0, 0,
0, 1, -1/2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, -1, 0, 1, 0, 1, 0, 0, 0, -1/2, 0,
0, 0, 0, 0, 0, -1, 0, 0, -1, -1, 1, 0, 0, 0, 2, 1, 0, -1, 0, -1, 1, -1, 0, 0, 0,
0, 0, 0, -1/2, 0, 0, 0, 0, 0, 0, -1, 0, 1, -1/2), (0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 0, 0, 1, 0, -1, 0, -1, 0, 0, 0, 1/2, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
-1, 0, 0, 0, 0, 0, 0, 1, 0, 0, -1, 1, 0, 0, 0, 0, 0, 0, 1/2, 0, 0, 0, 0, 0, 0,
0, 0, -1, 1/2), (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 4, 0, -4, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0), (0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0,
```

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

We have 1 Hamiltonian graph that evaluates to 0.
The Hamiltonian graph evaluated to 0 is:

Hamiltonian 9

The nullity among the skewed Hamiltonian vector fields is  13 .

Explicitly, these are the linear combinations that evaluate to 0: [
(0, 1, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0),
(0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0),
(0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, -1, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, -1, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, -1, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, -1, 0)
]

Theorem 17: The shifts in the cocycle space are Hamiltonian.
The shift # 1 is the following linear combination of Hamiltonian vector fields:
(0, 1, 0, 1/4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
The shift # 2 is the following linear combination of Hamiltonian vector fields:
(0, 0, 0, -1/4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
The shift # 3 is the following linear combination of Hamiltonian vector fields:
(0, 0, 0, 0, 1/2, 0, 0, 0, 0, 0, 0, 0, 0, -1/2, 0, 0, 0, 0, 0, 0, -1/16)
The shift # 4 is the following linear combination of Hamiltonian vector fields:
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1/2, 0, 0, 0, 0, 0, 0, 1/16)
The shift # 5 is the following linear combination of Hamiltonian vector fields:
(0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
The shift # 6 is the following linear combination of Hamiltonian vector fields:
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
The shift # 7 is the following linear combination of Hamiltonian vector fields:
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1/8)
```

Encodings of the other graph pairs

October 1, 2024

[ ]:
```
# In 3D and 4D, we have not only checked the descendants of one particular pair␣
 ↪(the Sunflower pair), but also of the
# other 27 pairs that solve the equation in 2D. The code that should be ran is␣
 ↪exactly as in the 3D and 4D cases of the
# Sunflower pair that has been completely worked out. The only thing that␣
 ↪changed is the input (in terms of the
# encodings/graphs we feed the code to compute with. Below, you can find the␣
 ↪(code generating the) encodings for the ALL 28
# pairs in 3D (the Sunflower is pair G11 and G12) and 5 pairs in 4D. From these␣
 ↪28 pairs, we  find a solution over 5 of these
# pairs (G11 and G12,  G2 and G8, G2 and G12, G7 and G11, and G8 and G11). It is␣
 ↪impossible for there to be a solution in 4D,
# but NOT in 3D over the descendants of the same graphs, so in 4D we only␣
 ↪checked the 5 remaining pairs. From these, we find a
# solution over only 2 pairs (G11 and G12,  and G2 and G12).


#The 3D encodings of pairs
g1_and_g_2_to_3d_graphs="(0,1,4;2,3,5;1,3,6)+(0,1,4;2,3,5;3,4,6)+(0,1,4;2,5,6;
 ↪1,3,6)+(0,1,4;2,5,6;3,4,6)+(0,1,4;1,2,5;1,3,6)+(0,1,4;1,2,5;3,4,6)+(0,1,4;
 ↪2,4,5;1,3,6)+(0,1,4;2,4,5;3,4,6)"

g1_and_g_4_to_3d_graphs="(0,1,4;2,3,5;1,3,6)+(0,1,4;2,3,5;3,4,6)+(0,1,4;2,5,6;
 ↪1,3,6)+(0,1,4;2,5,6;3,4,6)+(0,3,4;2,3,5;1,3,6)+(0,3,4;2,3,5;3,4,6)+(0,3,4;
 ↪2,5,6;1,3,6)+(0,3,4;2,5,6;3,4,6)+(0,4,6;2,3,5;1,3,6)+(0,4,6;2,3,5;
 ↪3,4,6)+(0,4,6;2,5,6;1,3,6)+(0,4,6;2,5,6;3,4,6)"

g1_and_g_9_to_3d_graphs="(0,1,4;2,3,5;1,3,6)+(0,1,4;2,3,5;3,4,6)+(0,1,4;2,5,6;
 ↪1,3,6)+(0,1,4;2,5,6;3,4,6)+(0,2,4;2,3,5;1,2,6)+(0,2,4;2,3,5;1,5,6)+(0,2,4;
 ↪2,3,5;2,4,6)+(0,2,4;2,3,5;4,5,6)+(0,2,4;2,5,6;1,2,6)+(0,2,4;2,5,6;
 ↪1,5,6)+(0,2,4;2,5,6;2,4,6)+(0,2,4;2,5,6;4,5,6)+(0,4,5;2,3,5;1,2,6)+(0,4,5;
 ↪2,3,5;1,5,6)+(0,4,5;2,3,5;2,4,6)+(0,4,5;2,3,5;4,5,6)+(0,4,5;2,5,6;
 ↪1,2,6)+(0,4,5;2,5,6;1,5,6)+(0,4,5;2,5,6;2,4,6)+(0,4,5;2,5,6;4,5,6)"
```

```
g1_and_g_11_to_3d_graphs="(0,1,4;2,3,5;1,3,6)+(0,1,4;2,3,5;3,4,6)+(0,1,4;2,5,6;
 ↪1,3,6)+(0,1,4;2,5,6;3,4,6)+(0,1,4;1,3,5;1,2,6)+(0,1,4;1,3,5;2,4,6)+(0,1,4;
 ↪1,3,5;1,5,6)+(0,1,4;1,3,5;4,5,6)+(0,1,4;3,4,5;1,2,6)+(0,1,4;3,4,5;
 ↪2,4,6)+(0,1,4;3,4,5;1,5,6)+(0,1,4;3,4,5;4,5,6)+(0,1,4;1,5,6;1,2,6)+(0,1,4;
 ↪1,5,6;2,4,6)+(0,1,4;1,5,6;1,5,6)+(0,1,4;1,5,6;4,5,6)+(0,1,4;4,5,6;
 ↪1,2,6)+(0,1,4;4,5,6;2,4,6)+(0,1,4;4,5,6;1,5,6)+(0,1,4;4,5,6;4,5,6)"


g5_and_g_2_to_3d_graphs="(0,2,4;2,3,5;1,3,6)+(0,2,4;2,3,5;3,4,6)+(0,2,4;2,5,6;
 ↪1,3,6)+(0,2,4;2,5,6;3,4,6)+(0,4,5;2,3,5;1,3,6)+(0,4,5;2,3,5;3,4,6)+(0,4,5;
 ↪2,5,6;1,3,6)+(0,4,5;2,5,6;3,4,

g5_and_g_4_to_3d_graphs="(0,2,4;2,3,5;1,3,6)+(0,2,4;2,3,5;3,4,6)+(0,2,4;2,5,6;
 ↪1,3,6)+(0,2,4;2,5,6;3,4,6)+(0,4,5;2,3,5;1,3,6)+(0,4,5;2,3,5;3,4,6)+(0,4,5;
 ↪2,5,6;1,3,6)+(0,4,5;2,5,6;3,4,6)+(0,3,4;2,3,5;1,3,6)+(0,3,4;2,3,5;
 ↪3,4,6)+(0,3,4;2,5,6;1,3,6)+(0,3,4;2,5,6;3,4,6)+(0,4,6;2,3,5;1,3,6)+(0,4,6;
 ↪2,3,5;3,4,6)+(0,4,6;2,5,6;1,3,6)+(0,4,6;2,5,6;3,4,6)"

g5_and_g_9_to_3d_graphs="(0,2,4;2,3,5;1,3,6)+(0,2,4;2,3,5;3,4,6)+(0,2,4;2,5,6;
 ↪1,3,6)+(0,2,4;2,5,6;3,4,6)+(0,4,5;2,3,5;1,3,6)+(0,4,5;2,3,5;3,4,6)+(0,4,5;
 ↪2,5,6;1,3,6)+(0,4,5;2,5,6;3,4,6)+(0,2,4;2,3,5;1,2,6)+(0,2,4;2,3,5;
 ↪1,5,6)+(0,2,4;2,3,5;2,4,6)+(0,2,4;2,3,5;4,5,6)+(0,2,4;2,5,6;1,2,6)+(0,2,4;
 ↪2,5,6;1,5,6)+(0,2,4;2,5,6;2,4,6)+(0,2,4;2,5,6;4,5,6)+(0,4,5;2,3,5;
 ↪1,2,6)+(0,4,5;2,3,5;1,5,6)+(0,4,5;2,3,5;2,4,6)+(0,4,5;2,3,5;4,5,6)+(0,4,5;
 ↪2,5,6;1,2,6)+(0,4,5;2,5,6;1,5,6)+(0,4,5;2,5,6;2,4,6)+(0,4,5;2,5,6;4,5,6)"

g5_and_g11_to_3d_graphs="(0,2,4;2,3,5;1,3,6)+(0,2,4;2,3,5;3,4,6)+(0,2,4;2,5,6;
 ↪1,3,6)+(0,2,4;2,5,6;3,4,6)+(0,4,5;2,3,5;1,3,6)+(0,4,5;2,3,5;3,4,6)+(0,4,5;
 ↪2,5,6;1,3,6)+(0,4,5;2,5,6;3,4,6)+(0,1,4;1,3,5;1,2,6)+(0,1,4;1,3,5;
 ↪2,4,6)+(0,1,4;1,3,5;1,5,6)+(0,1,4;1,3,5;4,5,6)+(0,1,4;3,4,5;1,2,6)+(0,1,4;
 ↪3,4,5;2,4,6)+(0,1,4;3,4,5;1,5,6)+(0,1,4;3,4,5;4,5,6)+(0,1,4;1,5,6;
 ↪1,2,6)+(0,1,4;1,5,6;2,4,6)+(0,1,4;1,5,6;1,5,6)+(0,1,4;1,5,6;4,5,6)+(0,1,4;
 ↪4,5,6;1,2,6)+(0,1,4;4,5,6;2,4,6)+(0,1,4;4,5,6;1,5,6)+(0,1,4;4,5,6;4,5,6)"


g6_and_g_2_to_3d_graphs="(0,3,4;1,2,5; 1,3,6)+(0,3,4;1,2,5;3,4,6)+(0,3,4;2,4,5;
 ↪1,3,6)+(0,3,4;2,4,5;3,4,6)+(0,4,6;1,2,5; 1,3,6)+(0,4,6;1,2,5;3,4,6)+(0,4,6;
 ↪2,4,5;1,3,6)+(0,4,6;2,4,5;3,4,6)+(0,1,4;1,2,5;1,3,6)+(0,1,4;1,2,5;
 ↪3,4,6)+(0,1,4;2,4,5;1,3,6)+(0,1,4;2,4,5;3,4,6)"

g6_and_g_4_to_3d_graphs="(0,3,4;1,2,5; 1,3,6)+(0,3,4;1,2,5;3,4,6)+(0,3,4;2,4,5;
 ↪1,3,6)+(0,3,4;2,4,5;3,4,6)+(0,4,6;1,2,5; 1,3,6)+(0,4,6;1,2,5;3,4,6)+(0,4,6;
 ↪2,4,5;1,3,6)+(0,4,6;2,4,5;3,4,6)+(0,3,4;2,3,5;1,3,6)+(0,3,4;2,3,5;
 ↪3,4,6)+(0,3,4;2,5,6;1,3,6)+(0,3,4;2,5,6;3,4,6)+(0,4,6;2,3,5;1,3,6)+(0,4,6;
 ↪2,3,5;3,4,6)+(0,4,6;2,5,6;1,3,6)+(0,4,6;2,5,6;3,4,6)"
```

```
g6_and_g_9_to_3d_graphs="(0,3,4;1,2,5; 1,3,6)+(0,3,4;1,2,5;3,4,6)+(0,3,4;2,4,5;
 ↪1,3,6)+(0,3,4;2,4,5;3,4,6)+(0,4,6;1,2,5; 1,3,6)+(0,4,6;1,2,5;3,4,6)+(0,4,6;
 ↪2,4,5;1,3,6)+(0,4,6;2,4,5;3,4,6)+(0,2,4;2,3,5;1,2,6)+(0,2,4;2,3,5;
 ↪1,5,6)+(0,2,4;2,3,5;2,4,6)+(0,2,4;2,3,5;4,5,6)+(0,2,4;2,5,6;1,2,6)+(0,2,4;
 ↪2,5,6;1,5,6)+(0,2,4;2,5,6;2,4,6)+(0,2,4;2,5,6;4,5,6)+(0,4,5;2,3,5;
 ↪1,2,6)+(0,4,5;2,3,5;1,5,6)+(0,4,5;2,3,5;2,4,6)+(0,4,5;2,3,5;4,5,6)+(0,4,5;
 ↪2,5,6;1,2,6)+(0,4,5;2,5,6;1,5,6)+(0,4,5;2,5,6;2,4,6)+(0,4,5;2,5,6;4,5,6)"

g6_and_g_11_to_3d_graphs="(0,3,4;1,2,5; 1,3,6)+(0,3,4;1,2,5;3,4,6)+(0,3,4;2,4,5;
 ↪1,3,6)+(0,3,4;2,4,5;3,4,6)+(0,4,6;1,2,5; 1,3,6)+(0,4,6;1,2,5;3,4,6)+(0,4,6;
 ↪2,4,5;1,3,6)+(0,4,6;2,4,5;3,4,6)+(0,1,4;1,3,5;1,2,6)+(0,1,4;1,3,5;
 ↪2,4,6)+(0,1,4;1,3,5;1,5,6)+(0,1,4;1,3,5;4,5,6)+(0,1,4;3,4,5;1,2,6)+(0,1,4;
 ↪3,4,5;2,4,6)+(0,1,4;3,4,5;1,5,6)+(0,1,4;3,4,5;4,5,6)+(0,1,4;1,5,6;
 ↪1,2,6)+(0,1,4;1,5,6;2,4,6)+(0,1,4;1,5,6;1,5,6)+(0,1,4;1,5,6;4,5,6)+(0,1,4;
 ↪4,5,6;1,2,6)+(0,1,4;4,5,6;2,4,6)+(0,1,4;4,5,6;1,5,6)+(0,1,4;4,5,6;4,5,6)"


g7_and_g_2_to_3d_graphs="(0,3,4;2,3,5;1,2,6)+(0,3,4;2,3,5;2,4,6)+(0,3,4;2,3,5;
 ↪1,5,6)+(0,3,4;2,3,5;4,5,6)+(0,3,4;2,5,6;1,2,6)+(0,3,4;2,5,6;2,4,6)+(0,3,4;
 ↪2,5,6;1,5,6)+(0,3,4;2,5,6;4,5,6)+(0,4,6;2,3,5;1,2,6)+(0,4,6;2,3,5;
 ↪2,4,6)+(0,4,6;2,3,5;1,5,6)+(0,4,6;2,3,5;4,5,6)+(0,4,6;2,5,6;1,2,6)+(0,4,6;
 ↪2,5,6;2,4,6)+(0,4,6;2,5,6;1,5,6)+(0,4,6;2,5,6;4,5,6)+(0,1,4;1,2,5;
 ↪1,3,6)+(0,1,4;1,2,5;3,4,6)+(0,1,4;2,4,5;1,3,6)+(0,1,4;2,4,5;3,4,6)"

g7_and_g_4_to_3d_graphs="(0,3,4;2,3,5;1,2,6)+(0,3,4;2,3,5;2,4,6)+(0,3,4;2,3,5;
 ↪1,5,6)+(0,3,4;2,3,5;4,5,6)+(0,3,4;2,5,6;1,2,6)+(0,3,4;2,5,6;2,4,6)+(0,3,4;
 ↪2,5,6;1,5,6)+(0,3,4;2,5,6;4,5,6)+(0,4,6;2,3,5;1,2,6)+(0,4,6;2,3,5;
 ↪2,4,6)+(0,4,6;2,3,5;1,5,6)+(0,4,6;2,3,5;4,5,6)+(0,4,6;2,5,6;1,2,6)+(0,4,6;
 ↪2,5,6;2,4,6)+(0,4,6;2,5,6;1,5,6)+(0,4,6;2,5,6;4,5,6)+(0,3,4;2,3,5;
 ↪1,3,6)+(0,3,4;2,3,5;3,4,6)+(0,3,4;2,5,6;1,3,6)+(0,3,4;2,5,6;3,4,6)+(0,4,6;
 ↪2,3,5;1,3,6)+(0,4,6;2,3,5;3,4,6)+(0,4,6;2,5,6;1,3,6)+(0,4,6;2,5,6;3,4,6)"

g7_and_g_9_to_3d_graphs="(0,3,4;2,3,5;1,2,6)+(0,3,4;2,3,5;2,4,6)+(0,3,4;2,3,5;
 ↪1,5,6)+(0,3,4;2,3,5;4,5,6)+(0,3,4;2,5,6;1,2,6)+(0,3,4;2,5,6;2,4,6)+(0,3,4;
 ↪2,5,6;1,5,6)+(0,3,4;2,5,6;4,5,6)+(0,4,6;2,3,5;1,2,6)+(0,4,6;2,3,5;
 ↪2,4,6)+(0,4,6;2,3,5;1,5,6)+(0,4,6;2,3,5;4,5,6)+(0,4,6;2,5,6;1,2,6)+(0,4,6;
 ↪2,5,6;2,4,6)+(0,4,6;2,5,6;1,5,6)+(0,4,6;2,5,6;4,5,6)+(0,2,4;2,3,5;
 ↪1,2,6)+(0,2,4;2,3,5;1,5,6)+(0,2,4;2,3,5;2,4,6)+(0,2,4;2,3,5;4,5,6)+(0,2,4;
