Examples of symbolic and numerical computation in Poisson geometry ⋆

M. Evangelista–Alvarado[1] [0000−0003−0997−703X],
J. C. Ruíz–Pantaleón[2] ✉ [0000−0001−7642−8298], and
P. Suárez–Serrato[3] [0000−0002−1138−0921]

Instituto de Matemáticas, Universidad Nacional Autónoma de México,
Mexico City, Mexico
[1]miguel.eva.alv@matem.unam.mx, [2]✉jcpanta@im.unam.mx, [3]pablo@im.unam.mx

Abstract. We developed two Python modules for symbolic and numerical computation in Poisson geometry. We explain how to use them through several illustrative examples, which include the computation of Hamiltonian and modular vector fields.

Keywords: Poisson geometry · Computer algebra · Python.

1 Introduction

This paper serves as an open invitation to try our two Python modules for symbolic and numerical computation in Poisson Geometry. Leisurely introductions to the history, relevance and multiple application of this branch of Differential Geometry can be found in our two papers that detail these algorithms, and the references found therein [2,3]. Comprehensive treatments of Poisson Geometry are also available [6,1,4].

We will first explain the syntax required to code objects such as functions and bivectors, so that they can be used as inputs. This is covered in Section 2. Then we start with an overview of examples in Section 3, serving as simple illustrations of how our two modules can be used. We hope to encourage users to perform their own experiments and computations. The results can be included in research papers with LaTeX code as output, or be used in numerical experiments with NumPy, TensorFlow or PyTorch outputs, as we explain below.

⋆ This research was partially supported by CONACyT and UNAM-DGAPA-PAPIIT-IN104819. JCRP thanks CONACyT for a postdoctoral fellowship held during the production of this work.

2 Installation and Syntax

Our modules work with Python 3. To download the latest stable versions via pip[1][2], run: `>>> pip install poissongeometry` for PoissonGeometry[3], and `>>> pip install numericalpoissongeometry` for NumPoissonGeometry[4].

The syntax of PoissonGeometry and NumPoissonGeometry are the same.

Scalar Functions. A scalar function is written using string type expressions.

Example 1. The function $f = ax_1 + bx_2^2 + cx_3^3$ on $\mathbf{R}^3$, with $a, b, c \in \mathbf{R}$, should be written exactly as follows: `>>> "a*x1 + b*x2**2 + c*x3**3"`.

Here, `x1`, `x2`, `x3` are symbolic variables that our modules define by default and that here represent the coordinates (x_1, x_2, x_3) on $\mathbf{R}^3$. A character that is not a coordinate is treated as a symbolic parameter, like `a`, `b`, `c` above.

Multivector fields. A multivector field is written using python dictionaries where all the keys are tuples of integers and all the values are string type expressions, i.e., when we have a multivector A of degree a on $\mathbf{R}^m$

$$A = \sum_{1 \leq i_1 < i_2 < \cdots < i_a \leq m} A^{i_1 i_2 \cdots i_a} \frac{\partial}{\partial x_{i_1}} \wedge \frac{\partial}{\partial x_{i_2}} \wedge \cdots \wedge \frac{\partial}{\partial x_{i_a}},$$

the keys of the dictionary are tuples $(i_1, i_2, \ldots, i_a)$ corresponding to the ordered indices $i_1 i_2 \cdots i_a$ of $A^{i_1 i_2 \cdots i_a}$ and the values the corresponding string expression of the coefficient (scalar function) $A^{i_1 i_2 \cdots i_a}$.

Example 2. The vector field $x_1 \frac{\partial}{\partial x_1} + x_2 \frac{\partial}{\partial x_2} + x_3 \frac{\partial}{\partial x_3}$ on $\mathbf{R}^3$ it should be written as: `>>> {(1,): "x1", (2,): "x2", (3,): "x3"}`.

In this case it is very important to use commas in the dictionary keys because in Python the expression `(1)` is a number. On the other hand, it is not necessary to write keys with null values.