 ↪2,5,6;1,2,6)+(0,2,4;2,5,6;1,5,6)+(0,2,4;2,5,6;2,4,6)+(0,2,4;2,5,6;
 ↪4,5,6)+(0,4,5;2,3,5;1,2,6)+(0,4,5;2,3,5;1,5,6)+(0,4,5;2,3,5;2,4,6)+(0,4,5;
 ↪2,3,5;4,5,6)+(0,4,5;2,5,6;1,2,6)+(0,4,5;2,5,6;1,5,6)+(0,4,5;2,5,6;
 ↪2,4,6)+(0,4,5;2,5,6;4,5,6)"
```

```
g7_and_g_11_to_3d_graphs="(0,3,4;2,3,5;1,2,6)+(0,3,4;2,3,5;2,4,6)+(0,3,4;2,3,5;
 ↪1,5,6)+(0,3,4;2,3,5;4,5,6)+(0,3,4;2,5,6;1,2,6)+(0,3,4;2,5,6;2,4,6)+(0,3,4;
 ↪2,5,6;1,5,6)+(0,3,4;2,5,6;4,5,6)+(0,4,6;2,3,5;1,2,6)+(0,4,6;2,3,5;
 ↪2,4,6)+(0,4,6;2,3,5;1,5,6)+(0,4,6;2,3,5;4,5,6)+(0,4,6;2,5,6;1,2,6)+(0,4,6;
 ↪2,5,6;2,4,6)+(0,4,6;2,5,6;1,5,6)+(0,4,6;2,5,6;4,5,6)+(0,1,4;1,3,5;
 ↪1,2,6)+(0,1,4;1,3,5;2,4,6)+(0,1,4;1,3,5;1,5,6)+(0,1,4;1,3,5;4,5,6)+(0,1,4;
 ↪3,4,5;1,2,6)+(0,1,4;3,4,5;2,4,6)+(0,1,4;3,4,5;1,5,6)+(0,1,4;3,4,5;
 ↪4,5,6)+(0,1,4;1,5,6;1,2,6)+(0,1,4;1,5,6;2,4,6)+(0,1,4;1,5,6;1,5,6)+(0,1,4;
 ↪1,5,6;4,5,6)+(0,1,4;4,5,6;1,2,6)+(0,1,4;4,5,6;2,4,6)+(0,1,4;4,5,6;
 ↪1,5,6)+(0,1,4;4,5,6;4,5,6)"


g8_and_g_2_to_3d_graphs="(0,3,4;1,2,5;1,2,6)+(0,3,4;1,2,5;1,5,6)+(0,3,4;1,2,5;
 ↪2,4,6)+(0,3,4;1,2,5;4,5,6)+(0,3,4;2,4,5;1,2,6)+(0,3,4;2,4,5;1,5,6)+(0,3,4;
 ↪2,4,5;2,4,6)+(0,3,4;2,4,5;4,5,6)+(0,4,6;1,2,5;1,2,6)+(0,4,6;1,2,5;
 ↪1,5,6)+(0,4,6;1,2,5;2,4,6)+(0,4,6;1,2,5;4,5,6)+(0,4,6;2,4,5;1,2,6)+(0,4,6;
 ↪2,4,5;1,5,6)+(0,4,6;2,4,5;2,4,6)+(0,4,6;2,4,5;4,5,6)+(0,1,4;1,2,5;
 ↪1,3,6)+(0,1,4;1,2,5;3,4,6)+(0,1,4;2,4,5;1,3,6)+(0,1,4;2,4,5;3,4,6)"

g8_and_g_4_to_3d_graphs="(0,3,4;1,2,5;1,2,6)+(0,3,4;1,2,5;1,5,6)+(0,3,4;1,2,5;
 ↪2,4,6)+(0,3,4;1,2,5;4,5,6)+(0,3,4;2,4,5;1,2,6)+(0,3,4;2,4,5;1,5,6)+(0,3,4;
 ↪2,4,5;2,4,6)+(0,3,4;2,4,5;4,5,6)+(0,4,6;1,2,5;1,2,6)+(0,4,6;1,2,5;
 ↪1,5,6)+(0,4,6;1,2,5;2,4,6)+(0,4,6;1,2,5;4,5,6)+(0,4,6;2,4,5;1,2,6)+(0,4,6;
 ↪2,4,5;1,5,6)+(0,4,6;2,4,5;2,4,6)+(0,4,6;2,4,5;4,5,6)+(0,3,4;2,3,5;
 ↪1,3,6)+(0,3,4;2,3,5;3,4,6)+(0,3,4;2,5,6;1,3,6)+(0,3,4;2,5,6;3,4,6)+(0,4,6;
 ↪2,3,5;1,3,6)+(0,4,6;2,3,5;3,4,6)+(0,4,6;2,5,6;1,3,6)+(0,4,6;2,5,6;3,4,6)"

g8_and_g_9_to_3d_graphs="(0,3,4;1,2,5;1,2,6)+(0,3,4;1,2,5;1,5,6)+(0,3,4;1,2,5;
 ↪2,4,6)+(0,3,4;1,2,5;4,5,6)+(0,3,4;2,4,5;1,2,6)+(0,3,4;2,4,5;1,5,6)+(0,3,4;
 ↪2,4,5;2,4,6)+(0,3,4;2,4,5;4,5,6)+(0,4,6;1,2,5;1,2,6)+(0,4,6;1,2,5;
 ↪1,5,6)+(0,4,6;1,2,5;2,4,6)+(0,4,6;1,2,5;4,5,6)+(0,4,6;2,4,5;1,2,6)+(0,4,6;
 ↪2,4,5;1,5,6)+(0,4,6;2,4,5;2,4,6)+(0,4,6;2,4,5;4,5,6)+(0,2,4;2,3,5;
 ↪1,2,6)+(0,2,4;2,3,5;1,5,6)+(0,2,4;2,3,5;2,4,6)+(0,2,4;2,3,5;4,5,6)+(0,2,4;
 ↪2,5,6;1,2,6)+(0,2,4;2,5,6;1,5,6)+(0,2,4;2,5,6;2,4,6)+(0,2,4;2,5,6;
 ↪4,5,6)+(0,4,5;2,3,5;1,2,6)+(0,4,5;2,3,5;1,5,6)+(0,4,5;2,3,5;2,4,6)+(0,4,5;
 ↪2,3,5;4,5,6)+(0,4,5;2,5,6;1,2,6)+(0,4,5;2,5,6;1,5,6)+(0,4,5;2,5,6;
 ↪2,4,6)+(0,4,5;2,5,6;4,5,6)"
```

```
g8_and_g_11_to_3d_graphs="(0,3,4;1,2,5;1,2,6)+(0,3,4;1,2,5;1,5,6)+(0,3,4;1,2,5;
 ↪2,4,6)+(0,3,4;1,2,5;4,5,6)+(0,3,4;2,4,5;1,2,6)+(0,3,4;2,4,5;1,5,6)+(0,3,4;
 ↪2,4,5;2,4,6)+(0,3,4;2,4,5;4,5,6)+(0,4,6;1,2,5;1,2,6)+(0,4,6;1,2,5;
 ↪1,5,6)+(0,4,6;1,2,5;2,4,6)+(0,4,6;1,2,5;4,5,6)+(0,4,6;2,4,5;1,2,6)+(0,4,6;
 ↪2,4,5;1,5,6)+(0,4,6;2,4,5;2,4,6)+(0,4,6;2,4,5;4,5,6)+(0,1,4;1,3,5;
 ↪1,2,6)+(0,1,4;1,3,5;2,4,6)+(0,1,4;1,3,5;1,5,6)+(0,1,4;1,3,5;4,5,6)+(0,1,4;
 ↪3,4,5;1,2,6)+(0,1,4;3,4,5;2,4,6)+(0,1,4;3,4,5;1,5,6)+(0,1,4;3,4,5;
 ↪4,5,6)+(0,1,4;1,5,6;1,2,6)+(0,1,4;1,5,6;2,4,6)+(0,1,4;1,5,6;1,5,6)+(0,1,4;
 ↪1,5,6;4,5,6)+(0,1,4;4,5,6;1,2,6)+(0,1,4;4,5,6;2,4,6)+(0,1,4;4,5,6;
 ↪1,5,6)+(0,1,4;4,5,6;4,5,6)"


g12_and_g_2_to_3d_graphs="(0,3,4;1,3,5;1,2,6)+(0,3,4;1,3,5;1,5,6)+(0,3,4;1,3,5;
 ↪2,4,6)+(0,3,4;1,3,5;4,5,6)+(0,3,4;1,5,6;1,2,6)+(0,3,4;1,5,6;1,5,6)+(0,3,4;
 ↪1,5,6;2,4,6)+(0,3,4;1,5,6;4,5,6)+(0,3,4;3,4,5;1,2,6)+(0,3,4;3,4,5;
 ↪1,5,6)+(0,3,4;3,4,5;2,4,6)+(0,3,4;3,4,5;4,5,6)+(0,3,4;4,5,6;1,2,6)+(0,3,4;
 ↪4,5,6;1,5,6)+(0,3,4;4,5,6;2,4,6)+(0,3,4;4,5,6;4,5,6)+(0,4,6;1,3,5;
 ↪1,2,6)+(0,4,6;1,3,5;1,5,6)+(0,4,6;1,3,5;2,4,6)+(0,4,6;1,3,5;4,5,6)+(0,4,6;
 ↪1,5,6;1,2,6)+(0,4,6;1,5,6;1,5,6)+(0,4,6;1,5,6;2,4,6)+(0,4,6;1,5,6;
 ↪4,5,6)+(0,4,6;3,4,5;1,2,6)+(0,4,6;3,4,5;1,5,6)+(0,4,6;3,4,5;2,4,6)+(0,4,6;
 ↪3,4,5;4,5,6)+(0,4,6;4,5,6;1,2,6)+(0,4,6;4,5,6;1,5,6)+(0,4,6;4,5,6;
 ↪2,4,6)+(0,4,6;4,5,6;4,5,6)+(0,1,4;1,2,5;1,3,6)+(0,1,4;1,2,5;3,4,6)+(0,1,4;
 ↪2,4,5;1,3,6)+(0,1,4;2,4,5;3,4,6)"

g12_and_g_4_to_3d_graphs="(0,3,4;1,3,5;1,2,6)+(0,3,4;1,3,5;1,5,6)+(0,3,4;1,3,5;
 ↪2,4,6)+(0,3,4;1,3,5;4,5,6)+(0,3,4;1,5,6;1,2,6)+(0,3,4;1,5,6;1,5,6)+(0,3,4;
 ↪1,5,6;2,4,6)+(0,3,4;1,5,6;4,5,6)+(0,3,4;3,4,5;1,2,6)+(0,3,4;3,4,5;
 ↪1,5,6)+(0,3,4;3,4,5;2,4,6)+(0,3,4;3,4,5;4,5,6)+(0,3,4;4,5,6;1,2,6)+(0,3,4;
 ↪4,5,6;1,5,6)+(0,3,4;4,5,6;2,4,6)+(0,3,4;4,5,6;4,5,6)+(0,4,6;1,3,5;
 ↪1,2,6)+(0,4,6;1,3,5;1,5,6)+(0,4,6;1,3,5;2,4,6)+(0,4,6;1,3,5;4,5,6)+(0,4,6;
 ↪1,5,6;1,2,6)+(0,4,6;1,5,6;1,5,6)+(0,4,6;1,5,6;2,4,6)+(0,4,6;1,5,6;
 ↪4,5,6)+(0,4,6;3,4,5;1,2,6)+(0,4,6;3,4,5;1,5,6)+(0,4,6;3,4,5;2,4,6)+(0,4,6;
 ↪3,4,5;4,5,6)+(0,4,6;4,5,6;1,2,6)+(0,4,6;4,5,6;1,5,6)+(0,4,6;4,5,6;
 ↪2,4,6)+(0,4,6;4,5,6;4,5,6)+(0,3,4;2,3,5;1,3,6)+(0,3,4;2,3,5;3,4,6)+(0,3,4;
 ↪2,5,6;1,3,6)+(0,3,4;2,5,6;3,4,6)+(0,4,6;2,3,5;1,3,6)+(0,4,6;2,3,5;
 ↪3,4,6)+(0,4,6;2,5,6;1,3,6)+(0,4,6;2,5,6;3,4,6)"
```

```
g12_and_g_9_to_3d_graphs="(0,3,4;1,3,5;1,2,6)+(0,3,4;1,3,5;1,5,6)+(0,3,4;1,3,5;
 ↪2,4,6)+(0,3,4;1,3,5;4,5,6)+(0,3,4;1,5,6;1,2,6)+(0,3,4;1,5,6;1,5,6)+(0,3,4;
 ↪1,5,6;2,4,6)+(0,3,4;1,5,6;4,5,6)+(0,3,4;3,4,5;1,2,6)+(0,3,4;3,4,5;
 ↪1,5,6)+(0,3,4;3,4,5;2,4,6)+(0,3,4;3,4,5;4,5,6)+(0,3,4;4,5,6;1,2,6)+(0,3,4;
 ↪4,5,6;1,5,6)+(0,3,4;4,5,6;2,4,6)+(0,3,4;4,5,6;4,5,6)+(0,4,6;1,3,5;
 ↪1,2,6)+(0,4,6;1,3,5;1,5,6)+(0,4,6;1,3,5;2,4,6)+(0,4,6;1,3,5;4,5,6)+(0,4,6;
 ↪1,5,6;1,2,6)+(0,4,6;1,5,6;1,5,6)+(0,4,6;1,5,6;2,4,6)+(0,4,6;1,5,6;
 ↪4,5,6)+(0,4,6;3,4,5;1,2,6)+(0,4,6;3,4,5;1,5,6)+(0,4,6;3,4,5;2,4,6)+(0,4,6;
 ↪3,4,5;4,5,6)+(0,4,6;4,5,6;1,2,6)+(0,4,6;4,5,6;1,5,6)+(0,4,6;4,5,6;
 ↪2,4,6)+(0,4,6;4,5,6;4,5,6)+(0,2,4;2,3,5;1,2,6)+(0,2,4;2,3,5;1,5,6)+(0,2,4;
 ↪2,3,5;2,4,6)+(0,2,4;2,3,5;4,5,6)+(0,2,4;2,5,6;1,2,6)+(0,2,4;2,5,6;
 ↪1,5,6)+(0,2,4;2,5,6;2,4,6)+(0,2,4;2,5,6;4,5,6)+(0,4,5;2,3,5;1,2,6)+(0,4,5;
 ↪2,3,5;1,5,6)+(0,4,5;2,3,5;2,4,6)+(0,4,5;2,3,5;4,5,6)+(0,4,5;2,5,6;
 ↪1,2,6)+(0,4,5;2,5,6;1,5,6)+(0,4,5;2,5,6;2,4,6)+(0,4,5;2,5,6;4,5,6)"

g12_and_g_11_to_3d_graphs="(0,3,4;1,3,5;1,2,6)+(0,3,4;1,3,5;1,5,6)+(0,3,4;1,3,5;
 ↪2,4,6)+(0,3,4;1,3,5;4,5,6)+(0,3,4;1,5,6;1,2,6)+(0,3,4;1,5,6;1,5,6)+(0,3,4;
 ↪1,5,6;2,4,6)+(0,3,4;1,5,6;4,5,6)+(0,3,4;3,4,5;1,2,6)+(0,3,4;3,4,5;
 ↪1,5,6)+(0,3,4;3,4,5;2,4,6)+(0,3,4;3,4,5;4,5,6)+(0,3,4;4,5,6;1,2,6)+(0,3,4;
 ↪4,5,6;1,5,6)+(0,3,4;4,5,6;2,4,6)+(0,3,4;4,5,6;4,5,6)+(0,4,6;1,3,5;
 ↪1,2,6)+(0,4,6;1,3,5;1,5,6)+(0,4,6;1,3,5;2,4,6)+(0,4,6;1,3,5;4,5,6)+(0,4,6;
 ↪1,5,6;1,2,6)+(0,4,6;1,5,6;1,5,6)+(0,4,6;1,5,6;2,4,6)+(0,4,6;1,5,6;
 ↪4,5,6)+(0,4,6;3,4,5;1,2,6)+(0,4,6;3,4,5;1,5,6)+(0,4,6;3,4,5;2,4,6)+(0,4,6;
 ↪3,4,5;4,5,6)+(0,4,6;4,5,6;1,2,6)+(0,4,6;4,5,6;1,5,6)+(0,4,6;4,5,6;
 ↪2,4,6)+(0,4,6;4,5,6;4,5,6)+(0,1,4;1,3,5;1,2,6)+(0,1,4;1,3,5;2,4,6)+(0,1,4;
 ↪1,3,5;1,5,6)+(0,1,4;1,3,5;4,5,6)+(0,1,4;3,4,5;1,2,6)+(0,1,4;3,4,5;
 ↪2,4,6)+(0,1,4;3,4,5;1,5,6)+(0,1,4;3,4,5;4,5,6)+(0,1,4;1,5,6;1,2,6)+(0,1,4;
 ↪1,5,6;2,4,6)+(0,1,4;1,5,6;1,5,6)+(0,1,4;1,5,6;4,5,6)+(0,1,4;4,5,6;
 ↪1,2,6)+(0,1,4;4,5,6;2,4,6)+(0,1,4;4,5,6;1,5,6)+(0,1,4;4,5,6;4,5,6)"


g13_and_g_2_to_3d_graphs="(0,1,4;1,3,5;2,3,6)+(0,1,4;1,3,5;3,5,6)+(0,1,4;1,5,6;
 ↪2,3,6)+(0,1,4;1,5,6;3,5,6)+(0,1,4;3,4,5;2,3,6)+(0,1,4;3,4,5;3,5,6)+(0,1,4;
 ↪4,5,6;2,3,6)+(0,1,4;4,5,6;3,5,6)+(0,1,4;1,2,5;1,3,6)+(0,1,4;1,2,5;
 ↪3,4,6)+(0,1,4;2,4,5;1,3,6)+(0,1,4;2,4,5;3,4,6)"

g13_and_g_4_to_3d_graphs="(0,1,4;1,3,5;2,3,6)+(0,1,4;1,3,5;3,5,6)+(0,1,4;1,5,6;
 ↪2,3,6)+(0,1,4;1,5,6;3,5,6)+(0,1,4;3,4,5;2,3,6)+(0,1,4;3,4,5;3,5,6)+(0,1,4;
 ↪4,5,6;2,3,6)+(0,1,4;4,5,6;3,5,6)+(0,3,4;2,3,5;1,3,6)+(0,3,4;2,3,5;
 ↪3,4,6)+(0,3,4;2,5,6;1,3,6)+(0,3,4;2,5,6;3,4,6)+(0,4,6;2,3,5;1,3,6)+(0,4,6;
 ↪2,3,5;3,4,6)+(0,4,6;2,5,6;1,3,6)+(0,4,6;2,5,6;3,4,6)"
```

```
g13_and_g_9_to_3d_graphs="(0,1,4;1,3,5;2,3,6)+(0,1,4;1,3,5;3,5,6)+(0,1,4;1,5,6;
 ↪2,3,6)+(0,1,4;1,5,6;3,5,6)+(0,1,4;3,4,5;2,3,6)+(0,1,4;3,4,5;3,5,6)+(0,1,4;
 ↪4,5,6;2,3,6)+(0,1,4;4,5,6;3,5,6)+(0,2,4;2,3,5;1,2,6)+(0,2,4;2,3,5;
 ↪1,5,6)+(0,2,4;2,3,5;2,4,6)+(0,2,4;2,3,5;4,5,6)+(0,2,4;2,5,6;1,2,6)+(0,2,4;
 ↪2,5,6;1,5,6)+(0,2,4;2,5,6;2,4,6)+(0,2,4;2,5,6;4,5,6)+(0,4,5;2,3,5;
 ↪1,2,6)+(0,4,5;2,3,5;1,5,6)+(0,4,5;2,3,5;2,4,6)+(0,4,5;2,3,5;4,5,6)+(0,4,5;
 ↪2,5,6;1,2,6)+(0,4,5;2,5,6;1,5,6)+(0,4,5;2,5,6;2,4,6)+(0,4,5;2,5,6;4,5,6)"

g13_and_g_11_to_3d_graphs="(0,1,4;1,3,5;2,3,6)+(0,1,4;1,3,5;3,5,6)+(0,1,4;1,5,6;
 ↪2,3,6)+(0,1,4;1,5,6;3,5,6)+(0,1,4;3,4,5;2,3,6)+(0,1,4;3,4,5;3,5,6)+(0,1,4;
 ↪4,5,6;2,3,6)+(0,1,4;4,5,6;3,5,6)+(0,1,4;1,3,5;1,2,6)+(0,1,4;1,3,5;
 ↪2,4,6)+(0,1,4;1,3,5;1,5,6)+(0,1,4;1,3,5;4,5,6)+(0,1,4;3,4,5;1,2,6)+(0,1,4;
 ↪3,4,5;2,4,6)+(0,1,4;3,4,5;1,5,6)+(0,1,4;3,4,5;4,5,6)+(0,1,4;1,5,6;
 ↪1,2,6)+(0,1,4;1,5,6;2,4,6)+(0,1,4;1,5,6;1,5,6)+(0,1,4;1,5,6;4,5,6)+(0,1,4;
 ↪4,5,6;1,2,6)+(0,1,4;4,5,6;2,4,6)+(0,1,4;4,5,6;1,5,6)+(0,1,4;4,5,6;4,5,6)"


# Generating 4D encodings of the 5 remaining pairs:
import itertools
# Descendants graph 2.
for (i,j) in itertools.product([1,4,7],[1,4,7]):
    X_graph_encodings.append((0,1,4,7,i,2,5,8,j,3,6,9))

# Descendants graph 7
for (i,j,k,l) in itertools.product([3,6,9],[3,6,9],[1,4,7],[2,5,8]):
    X_graph_encodings.append((0,i,4,7,2,j,5,8,k,l,6,9))

# Descendants of graph 8.
for (i,j,k,l) in itertools.product([3,6,9],[1,4,7],[1,4,7],[2,5,8]):
    X_graph_encodings.append((0,i,4,7,j,2,5,8,k,l,6,9))

# Descendants graph 11.
for (i,j,k,l) in itertools.product([1,4,7],[3,6,9],[1,4,7],[2,5,8]):
    X_graph_encodings.append((0,1,4,7,i,j,5,8,k,l,6,9))

# descendants graph 12.
for (i,j,k,l,m) in itertools.product ([3,6,9],[1,4,7],[3,6,9],[1,4,7],[2,5,8]):
    X_graph_encodings.append((0,i,4,7,j,k,5,8,l,m,6,9))
```

Finding $\vec{X}_{2D}^{\gamma_5}$ (Brute-force method)

November 23, 2024

[1]:
```
# We import the following (see https://github.com/rburing/gcaops) to be able to␣
↪run the code.
from gcaops.graph.formality_graph import FormalityGraph
from gcaops.algebra.differential_polynomial_ring import␣
↪DifferentialPolynomialRing
from gcaops.algebra.superfunction_algebra import SuperfunctionAlgebra
from gcaops.graph.undirected_graph_complex import UndirectedGraphComplex
from gcaops.graph.directed_graph_complex import DirectedGraphComplex
import itertools
from multiprocessing import Pool

# Depending on your machine, you want to change the number of processors.
ProcessorNumber = 14

# We create all the Kontsevich graphs on 5 vertices and 1 sink in 2D.
encodings=[]
for i1 in [1,2,3,4,5]:
    for index_choice_1 in itertools.product(itertools.combinations(range(5),␣
↪int(2))):
        for index_choice_2 in itertools.product(itertools.combinations(range(5),␣
↪int(2))):
            for index_choice_3 in itertools.product(itertools.
↪combinations(range(5), int(2))):
                for index_choice_4 in itertools.product(itertools.
↪combinations(range(5), int(2))):
                    encodings.append((0, i1, index_choice_1[0][0]+1,␣
↪index_choice_1[0][1]+1, index_choice_2[0][0]+1, index_choice_2[0][1]+1,␣
↪index_choice_3[0][0]+1, index_choice_3[0][1]+1, index_choice_4[0][0]+1,␣
↪index_choice_4[0][1]+1))

# To move from the encodings of the graphs to actual graphs, we use the next␣
↪function. The function splits the
# encoding by vertex via ;, and then the target vertices by ,. A graph is␣
↪returned on 5 vertices, 1 sink, and with edges
# (origin vertex, target vertex).
```

```
def encoding_to_graph(encoding):
    targets = [encoding[0:2], encoding[2:4], encoding[4:6], encoding[6:8],␣
 ↪encoding[8:10]]
    edges = sum([[(k+1,v) for v in t] for (k,t) in enumerate(targets)], [])
    return FormalityGraph(1, 5, edges)

# We compute the corresponding graphs
graphs= []
for encoding in encodings:
    graphs.append(encoding_to_graph(encoding))
print('We have', len(graphs), 'graphs.\n')

# Let's see how many are actually non-isomorphic
graphs_iso={}
for g in graphs:
    h=tuple(g.canonical_form(aerial_vertex_partition=[[1],[2,3,4,5]]).edges())
    if not h in graphs_iso:
        graphs_iso[h]=g
graphs_iso=list(graphs_iso.values())
print('Lemma 3: We have', len(graphs_iso), 'non-isomorphic graphs. \n')

# Create the differential polynomial ring. We are working in 2D, so we only have␣
 ↪even coordinates x,y and the corresponding
# odd coordinates xi[0] and xi[1]. rho is exactly the rho in a 2D Poisson␣
 ↪bracket (P= rho d/dx d/dy). Finally,
# max_differential_orders tells the programme how many times rho can be␣
 ↪differentiated. The maximum is stipulated by the graphs,
# as we cannot have double edges. Thus, the maximum in degree of each vertex is␣
 ↪5. As we will be looking at [[P,X]], we add
# an extra +1 to the differential orders since taking the Schouten bracket with␣
 ↪P introduces an extra derivative.
D2=DifferentialPolynomialRing(QQ,('rho', ), ('x','y'),␣
 ↪max_differential_orders=[5+1])
rho, =D2.fibre_variables()
x,y= D2.base_variables()
even_coords=[x,y]

S2.<xi0,xi1>=SuperfunctionAlgebra(D2, D2.base_variables())
xi=S2.gens()
odd_coords=xi

# We now compute the vector fields corresponding to the graphs. E is the Euler␣
 ↪vector field in the sink (vertex 0), and
# epsilon is the Levi-Civita tensor. Note that we have a Levi-Civita tensor at␣
 ↪each of the vertices 1, 2, 3, 4, 5. We first
```

```
# compute the sign of each term appearing in the formulas, and then compute the␣
 ↪differential polynomial.

E=x*xi[0]+y*xi[1]
epsilon = xi[0]*xi[1]
def graphMultiprocessor(g):
    term = S2.zero()
    for index_choice in itertools.product(itertools.permutations(range(2)),␣
 ↪repeat=5):
        sign = epsilon[index_choice[0]] * epsilon[index_choice[1]] *␣
 ↪epsilon[index_choice[2]] * epsilon[index_choice[3]] * epsilon[index_choice[4]]
        vertex_content = [E, S2(rho), S2(rho), S2(rho), S2(rho), S2(rho)]
        for ((source, target), index) in zip(g.edges(), sum(map(list,␣
 ↪index_choice), [])):
            vertex_content[target] = diff(vertex_content[target],␣
 ↪even_coords[index])
        term += sign * prod(vertex_content)
    return term

X_vector_fields=[]
with Pool(processes=ProcessorNumber) as pool:
    X_vector_fields = list(pool.imap(graphMultiprocessor, graphs))

# We check how many graphs evaluate to 0.
zeros=X_vector_fields.count(0)
print('There are', zeros, 'graphs that evaluate to 0 under the morphism from␣
 ↪graphs to multivectors.\n')

# We remove the zero graphs, as well as duplicate vector fields from the list.␣
 ↪In doing this, the system we consider
# is significantly smaller.
vanishing_graphs=[k for (k,X) in enumerate (X_vector_fields) if X==0]
reversed_vanishing_graphs= vanishing_graphs[::-1]

for i in range(len(reversed_vanishing_graphs)):
    graphs.pop(reversed_vanishing_graphs[i])
    X_vector_fields.pop(reversed_vanishing_graphs[i])
    encodings.pop(reversed_vanishing_graphs[i])

X_vector_fields_duplicates_removed = []
i=-1
j=0
for formula in X_vector_fields:
    i+=1
    if formula not in X_vector_fields_duplicates_removed:
        X_vector_fields_duplicates_removed.append(formula)
```

```
    else:
        graphs.pop(i-j)
        encodings.pop(i-j)
        j+=1

if len(X_vector_fields_duplicates_removed)!=len(graphs) or len(graphs)!
 ↪=len(encodings):
    print('Something went wrong with removing duplicate vector fields')

print('We have', len(X_vector_fields_duplicates_removed), 'graphs left that do␣
 ↪not evaluate into the exact same vector fields.\n')

# The next part is to find out linear relations of the vector fields we just␣
 ↪computed. We look at the monomials that appear
# in each xi[0] and xi[1] parts of the vector field, and store them in␣
 ↪X_monomial_basis.
X_monomial_basis = [set([]) for i in range(2)]
for i in range(2):
    for X in X_vector_fields_duplicates_removed:
        X_monomial_basis[i]|=set(X[i].monomials())
X_monomial_basis=[list(b) for b in X_monomial_basis]
X_monomial_index= [{m:k for k,m in enumerate(b)} for b in X_monomial_basis]
X_monomial_count= sum(len(b) for b in X_monomial_basis)

# Next, we use this monomial basis to create a matrix that identifies each␣
 ↪vector field by the monomials that appear in it.
X_evaluation_matrix= matrix(QQ, X_monomial_count,␣
 ↪len(X_vector_fields_duplicates_removed), sparse=True)
for i in range(len(X_vector_fields_duplicates_removed)):
    v=vector(QQ, X_monomial_count, sparse=True)
    index_shift=0
    for j in range(2):
        f=X_vector_fields_duplicates_removed[i][j]
        for coeff, monomial in zip(f.coefficients(), f.monomials()):
            monomial_index=X_monomial_index[j][monomial]
            v[index_shift+monomial_index]=coeff
        index_shift+=len(X_monomial_basis[j])
    X_evaluation_matrix.set_column(i,v)

# We can now detect linear relation by computing the nullity of this matrix
nullity = X_evaluation_matrix.right_nullity()
print('Claim 4: The remaining vector fields have', nullity, 'linear relations␣
 ↪among themselves.')
print('In other words, we have 22 linearly independent vector fields left.\n')
```

```
# We find a maximally linearly independent subset of these graphs, and save them␣
 ↪in a new list. We also create a list
# of corresponding graph encodings.
pivots2 = X_evaluation_matrix.pivots()
print('A maximal subset of linearly independent graphs is given by:',␣
 ↪list(pivots2),'\n')

lin_ind_vector_fields=[]
for i in pivots2:
    lin_ind_vector_fields.append(X_vector_fields_duplicates_removed[i])

lin_ind_encodings=[]
for i in pivots2:
    lin_ind_encodings.append(encodings[i])

# We now compute the pentagon wheel flow. First, we create the Poisson bivector␣
 ↪P and check that it satisfies [[P,P]]=0
P= rho*epsilon
if P.bracket(P)!=0:
    print('P is not a Poisson bivector. \n')

# Introduce the graph complex, and find the pentagon wheel as a linear␣
 ↪combination of 2 graph on 6 vertices and 10 edges
# in the graph complex. These graphs are unoriented, so we orient them. The next␣
 ↪step is to make sure that the graph is
# built of wedges. This means that there cannot be more than 2 outgoing edges at␣
 ↪each vertex.
# Use fivewheel_oriented_filtered.show() to see the directed graphs we get.
# Finally, the bivector corresponding to the graph cocycle is computed.

GC = UndirectedGraphComplex(QQ, implementation='vector', sparse=True)
fivewheel_cocycle = GC.cohomology_basis(6,10)[0]; fivewheel_cocycle
dGC = DirectedGraphComplex(QQ, implementation='vector')
fivewheel_oriented = dGC(fivewheel_cocycle)
fivewheel_oriented_filtered = fivewheel_oriented.filter(max_out_degree=2)
fivewheel_operation = S2.graph_operation(fivewheel_oriented_filtered)
Q_fivewheel= fivewheel_operation(P,P,P,P,P,P)
print('The pentagon wheel flow in 2D is', Q_fivewheel, '\n')

# Now that we have the pentagon wheel flow, we see if we can create a vector␣
 ↪field X from our previously computed
# graphs-to-vector fields such that [[P,X]]= Q_fivewheel. We first look which␣
 ↪bivectors X_bivectors the vector fields become
# after taking the Schouten bracket with P.
X_bivectors=[]
for X in lin_ind_vector_fields:
```

```
    X_bivectors.append(P.bracket(X))

zero_bivectors = X_bivectors.count(0)
print('There is', zero_bivectors, 'vector field from the linearly independent␣
 ↪vector fields that evaluates to 0 bivector after taking the Schouten bracket␣
 ↪with P.')
print('This vector field that evaluate to 0 is obtained from the graph:')
for (k,X) in enumerate(X_bivectors):
    if X==0:
        print('graph', k+1,)
print()

# Now, we extract the monomials appearing in these bivectors (as well as in␣
 ↪Q_fivewheel).
Q_monomial_basis={}
from itertools import combinations
for i,j in combinations(range(2),2):
    Q_monomial_basis[i,j]=set(Q_fivewheel[i,j].monomials())
    for P_X in X_bivectors:
        Q_monomial_basis[i,j]|= set(P_X[i,j].monomials())

Q_monomial_basis={idx: list(b) for idx, b in Q_monomial_basis.items()}
Q_monomial_index= {idx:{m:k for k,m in enumerate(b)} for idx, b in␣
 ↪Q_monomial_basis.items()}
Q_monomial_count=sum(len(b) for b in Q_monomial_basis.values())

# We create the vector representation of Q_fivewheel in terms of the monomials.
Q_fivewheel_vector= vector(QQ, Q_monomial_count, sparse=True)
index_shift=0
for i,j in Q_monomial_basis:
    for coeff, monomial in Q_fivewheel[i,j]:
        monomial_index= Q_monomial_index[i,j][monomial]
        Q_fivewheel_vector[monomial_index+index_shift]=coeff
    index_shift+=len(Q_monomial_basis[i,j])

# We create the matrix that represents the X_bivectors in terms of the monomials.
X_bivector_evaluation_matrix= matrix(QQ, Q_monomial_count, len(X_bivectors),␣
 ↪sparse=True)
for k in range(len(X_bivectors)):
    P_X=X_bivectors[k]
    v=vector(QQ,Q_monomial_count, sparse=True)
    index_shift=0
    for i,j in Q_monomial_basis:
        for coeff, monomial in P_X[i,j]:
            monomial_index=Q_monomial_index[i,j][monomial]
            v[monomial_index+index_shift]=coeff
        index_shift+=len(Q_monomial_basis[i,j])
```

```
    X_bivector_evaluation_matrix.set_column(k,v)

# We solve the linear system.
X_solution=X_bivector_evaluation_matrix.solve_right(Q_fivewheel_vector)
print('Proposition 5: The solution on vector fields is given by', X_solution,'␣
 ↪\n')

# Finally, we will check the homogeneous system. We look at the kernel of the␣
 ↪X_bivector_evaluation_matrix and filter out the
# nullity of the X_evaluation matrix (restricted to the linealy independent␣
 ↪vector fields)
X_lin_ind_evaluation_matrix= matrix(QQ, X_monomial_count,␣
 ↪len(lin_ind_vector_fields), sparse=True)
for i in range(len(lin_ind_vector_fields)):
    v=vector(QQ, X_monomial_count, sparse=True)
    index_shift=0
    for j in range(2):
        f=lin_ind_vector_fields[i][j]
        for coeff, monomial in zip(f.coefficients(), f.monomials()):
            monomial_index=X_monomial_index[j][monomial]
            v[index_shift+monomial_index]=coeff
        index_shift+=len(X_monomial_basis[j])
    X_lin_ind_evaluation_matrix.set_column(i,v)

X_cocycle_space= X_bivector_evaluation_matrix.right_kernel().
 ↪quotient(X_lin_ind_evaluation_matrix.right_kernel())
X_cocycles=[X_cocycle_space.lift(v) for v in X_cocycle_space.basis()]; X_cocycles
if all(X_bivector_evaluation_matrix*X_cocycle==0 for X_cocycle in X_cocycles)!=␣
 ↪True:
    print('There is an error in the cocycle space.')
print('The space of solutions to the homogeneous system has dimension',␣
 ↪X_cocycle_space.dimension(), )
print ('Proposition 7: These shifts are given by', X_cocycles, '\n')

# Let us write the vector fields as vector fields X such that [[P,X]]=0, rather␣
 ↪than just the combination of graphs.
shifts_formulas = [sum(X_cocycle[j]*lin_ind_vector_fields[j] for j in␣
 ↪range(len(lin_ind_vector_fields))) for X_cocycle in X_cocycles]

# We check that the shifts above are induced by Hamiltonian vector fields.
hamiltonian_encodings=[]
for index_choice_1 in itertools.product(itertools.combinations(range(4),␣
 ↪int(2))):
        for index_choice_2 in itertools.product(itertools.combinations(range(4),␣
 ↪int(2))):
```

```
            for index_choice_3 in itertools.product(itertools.
 ↪combinations(range(4), int(2))):
                for index_choice_4 in itertools.product(itertools.
 ↪combinations(range(4), int(2))):
                    hamiltonian_encodings.append((index_choice_1[0][0],␣
 ↪index_choice_1[0][1], index_choice_2[0][0], index_choice_2[0][1],␣
 ↪index_choice_3[0][0], index_choice_3[0][1], index_choice_4[0][0],␣
 ↪index_choice_4[0][1]))
print('We generate', len(hamiltonian_encodings), 'Hamiltonian graphs.\n')

# We create a new encoding to graph definition, as this is a graph on 4 vertices␣
 ↪and no sink.
def ham_encoding_to_graph(encoding):
    targets = [encoding[0:2], encoding[2:4], encoding[4:6], encoding[6:8]]
    edges = sum([[(k,v) for v in t] for (k,t) in enumerate(targets)], [])
    return FormalityGraph(0, 4, edges)

ham_graphs= []
for encoding in hamiltonian_encodings:
    ham_graphs.append(ham_encoding_to_graph(encoding))

ham_graphs_iso={}
for g in ham_graphs:
    h=tuple(g.canonical_form().edges())
    if not h in ham_graphs_iso:
        ham_graphs_iso[h]=g
ham_graphs_iso=list(ham_graphs_iso.values())
print ('Lemma 8: There are', len(ham_graphs_iso), 'nonisomorphic Hamiltonian␣
 ↪graphs.\n')


# This computation is also slightly different; we do not have the Euler vector␣
 ↪field since we do not have a sink, and as
# we have only 4 copies of the Poisson bivector, the index_choice for the sign␣
 ↪is only on 4 repeats as well. Moreover, there
# are only 4 vertices to give vertex_content to.
hamiltonian_formulas = []

for h in ham_graphs:
    term = S2.zero()
    for index_choice in itertools.product(itertools.permutations(range(2)),␣
 ↪repeat=4):
        # TODO: Check that this way of evaluating is correct.
        sign = epsilon[index_choice[0]] * epsilon[index_choice[1]] *␣
 ↪epsilon[index_choice[2]] * epsilon[index_choice[3]]
        vertex_content = [S2(rho), S2(rho), S2(rho), S2(rho)]
```

```
        for ((source, target), index) in zip(h.edges(), sum(map(list,␣
↪index_choice), [])):
            vertex_content[target] = diff(vertex_content[target],␣
↪even_coords[index])
        term += sign * prod(vertex_content)
    hamiltonian_formulas.append(term)

# Let us check how many of these Hamiltonians are actually 0.
print('We have', hamiltonian_formulas.count(0), 'Hamiltonian graphs that␣
↪evaluate to 0.\n')

# We remove these Hamiltonians.
vanishing_hamiltonians=[k for (k,X) in enumerate (hamiltonian_formulas) if X==0]
reversed_vanishing_hamiltonians_graphs= vanishing_hamiltonians[::-1]

for i in range(len(reversed_vanishing_hamiltonians_graphs)):
    ham_graphs.pop(reversed_vanishing_hamiltonians_graphs[i])
    hamiltonian_formulas.pop(reversed_vanishing_hamiltonians_graphs[i])
    hamiltonian_encodings.pop(reversed_vanishing_hamiltonians_graphs[i])

# We create the vector fields [[P,H]] from the Hamiltonians H.
hamiltonian_vector_fields= []
for formula in hamiltonian_formulas:
    hamiltonian_vector_fields.append(P.bracket(formula))

# Create the basis of monomials.
hamiltonian_monomial_basis = {}
for j in range(len(shifts_formulas)):
    for i in range(2):
        hamiltonian_monomial_basis[i] = set(shifts_formulas[j][i].monomials())
        for formula in hamiltonian_vector_fields:
            hamiltonian_monomial_basis[i] |= set(formula[i].monomials())

hamiltonian_monomial_basis = {idx: list(b) for idx, b in␣
↪hamiltonian_monomial_basis.items()}
hamiltonian_monomial_index = {idx: {m : k for k, m in enumerate(b)} for idx, b␣
↪in hamiltonian_monomial_basis.items()}
hamiltonian_monomial_count = sum(len(b) for b in hamiltonian_monomial_basis.
↪values())

# Now, we write the shifts in terms of a vector of the monomials.
def shift_formula_to_vector(shift_formula):
        shift_vector = vector(QQ,hamiltonian_monomial_count, sparse=True)
        index_offset = 0
        for i in hamiltonian_monomial_basis:
            for coeff, monomial in shift_formula[i]:
```

```
                monomial_index = hamiltonian_monomial_index[i][monomial]
                shift_vector[monomial_index + index_offset] = coeff
            index_offset += len(hamiltonian_monomial_basis[i])
        return shift_vector

shifts_vectors = [shift_formula_to_vector(shift_formula) for shift_formula in␣
↪shifts_formulas]

# Create the evaluation matrix for the Hamiltonian vector fields.
hamiltonian_evaluation_matrix =␣
↪matrix(QQ,hamiltonian_monomial_count,len(hamiltonian_vector_fields),sparse=True)
for k in range(len(hamiltonian_vector_fields)):
    formula = hamiltonian_vector_fields[k]
    v = vector(QQ, hamiltonian_monomial_count, sparse=True)
    index_shift = 0
    for i in hamiltonian_monomial_basis:
        for coeff, monomial in formula[i]:
            monomial_index = hamiltonian_monomial_index[i][monomial]
            v[monomial_index +index_shift] = coeff
        index_shift += len(hamiltonian_monomial_basis[i])
    hamiltonian_evaluation_matrix.set_column(k, v)

nullity=hamiltonian_evaluation_matrix.right_nullity()
kernel_basis= hamiltonian_evaluation_matrix.right_kernel().basis()
print('The nullity among the Hamiltonian vector fields is ', nullity, '.\n')

pivots = hamiltonian_evaluation_matrix.pivots()
print('Lemma 8: A maximal subset of linearly independent hamiltonian graphs is␣
↪given by:', list(pivots),'\n')

# We can now solve the shifts! Note that if (at least one) shift is NOT␣
↪Hamiltonian, the code will break here as no solution
# can be found.
shifts_solutions = [hamiltonian_evaluation_matrix.solve_right(shift_vector) for␣
↪shift_vector in shifts_vectors]

# Write the solutions above from combinations of graphs to their vector field␣
↪forms to check validity.
solution_formulas = [sum(shift_solution[i]*hamiltonian_vector_fields[i] for i in␣
↪range(len(shift_solution))) for shift_solution in shifts_solutions]

if any(P.bracket(solution_formula) != 0 for solution_formula in␣
↪solution_formulas) or solution_formulas != shifts_formulas:
    print('There is an error in computing the shifts.')
else:
    print('Theorem 9: The shifts in the cocycle space are Hamiltonian.')
```

```
for k, shift_solution in enumerate(shifts_solutions):
        print('The shift #', k+1, 'is the following linear combination of␣
 ↪Hamiltonian vector fields:', shift_solution)
```

```
We have 50000 graphs.

Lemma 3: We have 2225 non-isomorphic graphs.

There are 8640 graphs that evaluate to 0 under the morphism from graphs to
multivectors.

We have 69 graphs left that do not evaluate into the exact same vector fields.

Claim 4: The vector fields have 47 linear relations among themselves.
In other words, we have 22 linearly independent vector fields left.

A maximal subset of linearly independent graphs is given by: [0, 1, 2, 3, 4, 5,
6, 7, 9, 10, 11, 13, 14, 17, 18, 20, 23, 25, 34, 36, 37, 39]

The pentagon wheel flow in 2D is (-10*rho_y^3*rho_xx*rho_yy*rho_xxx +
20*rho_x*rho_y^2*rho_xy*rho_yy*rho_xxx - 10*rho_x^2*rho_y*rho_yy^2*rho_xxx +
20*rho_y^3*rho_xx*rho_xy*rho_xxy - 40*rho_x*rho_y^2*rho_xy^2*rho_xxy +
10*rho_x*rho_y^2*rho_xx*rho_yy*rho_xxy + 10*rho_x^3*rho_yy^2*rho_xxy -
10*rho_y^3*rho_xx^2*rho_xyy + 40*rho_x^2*rho_y*rho_xy^2*rho_xyy -
10*rho_x^2*rho_y*rho_xx*rho_yy*rho_xyy - 20*rho_x^3*rho_xy*rho_yy*rho_xyy +
10*rho_x*rho_y^2*rho_xx^2*rho_yyy - 20*rho_x^2*rho_y*rho_xx*rho_xy*rho_yyy +
10*rho_x^3*rho_xx*rho_yy*rho_yyy - 10*rho_y^4*rho_xy*rho_xxxx +
10*rho_x*rho_y^3*rho_yy*rho_xxxx + 10*rho_y^4*rho_xx*rho_xxxy +
20*rho_x*rho_y^3*rho_xy*rho_xxxy - 30*rho_x^2*rho_y^2*rho_yy*rho_xxxy -
30*rho_x*rho_y^3*rho_xx*rho_xxyy + 30*rho_x^3*rho_y*rho_yy*rho_xxyy +
30*rho_x^2*rho_y^2*rho_xx*rho_xyyy - 20*rho_x^3*rho_y*rho_xy*rho_xyyy -
10*rho_x^4*rho_yy*rho_xyyy - 10*rho_x^3*rho_y*rho_xx*rho_yyyy +
10*rho_x^4*rho_xy*rho_yyyy - 2*rho_y^5*rho_xxxxx + 10*rho_x*rho_y^4*rho_xxxxy -
20*rho_x^2*rho_y^3*rho_xxxyy + 20*rho_x^3*rho_y^2*rho_xxyyy -
10*rho_x^4*rho_y*rho_xyyyy + 2*rho_x^5*rho_yyyyy)*xi0*xi1

There is 1 vector field from the linearly independent vector fields that
evaluates to 0 bivector after taking the Schouten bracket with P.
This vector field that evaluate to 0 is obtained from the graph:
graph 12

Proposition 5: The solution on vector fields is given by (0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, -10, -2, -2, 0, 2, 4, 0, 8, 4, -12)

The space of solutions to the homogeneous system has dimension 8
Proposition 7: These shifts are given by [(1, 0, -2, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0), (0, 1, -1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0,
```

```
-3, 0, 0, 0, 0, 0, 0), (0, 0, 0, 1, 0, 0, 0, 0, 2, 0, 2, 0, 0, 0, 0, -1, 0, 0,
-1, 0, 0, 0), (0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, -2, 0, 0,
0), (0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0), (0, 0,
0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, -2, 0, 0, 0, 0, 0, 0), (0, 0, 0, 0, 0, 0,
0, 1, 0, 0, 2, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0), (0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0)]

We generate 1296 Hamiltonian graphs.

Lemma 8: There are 66 nonisomorphic Hamiltonian graphs.

We have 312 Hamiltonian graphs that evaluate to 0.

The nullity among the Hamiltonian vector fields is  976 .

Lemma 8: A maximal subset of linearly independent hamiltonian graphs is given
by: [0, 1, 6, 7, 8, 9, 10, 33]

Theorem 9: The shifts in the cocycle space are Hamiltonian.
The shift # 1 is the following linear combination of Hamiltonian vector fields:
(1/2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
The shift # 2 is the following linear combination of Hamiltonian vector fields:
(0, 1, 0, 0, 0, 0, 0, 0, 1/2, 1/2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
The shift # 3 is the following linear combination of Hamiltonian vector fields:
```

```
(0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1/8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
The shift # 4 is the following linear combination of Hamiltonian vector fields:
(0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
The shift # 5 is the following linear combination of Hamiltonian vector fields:
(0, 0, 0, 0, 0, 0, 0, 0, 1/2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
The shift # 6 is the following linear combination of Hamiltonian vector fields:
(0, 0, 0, 0, 0, 0, 0, 0, 0, 1/2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
The shift # 7 is the following linear combination of Hamiltonian vector fields:
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1/8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
The shift # 8 is the following linear combination of Hamiltonian vector fields:
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1/4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
```

Finding $\vec{X}_{2D}^{\gamma_5}$ (Hamiltonian method)