Example 3. The bivector field $x_3 \frac{\partial}{\partial x_1} \wedge \frac{\partial}{\partial x_2} - x_2 \frac{\partial}{\partial x_1} \wedge \frac{\partial}{\partial x_3} + x_1 \frac{\partial}{\partial x_2} \wedge \frac{\partial}{\partial x_3}$ on $\mathbf{R}^m$ should be written as: `>>> {(1,2): "x3", (1,3): "-x2", (2,3): "x1"}`.

The order of the indexes in each tuple can be changed, according to the skew–symmetry of the exterior algebra operations.

Example 4. The bivector field on $x_1 x_2 \frac{\partial}{\partial x_1} \wedge \frac{\partial}{\partial x_2}$ on $\mathbf{R}^m$ can be written as `>>> {(1,2): "x1"}` or as `>>> {(2,1): "-x1"}`, where this last dictionary corresponds to the bivector field $-x_1 x_2 \frac{\partial}{\partial x_2} \wedge \frac{\partial}{\partial x_1}$.

[1] https://pypi.org/project/poissongeometry/
[2] https://pypi.org/project/numericalpoissongeometry/
[3] https://github.com/appliedgeometry/poissongeometry
[4] https://github.com/appliedgeometry/NumericalPoissonGeometry

Differential forms. The syntax for differential forms is the same as for multivectors fields.

Example 5. The differential form $-\mathrm{d}x_1 \wedge \mathrm{d}x_2 - (x_3 + x_4)\,\mathrm{d}x_5 \wedge \mathrm{d}x_6$ on $\mathbf{R}^m$ is written as `>>> {(1,2): "-1", (5,6): "-(x3 + x4)"}`.

3 Examples of our Symbolic and Numerical Methods

First instantiate PoissonGeometry or NumPoissonGeometry by entering the dimension and the letter/word for the symbolic variable to be used.

Example 6. To work with PoissonGeometry in dimension 6 and with coordinates $(x_1, \ldots, x_6)$, execute:

```
# Import the module PoissonGeometry with the alias pg for simplicity
>>> from poisson.poisson import PoissonGeometry as pg
# Instantiate the module to work in dimension 6 and with the symbolic variable x
>>> pg6 = pg(6)
```

Example 7. To work with NumPoissonGeometry in dimension 6 and with coordinates $(z_1, \ldots, z_6)$:

```
# Import the module PoissonGeometry with the alias npg for simplicity
>>> from numpoisson.numpoisson import NumPoissonGeometry as npg
# Instantiate the module to work in dimension 6 and with the symbolic variable z
>>> npg6 = npg(6, variable="z")
```

Furthermore, in `NumPoissonGeometry` it is necessary to define meshes to evaluate the objects described in Section 2. The inputs have to be written as lists of lists, for example as a NumPy array.

Example 8. To create a $(10^6, 6)$ NumPy array with random samples from a uniform distribution over $[0, 1)$ execute `>>> numpy.random.rand(10**6, 6)`.

3.1 Poisson Brackets

Consider the Lie–Poisson bivector field on $\mathbf{R}^6$, used in the context of the infinitesimal geometry of Poisson submanifolds [5]:

$$\begin{aligned}\Pi = {} & x_3 \frac{\partial}{\partial x_1} \wedge \frac{\partial}{\partial x_2} - x_2 \frac{\partial}{\partial x_1} \wedge \frac{\partial}{\partial x_3} + x_6 \frac{\partial}{\partial x_1} \wedge \frac{\partial}{\partial x_5} - x_5 \frac{\partial}{\partial x_1} \wedge \frac{\partial}{\partial x_6} + x_1 \frac{\partial}{\partial x_2} \wedge \frac{\partial}{\partial x_3} \\ & - x_6 \frac{\partial}{\partial x_2} \wedge \frac{\partial}{\partial x_4} + x_4 \frac{\partial}{\partial x_2} \wedge \frac{\partial}{\partial x_6} + x_5 \frac{\partial}{\partial x_3} \wedge \frac{\partial}{\partial x_4} - x_4 \frac{\partial}{\partial x_3} \wedge \frac{\partial}{\partial x_5}\end{aligned}$$