November 23, 2024

[2]:
```
from gcaops.algebra.differential_polynomial_ring import␣
↪DifferentialPolynomialRing
from gcaops.algebra.superfunction_algebra import SuperfunctionAlgebra
from gcaops.algebra.homogeneous_polynomial_poisson_complex import PoissonComplex
from gcaops.graph.undirected_graph import UndirectedGraph
from gcaops.graph.undirected_graph_complex import UndirectedGraphComplex
from gcaops.graph.directed_graph import DirectedGraph
from gcaops.graph.directed_graph_complex import DirectedGraphComplex
from gcaops.algebra.differential_polynomial_solver import␣
↪solve_homogeneous_diffpoly
from gcaops.graph.formality_graph import FormalityGraph


# We introduce the differential polynomial ring. Explicitly, we have P=rho d/dx␣
↪d/dy, V0 and V1 are components of a vector
# field V0 d/dx +v1 d/dy and H is a function.
D2=DifferentialPolynomialRing(QQ,('rho', 'V0', 'V1', 'H' ), ('x','y'),␣
↪max_differential_orders=[5+1+2, 1, 1, 1])
rho, V0, V1, H =D2.fibre_variables()
x,y= D2.base_variables()
even_coords=[x,y]

S2.<xi0,xi1>=SuperfunctionAlgebra(D2, D2.base_variables())
xi=S2.gens()
odd_coords=xi
epsilon = xi[0]*xi[1]

# We create a Poisson bivector.
P = rho*xi[0]*xi[1]

# We introduce the graph complex to compute Q_fivewheel.
GC = UndirectedGraphComplex(QQ, connected=True, biconnected=True, min_degree=3,␣
↪implementation='vector')
dGC = DirectedGraphComplex(QQ, connected=True, biconnected=True, min_degree=3,␣
↪loops=False, implementation='vector', sparse=True)
dfGC = DirectedGraphComplex(QQ, connected=True, implementation='vector',␣
↪sparse=True)
```

```
fivewheel_cocycle = GC.cohomology_basis(6,10)[0]
fivewheel_operation2 = S2.graph_operation(fivewheel_cocycle)
Q_fivewheel2 = fivewheel_operation2(P,P,P,P,P,P)

# We create a vector field, and consider the bivector it gives rise to.
V = V0*xi[0] + V1*xi[1]
PbracketV = P.bracket(V)

# We solve Q_fivewheel=[[P,V]] for the vector field V
cX_fivewheel2 = solve_homogeneous_diffpoly(Q_fivewheel2[0,1], PbracketV[0,1],␣
 ↪[V0,V1])
X_fivewheel2 = cX_fivewheel2[V0]*xi[0] + cX_fivewheel2[V1]*xi[1]

if Q_fivewheel2 != P.bracket(X_fivewheel2):
    print('Something went wrong in computing the trivializing vector field.')

# We now find a function H such that V0= d/dy H, and V1=-d/dx H
cH_fivewheel2 = solve_homogeneous_diffpoly(cX_fivewheel2[V0], diff(H,y), [H])
cH_fivewheel2 = solve_homogeneous_diffpoly(cX_fivewheel2[V1],-diff(H,x), [H])
H_fivewheel2 = cH_fivewheel2[H]

if diff(H_fivewheel2, y) != X_fivewheel2[0] or -diff(H_fivewheel2, x) !=␣
 ↪X_fivewheel2[1]:
    print('Something went wronf in computing the Hamiltonian function.')

# We list all the graphs on 5 vertices, with exactly 4 vertices having 2␣
 ↪outgoing edges.
wedge_graphs=[DirectedGraph(5, [(0, 3), (0, 4), (1, 3), (1, 4), (2, 1), (2, 4),␣
 ↪(3, 2), (3, 4)]),
 DirectedGraph(5, [(0, 2), (0, 3), (1, 3), (1, 4), (2, 1), (2, 4), (3, 2), (3,␣
 ↪4)]),
 DirectedGraph(5, [(0, 1), (0, 4), (1, 3), (1, 4), (3, 0), (3, 2), (4, 2), (4,␣
 ↪3)]),
 DirectedGraph(5, [(0, 3), (0, 4), (1, 0), (1, 3), (2, 1), (2, 4), (3, 2), (3,␣
 ↪4)]),
 DirectedGraph(5, [(0, 3), (0, 4), (1, 2), (1, 4), (2, 0), (2, 4), (3, 1), (3,␣
 ↪4)]),
 DirectedGraph(5, [(0, 3), (0, 4), (1, 3), (1, 4), (3, 2), (3, 4), (4, 2), (4,␣
 ↪3)]),
 DirectedGraph(5, [(0, 3), (0, 4), (1, 3), (1, 4), (2, 3), (2, 4), (3, 2), (3,␣
 ↪4)]),
 DirectedGraph(5, [(0, 3), (0, 4), (1, 3), (1, 4), (3, 2), (3, 4), (4, 1), (4,␣
 ↪2)]),
 DirectedGraph(5, [(0, 3), (0, 4), (1, 2), (1, 4), (2, 3), (2, 4), (4, 2), (4,␣
 ↪3)]),
```

```
DirectedGraph(5, [(0, 3), (0, 4), (1, 2), (1, 4), (2, 3), (2, 4), (3, 2), (3,␣
↪4)]),
DirectedGraph(5, [(0, 3), (0, 4), (1, 2), (1, 4), (3, 1), (3, 4), (4, 2), (4,␣
↪3)]),
DirectedGraph(5, [(0, 1), (0, 4), (1, 3), (1, 4), (3, 2), (3, 4), (4, 2), (4,␣
↪3)]),
DirectedGraph(5, [(0, 3), (0, 4), (2, 1), (2, 3), (3, 2), (3, 4), (4, 1), (4,␣
↪2)]),
DirectedGraph(5, [(0, 2), (0, 4), (1, 3), (1, 4), (2, 3), (2, 4), (4, 1), (4,␣
↪3)]),
DirectedGraph(5, [(0, 3), (0, 4), (1, 3), (1, 4), (2, 1), (2, 3), (3, 2), (3,␣
↪4)]),
DirectedGraph(5, [(0, 3), (0, 4), (1, 0), (1, 4), (2, 1), (2, 4), (4, 2), (4,␣
↪3)]),
DirectedGraph(5, [(0, 2), (0, 4), (1, 3), (1, 4), (2, 3), (2, 4), (3, 1), (3,␣
↪4)]),
DirectedGraph(5, [(0, 3), (0, 4), (1, 2), (1, 3), (2, 1), (2, 4), (3, 2), (3,␣
↪4)]),
DirectedGraph(5, [(0, 3), (0, 4), (1, 2), (1, 4), (3, 1), (3, 2), (4, 0), (4,␣
↪3)]),
DirectedGraph(5, [(0, 2), (0, 4), (1, 3), (1, 4), (3, 0), (3, 1), (4, 2), (4,␣
↪3)]),
DirectedGraph(5, [(1, 3), (1, 4), (2, 1), (2, 4), (3, 2), (3, 4), (4, 0), (4,␣
↪3)]),
DirectedGraph(5, [(1, 3), (1, 4), (2, 1), (2, 4), (3, 0), (3, 2), (4, 2), (4,␣
↪3)]),
DirectedGraph(5, [(0, 3), (0, 4), (1, 2), (1, 4), (2, 3), (2, 4), (3, 1), (3,␣
↪2)]),
DirectedGraph(5, [(0, 1), (0, 3), (1, 2), (1, 4), (2, 3), (2, 4), (3, 2), (3,␣
↪4)]),
DirectedGraph(5, [(0, 1), (0, 2), (1, 3), (1, 4), (3, 0), (3, 4), (4, 2), (4,␣
↪3)]),
DirectedGraph(5, [(0, 1), (0, 4), (1, 2), (1, 3), (3, 0), (3, 4), (4, 2), (4,␣
↪3)]),
DirectedGraph(5, [(0, 2), (0, 4), (1, 2), (1, 3), (3, 0), (3, 4), (4, 1), (4,␣
↪3)]),
DirectedGraph(5, [(0, 3), (0, 4), (1, 0), (1, 4), (2, 1), (2, 3), (3, 2), (3,␣
↪4)]),
DirectedGraph(5, [(0, 3), (0, 4), (1, 3), (1, 4), (2, 0), (2, 4), (3, 1), (3,␣
↪2)]),
DirectedGraph(5, [(0, 3), (0, 4), (2, 3), (2, 4), (3, 1), (3, 4), (4, 2), (4,␣
↪3)]),
DirectedGraph(5, [(1, 3), (1, 4), (2, 3), (2, 4), (3, 2), (3, 4), (4, 0), (4,␣
↪1)]),
```

```
DirectedGraph(5, [(0, 3), (0, 4), (2, 3), (2, 4), (3, 2), (3, 4), (4, 1), (4,␣
↪2)]),
DirectedGraph(5, [(0, 3), (0, 4), (1, 3), (1, 4), (3, 1), (3, 2), (4, 0), (4,␣
↪2)]),
DirectedGraph(5, [(0, 3), (0, 4), (2, 3), (2, 4), (3, 1), (3, 2), (4, 1), (4,␣
↪2)]),
DirectedGraph(5, [(0, 3), (0, 4), (1, 2), (1, 4), (2, 1), (2, 4), (4, 0), (4,␣
↪3)]),
DirectedGraph(5, [(0, 3), (0, 4), (1, 2), (1, 4), (2, 1), (2, 4), (3, 0), (3,␣
↪4)]),
DirectedGraph(5, [(0, 3), (0, 4), (1, 2), (1, 4), (2, 1), (2, 4), (4, 2), (4,␣
↪3)]),
DirectedGraph(5, [(0, 2), (0, 4), (1, 3), (1, 4), (3, 2), (3, 4), (4, 1), (4,␣
↪3)]),
DirectedGraph(5, [(0, 1), (0, 4), (2, 3), (2, 4), (3, 1), (3, 2), (4, 2), (4,␣
↪3)]),
DirectedGraph(5, [(0, 3), (0, 4), (1, 2), (1, 4), (3, 2), (3, 4), (4, 1), (4,␣
↪3)]),
DirectedGraph(5, [(0, 3), (0, 4), (1, 3), (1, 4), (2, 0), (2, 4), (4, 1), (4,␣
↪2)]),
DirectedGraph(5, [(0, 3), (0, 4), (1, 2), (1, 4), (3, 0), (3, 2), (4, 1), (4,␣
↪3)]),
DirectedGraph(5, [(0, 1), (0, 4), (1, 0), (1, 3), (3, 2), (3, 4), (4, 2), (4,␣
↪3)]),
DirectedGraph(5, [(0, 3), (0, 4), (1, 2), (1, 4), (2, 1), (2, 3), (3, 0), (3,␣
↪4)]),
DirectedGraph(5, [(0, 3), (0, 4), (1, 2), (1, 4), (2, 1), (2, 3), (3, 2), (3,␣
↪4)]),
DirectedGraph(5, [(0, 1), (0, 3), (1, 2), (1, 4), (3, 0), (3, 4), (4, 2), (4,␣
↪3)]),
DirectedGraph(5, [(0, 3), (0, 4), (2, 1), (2, 4), (3, 2), (3, 4), (4, 2), (4,␣
↪3)]),
DirectedGraph(5, [(1, 2), (1, 4), (2, 1), (2, 3), (3, 0), (3, 4), (4, 2), (4,␣
↪3)]),
DirectedGraph(5, [(0, 3), (0, 4), (2, 0), (2, 4), (3, 1), (3, 4), (4, 2), (4,␣
↪3)]),
DirectedGraph(5, [(1, 3), (1, 4), (2, 0), (2, 4), (3, 2), (3, 4), (4, 1), (4,␣
↪3)]),
DirectedGraph(5, [(1, 0), (1, 4), (2, 1), (2, 3), (3, 2), (3, 4), (4, 2), (4,␣
↪3)]),
DirectedGraph(5, [(1, 2), (1, 3), (2, 1), (2, 4), (3, 0), (3, 4), (4, 2), (4,␣
↪3)]),
DirectedGraph(5, [(1, 3), (1, 4), (2, 0), (2, 4), (3, 1), (3, 4), (4, 2), (4,␣
↪3)]),
```

```
 DirectedGraph(5, [(0, 2), (0, 4), (1, 2), (1, 3), (3, 1), (3, 4), (4, 0), (4,␣
 ↪3)])]

print('We have', len(wedge_graphs), 'graphs on 5 vertices, with exactly 4␣
 ↪vertices having 2 outgoing edges.\n')

# We compute the formulas the above graphs evaluate into
graphs_op_wedges=[]
for i in range(len(wedge_graphs)):
    formula=S2.graph_operation(dfGC(wedge_graphs[i]))(P,P,P,P,P)[0,1]
    graphs_op_wedges.append(formula)

# We check if there are any graphs that evaluate into 0
vanishing_graphs_index=[k for (k,X) in enumerate (graphs_op_wedges) if X==0]
vanishing_graphs= []
for i in vanishing_graphs_index:
    vanishing_graphs.append(wedge_graphs[i])

print( len(vanishing_graphs), 'of these graphs evaluate into 0.\n')

# We now create Hamiltonian (wrt the standard symplectic form) vector fields␣
 ↪from the formulas we found above.
hamiltonian_vector_fields=[]
for i in range(len(graphs_op_wedges)):
    hamiltonian_vector_fields.append(diff(graphs_op_wedges[i], y)*xi[0] -␣
 ↪diff(graphs_op_wedges[i], x)*xi[1])

# We now start with setting up the system we want to solve to find a graph␣
 ↪representation of the Hamiltonian
hamiltonian_monomial_basis = {}
for i in range(2):
    hamiltonian_monomial_basis[i] = set(X_fivewheel2[i].monomials())
    for formula in hamiltonian_vector_fields:
        hamiltonian_monomial_basis[i] |= set(formula[i].monomials())

hamiltonian_monomial_basis = {idx: list(b) for idx, b in␣
 ↪hamiltonian_monomial_basis.items()}
hamiltonian_monomial_index = {idx: {m : k for k, m in enumerate(b)} for idx, b␣
 ↪in hamiltonian_monomial_basis.items()}
{idx: len(b) for idx, b in hamiltonian_monomial_basis.items()}
hamiltonian_monomial_count = sum(len(b) for b in hamiltonian_monomial_basis.
 ↪values()); hamiltonian_monomial_count

# We write the trivializing vector field V from before as a vector of its␣
 ↪monomials
vector_field_vector = vector(QQ,hamiltonian_monomial_count, sparse=True)
```

```
index_shift = 0
for i in hamiltonian_monomial_basis:
    for coeff, monomial in X_fivewheel2[i]:
        monomial_index = hamiltonian_monomial_index[i][monomial]
        vector_field_vector[monomial_index + index_shift] = coeff
    index_shift += len(hamiltonian_monomial_basis[i])

# Create the evaluation matrix
hamiltonian_evaluation_matrix =␣
↪matrix(QQ,hamiltonian_monomial_count,len(hamiltonian_vector_fields),sparse=True)
for k in range(len(hamiltonian_vector_fields)):
    formula = hamiltonian_vector_fields[k]
    v = vector(QQ, hamiltonian_monomial_count, sparse=True)
    index_shift = 0
    for i in hamiltonian_monomial_basis:
        for coeff, monomial in formula[i]:
            monomial_index = hamiltonian_monomial_index[i][monomial]
            v[monomial_index +index_shift] = coeff
        index_shift += len(hamiltonian_monomial_basis[i])
    hamiltonian_evaluation_matrix.set_column(k, v)

# We find a linear combination of Hamiltonian vector fields coming from graphs␣
↪that equals the trivializing
# vector field V
vec_sol = hamiltonian_evaluation_matrix.solve_right(vector_field_vector)
print('The trivializing vector field can be realised as a linear combination of␣
↪Hamiltonian vector fields (with respect to the standard symplectic structure.')
print('Explicitly, the the linear combination is given by',vec_sol, '\n')

#Let us now check if we truly solve the problem. I switched around some of the␣
↪vertex labels. This changes
# the vector fields up to a minus sign.
encodings=[]
# wedge_graph[1]
for i1 in [1,2,3,4,5]:
    encodings.append((0,i1,3,5,4,5,1,2,1,4))
# wedge_graph[2]
for i1 in [1,2,3,4,5]:
    encodings.append((0,i1,1,5,2,5,1,3,1,4))
# wedge_graph[3]
for i1 in [1,2,3,4,5]:
    encodings.append((0,i1,1,5,1,4,1,2,1,3))

# We introduce the function to go from encodings to graphs.
def encoding_to_graph(encoding):
    targets = [encoding[0:2], encoding[2:4], encoding[4:6], encoding[6:8],␣
↪encoding[8:10]]
```

```
    edges = sum([[(k+1,v) for v in t] for (k,t) in enumerate(targets)], [])
    return FormalityGraph(1, 5, edges)

# We save the graphs in this list.
sol_graphs= []
for encoding in encodings:
    sol_graphs.append(encoding_to_graph(encoding))

# We compute the vector fields the graphs evaluate into.
sol_graphs_formulas=[]
import itertools
E=x*xi[0]+y*xi[1]
epsilon = xi[0]*xi[1]
for g in sol_graphs:
    term = S2.zero()
    for index_choice in itertools.product(itertools.permutations(range(2)),␣
 ↪repeat=5):
        # TODO: Check that this way of evaluating is correct.
        sign = epsilon[index_choice[0]] * epsilon[index_choice[1]] *␣
 ↪epsilon[index_choice[2]] * epsilon[index_choice[3]] * epsilon[index_choice[4]]
        vertex_content = [E, S2(rho), S2(rho), S2(rho), S2(rho)]
        for ((source, target), index) in zip(g.edges(), sum(map(list,␣
 ↪index_choice), [])):
            vertex_content[target] = diff(vertex_content[target],␣
 ↪even_coords[index])
        term += sign * prod(vertex_content)
    sol_graphs_formulas.append(term)

# We sum all the vector fields with the correct constants to create the␣
 ↪trivializing vector
# field
supposed_trivializing_vector_field= 0
for i in range(5):
    supposed_trivializing_vector_field+=-6*sol_graphs_formulas[i]
for j in range(5):
    supposed_trivializing_vector_field+=+2*sol_graphs_formulas[j+5]
for k in range(5):
    supposed_trivializing_vector_field+=-2*sol_graphs_formulas[k+10]

# We check that the vector field that we find trivializes Q_fivewheel
if P.bracket(supposed_trivializing_vector_field)!=Q_fivewheel2:
    print('The vector field found does not trivialize Q_fivewheel.')
else:
    print('Theorem 6: The vector field evaluated from the 15 graphs trivializes␣
 ↪Q_fivewheel.')
```

We have 54 graphs on 5 vertices, with exactly 4 vertices having 2 outgoing

```
edges.

10 of these graphs evaluate into 0.

The trivializing vector field can be realised as a linear combination of
Hamiltonian vector fields (with respect to the standard symplectic structure.
Explicitly, the the linear combination is given by (0, 0, 6, -2, -2, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)

Theorem 6: The vector field evaluated from the 15 graphs trivializes
Q_fivewheel.
```

Q_{γ_5} in 3D

November 25, 2024

[1]:
```
from gcaops.graph.formality_graph import FormalityGraph
```

[2]:
```
from gcaops.algebra.differential_polynomial_ring import␣
 ↪DifferentialPolynomialRing
D3 = DifferentialPolynomialRing(QQ, ('rho', 'a'), ('x','y', 'z'),␣
 ↪max_differential_orders=[5+1, 5+1+1])
rho, a = D3.fibre_variables()
x,y, z = D3.base_variables()
even_coords = [x,y,z]
```

[3]:
```
from gcaops.algebra.superfunction_algebra import SuperfunctionAlgebra

S3.<xi0,xi1,xi2> = SuperfunctionAlgebra(D3, D3.base_variables())
xi = S3.gens()
odd_coords = xi
epsilon = xi[0]*xi[1]*xi[2] # Levi-Civita tensor
```

[4]:
```
%%time
from gcaops.graph.undirected_graph_complex import UndirectedGraphComplex

GC = UndirectedGraphComplex(QQ, implementation='vector', sparse=True)
fivewheel_cocycle = GC.cohomology_basis(6,10)[0]; fivewheel_cocycle
```

```
CPU times: user 146 ms, sys: 27.6 ms, total: 173 ms
Wall time: 211 ms
```

[4]:
```
1*UndirectedGraph(6, [(0, 3), (0, 4), (0, 5), (1, 2), (1, 4), (1, 5), (2, 3),
(2, 5), (3, 5), (4, 5)]) + (5/2)*UndirectedGraph(6, [(0, 1), (0, 3), (0, 5), (1,
2), (1, 4), (2, 4), (2, 5), (3, 4), (3, 5), (4, 5)])
```

[5]:
```
%%time
from gcaops.graph.directed_graph_complex import DirectedGraphComplex
dGC = DirectedGraphComplex(QQ, implementation='vector')
fivewheel_oriented = dGC(fivewheel_cocycle)
fivewheel_oriented_filtered = fivewheel_oriented.filter(max_out_degree=2)
fivewheel_operation = S3.graph_operation(fivewheel_oriented_filtered)
```

```
CPU times: user 12.2 s, sys: 531 ms, total: 12.8 s
Wall time: 13.3 s
```

[6]:
```
P = rho*epsilon.bracket(a)
```

[7]:
```
%%time
Q_fivewheel= fivewheel_operation(P,P,P,P,P,P)
```

```
CPU times: user 16min 42s, sys: 1min 16s, total: 17min 59s
Wall time: 6h 54min 33s
```

[8]:
```
%%time
P.bracket(Q_fivewheel)
```

```
CPU times: user 10min 3s, sys: 7.65 s, total: 10min 10s
Wall time: 10min 10s
```

[8]:
```
0
```

[9]:
```
%%time
import pickle
file_path = 'q_fivewheel_3d.pickle'

with open(file_path,'wb') as file:
    pickle.dump(Q_fivewheel, file)
```

```
CPU times: user 5.82 s, sys: 369 ms, total: 6.19 s
Wall time: 6.6 s
```

Trivializing Q_{γ_5} in 3D (Descendant of the brute force 2D solution)

November 25, 2024

[ ]:
```
from gcaops.graph.formality_graph import FormalityGraph
from gcaops.algebra.differential_polynomial_ring import␣
 ↪DifferentialPolynomialRing
from gcaops.algebra.superfunction_algebra import SuperfunctionAlgebra
from gcaops.graph.undirected_graph_complex import UndirectedGraphComplex
from gcaops.graph.directed_graph_complex import DirectedGraphComplex

import itertools
from multiprocessing import Pool
import pickle

X_graph_encodings = []

for (i1,j1,j2,k1,l1) in itertools.product([1,6],[1,6],[2,7],[3,8],[3,8]):
    X_graph_encodings.append((0,1,6,i1,2,7,j1,j2,8,k1,4,9,l1,5,10))

for (i1,j1,k1,l1) in itertools.product([1,6],[1,6],[1,6],[1,6]):
    X_graph_encodings.append((0,1,6,i1,2,7,j1,3,8,k1,4,9,l1,5,10))

for (i1,j1,k1,l1) in itertools.product([1,6],[1,6],[1,6],[2,7]):
    X_graph_encodings.append((0,1,6,i1,2,7,j1,3,8,k1,4,9,l1,5,10))

for (i1,j1,k1,l1) in itertools.product([1,6],[1,6],[2,7],[2,7]):
    X_graph_encodings.append((0,1,6,i1,2,7,j1,3,8,k1,4,9,l1,5,10))

for (i1,j1,k1,l1) in itertools.product([1,6],[1,6],[2,7],[3,8]):
    X_graph_encodings.append((0,1,6,i1,2,7,j1,3,8,k1,4,9,l1,5,10))

for (i1,j1,k1,l1) in itertools.product([1,6],[2,7],[2,7],[2,7]):
    X_graph_encodings.append((0,1,6,i1,2,7,j1,3,8,k1,4,9,l1,5,10))

for (i1,j1,k1,l1) in itertools.product([1,6],[2,7],[2,7],[3,8]):
    X_graph_encodings.append((0,1,6,i1,2,7,j1,3,8,k1,4,9,l1,5,10))

for (i1,j1,k1,l1) in itertools.product([1,6],[2,7],[2,7],[4,9]):
    X_graph_encodings.append((0,1,6,i1,2,7,j1,3,8,k1,4,9,l1,5,10))
```

```
def encoding_to_graph(encoding):
    targets = [encoding[0:3], encoding[3:6], encoding[6:9], encoding[9:12],␣
 ↪encoding[12:15]]
    edges = sum([[(k+1,v) for v in t] for (k,t) in enumerate(targets)], [])
    return FormalityGraph(1, 10, edges)

X_graphs = [encoding_to_graph(e) for e in X_graph_encodings]
print("Number of graphs in X:", len(X_graphs), flush=True)

D3 = DifferentialPolynomialRing(QQ, ('rho','a'), ('x','y','z'),␣
 ↪max_differential_orders=[5+1,1+5+1])
rho, a = D3.fibre_variables()
x,y,z = D3.base_variables()
even_coords = [x,y,z]

S3.<xi0,xi1,xi2> = SuperfunctionAlgebra(D3, D3.base_variables()) #; S3
xi = S3.gens()
odd_coords = xi
epsilon = xi[0]*xi[1]*xi[2]

def evaluate_graph(g):
    E = x*xi[0] + y*xi[1] + z*xi[2] # Euler vector field, to insert into ground␣
 ↪vertex. Incoming derivative d/dx^i will result in xi[i].
    result = S3.zero()
    for index_choice in itertools.product(itertools.permutations(range(3)),␣
 ↪repeat=5):
        sign = epsilon[index_choice[0]] * epsilon[index_choice[1]] *␣
 ↪epsilon[index_choice[2]] * epsilon[index_choice[3]] * epsilon[index_choice[4]]
        # NOTE: This assumes the ground vertex is labeled 0, the vertices of␣
 ↪out-degree 4 are labeled 1, 2, 3, and the Casimirs are labeled 4, 5, 6; 7, 8,␣
 ↪9.
        vertex_content = [E, S3(rho), S3(rho), S3(rho), S3(rho), S3(rho), S3(a),␣
 ↪S3(a), S3(a), S3(a), S3(a)]
        for ((source, target), index) in zip(g.edges(), sum(map(list,␣
 ↪index_choice), [])):
            vertex_content[target] = vertex_content[target].
 ↪derivative(even_coords[index])
        result += sign * prod(vertex_content)
    return result

print("Calculating X_vector_fields", flush=True)
X_vector_fields = []

with Pool(processes=14) as pool:
    X_vector_fields = list(pool.imap(evaluate_graph, X_graphs))
```

```
print("Calculated X_vector_fields", flush=True)

X_vector_fields.count(0)

[k+1 for (k, X) in enumerate(X_vector_fields) if X == 0]

X_monomial_basis = [set([]) for i in range(4)]
for i in range(4):
    for X in X_vector_fields:
        X_monomial_basis[i] |= set(X[i].monomials())
X_monomial_basis = [list(b) for b in X_monomial_basis]
X_monomial_count = sum(len(b) for b in X_monomial_basis)

# Next, we use this monomial basis to create a matrix that identifies each␣
 ↪vector field by the monomials that appear in it.
X_monomial_to_index = [{monomial : idx for (idx,monomial) in␣
 ↪enumerate(X_monomial_basis[j])} for j in range(4)]
X_evaluation_matrix = matrix(QQ, X_monomial_count, len(X_vector_fields))
for i in range(len(X_vector_fields)):
    v = vector(QQ, X_monomial_count)
    index_shift = 0
    for j in range(3):
        f = X_vector_fields[i][j]
        for coeff, monomial in zip(f.coefficients(), f.monomials()):
            monomial_index = X_monomial_to_index[j][monomial]
            v[index_shift + monomial_index] = coeff
        index_shift += len(X_monomial_basis[j])
    X_evaluation_matrix.set_column(i, v)

nullity = X_evaluation_matrix.right_nullity()
print('The vector fields have', nullity, 'linear relations among themselves.\n')

pivots = X_evaluation_matrix.pivots()
print("Maximal subset of linearly independent graphs:", list(pivots), flush=True)

X_lin_ind_formulas=[]
for i in pivots:
    X_lin_ind_formulas.append(X_vector_fields[i])

lin_ind_encodings=[]
for i in pivots:
    lin_ind_encodings.append(X_graph_encodings[i])

P = rho*epsilon.bracket(a)
if P.bracket(P)!=0:
    print('P is not a Poisson bivector. \n')
```

```
#GC = UndirectedGraphComplex(QQ, implementation='vector', sparse=True)
#fivewheel_cocycle = GC.cohomology_basis(6,10)[0];
#dGC = DirectedGraphComplex(QQ, implementation='vector')
#fivewheel_oriented = dGC(fivewheel_cocycle)
#fivewheel_oriented_filtered = fivewheel_oriented.filter(max_out_degree=2)
#fivewheel_operation = S3.graph_operation(fivewheel_oriented_filtered)

#Q_fivewheel= fivewheel_operation(P,P,P,P,P,P)
#P.bracket(Q_fivewheel)
file_path = 'q_fivewheel_3d.pickle'
Q_fivewheel=None
with open(file_path,'rb') as file:
    Q_fivewheel=pickle.load(file)

X_bivectors=[]
for X in X_vector_fields:
    X_bivectors.append(P.bracket(X))

# There are no nonzero (new) vector fields evaluated from 1 graph such that␣
 ↪[[P,X_{graph}]]=0.
zero_bivectors = X_bivectors.count(0)
print('There are', zero_bivectors, 'vector fields in X_vector_fields that␣
 ↪evaluate to 0 bivectors after taking the Schouten bracket with P .')
print('These vector fields that evaluate to 0 are obtained from the graphs:')
for (k,X) in enumerate(X_bivectors):
    if X==0:
        print('graph', k+1,)
print()

# Now, we extract the monomials appearing in these bivectors (as well as in␣
 ↪Q_fivewheel).
Q_monomial_basis={}
from itertools import combinations
for i,j in combinations(range(2),2):
    Q_monomial_basis[i,j]=set(Q_fivewheel[i,j].monomials())
    for P_X in X_bivectors:
        Q_monomial_basis[i,j]|= set(P_X[i,j].monomials())

Q_monomial_basis={idx: list(b) for idx, b in Q_monomial_basis.items()}
Q_monomial_index= {idx:{m:k for k,m in enumerate(b)} for idx, b in␣
 ↪Q_monomial_basis.items()}
Q_monomial_count=sum(len(b) for b in Q_monomial_basis.values());

# We create the vector representation of Q_fivewheel in terms of the monomials.
Q_fivewheel_vector= vector(QQ, Q_monomial_count, sparse=True)
index_shift=0
```

```
for i,j in Q_monomial_basis:
    for coeff, monomial in Q_fivewheel[i,j]:
        monomial_index= Q_monomial_index[i,j][monomial]
        Q_fivewheel_vector[monomial_index+index_shift]=coeff
    index_shift+=len(Q_monomial_basis[i,j])

X_bivector_evaluation_matrix= matrix(QQ, Q_monomial_count, len(X_bivectors),␣
 ↪sparse=True)
for k in range(len(X_bivectors)):
    P_X=X_bivectors[k]
    v=vector(QQ,Q_monomial_count, sparse=True)
    index_shift=0
    for i,j in Q_monomial_basis:
        for coeff, monomial in P_X[i,j]:
            monomial_index=Q_monomial_index[i,j][monomial]
            v[monomial_index+index_shift]=coeff
        index_shift+=len(Q_monomial_basis[i,j])
    X_bivector_evaluation_matrix.set_column(k,v)

# We soule the linear system.
X_solution=X_bivector_evaluation_matrix.solve_right(Q_fivewheel_vector)
print('The solution on vector fields is given by', X_solution,' \n')
```

```
Number of graphs in X: 144
Calculating X_vector_fields
Calculated X_vector_fields
The vector fields have 69 linear relations among themselves.

Maximal subset of linearly independent graphs: [0, 3, 4, 7, 8, 11, 12, 15, 19,
20, 23, 24, 27, 28, 31, 32, 33, 35, 39, 47, 48, 49, 50, 51, 54, 55, 56, 57, 58,
59, 62, 63, 64, 65, 67, 68, 69, 71, 72, 73, 75, 76, 77, 79, 80, 81, 83, 84, 85,
86, 87, 92, 93, 95, 96, 97, 99, 103, 104, 105, 107, 111, 112, 113, 114, 115,
118, 119, 120, 121, 123, 124, 125, 126, 127]
There are 16 vector fields in X_vector_fields that evaluate to 0 bivectors after
taking the Schouten bracket with P .
These vector fields that evaluate to 0 are obtained from the graphs:
graph 2
graph 3
graph 6
graph 7
graph 10
graph 11
graph 14
graph 15
graph 18
graph 19
graph 22
graph 23
```

```
graph 26
graph 27
graph 30
graph 31
```

[ ]:

Trivializing Q_{γ_5} in 3D (Descendants of the 2D solution of the Hamiltonian method)

November 25, 2024

[ ]:
```
from gcaops.graph.formality_graph import FormalityGraph
from gcaops.algebra.differential_polynomial_ring import␣
↪DifferentialPolynomialRing
from gcaops.algebra.superfunction_algebra import SuperfunctionAlgebra
from gcaops.graph.undirected_graph_complex import UndirectedGraphComplex
from gcaops.graph.directed_graph_complex import DirectedGraphComplex

import itertools
from multiprocessing import Pool
import pickle

X_graph_encodings=[]

X_graph_encodings = []
for (i1,j1,j2,k1,k2,l1,l2,m1,m2) in itertools.
↪product([1,2,3,4,5,7,8,9,10],[3,8],[5,10],[4,9],[5,10],[1,6],[2,7],[1,6],[4,9]):
↪
    X_graph_encodings.append((0,i1,6,j1,j2,7,k1,k2,8,l1,l2,9,m1,m2,10))

for (i1,j1,j2,k1,k2,l1,l2,m1,m2) in itertools.
↪product([1,2,3,4,5,7,8,9,10],[1,6],[5,10],[2,7],[5,10],[1,6],[3,8],[1,6],[4,9]):
↪
    X_graph_encodings.append((0,i1,6,j1,j2,7,k1,k2,8,l1,l2,9,m1,m2,10))

for (i1,j1,j2,k1,k2,l1,l2,m1,m2) in itertools.
↪product([1,2,3,4,5,7,8,9,10],[1,6],[5,10],[1,6],[4,9],[1,6],[2,7],[1,6],[3,8]):
    X_graph_encodings.append((0,i1,6,j1,j2,7,k1,k2,8,l1,l2,9,m1,m2,10))

def encoding_to_graph(encoding):
    targets = [encoding[0:3], encoding[3:6], encoding[6:9], encoding[9:12],␣
↪encoding[12:15]]
    edges = sum([[(k+1,v) for v in t] for (k,t) in enumerate(targets)], [])
    return FormalityGraph(1, 10, edges)

X_graphs = [encoding_to_graph(e) for e in X_graph_encodings]
```

```
print("Number of graphs in X:", len(X_graphs), flush=True)

D3 = DifferentialPolynomialRing(QQ, ('rho','a'), ('x','y','z'),␣
↪max_differential_orders=[5+1,1+5+1])
rho, a = D3.fibre_variables()
x,y,z = D3.base_variables()
even_coords = [x,y,z]

S3.<xi0,xi1,xi2> = SuperfunctionAlgebra(D3, D3.base_variables()) #; S3
xi = S3.gens()
odd_coords = xi
epsilon = xi[0]*xi[1]*xi[2]

def evaluate_graph(g):
    E = x*xi[0] + y*xi[1] + z*xi[2] # Euler vector field, to insert into ground␣
↪vertex. Incoming derivative d/dx^i will result in xi[i].
    result = S3.zero()
    for index_choice in itertools.product(itertools.permutations(range(3)),␣
↪repeat=5):
        sign = epsilon[index_choice[0]] * epsilon[index_choice[1]] *␣
↪epsilon[index_choice[2]] * epsilon[index_choice[3]] * epsilon[index_choice[4]]
        # NOTE: This assumes the ground vertex is labeled 0, the vertices of␣
↪out-degree 4 are labeled 1, 2, 3, and the Casimirs are labeled 4, 5, 6; 7, 8,␣
↪9.
        vertex_content = [E, S3(rho), S3(rho), S3(rho), S3(rho), S3(a),␣
↪S3(a), S3(a), S3(a), S3(a)]
        for ((source, target), index) in zip(g.edges(), sum(map(list,␣
↪index_choice), [])):
            vertex_content[target] = vertex_content[target].
↪derivative(even_coords[index])
        result += sign * prod(vertex_content)
    return result

print("Calculating X_vector_fields", flush=True)
X_vector_fields = []

with Pool(processes=14) as pool:
    X_vector_fields = list(pool.imap(evaluate_graph, X_graphs))

print("Calculated X_vector_fields", flush=True)

X_vector_fields.count(0)

[k+1 for (k, X) in enumerate(X_vector_fields) if X == 0]

X_monomial_basis = [set([]) for i in range(4)]
```

```
for i in range(4):
    for X in X_vector_fields:
        X_monomial_basis[i] |= set(X[i].monomials())
X_monomial_basis = [list(b) for b in X_monomial_basis]
X_monomial_count = sum(len(b) for b in X_monomial_basis)

# Next, we use this monomial basis to create a matrix that identifies each␣
 ↪vector field by the monomials that appear in it.
X_monomial_to_index = [{monomial : idx for (idx,monomial) in␣
 ↪enumerate(X_monomial_basis[j])} for j in range(4)]
X_evaluation_matrix = matrix(QQ, X_monomial_count, len(X_vector_fields))
for i in range(len(X_vector_fields)):
    v = vector(QQ, X_monomial_count)
    index_shift = 0
    for j in range(3):
        f = X_vector_fields[i][j]
        for coeff, monomial in zip(f.coefficients(), f.monomials()):
            monomial_index = X_monomial_to_index[j][monomial]
            v[index_shift + monomial_index] = coeff
        index_shift += len(X_monomial_basis[j])
    X_evaluation_matrix.set_column(i, v)

nullity = X_evaluation_matrix.right_nullity()
print('The vector fields have', nullity, 'linear relations among themselves.\n')

pivots = X_evaluation_matrix.pivots()
print("Maximal subset of linearly independent graphs:", list(pivots), flush=True)

X_lin_ind_formulas=[]
for i in pivots:
    X_lin_ind_formulas.append(X_vector_fields[i])

lin_ind_encodings=[]
for i in pivots:
    lin_ind_encodings.append(X_graph_encodings[i])

P = rho*epsilon.bracket(a)
if P.bracket(P)!=0:
    print('P is not a Poisson bivector. \n')

#GC = UndirectedGraphComplex(QQ, implementation='vector', sparse=True)
#fivewheel_cocycle = GC.cohomology_basis(6,10)[0];
#dGC = DirectedGraphComplex(QQ, implementation='vector')
#fivewheel_oriented = dGC(fivewheel_cocycle)
#fivewheel_oriented_filtered = fivewheel_oriented.filter(max_out_degree=2)
#fivewheel_operation = S3.graph_operation(fivewheel_oriented_filtered)
```

```
#Q_fivewheel= fivewheel_operation(P,P,P,P,P,P)
#P.bracket(Q_fivewheel)
file_path = 'q_fivewheel_3d.pickle'
Q_fivewheel=None
with open(file_path,'rb') as file:
    Q_fivewheel=pickle.load(file)

X_bivectors=[]
for X in X_vector_fields:
    X_bivectors.append(P.bracket(X))

# There are no nonzero (new) vector fields evaluated from 1 graph such that␣
 ↪[[P,X_{graph}]]=0.
zero_bivectors = X_bivectors.count(0)
print('There are', zero_bivectors, 'vector fields in X_vector_fields that␣
 ↪evaluate to 0 bivectors after taking the Schouten bracket with P .')
print('These vector fields that evaluate to 0 are obtained from the graphs:')
for (k,X) in enumerate(X_bivectors):
    if X==0:
        print('graph', k+1,)
print()

# Now, we extract the monomials appearing in these bivectors (as well as in␣
 ↪Q_fivewheel).
Q_monomial_basis={}
from itertools import combinations
for i,j in combinations(range(2),2):
    Q_monomial_basis[i,j]=set(Q_fivewheel[i,j].monomials())
    for P_X in X_bivectors:
        Q_monomial_basis[i,j]|= set(P_X[i,j].monomials())

Q_monomial_basis={idx: list(b) for idx, b in Q_monomial_basis.items()}
Q_monomial_index= {idx:{m:k for k,m in enumerate(b)} for idx, b in␣
 ↪Q_monomial_basis.items()}
Q_monomial_count=sum(len(b) for b in Q_monomial_basis.values());

# We create the vector representation of Q_fivewheel in terms of the monomials.
Q_fivewheel_vector= vector(QQ, Q_monomial_count, sparse=True)
index_shift=0
for i,j in Q_monomial_basis:
    for coeff, monomial in Q_fivewheel[i,j]:
        monomial_index= Q_monomial_index[i,j][monomial]
        Q_fivewheel_vector[monomial_index+index_shift]=coeff
    index_shift+=len(Q_monomial_basis[i,j])

X_bivector_evaluation_matrix= matrix(QQ, Q_monomial_count, len(X_bivectors),␣
 ↪sparse=True)
```

```
for k in range(len(X_bivectors)):
    P_X=X_bivectors[k]
    v=vector(QQ,Q_monomial_count, sparse=True)
    index_shift=0
    for i,j in Q_monomial_basis:
        for coeff, monomial in P_X[i,j]:
            monomial_index=Q_monomial_index[i,j][monomial]
            v[monomial_index+index_shift]=coeff
        index_shift+=len(Q_monomial_basis[i,j])
    X_bivector_evaluation_matrix.set_column(k,v)

# We soule the linear system.
X_solution=X_bivector_evaluation_matrix.solve_right(Q_fivewheel_vector)
print('The solution on vector fields is given by', X_solution,' \n')
```

Trivializing Q_{γ_5} in 3D (Descendants of all the 2D graphs Hamiltonian method)

November 25, 2024

[ ]:
```
from gcaops.graph.formality_graph import FormalityGraph
from gcaops.algebra.differential_polynomial_ring import␣
↪DifferentialPolynomialRing
from gcaops.algebra.superfunction_algebra import SuperfunctionAlgebra
from gcaops.graph.undirected_graph_complex import UndirectedGraphComplex
from gcaops.graph.directed_graph_complex import DirectedGraphComplex

import itertools
from multiprocessing import Pool
import pickle

X_graph_encodings=[]
for i1 in [1,2,3,4,5,7,8,9,10]:
    for index_choice_1 in itertools.combinations(range(10), int(2)):
        if index_choice_1[0]==1 or index_choice_1[1]==1 or␣
↪index_choice_1[0]+5==index_choice_1[1]:
            continue
        for index_choice_2 in itertools.combinations(range(10), int(2)):
            if index_choice_2[0]==2 or index_choice_2[1]==2 or␣
↪index_choice_2[0]+5==index_choice_2[1]:
                continue
            for index_choice_3 in itertools.combinations(range(10), int(2)):
                if index_choice_3[0]==3 or index_choice_3[1]==3 or␣
↪index_choice_3[0]+5==index_choice_3[1]:
                    continue
                for index_choice_4 in itertools.combinations(range(10), int(2)):
                    if index_choice_4[0]==4 or index_choice_4[1]==4 or␣
↪index_choice_4[0]+5==index_choice_4[1]:
                        continue
                    X_graph_encodings.append((0, i1, 6, index_choice_1[0]+1,␣
↪index_choice_1[1]+1, 7, index_choice_2[0]+1, index_choice_2[1]+1, 8,␣
↪index_choice_3[0]+1, index_choice_3[1]+1, 9, index_choice_4[0]+1,␣
↪index_choice_4[1]+1, 10))

X_graphs = [encoding_to_graph(e) for e in X_graph_encodings]
```

```
print("Number of graphs in X:", len(X_graphs), flush=True)

D3 = DifferentialPolynomialRing(QQ, ('rho','a'), ('x','y','z'),␣
↪max_differential_orders=[5+1,1+5+1])
rho, a = D3.fibre_variables()
x,y,z = D3.base_variables()
even_coords = [x,y,z]

S3.<xi0,xi1,xi2> = SuperfunctionAlgebra(D3, D3.base_variables()) #; S3
xi = S3.gens()
odd_coords = xi
epsilon = xi[0]*xi[1]*xi[2]

def evaluate_graph(g):
    E = x*xi[0] + y*xi[1] + z*xi[2] # Euler vector field, to insert into ground␣
↪vertex. Incoming derivative d/dx^i will result in xi[i].
    result = S3.zero()
    for index_choice in itertools.product(itertools.permutations(range(3)),␣
↪repeat=5):
        sign = epsilon[index_choice[0]] * epsilon[index_choice[1]] *␣
↪epsilon[index_choice[2]] * epsilon[index_choice[3]] * epsilon[index_choice[4]]
        # NOTE: This assumes the ground vertex is labeled 0, the vertices of␣
↪out-degree 4 are labeled 1, 2, 3, and the Casimirs are labeled 4, 5, 6; 7, 8,␣
↪9.
        vertex_content = [E, S3(rho), S3(rho), S3(rho), S3(rho), S3(a),␣
↪S3(a), S3(a), S3(a), S3(a)]
        for ((source, target), index) in zip(g.edges(), sum(map(list,␣
↪index_choice), [])):
            vertex_content[target] = vertex_content[target].
↪derivative(even_coords[index])
        result += sign * prod(vertex_content)
    return result

print("Calculating X_vector_fields", flush=True)
X_vector_fields = []

with Pool(processes=14) as pool:
    X_vector_fields = list(pool.imap(evaluate_graph, X_graphs))

print("Calculated X_vector_fields", flush=True)

X_vector_fields.count(0)

[k+1 for (k, X) in enumerate(X_vector_fields) if X == 0]

X_monomial_basis = [set([]) for i in range(4)]
```

```
for i in range(4):
    for X in X_vector_fields:
        X_monomial_basis[i] |= set(X[i].monomials())
X_monomial_basis = [list(b) for b in X_monomial_basis]
X_monomial_count = sum(len(b) for b in X_monomial_basis)

# Next, we use this monomial basis to create a matrix that identifies each␣
 ↪vector field by the monomials that appear in it.
X_monomial_to_index = [{monomial : idx for (idx,monomial) in␣
 ↪enumerate(X_monomial_basis[j])} for j in range(4)]
X_evaluation_matrix = matrix(QQ, X_monomial_count, len(X_vector_fields))
for i in range(len(X_vector_fields)):
    v = vector(QQ, X_monomial_count)
    index_shift = 0
    for j in range(3):
        f = X_vector_fields[i][j]
        for coeff, monomial in zip(f.coefficients(), f.monomials()):
            monomial_index = X_monomial_to_index[j][monomial]
            v[index_shift + monomial_index] = coeff
        index_shift += len(X_monomial_basis[j])
    X_evaluation_matrix.set_column(i, v)

nullity = X_evaluation_matrix.right_nullity()
print('The vector fields have', nullity, 'linear relations among themselves.\n')

pivots = X_evaluation_matrix.pivots()
print("Maximal subset of linearly independent graphs:", list(pivots), flush=True)

X_lin_ind_formulas=[]
for i in pivots:
    X_lin_ind_formulas.append(X_vector_fields[i])

lin_ind_encodings=[]
for i in pivots:
    lin_ind_encodings.append(X_graph_encodings[i])

P = rho*epsilon.bracket(a)
if P.bracket(P)!=0:
    print('P is not a Poisson bivector. \n')

#GC = UndirectedGraphComplex(QQ, implementation='vector', sparse=True)
#fivewheel_cocycle = GC.cohomology_basis(6,10)[0];
#dGC = DirectedGraphComplex(QQ, implementation='vector')
#fivewheel_oriented = dGC(fivewheel_cocycle)
#fivewheel_oriented_filtered = fivewheel_oriented.filter(max_out_degree=2)
#fivewheel_operation = S3.graph_operation(fivewheel_oriented_filtered)
```

```
#Q_fivewheel= fivewheel_operation(P,P,P,P,P,P)
#P.bracket(Q_fivewheel)
file_path = 'q_fivewheel_3d.pickle'
Q_fivewheel=None
with open(file_path,'rb') as file:
    Q_fivewheel=pickle.load(file)

X_bivectors=[]
for X in X_vector_fields:
    X_bivectors.append(P.bracket(X))

zero_bivectors = X_bivectors.count(0)
print('There are', zero_bivectors, 'vector fields in X_vector_fields that␣
 ↪evaluate to 0 bivectors after taking the Schouten bracket with P .')
print('These vector fields that evaluate to 0 are obtained from the graphs:')
for (k,X) in enumerate(X_bivectors):
    if X==0:
        print('graph', k+1,)
print()

# Now, we extract the monomials appearing in these bivectors (as well as in␣
 ↪Q_fivewheel).
Q_monomial_basis={}
from itertools import combinations
for i,j in combinations(range(2),2):
    Q_monomial_basis[i,j]=set(Q_fivewheel[i,j].monomials())
    for P_X in X_bivectors:
        Q_monomial_basis[i,j]|= set(P_X[i,j].monomials())

Q_monomial_basis={idx: list(b) for idx, b in Q_monomial_basis.items()}
Q_monomial_index= {idx:{m:k for k,m in enumerate(b)} for idx, b in␣
 ↪Q_monomial_basis.items()}
Q_monomial_count=sum(len(b) for b in Q_monomial_basis.values());

# We create the vector representation of Q_fivewheel in terms of the monomials.
Q_fivewheel_vector= vector(QQ, Q_monomial_count, sparse=True)
index_shift=0
for i,j in Q_monomial_basis:
    for coeff, monomial in Q_fivewheel[i,j]:
        monomial_index= Q_monomial_index[i,j][monomial]
        Q_fivewheel_vector[monomial_index+index_shift]=coeff
    index_shift+=len(Q_monomial_basis[i,j])

X_bivector_evaluation_matrix= matrix(QQ, Q_monomial_count, len(X_bivectors),␣
 ↪sparse=True)
for k in range(len(X_bivectors)):
    P_X=X_bivectors[k]
```

```
    v=vector(QQ,Q_monomial_count, sparse=True)
    index_shift=0
    for i,j in Q_monomial_basis:
        for coeff, monomial in P_X[i,j]:
            monomial_index=Q_monomial_index[i,j][monomial]
            v[monomial_index+index_shift]=coeff
        index_shift+=len(Q_monomial_basis[i,j])
    X_bivector_evaluation_matrix.set_column(k,v)

# We soule the linear system.
X_solution=X_bivector_evaluation_matrix.solve_right(Q_fivewheel_vector)
print('The solution on vector fields is given by', X_solution,' \n')
```

Finding $\vec{X}_{3D}^{\gamma_5}$: Brute force method

November 25, 2024

[ ]:
```
from gcaops.graph.formality_graph import FormalityGraph
from gcaops.algebra.differential_polynomial_ring import␣
↪DifferentialPolynomialRing
from gcaops.algebra.superfunction_algebra import SuperfunctionAlgebra
from gcaops.graph.undirected_graph_complex import UndirectedGraphComplex
from gcaops.graph.directed_graph_complex import DirectedGraphComplex

import itertools
import numpy as np
from multiprocessing import Pool

X_graph_encodings=[]
for i1 in [1,2,3,4,5,7,8,9,10]:
    for index_choice_1 in itertools.combinations(range(10), int(2)):
        if index_choice_1[0]==1 or index_choice_1[1]==1 or␣
↪index_choice_1[0]+5==index_choice_1[1]:
            continue
        for index_choice_2 in itertools.combinations(range(10), int(2)):
            if index_choice_2[0]==2 or index_choice_2[1]==2 or␣
↪index_choice_2[0]+5==index_choice_2[1]:
                continue
            for index_choice_3 in itertools.combinations(range(10), int(2)):
                if index_choice_3[0]==3 or index_choice_3[1]==3 or␣
↪index_choice_3[0]+5==index_choice_3[1]:
                    continue
                for index_choice_4 in itertools.combinations(range(10), int(2)):
                    if index_choice_4[0]==4 or index_choice_4[1]==4 or␣
↪index_choice_4[0]+5==index_choice_4[1]:
                        continue
                    X_graph_encodings.append((0, i1, 6, index_choice_1[0]+1,␣
↪index_choice_1[1]+1, 7, index_choice_2[0]+1, index_choice_2[1]+1, 8,␣
↪index_choice_3[0]+1, index_choice_3[1]+1, 9, index_choice_4[0]+1,␣
↪index_choice_4[1]+1, 10))

def encoding_to_graph(encoding):
    targets = [encoding[0:3], encoding[3:6], encoding[6:9], encoding[9:12],␣
↪encoding[12:15]]
```

```
    edges = sum([[(k+1,v) for v in t] for (k,t) in enumerate(targets)], [])
    return FormalityGraph(1, 10, edges)

X_graphs = []
with Pool(processes=32) as pool:
    X_graphs = list(pool.imap(encoding_to_graph, X_graph_encodings))
print('We have', len(X_graphs), 'graphs.\n')

X_graphs_iso={}
X_encodings_iso=[]
i=-1
for g in X_graphs:
    i+= 1
    h=tuple(g.canonical_form().edges())
    if not h in X_graphs_iso:
        X_graphs_iso[h]=g
        X_encodings_iso.append(X_graph_encodings[i])
X_graphs_iso=list(X_graphs_iso.values())

if len(X_graphs_iso)!= len(X_encodings_iso):
    print('There is a mistake computing the encodings corresponding to the␣
 ↪nonisomorphic graphs.')

print ('There are', len(X_graphs_iso), 'nonisomorphic graphs.\n')

D3 = DifferentialPolynomialRing(QQ, ('rho','a'), ('x','y','z'),␣
 ↪max_differential_orders=[5+1,5+1])
rho, a = D3.fibre_variables()
x,y,z = D3.base_variables()
even_coords = [x,y,z]

S3.<xi0,xi1,xi2> = SuperfunctionAlgebra(D3, D3.base_variables())
xi = S3.gens()
odd_coords = xi

epsilon = xi[0]*xi[1]*xi[2]
E = x*xi[0] + y*xi[1] + z*xi[2]
def evaluate_graph(g):
    result = S3.zero()
    for index_choice in itertools.product(itertools.permutations(range(3)),␣
 ↪repeat=5):
        sign = epsilon[index_choice[0]] * epsilon[index_choice[1]] *␣
 ↪epsilon[index_choice[2]] * epsilon[index_choice[3]] * epsilon[index_choice[4]]
        vertex_content = [E, S3(rho), S3(rho), S3(rho), S3(rho), S3(a),␣
 ↪S3(a), S3(a), S3(a), S3(a)]
        for ((source, target), index) in zip(g.edges(), sum(map(list,␣
 ↪index_choice), [])):
```

```
            vertex_content[target] = vertex_content[target].
 ↪derivative(even_coords[index])
        result += sign * prod(vertex_content)
    return result

X_vector_fields = []
with Pool(processes=32) as pool:
    X_vector_fields = list(pool.imap(evaluate_graph, X_graphs_iso))

zeros=X_vector_fields.count(0)
print('There are', zeros, 'graphs that evaluate to 0 under the morhpism from␣
 ↪graphs to multivectors.')

X_monomial_basis = [set([]) for i in range(3)]
for i in range(3):
    for X in X_vector_fields:
        X_monomial_basis[i] |= set(X[i].monomials())
X_monomial_basis = [list(b) for b in X_monomial_basis]
X_monomial_count = sum(len(b) for b in X_monomial_basis)

X_evaluation_matrix = matrix(QQ, X_monomial_count, len(X_vector_fields))
for i in range(len(X_vector_fields)):
    v = vector(QQ, X_monomial_count)
    index_shift = 0
    for j in range(3):
        f = X_vector_fields[i][j]
        for coeff, monomial in zip(f.coefficients(), f.monomials()):
            monomial_index = X_monomial_basis[j].index(monomial)
            v[index_shift + monomial_index] = coeff
        index_shift += len(X_monomial_basis[j])
    X_evaluation_matrix.set_column(i, v)

nullity = X_evaluation_matrix.right_nullity()
print('The vector fields have', nullity, 'linear relations among themselves.\n')

pivots = X_evaluation_matrix.pivots()

lin_ind_vector_fields=[]
lin_ind_encodings=[]
for i in pivots:
    lin_ind_vector_fields.append(X_vector_fields[i])
    lin_ind_encodings.append(X_encodings_iso[i])

print('We have', len(lin_ind_vector_fields), 'linearly independent vector fields.
 ↪\n')

P= (rho*epsilon).bracket(a)
```

```
if P.bracket(P)!=0:
    print('P is not a Poisson bivector. \n')

GC = UndirectedGraphComplex(QQ, implementation='vector', sparse=True)
fivewheel_cocycle = GC.cohomology_basis(6,10)[0]; fivewheel_cocycle
dGC = DirectedGraphComplex(QQ, implementation='vector')
fivewheel_oriented = dGC(fivewheel_cocycle)
fivewheel_oriented_filtered = fivewheel_oriented.filter(max_out_degree=2)
fivewheel_operation = S3.graph_operation(fivewheel_oriented_filtered)
Q_fivewheel= fivewheel_operation(P,P,P,P,P,P)

X_bivectors=[]
for X in X_lin_ind_vector_fields:
    X_bivectors.append(P.bracket(X))

zero_bivectors = X_bivectors.count(0)
print('There are', zero_bivectors, 'vector fields in lin_ind_vector_fields that␣
 ↪evaluate to 0 bivectors after taking the Schouten bracket with P .')

Q_monomial_basis={}
from itertools import combinations
for i,j in combinations(range(2),2):
    Q_monomial_basis[i,j]=set(Q_fivewheel[i,j].monomials())
    for P_X in X_bivectors:
        Q_monomial_basis[i,j]|= set(P_X[i,j].monomials())

Q_monomial_basis={idx: list(b) for idx, b in Q_monomial_basis.items()}
Q_monomial_index= {idx:{m:k for k,m in enumerate(b)} for idx, b in␣
 ↪Q_monomial_basis.items()}
Q_monomial_count=sum(len(b) for b in Q_monomial_basis.values());

Q_fivewheel_vector= vector(QQ, Q_monomial_count, sparse=True)
index_shift=0
for i,j in Q_monomial_basis:
    for coeff, monomial in Q_fivewheel[i,j]:
        monomial_index= Q_monomial_index[i,j][monomial]
        Q_fivewheel_vector[monomial_index+index_shift]=coeff
    index_shift+=len(Q_monomial_basis[i,j])

# We create the matrix that represents the X_bivectors in terms of the monomials.
X_bivector_evaluation_matrix= matrix(QQ, Q_monomial_count, len(X_bivectors),␣
 ↪sparse=True)
for k in range(len(X_bivectors)):
    P_X=X_bivectors[k]
    v=vector(QQ,Q_monomial_count, sparse=True)
    index_shift=0
    for i,j in Q_monomial_basis:
```

```
        for coeff, monomial in P_X[i,j]:
            monomial_index=Q_monomial_index[i,j][monomial]
            v[monomial_index+index_shift]=coeff
        index_shift+=len(Q_monomial_basis[i,j])
    X_bivector_evaluation_matrix.set_column(k,v)

X_solution=X_bivector_evaluation_matrix.solve_right(Q_fivewheel_vector)

length_sol=np.nonzero(X_solution)
print('The are in total', length_sol[0].size, 'graph appearing in the solution.␣
 ↪The following graphs appear in the solution', length_sol, 'with the␣
 ↪coefficients \n')
for i in range(length_sol[0].size):
    print(X_solution[length_sol[0][i]])
print('The corresponding graph encodings are \n')
for i in range(length_sol[0].size):
    print(lin_ind_encodings[length_sol[0][i]])

X_cocycle_space= X_bivector_evaluation_matrix.right_kernel().
 ↪quotient(X_evaluation_matrix.right_kernel())
X_cocycles=[X_cocycle_space.lift(v) for v in X_cocycle_space.basis()]
if all(X_bivector_evaluation_matrix*X_cocycle==0 for X_cocycle in X_cocycles)!=␣
 ↪True:
    print('There is an error in the cocycle space.')
print('The space of solutions to the homogeneous system has dimension',␣
 ↪X_cocycle_space.dimension(), )
```