Example 9. We can verify that Π is a Poisson bivector field with the function `is_poisson_bivector` as follows:

```
>>> P = {(1,2): "x3", (1,3): "-x2", (1,5): "x6", (1,6): "-x5",
         (2,3): "x1", (2,4): "-x6", (2,6): "x4", (3,4):  "x5",
         (3,5): "-x4"}                               # dictionary encoding Π
>>> pg6.is_poisson_bivector(P)              # run is_poisson_bivector function

True
```

Example 10. The Poisson bracket of $f = x_1^2 + x_2^2 + x_3^2$ and $g = x_4 + x_5 + x_6$, induced by Π, can be computed using the function `poisson_bracket`:

```
>>> f, g = "x1**2 + x2**2 + x3**2",  "x4 + x5 + x6"
                                         # string variables encoding f and g
>>> pg6.poisson_bracket(P, f, g)             # run poisson_bracket function

-2*x1*x5 + 2*x1*x6 + 2*x2*x4 - 2*x2*x6 - 2*x3*x4 + 2*x3*x5
```

Hence $\{f, g\}_\Pi = -2x_1x_5 + 2x_1x_6 + 2x_2x_4 - 2x_2x_6 - 2x_3x_4 + 2x_3x_5$. We can easily conclude that $\{f, g\}_\Pi = 0$ at points $x \in \mathbf{R}^6$ such that $x_4 = x_5 = x_6$.

Example 11. We can check this fact with the function `num_poisson_bracket` using a random `mesh` of the form:

$$\{a_1, b_1\} \times \{a_2, b_2\} \times \{a_3, b_3\} \times \{1\} \times \{1\} \times \{1\}$$

Here the samples a_i, b_i are uniformly randomly selected from $[0, 1)$. In this example the output is a PyTorch tensor (using the flag `>>> pt_output=True`):

```
>>> npg6.num_poisson_bracket(P, f, g, mesh, pt_output=True)
                          # run num_poisson_bracket function with pt_output flag

tensor(-0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0,
       -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0,
       -0.0, -0.0 dtype=torch.float64)
```

Example 12. The output of Example 10 may be obtained in LaTeX code, adding the flag `>>> latex_format=True`:

```
>>> pg6.poisson_bracket(P, f, g, latex_format=True)
                        # run poisson_bracket function with latex_format flag

"- 2x_{1}x_{5} + 2x_{1}x_{6} + 2x_{2}x_{4} - 2x_{2}x_{6}
 - 2x_{3}x_{4} + 2x_{3}x_{5}"
```

3.2 Hamiltonian Vector Fields

Consider the Poisson bivector field $\Pi = \frac{\partial}{\partial x_1} \wedge \frac{\partial}{\partial x_4} + \frac{\partial}{\partial x_2} \wedge \frac{\partial}{\partial x_5} + \frac{\partial}{\partial x_3} \wedge \frac{\partial}{\partial x_6}$ on $\mathbf{R}^6$, and the function $h = \frac{1}{2}(x_4^2 + x_5^2 + x_6^2) - \frac{1}{x_2 - x_1} - \frac{1}{x_3 - x_1} - \frac{1}{x_3 - x_2}$.