Linear relations of the 4D Hamiltonians

November 24, 2024

[1]:
```
NPROCS=14

hamiltonian_encodings= [(0, 1, 2, 4, 0, 1, 3, 5), (0, 1, 2, 4,  1, 2, 3, 5), (0,␣
 ↪1, 2, 4, 1, 3, 4, 5), (0, 2, 3, 4, 1, 2, 3, 5), (0, 2, 3, 4, 1, 3, 4, 5), (0,␣
 ↪2, 4, 5, 1, 3, 4, 5), (0, 1, 2, 4, 0, 2, 3, 5), (0, 1, 2, 4, 0, 3, 4, 5), (0,␣
 ↪1, 2, 4, 2, 3, 4, 5), (0, 2, 3, 4, 0, 2, 3, 5), (0, 2, 4, 5, 0, 2, 3, 5), (0,␣
 ↪2, 3, 4, 0, 3, 4, 5), (0, 2, 4, 5, 0, 3, 4, 5), (0, 2, 3, 4, 2, 3, 4, 5), (0,␣
 ↪2, 4, 5, 2, 3, 4, 5), (1, 2, 3, 4, 0, 2, 3, 5), (1, 2, 4, 5, 0, 3, 4, 5), (1,␣
 ↪2, 3, 4, 0, 3, 4, 5), (1, 2, 3, 4, 2, 3, 4, 5), (1, 2, 4, 5, 2, 3, 4, 5), (2,␣
 ↪3, 4, 5, 2, 3, 4, 5)]

from gcaops.graph.formality_graph import FormalityGraph
def encoding_to_graph(encoding):
    targets = [encoding[0:4], encoding[4:8]]
    edges = sum([[(k+1,v) for v in t] for (k,t) in enumerate(targets)], [])
    return FormalityGraph(0, 6, edges)

hamiltonian_graphs = [encoding_to_graph(e) for e in hamiltonian_encodings]
print("Number of 4D Hamiltonians:", len(hamiltonian_graphs), flush=True)


from gcaops.algebra.differential_polynomial_ring import␣
 ↪DifferentialPolynomialRing
from gcaops.algebra.superfunction_algebra import SuperfunctionAlgebra

D4 = DifferentialPolynomialRing(QQ, ('rho','a1','a2'), ('x','y','z','w'),␣
 ↪max_differential_orders=[3+1,1+3+1,1+3+1])
rho, a1, a2 = D4.fibre_variables()
x,y,z,w= D4.base_variables()
even_coords = [x,y,z,w]

S4.<xi0,xi1,xi2,xi3> = SuperfunctionAlgebra(D4, D4.base_variables())
xi = S4.gens()
odd_coords = xi
epsilon = xi[0]*xi[1]*xi[2]*xi[3]
```

```
import itertools
from multiprocessing import Pool

def evaluate_graph(g):
    E = x*xi[0] + y*xi[1] + z*xi[2] + w*xi[3]
    result = S4.zero()
    for index_choice in itertools.product(itertools.permutations(range(4)),␣
 ↪repeat=2):
        sign = epsilon[index_choice[0]] * epsilon[index_choice[1]]
        vertex_content = [S4(rho), S4(rho), S4(a1), S4(a1), S4(a2), S4(a2)]
        for ((source, target), index) in zip(g.edges(), sum(map(list,␣
 ↪index_choice), [])):
            vertex_content[target] = vertex_content[target].
 ↪derivative(even_coords[index])
        result += sign * prod(vertex_content)
    return result

print("Calculating 4D hamiltonian_formulas", flush=True)
hamiltonian_formulas = []
with Pool(processes=NPROCS) as pool:
    hamiltonian_formulas = list(pool.imap(evaluate_graph, hamiltonian_graphs))
print("Calculated 4D hamiltonian_formulas", flush=True)

hamiltonian_formulas.count(0)
[k for (k, ham) in enumerate(hamiltonian_formulas) if ham == 0]

from itertools import combinations
for i,j in combinations(range(len(hamiltonian_formulas)),2):
    if i!=j and hamiltonian_formulas[i]==hamiltonian_formulas[j]:
        print("4D Hamiltonians", i,j, "share the same formula")

for i,j in combinations(range(len(hamiltonian_formulas)),2):
    if i!=j and hamiltonian_formulas[i]==-hamiltonian_formulas[j]:
        print("4D Hamiltonians", i,j, "share the same formula with a - sign")
```

```
Number of 4D Hamiltonians: 21
Calculating 4D hamiltonian_formulas
Calculated 4D hamiltonian_formulas
4D Hamiltonians 3 15 share the same formula
4D Hamiltonians 4 17 share the same formula
4D Hamiltonians 5 16 share the same formula
4D Hamiltonians 10 11 share the same formula
4D Hamiltonians 13 18 share the same formula
4D Hamiltonians 14 19 share the same formula
4D Hamiltonians 1 6 share the same formula with a - sign
4D Hamiltonians 2 7 share the same formula with a - sign
```

Linear relations of the 4D Hamiltonians embedded to 5D

November 24, 2024

[1]:
```
NPROCS=14

hamiltonian_encodings= [(0, 1, 2, 4, 6, 0, 1, 3, 5, 7), (0, 1, 2, 4, 6, 1, 2, 3,␣
 ↪5, 7), (0, 1, 2, 4, 6, 1, 3, 4, 5, 7), (0, 2, 3, 4, 6, 1, 2, 3, 5, 7), (0, 2,␣
 ↪3, 4, 6, 1, 3, 4, 5, 7), (0, 2, 4, 5, 6, 1, 3, 4, 5, 7), (0, 1, 2, 4, 6, 0, 2,␣
 ↪3, 5, 7), (0, 1, 2, 4, 6, 0, 3, 4, 5, 7), (0, 1, 2, 4, 6, 2, 3, 4, 5, 7), (0,␣
 ↪2, 3, 4, 6, 0, 2, 3, 5, 7), (0, 2, 4, 5, 6, 0, 2, 3, 5, 7), (0, 2, 3, 4, 6, 0,␣
 ↪3, 4, 5, 7), (0, 2, 4, 5, 6,  0, 3, 4, 5, 7), (0, 2, 3, 4, 6, 2, 3, 4, 5, 7),␣
 ↪(0, 2, 4, 5, 6, 2, 3, 4, 5, 7), (1, 2, 3, 4, 6, 0, 2, 3, 5, 7), (1, 2, 4, 5,␣
 ↪6, 0, 3, 4, 5, 7), (1, 2, 3, 4, 6, 0, 3, 4, 5, 7), (1, 2, 3, 4, 6, 2, 3, 4, 5,␣
 ↪7), (1, 2, 4, 5, 6, 2, 3, 4, 5, 7), (2, 3, 4, 5, 6, 2, 3, 4, 5, 7)]

from gcaops.graph.formality_graph import FormalityGraph
def encoding_to_graph(encoding):
    targets = [encoding[0:5], encoding[5:10]]
    edges = sum([[(k+1,v) for v in t] for (k,t) in enumerate(targets)], [])
    return FormalityGraph(0, 8, edges)

hamiltonian_graphs = [encoding_to_graph(e) for e in hamiltonian_encodings]
print("Number of 5D Hamiltonians:", len(hamiltonian_graphs), flush=True)

from gcaops.algebra.differential_polynomial_ring import␣
 ↪DifferentialPolynomialRing
from gcaops.algebra.superfunction_algebra import SuperfunctionAlgebra

D5 = DifferentialPolynomialRing(QQ, ('rho','a1','a2', 'a3'), ('x','y','z','w',␣
 ↪'v'), max_differential_orders=[3+1,1+3+1,1+3+1, 1+3+1])
rho, a1, a2, a3 = D5.fibre_variables()
x,y,z,w, v = D5.base_variables()
even_coords = [x,y,z,w,v]

S5.<xi0,xi1,xi2,xi3,xi4> = SuperfunctionAlgebra(D5, D5.base_variables())
xi = S5.gens()
odd_coords = xi
epsilon = xi[0]*xi[1]*xi[2]*xi[3]*xi[4]

import itertools
```

```
from multiprocessing import Pool

def evaluate_graph(g):
    E = x*xi[0] + y*xi[1] + z*xi[2] + w*xi[3] + v*xi[4]
    result = S5.zero()
    for index_choice in itertools.product(itertools.permutations(range(5)),␣
 ↪repeat=2):
        sign = epsilon[index_choice[0]] * epsilon[index_choice[1]]
        vertex_content = [S5(rho), S5(rho), S5(a1), S5(a1), S5(a2), S5(a2),␣
 ↪S5(a3), S5(a3)]
        for ((source, target), index) in zip(g.edges(), sum(map(list,␣
 ↪index_choice), [])):
            vertex_content[target] = vertex_content[target].
 ↪derivative(even_coords[index])
        result += sign * prod(vertex_content)
    return result

print("Calculating 5D hamiltonian_formulas", flush=True)
hamiltonian_formulas = []
with Pool(processes=NPROCS) as pool:
    hamiltonian_formulas = list(pool.imap(evaluate_graph, hamiltonian_graphs))
print("Calculated 5D hamiltonian_formulas", flush=True)

hamiltonian_formulas.count(0)
[k for (k, ham) in enumerate(hamiltonian_formulas) if ham == 0]

from itertools import combinations
for i,j in combinations(range(len(hamiltonian_formulas)),2):
    if i!=j and hamiltonian_formulas[i]==hamiltonian_formulas[j]:
        print("5D Hamiltonians", i,j, "share the same formula")

for i,j in combinations(range(len(hamiltonian_formulas)),2):
    if i!=j and hamiltonian_formulas[i]==-hamiltonian_formulas[j]:
        print("5D Hamiltonians", i,j, "share the same formula with a - sign")
```

```
Number of 5D Hamiltonians: 21
Calculating 5D hamiltonian_formulas
Calculated 5D hamiltonian_formulas
5D Hamiltonians 3 15 share the same formula
5D Hamiltonians 4 17 share the same formula
5D Hamiltonians 5 16 share the same formula
5D Hamiltonians 10 11 share the same formula
5D Hamiltonians 13 18 share the same formula
5D Hamiltonians 14 19 share the same formula
5D Hamiltonians 1 6 share the same formula with a - sign
5D Hamiltonians 2 7 share the same formula with a - sign
```

Vanishing graph in 3D

November 24, 2024

[1]:
```
encoding= [0,3,4,3,4,5,1,2,6]
from gcaops.graph.formality_graph import FormalityGraph
def encoding_to_graph(encoding):
    targets = [encoding[0:3], encoding[3:6],encoding[6:9]]
    edges = sum([[(k+1,v) for v in t] for (k,t) in enumerate(targets)], [])
    return FormalityGraph(1, 7, edges)

graph = encoding_to_graph(encoding)

from gcaops.algebra.differential_polynomial_ring import␣
↪DifferentialPolynomialRing
from gcaops.algebra.superfunction_algebra import SuperfunctionAlgebra

D3 = DifferentialPolynomialRing(QQ, ('rho','a1'), ('x','y','z'),␣
↪max_differential_orders=[3+1,1+3+1])
rho, a1 = D3.fibre_variables()
x,y,z= D3.base_variables()
even_coords = [x,y,z]

S3.<xi0,xi1,xi2> = SuperfunctionAlgebra(D3, D3.base_variables())
xi = S3.gens()
odd_coords = xi
epsilon = xi[0]*xi[1]*xi[2]


import itertools
from multiprocessing import Pool

def evaluate_graph(g):
    E = x*xi[0] + y*xi[1] + z*xi[2]
    result = S3.zero()
    for index_choice in itertools.product(itertools.permutations(range(3)),␣
↪repeat=3):
        sign = epsilon[index_choice[0]] * epsilon[index_choice[1]] *␣
↪epsilon[index_choice[2]]
        vertex_content = [E, S3(rho), S3(rho), S3(rho), S3(a1), S3(a1), S3(a1)]
```

```
        for ((source, target), index) in zip(g.edges(), sum(map(list,␣
↪index_choice), [])):
            vertex_content[target] = vertex_content[target].
↪derivative(even_coords[index])
        result += sign * prod(vertex_content)
    return result

vector_field = evaluate_graph(graph)

print('The 3D vector field evaluates to', vector_field, '.')
```

```
The vector field evaluates to 0 .
```

Vanishing graph in 3D embedded to 4D

November 24, 2024

[1]:
```
encoding= [0,3,4,7,3,4,5,8,1,2,6,9]
from gcaops.graph.formality_graph import FormalityGraph
def encoding_to_graph(encoding):
    targets = [encoding[0:4], encoding[4:8],encoding[8:12]]
    edges = sum([[(k+1,v) for v in t] for (k,t) in enumerate(targets)], [])
    return FormalityGraph(1, 10, edges)

graph = encoding_to_graph(encoding)

from gcaops.algebra.differential_polynomial_ring import␣
↪DifferentialPolynomialRing
from gcaops.algebra.superfunction_algebra import SuperfunctionAlgebra

D4 = DifferentialPolynomialRing(QQ, ('rho','a1', 'a2'), ('x','y','z', 'w'),␣
↪max_differential_orders=[3+1,1+3+1,1+3+1])
rho, a1, a2 = D4.fibre_variables()
x,y,z,w= D4.base_variables()
even_coords = [x,y,z,w]

S4.<xi0,xi1,xi2,xi3> = SuperfunctionAlgebra(D4, D4.base_variables())
xi = S4.gens()
odd_coords = xi
epsilon = xi[0]*xi[1]*xi[2]*xi[3]


import itertools
from multiprocessing import Pool

def evaluate_graph(g):
    E = x*xi[0] + y*xi[1] + z*xi[2] +w*xi[3]
    result = S4.zero()
    for index_choice in itertools.product(itertools.permutations(range(4)),␣
↪repeat=3):
        sign = epsilon[index_choice[0]] * epsilon[index_choice[1]] *␣
↪epsilon[index_choice[2]]
        vertex_content = [E, S4(rho), S4(rho), S4(rho), S4(a1), S4(a1), S4(a1),␣
↪S4(a2), S4(a2), S4(a2)]
```

```
        for ((source, target), index) in zip(g.edges(), sum(map(list,␣
↪index_choice), [])):
            vertex_content[target] = vertex_content[target].
↪derivative(even_coords[index])
        result += sign * prod(vertex_content)
    return result

vector_field = evaluate_graph(graph)

print('The 4D vector field evaluates to', vector_field, '.')
```

```
The 4D vector field evaluates to 0 .
```

Linear relation of 3 3D graphs

November 24, 2024

[1]:
```
encodings= [[0,1,4,1,6,5,4,2,6], [0,2,4,1,6,5,1,5,6], [0,5,4,1,6,5,1,2,6]]
from gcaops.graph.formality_graph import FormalityGraph
def encoding_to_graph(encoding):
    targets = [encoding[0:3], encoding[3:6],encoding[6:9]]
    edges = sum([[(k+1,v) for v in t] for (k,t) in enumerate(targets)], [])
    return FormalityGraph(1, 7, edges)

graphs = [encoding_to_graph(encodings[0]), encoding_to_graph(encodings[1]),␣
 ↪encoding_to_graph(encodings[2])]

from gcaops.algebra.differential_polynomial_ring import␣
 ↪DifferentialPolynomialRing
from gcaops.algebra.superfunction_algebra import SuperfunctionAlgebra

D3 = DifferentialPolynomialRing(QQ, ('rho','a1'), ('x','y','z'),␣
 ↪max_differential_orders=[3+1,1+3+1])
rho, a1 = D3.fibre_variables()
x,y,z= D3.base_variables()
even_coords = [x,y,z]

S3.<xi0,xi1,xi2> = SuperfunctionAlgebra(D3, D3.base_variables())
xi = S3.gens()
odd_coords = xi
epsilon = xi[0]*xi[1]*xi[2]

import itertools
def evaluate_graph(g):
    E = x*xi[0] + y*xi[1] + z*xi[2]
    result = S3.zero()
    for index_choice in itertools.product(itertools.permutations(range(3)),␣
 ↪repeat=3):
        sign = epsilon[index_choice[0]] * epsilon[index_choice[1]] *␣
 ↪epsilon[index_choice[2]]
        vertex_content = [E, S3(rho), S3(rho), S3(rho), S3(a1), S3(a1), S3(a1)]
        for ((source, target), index) in zip(g.edges(), sum(map(list,␣
 ↪index_choice), [])):
```

```
            vertex_content[target] = vertex_content[target].
 ↪derivative(even_coords[index])
        result += sign * prod(vertex_content)
    return result

vector_fields = [evaluate_graph(graphs[0]),evaluate_graph(graphs[1]),␣
 ↪evaluate_graph(graphs[2])]
vector_field=vector_fields[0]+1/2*vector_fields[1]-1*vector_fields[2]

print('The linear combination of the 3 3D vector fields with coefficients 1,1/
 ↪2,-1 evaluates to',vector_field, '.')
```

```
The linear combination of the 3 3D vector fields with coefficients 1,1/2,-1
evaluates to 0 .
```

Linear relation of 3 3D graphs embedded to 4D

November 24, 2024

[1]:
```
encodings= [[0,1,4,7,1,6,5,8,4,2,6,9], [0,2,4,7,1,6,5,8,1,5,6,9],␣
↪[0,5,4,7,1,6,5,8,1,2,6,9]]
from gcaops.graph.formality_graph import FormalityGraph
def encoding_to_graph(encoding):
    targets = [encoding[0:4], encoding[4:8],encoding[8:12]]
    edges = sum([[(k+1,v) for v in t] for (k,t) in enumerate(targets)], [])
    return FormalityGraph(1, 10, edges)

graphs = [encoding_to_graph(encodings[0]), encoding_to_graph(encodings[1]),␣
↪encoding_to_graph(encodings[2])]

from gcaops.algebra.differential_polynomial_ring import␣
↪DifferentialPolynomialRing
from gcaops.algebra.superfunction_algebra import SuperfunctionAlgebra

D4 = DifferentialPolynomialRing(QQ, ('rho','a1', 'a2'), ('x','y','z', 'w'),␣
↪max_differential_orders=[3+1,1+3+1,1+3+1])
rho, a1, a2 = D4.fibre_variables()
x,y,z,w= D4.base_variables()
even_coords = [x,y,z,w]

S4.<xi0,xi1,xi2,xi3> = SuperfunctionAlgebra(D4, D4.base_variables())
xi = S4.gens()
odd_coords = xi
epsilon = xi[0]*xi[1]*xi[2]*xi[3]

import itertools

def evaluate_graph(g):
    E = x*xi[0] + y*xi[1] + z*xi[2] +w*xi[3]
    result = S4.zero()
    for index_choice in itertools.product(itertools.permutations(range(4)),␣
↪repeat=3):
        sign = epsilon[index_choice[0]] * epsilon[index_choice[1]] *␣
↪epsilon[index_choice[2]]
        vertex_content = [E, S4(rho), S4(rho), S4(rho), S4(a1), S4(a1), S4(a1),␣
↪S4(a2), S4(a2), S4(a2)]
```

```
        for ((source, target), index) in zip(g.edges(), sum(map(list,␣
↪index_choice), [])):
            vertex_content[target] = vertex_content[target].
↪derivative(even_coords[index])
        result += sign * prod(vertex_content)
    return result

vector_fields = [evaluate_graph(graphs[0]),evaluate_graph(graphs[1]),␣
↪evaluate_graph(graphs[2])]
vector_field=vector_fields[0]+1/2*vector_fields[1]-1*vector_fields[2]

print('The linear combination of the 3 4D vector fields with coefficients 1,1/
↪2,-1 evaluates to',vector_field, '.')
```

```
The linear combination of the 3 4D vector fields with coefficients 1,1/2,-1
evaluates to 0 .
```

Non-isomorphic Hamiltonians in 2D, 3D and 4D for γ_3, γ_5 and γ_7

November 25, 2024

[1]:
```
%%time
from gcaops.graph.formality_graph import FormalityGraph
import itertools
hamiltonian_encodings=[]
for index_choice_1 in itertools.product(itertools.combinations(range(2),␣
↪int(2)), repeat=1):
    for index_choice_2 in itertools.product(itertools.combinations(range(2),␣
↪int(2)), repeat=1):
        hamiltonian_encodings.append((index_choice_1[0][0],␣
↪index_choice_1[0][1], index_choice_2[0][0], index_choice_2[0][1]))

def ham_encoding_to_graph(encoding):
    targets = [encoding[0:2], encoding[2:4]]
    edges = sum([[(k,v) for v in t] for (k,t) in enumerate(targets)], [])
    return FormalityGraph(0, 2, edges)

ham_graphs= []
for encoding in hamiltonian_encodings:
    ham_graphs.append(ham_encoding_to_graph(encoding))

ham_graphs_iso={}
for g in ham_graphs:
    h=tuple(g.canonical_form().edges())
    if not h in ham_graphs_iso:
        ham_graphs_iso[h]=g
ham_graphs_iso=list(ham_graphs_iso.values())
print ('There are', len(ham_graphs_iso), 'nonisomorphic 2D Hamiltonian graphs␣
↪for gamma_3.\n')
```

```
There are 1 nonisomorphic 2D Hamiltonian graphs for gamma_3.

CPU times: user 5.44 ms, sys: 861 µs, total: 6.3 ms
Wall time: 7.3 ms
```

[2]:
```
%%time
hamiltonian_encodings=[]
for index_choice_1 in itertools.product(itertools.combinations(range(4),␣
↪int(2)), repeat=1):
```

```
    if index_choice_1[0][0]==2 or index_choice_1[0][1]==2:
        continue
    for index_choice_2 in itertools.product(itertools.combinations(range(4),␣
 ↪int(2)), repeat=1):
        if index_choice_2[0][0]==3 or index_choice_2[0][1]==3:
            continue
        hamiltonian_encodings.append((2,index_choice_1[0][0],␣
 ↪index_choice_1[0][1], 3, index_choice_2[0][0], index_choice_2[0][1]))

def ham_encoding_to_graph(encoding):
    targets = [encoding[0:3], encoding[3:6]]
    edges = sum([[(k,v) for v in t] for (k,t) in enumerate(targets)], [])
    return FormalityGraph(0, 4, edges)

ham_graphs= []
for encoding in hamiltonian_encodings:
    ham_graphs.append(ham_encoding_to_graph(encoding))
ham_graphs_iso={}
for g in ham_graphs:
    h=tuple(g.canonical_form().edges())
    if not h in ham_graphs_iso:
        ham_graphs_iso[h]=g
ham_graphs_iso=list(ham_graphs_iso.values())
print ('There are', len(ham_graphs_iso), 'nonisomorphic 3D Hamiltonian graphs␣
 ↪for gamma_3.\n')
```

```
There are 6 nonisomorphic 3D Hamiltonian graphs for gamma_3.

CPU times: user 3.18 ms, sys: 500 µs, total: 3.68 ms
Wall time: 3.6 ms
```

[3]:
```
%%time
hamiltonian_encodings=[]
for index_choice_1 in itertools.product(itertools.combinations(range(6),␣
 ↪int(2)), repeat=1):
    if index_choice_1[0][0]==2 or index_choice_1[0][1]==2 or␣
 ↪index_choice_1[0][0]==4 or index_choice_1[0][1]==4:
        continue
    for index_choice_2 in itertools.product(itertools.combinations(range(6),␣
 ↪int(2)), repeat=1):
        if index_choice_2[0][0]==3 or index_choice_2[0][1]==3 or␣
 ↪index_choice_2[0][0]==5 or index_choice_2[0][1]==5:
            continue
        hamiltonian_encodings.append((2, 4, index_choice_1[0][0],␣
 ↪index_choice_1[0][1], 3,5, index_choice_2[0][0], index_choice_2[0][1]))

def ham_encoding_to_graph(encoding):
```

```
    targets = [encoding[0:4], encoding[4:8]]
    edges = sum([[(k,v) for v in t] for (k,t) in enumerate(targets)], [])
    return FormalityGraph(0, 6, edges)

ham_graphs= []
for encoding in hamiltonian_encodings:
    ham_graphs.append(ham_encoding_to_graph(encoding))

ham_graphs_iso={}
for g in ham_graphs:
    h=tuple(g.canonical_form(aerial_vertex_partition=[[0,1], [2,3], [4,5]]).
 ↪edges())
    if not h in ham_graphs_iso:
        ham_graphs_iso[h]=g
ham_graphs_iso=list(ham_graphs_iso.values())
print ('There are', len(ham_graphs_iso), 'nonisomorphic 4D Hamiltonian graphs␣
 ↪for gamma_3.\n')
```

```
There are 21 nonisomorphic 4D Hamiltonian graphs for gamma_3.

CPU times: user 12.8 ms, sys: 400 µs, total: 13.2 ms
Wall time: 12.7 ms
```

[4]:
```
%%time
from gcaops.graph.formality_graph import FormalityGraph
import itertools
hamiltonian_encodings=[]
for index_choice_1 in itertools.product(itertools.combinations(range(4),␣
 ↪int(2)), repeat=1):
    for index_choice_2 in itertools.product(itertools.combinations(range(4),␣
 ↪int(2)), repeat=1):
        for index_choice_3 in itertools.product(itertools.combinations(range(4),␣
 ↪int(2)), repeat=1):
            for index_choice_4 in itertools.product(itertools.
 ↪combinations(range(4), int(2)), repeat=1):
                hamiltonian_encodings.append((index_choice_1[0][0],␣
 ↪index_choice_1[0][1], index_choice_2[0][0], index_choice_2[0][1],␣
 ↪index_choice_3[0][0], index_choice_3[0][1], index_choice_4[0][0],␣
 ↪index_choice_4[0][1]))

def ham_encoding_to_graph(encoding):
    targets = [encoding[0:2], encoding[2:4], encoding[4:6], encoding[6:8]]
    edges = sum([[(k,v) for v in t] for (k,t) in enumerate(targets)], [])
    return FormalityGraph(0, 4, edges)

ham_graphs= []
for encoding in hamiltonian_encodings:
```

```
    ham_graphs.append(ham_encoding_to_graph(encoding))

ham_graphs_iso={}
for g in ham_graphs:
    h=tuple(g.canonical_form().edges())
    if not h in ham_graphs_iso:
        ham_graphs_iso[h]=g
ham_graphs_iso=list(ham_graphs_iso.values())
print ('There are', len(ham_graphs_iso), 'nonisomorphic 2D Hamiltonian graphs␣
 ↪for gamma_5.\n')
```

```
There are 66 nonisomorphic 2D Hamiltonian graphs for gamma_5.

CPU times: user 360 ms, sys: 19.3 ms, total: 379 ms
Wall time: 378 ms
```

[5]:
```
%%time
from gcaops.graph.formality_graph import FormalityGraph
import itertools
hamiltonian_encodings=[]
for index_choice_1 in itertools.product(itertools.combinations(range(8),␣
 ↪int(2)), repeat=1):
    if index_choice_1[0][0]==4 or index_choice_1[0][1]==4 :
        continue
    for index_choice_2 in itertools.product(itertools.combinations(range(8),␣
 ↪int(2)), repeat=1):
        if index_choice_2[0][0]==5 or index_choice_2[0][1]==5 :
            continue
        for index_choice_3 in itertools.product(itertools.combinations(range(8),␣
 ↪int(2)), repeat=1):
            if index_choice_3[0][0]==6 or index_choice_3[0][1]==6 :
                continue
            for index_choice_4 in itertools.product(itertools.
 ↪combinations(range(8), int(2)), repeat=1):
                if index_choice_4[0][0]==7 or index_choice_4[0][1]==7 :
                    continue
                hamiltonian_encodings.append((4, index_choice_1[0][0],␣
 ↪index_choice_1[0][1], 5, index_choice_2[0][0], index_choice_2[0][1], 6,␣
 ↪index_choice_3[0][0], index_choice_3[0][1], 7, index_choice_4[0][0],␣
 ↪index_choice_4[0][1]))

def ham_encoding_to_graph(encoding):
    targets = [encoding[0:3], encoding[3:6], encoding[6:9], encoding[9:12]]
    edges = sum([[(k,v) for v in t] for (k,t) in enumerate(targets)], [])
    return FormalityGraph(0, 8, edges)

ham_graphs= []
```

```
for encoding in hamiltonian_encodings:
    ham_graphs.append(ham_encoding_to_graph(encoding))

ham_graphs_iso={}
for g in ham_graphs:
    h=tuple(g.canonical_form().edges())
    if not h in ham_graphs_iso:
        ham_graphs_iso[h]=g
ham_graphs_iso=list(ham_graphs_iso.values())
print ('There are', len(ham_graphs_iso), 'nonisomorphic 3D Hamiltonian graphs␣
↪for gamma_5.\n')
```

```
There are 6548 nonisomorphic 3D Hamiltonian graphs for gamma_5.

CPU times: user 50.1 s, sys: 1.01 s, total: 51.1 s
Wall time: 51 s
```

[6]:
```
%%time
from gcaops.graph.formality_graph import FormalityGraph
import itertools
hamiltonian_encodings=[]
for index_choice_1 in itertools.product(itertools.combinations(range(12),␣
↪int(2)), repeat=1):
    if index_choice_1[0][0]==4 or index_choice_1[0][1]==4 or␣
↪index_choice_1[0][0]==8 or index_choice_1[0][1]==8:
        continue
    for index_choice_2 in itertools.product(itertools.combinations(range(12),␣
↪int(2)), repeat=1):
        if index_choice_2[0][0]==5 or index_choice_2[0][1]==5 or␣
↪index_choice_2[0][0]==9 or index_choice_2[0][1]==9:
            continue
        for index_choice_3 in itertools.product(itertools.
↪combinations(range(12), int(2)), repeat=1):
            if index_choice_3[0][0]==6 or index_choice_3[0][1]==6 or␣
↪index_choice_3[0][0]==10 or index_choice_3[0][1]==10:
                continue
            for index_choice_4 in itertools.product(itertools.
↪combinations(range(12), int(2)), repeat=1):
                if index_choice_4[0][0]==7 or index_choice_4[0][1]==7 or␣
↪index_choice_4[0][0]==11 or index_choice_4[0][1]==11:
                    continue
                hamiltonian_encodings.append((4,8, index_choice_1[0][0],␣
↪index_choice_1[0][1], 5,9, index_choice_2[0][0], index_choice_2[0][1], 6,10,␣
↪index_choice_3[0][0], index_choice_3[0][1], 7, 11, index_choice_4[0][0],␣
↪index_choice_4[0][1]))

def ham_encoding_to_graph(encoding):
```

```
    targets = [encoding[0:4], encoding[4:8], encoding[8:12], encoding[12:16]]
    edges = sum([[(k,v) for v in t] for (k,t) in enumerate(targets)], [])
    return FormalityGraph(0, 12, edges)

ham_graphs= []
for encoding in hamiltonian_encodings:
    ham_graphs.append(ham_encoding_to_graph(encoding))

ham_graphs_iso={}
for g in ham_graphs:
    h=tuple(g.
 ↪canonical_form(aerial_vertex_partition=[[0,1,2,3],[4,5,6,7],[8,9,10,11]]).
 ↪edges())
    if not h in ham_graphs_iso:
        ham_graphs_iso[h]=g
ham_graphs_iso=list(ham_graphs_iso.values())
print ('There are', len(ham_graphs_iso), 'nonisomorphic 4D Hamiltonian graphs␣
 ↪for gamma_5.\n')
```

```
There are 141571 nonisomorphic 4D Hamiltonian graphs for gamma_5.

CPU times: user 16min 26s, sys: 17.5 s, total: 16min 44s
Wall time: 16min 43s
```

[11]:
```
%%time
from gcaops.graph.formality_graph import FormalityGraph
import itertools
hamiltonian_encodings=[]
for index_choice_1 in itertools.product(itertools.combinations(range(6),␣
 ↪int(2)), repeat=1):
    for index_choice_2 in itertools.product(itertools.combinations(range(6),␣
 ↪int(2)), repeat=1):
        for index_choice_3 in itertools.product(itertools.combinations(range(6),␣
 ↪int(2)), repeat=1):
            for index_choice_4 in itertools.product(itertools.
 ↪combinations(range(6), int(2)), repeat=1):
                for index_choice_5 in itertools.product(itertools.
 ↪combinations(range(6), int(2)), repeat=1):
                    for index_choice_6 in itertools.product(itertools.
 ↪combinations(range(6), int(2)), repeat=1):
                        hamiltonian_encodings.append((index_choice_1[0][0],␣
 ↪index_choice_1[0][1], index_choice_2[0][0], index_choice_2[0][1],␣
 ↪index_choice_3[0][0], index_choice_3[0][1], index_choice_4[0][0],␣
 ↪index_choice_4[0][1],index_choice_5[0][0], index_choice_5[0][1],␣
 ↪index_choice_6[0][0], index_choice_6[0][1]))

def ham_encoding_to_graph(encoding):
```

```
    targets = [encoding[0:2], encoding[2:4], encoding[4:6], encoding[6:
↪8],encoding[8:10],encoding[12:12]]
    edges = sum([[(k,v) for v in t] for (k,t) in enumerate(targets)], [])
    return FormalityGraph(0, 6, edges)

ham_graphs= []
for encoding in hamiltonian_encodings:
    ham_graphs.append(ham_encoding_to_graph(encoding))

ham_graphs_iso={}
for g in ham_graphs:
    h=tuple(g.canonical_form().edges())
    if not h in ham_graphs_iso:
        ham_graphs_iso[h]=g
ham_graphs_iso=list(ham_graphs_iso.values())
print ('There are', len(ham_graphs_iso), 'nonisomorphic 2D Hamiltonian graphs␣
↪for gamma_7.\n')
```

```
There are 6874 nonisomorphic 2D Hamiltonian graphs for gamma_7.

CPU times: user 36min 2s, sys: 36.8 s, total: 36min 39s
Wall time: 36min 38s
```

[ ]:
```
%%time
# This one could not be run on my laptop
from gcaops.graph.formality_graph import FormalityGraph
import itertools
hamiltonian_encodings=[]
for index_choice_1 in itertools.product(itertools.combinations(range(12),␣
↪int(2)), repeat=1):
    if index_choice_1[0][0]==6 or index_choice_1[0][1]==6:
        continue
    for index_choice_2 in itertools.product(itertools.combinations(range(12),␣
↪int(2)), repeat=1):
        if index_choice_2[0][0]==7 or index_choice_2[0][1]==7:
            continue
        for index_choice_3 in itertools.product(itertools.
↪combinations(range(12), int(2)), repeat=1):
            if index_choice_3[0][0]==8 or index_choice_3[0][1]==8:
                continue
            for index_choice_4 in itertools.product(itertools.
↪combinations(range(12), int(2)), repeat=1):
                if index_choice_4[0][0]==9 or index_choice_4[0][1]==9:
                    continue
                for index_choice_5 in itertools.product(itertools.
↪combinations(range(12), int(2)), repeat=1):
                    if index_choice_5[0][0]==10 or index_choice_5[0][1]==10:
```

```
                        continue
                    for index_choice_6 in itertools.product(itertools.
 ↪combinations(range(12), int(2)), repeat=1):
                        if index_choice_6[0][0]==11 or index_choice_6[0][1]==11:
                            continue
                        hamiltonian_encodings.append((6,index_choice_1[0][0],␣
 ↪index_choice_1[0][1],7, index_choice_2[0][0], index_choice_2[0][1],␣
 ↪8,index_choice_3[0][0], index_choice_3[0][1], 9,index_choice_4[0][0],␣
 ↪index_choice_4[0][1],10,index_choice_5[0][0], index_choice_5[0][1], 11,␣
 ↪index_choice_6[0][0], index_choice_6[0][1]))
print(len(encodings))
def ham_encoding_to_graph(encoding):
    targets = [encoding[0:3], encoding[3:6], encoding[6:9], encoding[9:
 ↪12],encoding[12:15],encoding[15:18]]
    edges = sum([[(k,v) for v in t] for (k,t) in enumerate(targets)], [])
    return FormalityGraph(0, 12, edges)

ham_graphs= []
for encoding in hamiltonian_encodings:
    ham_graphs.append(ham_encoding_to_graph(encoding))

ham_graphs_iso={}
for g in ham_graphs:
    h=tuple(g.canonical_form().edges())
    if not h in ham_graphs_iso:
        ham_graphs_iso[h]=g
ham_graphs_iso=list(ham_graphs_iso.values())
print ('There are', len(ham_graphs_iso), 'nonisomorphic 3D Hamiltonian graphs␣
 ↪for gamma_7.\n')
```

[ ]:
```
%%time
# This one could not be run on my laptop
from gcaops.graph.formality_graph import FormalityGraph
import itertools
hamiltonian_encodings=[]
for index_choice_1 in itertools.product(itertools.combinations(range(18),␣
 ↪int(2)), repeat=1):
    if index_choice_1[0][0]==6 or index_choice_1[0][1]==6 or␣
 ↪index_choice_1[0][0]==12 or index_choice_1[0][1]==12:
        continue
    for index_choice_2 in itertools.product(itertools.combinations(range(18),␣
 ↪int(2)), repeat=1):
        if index_choice_2[0][0]==7 or index_choice_2[0][1]==7 or␣
 ↪index_choice_2[0][0]==13 or index_choice_2[0][1]==13:
            continue
```

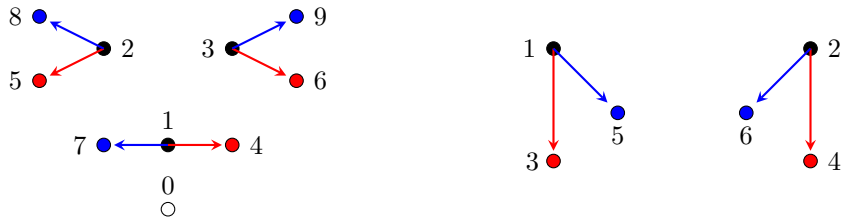
```
        for index_choice_3 in itertools.product(itertools.
↪combinations(range(18), int(2)), repeat=1):
            if index_choice_3[0][0]==8 or index_choice_3[0][1]==8 or␣
↪index_choice_3[0][0]==14 or index_choice_3[0][1]==14:
                continue
            for index_choice_4 in itertools.product(itertools.
↪combinations(range(18), int(2)), repeat=1):
                if index_choice_4[0][0]==9 or index_choice_4[0][1]==9 or␣
↪index_choice_4[0][0]==15 or index_choice_4[0][1]==15:
                    continue
                for index_choice_5 in itertools.product(itertools.
↪combinations(range(18), int(2)), repeat=1):
                    if index_choice_5[0][0]==10 or index_choice_5[0][1]==10 or␣
↪index_choice_5[0][0]==16 or index_choice_5[0][1]==16:
                        continue
                    for index_choice_6 in itertools.product(itertools.
↪combinations(range(18), int(2)), repeat=1):
                        if index_choice_6[0][0]==11 or index_choice_6[0][1]==11␣
↪or index_choice_6[0][0]==17 or index_choice_6[0][1]==17:
                            continue
                        hamiltonian_encodings.append((6, 12,␣
↪index_choice_1[0][0], index_choice_1[0][1],7,13, index_choice_2[0][0],␣
↪index_choice_2[0][1], 8,14,index_choice_3[0][0], index_choice_3[0][1],␣
↪9,15,index_choice_4[0][0], index_choice_4[0][1],10,16,index_choice_5[0][0],␣
↪index_choice_5[0][1], 11,17, index_choice_6[0][0], index_choice_6[0][1]))
print(len(encodings))
def ham_encoding_to_graph(encoding):
    targets = [encoding[0:4], encoding[4:8], encoding[8:12], encoding[12:
↪16],encoding[16:20],encoding[20:24]]
    edges = sum([[(k,v) for v in t] for (k,t) in enumerate(targets)], [])
    return FormalityGraph(0, 18, edges)

ham_graphs= []
for encoding in hamiltonian_encodings:
    ham_graphs.append(ham_encoding_to_graph(encoding))

ham_graphs_iso={}
for g in ham_graphs:
    h=tuple(g.canonical_form().edges())
    if not h in ham_graphs_iso:
        ham_graphs_iso[h]=g
ham_graphs_iso=list(ham_graphs_iso.values())
print ('There are', len(ham_graphs_iso), 'nonisomorphic 4D Hamiltonian graphs␣
↪for gamma_7.\n')
```

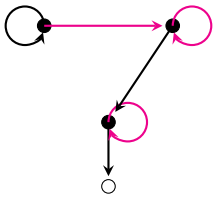
Appendix C. Graphs

This appendix contains an exhaustive list of all the graphs that are used in the papers [2], [3] and [4]. The edge ordering is determined by colors. For the $2D$ graphs, the black edge is ordered first, and the magenta colored edge is ordered second. For the $3D$ graphs the black edge is ordered first, the magenta colored edge is ordered second, and the red colored edge is ordered third (and by definition, must go to its own red colored a^1 Casimir vertex). For the $4D$ graphs the black edge is ordered first, the magenta colored edge is ordered second, the red colored edge is ordered third (and must go to its own red colored a^1 Casimir vertex) and the blue colored edge is ordered fourth (and must go to its own blue colored a^2 Casimir vertex). Moreover, for the graphs evaluating into vector fields, we give the sink vertex the label 0, the 3 Levi-Civita vertices labels 1, 2 and 3, and (when present) the corresponding a^1 Casimir vertices (respectively, a^2 Casimir vertices) the label 4, 5, 6 (respectively, 7, 8, 9). Similarly, for the graphs evaluating into Hamiltonians, we give the 2 Levi-Civita vertices labels 1 and 2, and (when present) the corresponding a^1 Casimir vertices (respectively, a^2 Casimir vertices) the label 3, 4 (respectively, 5, 6). Moreover, we fix where these vertices lie in the plane, as demonstrated below. The left graph is for graphs evaluating into vector fields, denoted by Γ_i^d, whereas the right graph is for graphs evaluating into Hamiltonians, denoted by $\Gamma_{H_i}^d$.

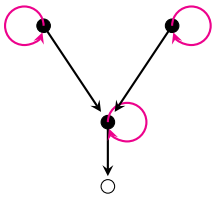


Warning: There are some differences between the $3D$ and $4D$ Hamiltonians here and in the paper [4]. First off, the Hamiltonian graphs $\Gamma_{H_4}^{3D}$ and $\Gamma_{H_7}^{3D}$ are actually isomorphic, and so only $\Gamma_{H_4}^{3D}$ is drawn. A second change is that the encodings are changed slightly to fit with how the ordering of the edges are defined here. Here, we have ordered the edges to the own Casimirs last, while in the paper, the ordering of edges in increasing with respect to the target vertex. As a result, this means that the formulas the Hamiltonian graphs $\Gamma_{H_i}^{3D}$ and $\Gamma_{H_i}^{4D}$ evaluate into here may differ by a minus sign compared to the encodings that are used in the paper.

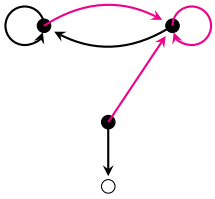
C.1. The relevant 2D graphs related to γ_3

$\Gamma_1^{2D} =$ 

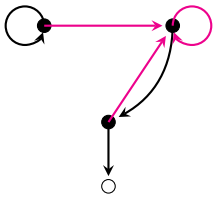
$[0,1;2,3;1,3]$

$\Gamma_2^{2D} =$ 

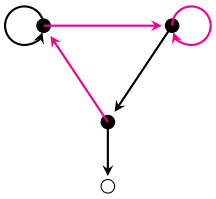
$[0,1;1,2;1,3]$

$\Gamma_3^{2D} =$ 

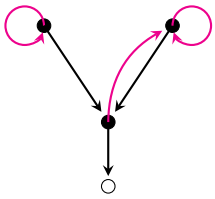
$[0,3;2,3;2,3]$

$\Gamma_4^{2D} =$ 

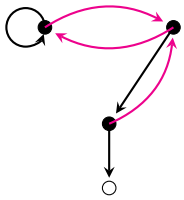
$[0,3;2,3;1,3]$

$\Gamma_5^{2D} =$ 

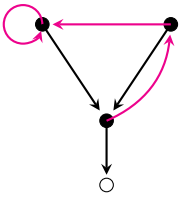
$[0,2;2,3;1,3]$

$\Gamma_6^{2D} =$ 

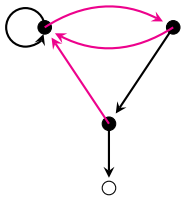
$[0,3;1,2;1,3]$

$\Gamma_7^{2D} =$ 

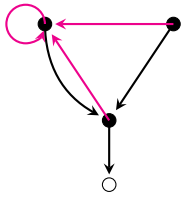
$[0,3;2,3;1,2]$

$\Gamma_8^{2D} =$ 

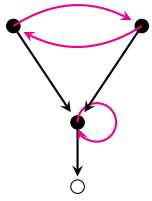
$[0,3;1,2;1,2]$

$\Gamma_9^{2D} =$ 

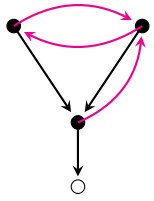
$[0,2;2,3;1,2]$

$\Gamma_{10}^{2D} =$ 

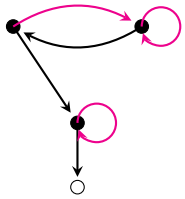
$[0,2;1,2;1,2]$

$\Gamma_{11}^{2D} =$ 

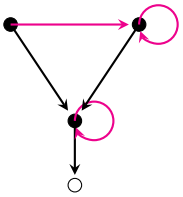
$[0,1;1,3;1,2]$

$\Gamma_{12}^{2D} =$ 

$[0,3;1,3;1,2]$

$\Gamma^{2D}_{13} =$ 

$[0,1;1,3;2,3]$

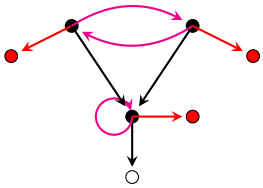
$\Gamma^{2D}_{14} =$ 

$[0,1;1,3;1,3]$

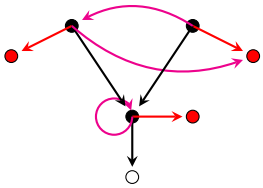
$\Gamma^{2D}_{H_1} =$ 

$[1,2;1,2]$

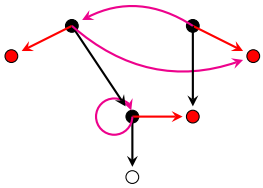
C.2. The relevant 3D graphs related to γ_3

$\Gamma^{3D}_{1} =$ 

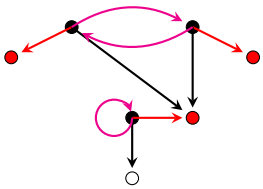
$[0,1,4;1,3,5;1,2,6]$

$\Gamma^{3D}_{2} =$ 

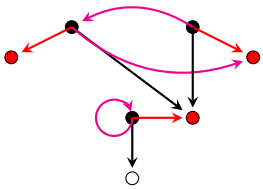
$[0,1,4;1,6,5;1,2,6]$

$\Gamma^{3D}_{4} =$ 

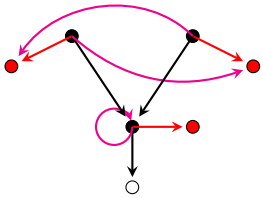
$[0,1,4;1,6,5;4,2,6]$

$\Gamma^{3D}_{7} =$ 

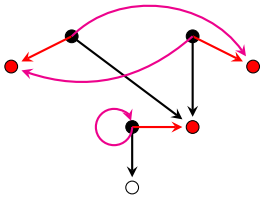
$[0,1,4;4,3,5;4,2,6]$

$\Gamma^{3D}_{8} =$ 

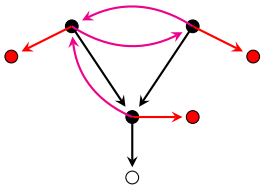
$[0,1,4;4,6,5;4,2,6]$

$\Gamma^{3D}_{10} =$ 

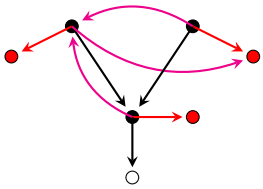
$[0,1,4;1,6,5;1,5,6]$

$\Gamma^{3D}_{16} =$ 

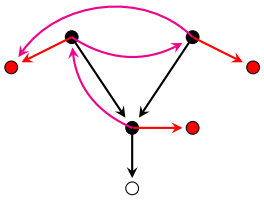
$[0,1,4;4,6,5;4,5,6]$

$\Gamma^{3D}_{17} =$ 

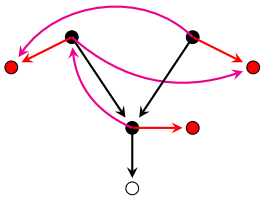
$[0,2,4;1,3,5;1,2,6]$

$\Gamma^{3D}_{18} =$ 

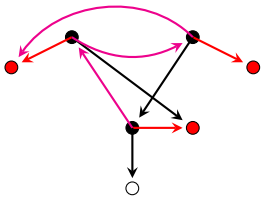
$[0,2,4;1,6,5;1,2,6]$

$\Gamma^{3D}_{25} =$ 

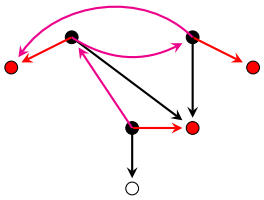
$[0, 2, 4; 1, 3, 5; 1, 5, 6]$

$\Gamma^{3D}_{26} =$ 

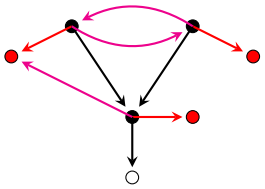
$[0, 2, 4; 1, 6, 5; 1, 5, 6]$

$\Gamma^{3D}_{29} =$ 

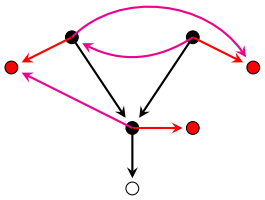
$[0, 2, 4; 4, 3, 5; 1, 5, 6]$

$\Gamma^{3D}_{31} =$ 

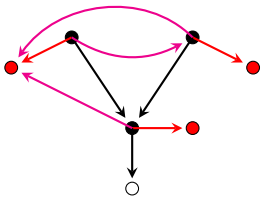
$[0, 2, 4; 4, 3, 5; 4, 5, 6]$

$\Gamma^{3D}_{33} =$ 

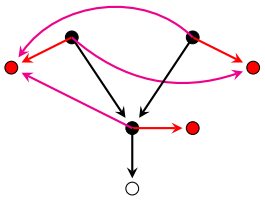
$[0, 5, 4; 1, 3, 5; 1, 2, 6]$

$\Gamma^{3D}_{34} =$ 

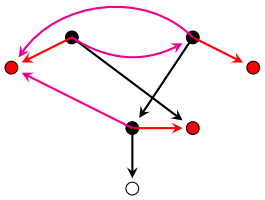
$[0, 5, 4; 1, 6, 5; 1, 2, 6]$

$\Gamma^{3D}_{41} =$ 

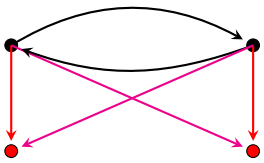
$[0, 5, 4; 1, 3, 5; 1, 5, 6]$

$\Gamma^{3D}_{42} =$ 

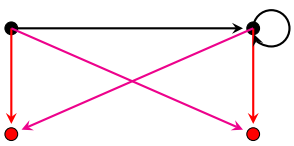
$[0, 5, 4; 1, 6, 5; 1, 5, 6]$

$\Gamma^{3D}_{45} =$ 

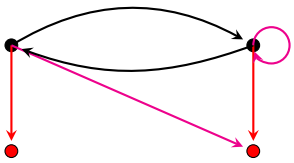
$[0, 5, 4; 4, 3, 5; 1, 5, 6]$

$\Gamma^{3D}_{H_1} =$ 

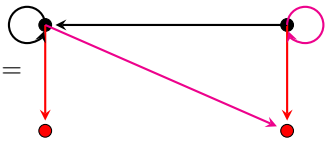
$[2, 4, 3; 1, 3, 4]$

$\Gamma^{3D}_{H_2} =$ 

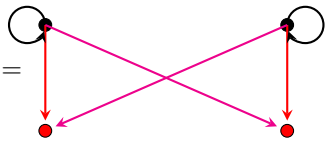
$[2, 4, 3; 2, 3, 4]$

$\Gamma^{3D}_{H_3} =$ 

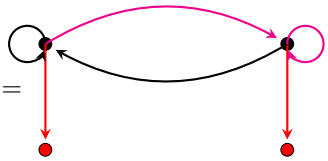
$[2, 4, 3; 1, 2, 4]$

$\Gamma^{3D}_{H_4} =$ 

$[1,4,3;1,2,4]$

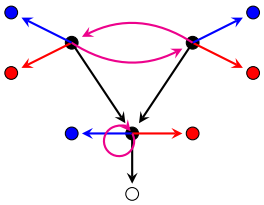
$\Gamma^{3D}_{H_5} =$ 

$[1,4,3;2,3,4]$

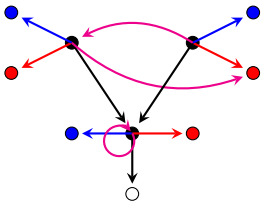
$\Gamma^{3D}_{H_6} =$ 

$[1,2,3;1,2,4]$

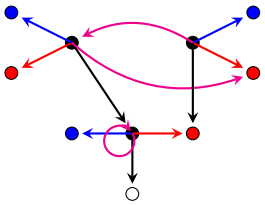
C.3. The relevant 4D graphs related to γ_3

$\Gamma^{4D}_{1} =$ 

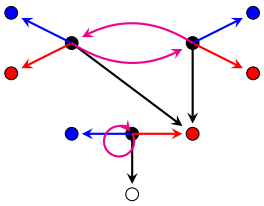
$[0,1,4,7;1,3,5,8;1,2,6,9]$

$\Gamma^{4D}_{2} =$ 

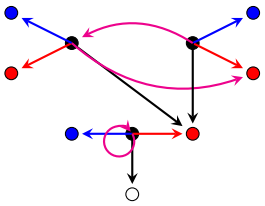
$[0,1,4,7;1,6,5,8;1,2,6,9]$

$\Gamma^{4D}_{4} =$ 

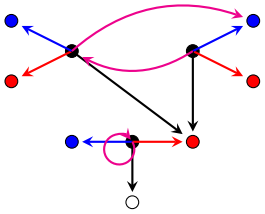
$[0,1,4,7;1,6,5,8;4,2,6,9]$

$\Gamma^{4D}_{8} =$ 

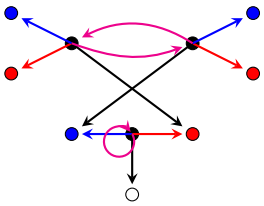
$[0,1,4,7;4,3,5,8;4,2,6,9]$

$\Gamma^{4D}_{9} =$ 

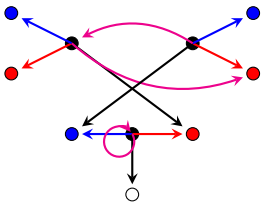
$[0,1,4,7;4,6,5,8;4,2,6,9]$

$\Gamma^{4D}_{10} =$ 

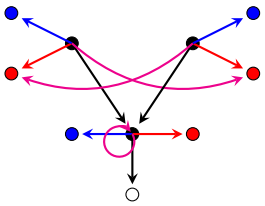
$[0,1,4,7;4,9,5,8;4,2,6,9]$

$\Gamma^{4D}_{11} =$ 

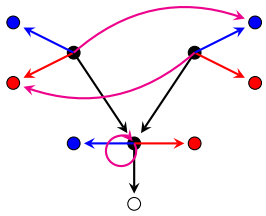
$[0,1,4,7;4,3,5,8;7,2,6,9]$

$\Gamma^{4D}_{12} =$ 

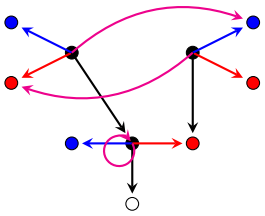
$[0,1,4,7;4,6,5,8;7,2,6,9]$

$\Gamma^{4D}_{14} =$ 

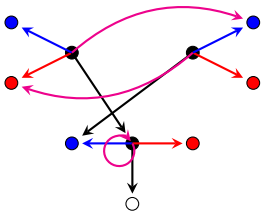
$[0,1,4,7;1,6,5,8;1,5,6,9]$

$\Gamma^{4D}_{15} =$ 

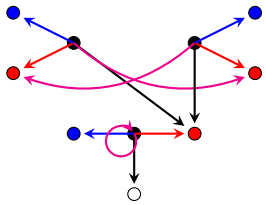
$[0, 1, 4, 7; 1, 9, 5, 8; 1, 5, 6, 9]$

$\Gamma^{4D}_{16} =$ 

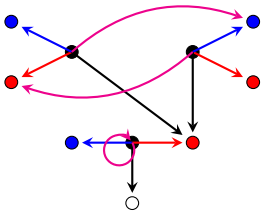
$[0, 1, 4, 7; 1, 9, 5, 8; 4, 5, 6, 9]$

$\Gamma^{4D}_{18} =$ 

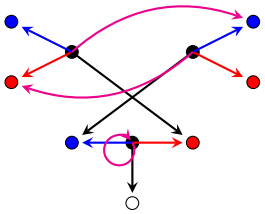
$[0, 1, 4, 7; 1, 9, 5, 8; 7, 5, 6, 9]$

$\Gamma^{4D}_{19} =$ 

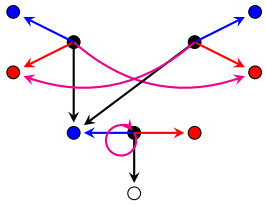
$[0, 1, 4, 7; 4, 6, 5, 8; 4, 5, 6, 9]$

$\Gamma^{4D}_{20} =$ 

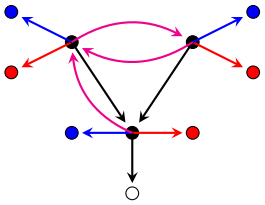
$[0, 1, 4, 7; 4, 9, 5, 8; 4, 5, 6, 9]$

$\Gamma^{4D}_{22} =$ 

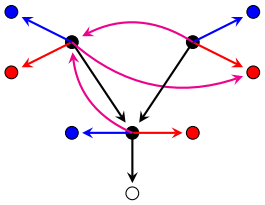
$[0, 1, 4, 7; 4, 9, 5, 8; 7, 5, 6, 9]$

$\Gamma^{4D}_{24} =$ 

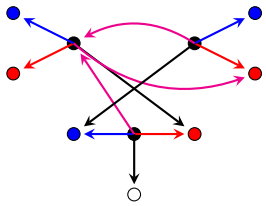
$[0, 1, 4, 7; 7, 6, 5, 8; 7, 5, 6, 9]$

$\Gamma^{4D}_{25} =$ 

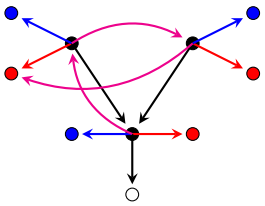
$[0, 2, 4, 7; 1, 3, 5, 8; 1, 2, 6, 9]$

$\Gamma^{4D}_{26} =$ 

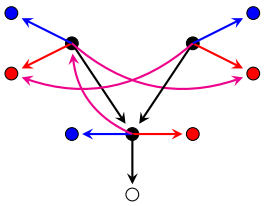
$[0, 2, 4, 7; 1, 6, 5, 8; 1, 2, 6, 9]$

$\Gamma^{4D}_{31} =$ 

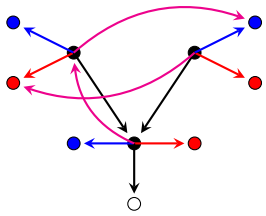
$[0, 2, 4, 7; 4, 6, 5, 8; 7, 2, 6, 9]$

$\Gamma^{4D}_{32} =$ 

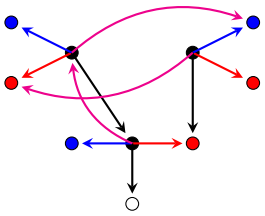
$[0, 2, 4, 7; 1, 3, 5, 8; 1, 5, 6, 9]$

$\Gamma^{4D}_{33} =$ 

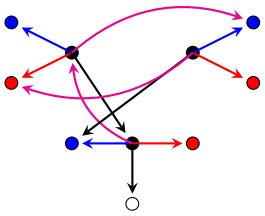
$[0, 2, 4, 7; 1, 6, 5, 8; 1, 5, 6, 9]$

$\Gamma^{4D}_{34} =$ 

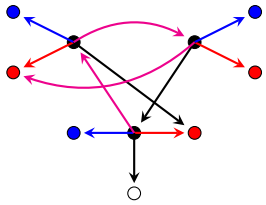
$[0, 2, 4, 7; 1, 9, 5, 8; 1, 5, 6, 9]$

$\Gamma^{4D}_{35} =$ 

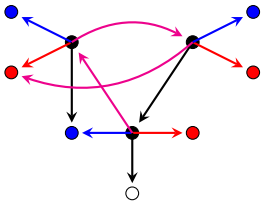
$[0, 2, 4, 7; 1, 9, 5, 8; 4, 5, 6, 9]$

$\Gamma^{4D}_{36} =$ 

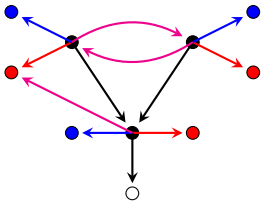
$[0, 2, 4, 7; 1, 9, 5, 8; 7, 5, 6, 9]$

$\Gamma^{4D}_{37} =$ 

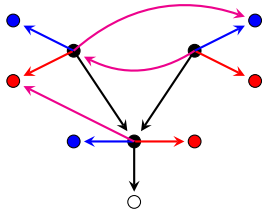
$[0, 2, 4, 7; 4, 3, 5, 8; 1, 5, 6, 9]$

$\Gamma^{4D}_{38} =$ 

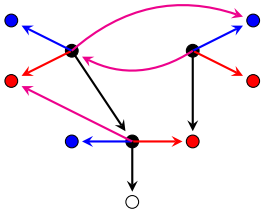
$[0, 2, 4, 7; 7, 3, 5, 8; 1, 5, 6, 9]$

$\Gamma^{4D}_{39} =$ 

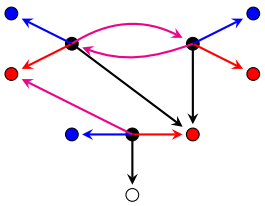
$[0, 5, 4, 7; 1, 3, 5, 8; 1, 2, 6, 9]$

$\Gamma^{4D}_{40} =$ 

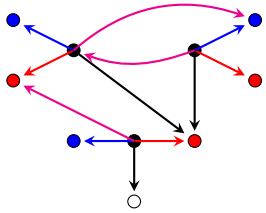
$[0, 5, 4, 7; 1, 9, 5, 8; 1, 2, 6, 9]$

$\Gamma^{4D}_{41} =$ 

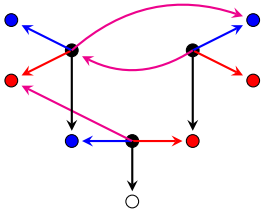
$[0, 5, 4, 7; 1, 9, 5, 8; 4, 2, 6, 9]$

$\Gamma^{4D}_{42} =$ 

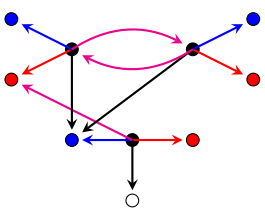
$[0, 5, 4, 7; 4, 3, 5, 8; 4, 2, 6, 9]$

$\Gamma^{4D}_{43} =$ 

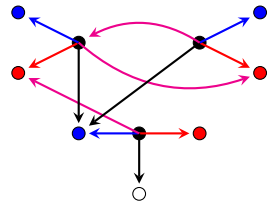
$[0, 5, 4, 7; 4, 9, 5, 8; 4, 2, 6, 9]$

$\Gamma^{4D}_{44} =$ 

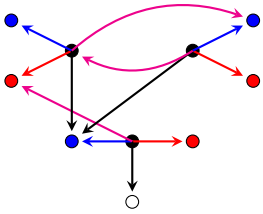
$[0, 5, 4, 7; 7, 9, 5, 8; 4, 2, 6, 9]$

$\Gamma^{4D}_{45} =$ 

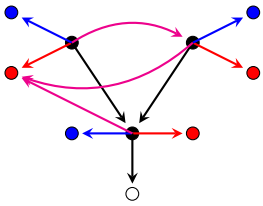
$[0, 5, 4, 7; 7, 3, 5, 8; 7, 2, 6, 9]$

$\Gamma^{4D}_{46} =$ 

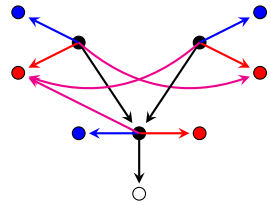
$[0, 5, 4, 7; 7, 6, 5, 8; 7, 2, 6, 9]$

$\Gamma^{4D}_{47} =$ 

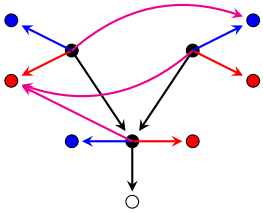
$[0, 5, 4, 7; 7, 9, 5, 8; 7, 2, 6, 9]$

$\Gamma^{4D}_{48} =$ 

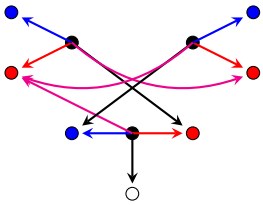
$[0, 5, 4, 7; 1, 3, 5, 8; 1, 5, 6, 9]$

$\Gamma^{4D}_{49} =$ 

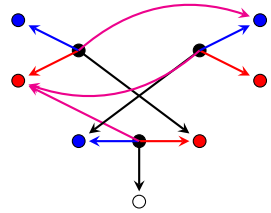
$[0, 5, 4, 7; 1, 6, 5, 8; 1, 5, 6, 9]$

$\Gamma^{4D}_{50} =$ 

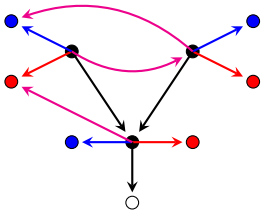
$[0, 5, 4, 7; 1, 9, 5, 8; 1, 5, 6, 9]$

$\Gamma^{4D}_{53} =$ 

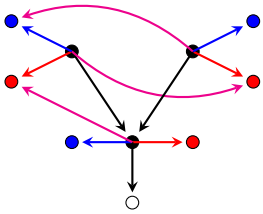
$[0, 5, 4, 7; 4, 6, 5, 8; 7, 5, 6, 9]$

$\Gamma^{4D}_{54} =$ 

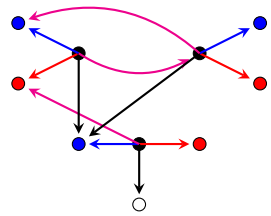
$[0, 5, 4, 7; 4, 9, 5, 8; 7, 5, 6, 9]$

$\Gamma^{4D}_{61} =$ 

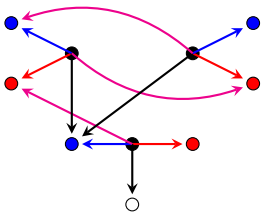
$[0, 5, 4, 7; 1, 3, 5, 8; 1, 8, 6, 9]$

$\Gamma^{4D}_{62} =$ 

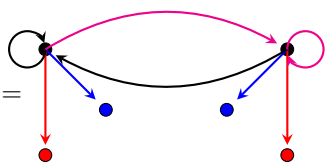
$[0, 5, 4, 7; 1, 6, 5, 8; 1, 8, 6, 9]$

$\Gamma^{4D}_{63} =$ 

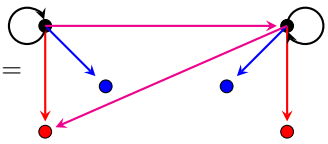
$[0, 5, 4, 7; 7, 3, 5, 8; 7, 8, 6, 9]$

$\Gamma^{4D}_{64} =$ 

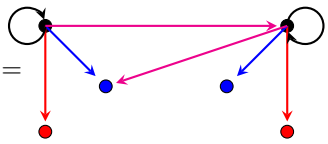
$[0, 5, 4, 7; 7, 6, 5, 8; 7, 8, 6, 9]$

$\Gamma^{4D}_{H_1} =$ 

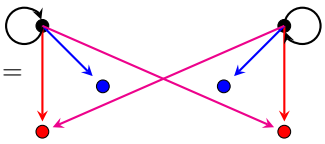
$[1, 2, 3, 5; 1, 2, 4, 6]$

$\Gamma^{4D}_{H_2} =$ 

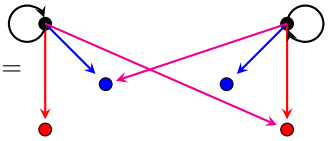
$[1,2,3,5;2,3,4,6]$

$\Gamma^{4D}_{H_3} =$ 

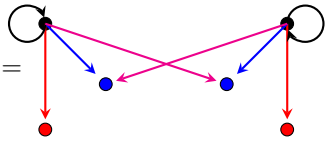
$[1,2,3,5;2,5,4,6]$

$H^{4D}_{4} =$ 

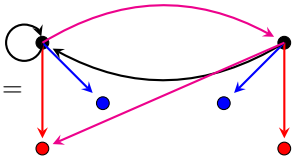
$[1,4,3,5;2,3,4,6]$

$\Gamma^{4D}_{H_5} =$ 

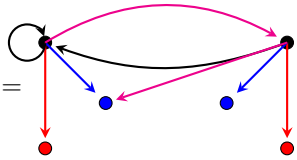
$[1,4,3,5;2,5,4,6]$

$\Gamma^{4D}_{H_6} =$ 

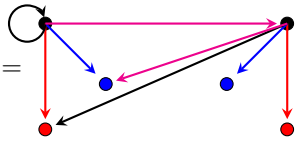
$[1,6,3,5;2,5,4,6]$

$H^{4D}_{7} =$ 

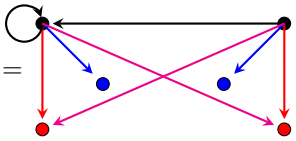
$[1,2,3,5;1,3,4,6]$

$\Gamma^{4D}_{H_8} =$ 

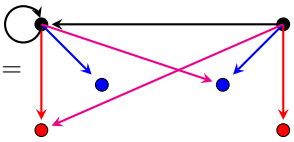
$[1,2,3,5;1,5,4,6]$

$\Gamma^{4D}_{H_9} =$ 

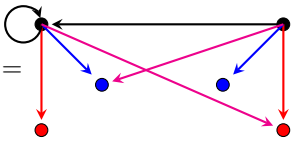
$[1,2,3,5;3,5,4,6]$

$H^{4D}_{10} =$ 

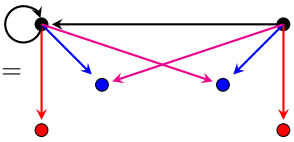
$[1,4,3,5;1,3,4,6]$

$\Gamma^{4D}_{H_{11}} =$ 

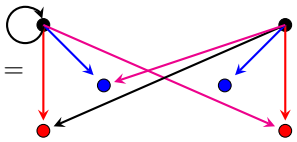
$[1,6,3,5;1,3,4,6]$

$\Gamma^{4D}_{H_{12}} =$ 

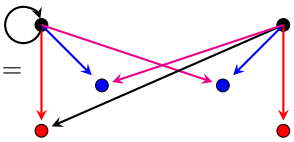
$[1,4,3,5;1,5,4,6]$

$H^{4D}_{13} =$ 

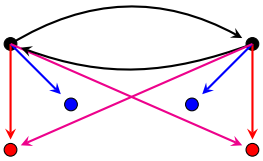
$[1,6,3,5;1,5,4,6]$

$\Gamma^{4D}_{H_{14}} =$ 

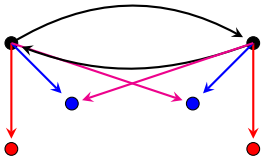
$[1, 4, 3, 5; 3, 5, 4, 6]$

$\Gamma^{4D}_{H_{15}} =$ 

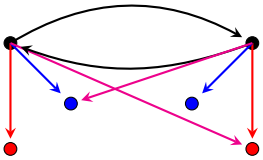
$[1, 6, 3, 5; 3, 5, 4, 6]$

$H^{4D}_{16} =$ 

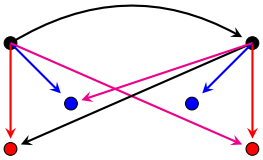
$[2, 4, 3, 5; 1, 3, 4, 6]$

$\Gamma^{4D}_{H_{17}} =$ 

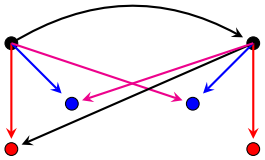
$[2, 6, 3, 5; 1, 5, 4, 6]$

$\Gamma^{4D}_{H_{18}} =$ 

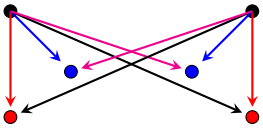
$[2, 4, 3, 5; 1, 5, 4, 6]$

$H^{4D}_{19} =$ 

$[2, 4, 3, 5; 3, 5, 4, 6]$

$\Gamma^{4D}_{H_{20}} =$ 

$[2, 6, 3, 5; 3, 5, 4, 6]$

$\Gamma^{4D}_{H_{21}} =$ 

$[4, 6, 3, 5; 3, 5, 4, 6]$