Example 13. We can compute the Hamiltonian vector field X_h of h with respect to Π in the following way:

```
>>> P = {(1,4): "1", (2,5): "1", (3,6): "1"}     # dictionary encoding Π
>>> h = "  1/2*(x4**2 + x5**2 + x6**2)           # string variable encoding h
         - 1/(x2 - x1) - 1/(x3 - x1) - 1/(x3 - x2)"
>>> pg6.hamiltonian_vf(P, h)                     # run hamiltonian_vf method

{(1,): "-x4", (2,): "-x5", (3,): -x6,
 (4,): "-1/(x1 - x3)**2 - 1/(x1 - x2)**2",
 (5,): "-1/(x2 - x3)**2 + (x1 - x2)**(-2)",
 (6,): "(x2 - x3)**(-2) + (x1 - x3)**(-2)"}
```

Therefore X_h can be written as:

$$X_h = -x_4 \frac{\partial}{\partial x_1} - x_5 \frac{\partial}{\partial x_2} - x_6 \frac{\partial}{\partial x_3} + \left[\frac{1}{(x_1 - x_3)|x_1 - x_3|} + \frac{1}{(x_1 - x_2)|x_1 - x_2|} \right] \frac{\partial}{\partial x_4}$$
$$+ \left[\frac{1}{(x_2 - x_3)|x_2 - x_3|} + \frac{1}{(x_1 - x_2)|x_1 - x_2|} \right] \frac{\partial}{\partial x_5} - \left[\frac{1}{(x_2 - x_3)|x_2 - x_3|} + \frac{1}{(x_1 - x_3)|x_1 - x_3|} \right] \frac{\partial}{\partial x_6}$$

Example 14. To evaluate X_h on a mesh in $\mathbf{R}^6$ we can use the method `num_hamiltonian_vf`. To this end we generate a $(10^6, 6)$ NumPy array following Example 8:

```
>>> mesh = numpy.random.rand(10**6, 6)
           # numpy array with random samples from a uniform distribution over [0,1)
>>> npg6.num_hamiltonian_vf(P, h, mesh, pt_output=True)
                                   # run num_hamiltonian_vf method with pt_output flag

tensor([[[-5.07], [-3.30], [-9.95], [-1.34], [ 1.28], [5.43]],
        ...
        [[-9.42], [-3.60], [-2.44], [-1.08], [-1.14], [2.23]]],
        dtype=torch.float64)
```

Observe that we only need the algebraic expression for Π and h to carry out the numerical evaluation (and not X_h explicitly).

3.3 Modular Vector Fields

Consider the Poisson bivector field on $\mathbf{R}^4$

$$\Pi = 2x_4 \frac{\partial}{\partial x_1} \wedge \frac{\partial}{\partial x_3} + 2x_3 \frac{\partial}{\partial x_1} \wedge \frac{\partial}{\partial x_4} - 2x_4 \frac{\partial}{\partial x_2} \wedge \frac{\partial}{\partial x_3} + 2x_3 \frac{\partial}{\partial x_2} \wedge \frac{\partial}{\partial x_4}$$
$$+ (x_1 - x_2) \frac{\partial}{\partial x_3} \wedge \frac{\partial}{\partial x_4}.$$

Our method `modular_vf` computes the modular vector field of a Poisson bivector field on $\mathbf{R}^m$ with respect to a volume form $f\Omega_0$, where f is a non–vanishing function and Ω_0 the euclidean volume form on $\mathbf{R}^m$.

Example 15. To compute the modular vector field Z of Π with respect to the Euclidean volume form on $\mathbf{R}^4$, run:

```
>>> P = {(1,3): "2*x4", (1,4): "2*x3", (2,3): "-2*x4",
         (2,4): "2*x3", (3,4):"x1-x2"}          # dictionary encoding Π
>>> pg4.modular_vf(P, 1)                          # run modular_vf method

{0:0}
```

We thus conclude that $Z = 0$ and hence Π is unimodular on all of $\mathbf{R}^4$.

Example 16. We can use random meshes to numerically test the triviality of Z:

```
>>> mesh = numpy.random.rand(10**6, 4)
      # (10^6, 4) numpy array with random samples from a uniform distribution over [0,1)
>>> npg4.num_modular_vf(P, 1, mesh, pt_output=True)
                                         # run modular_vf method with pt_output flag

 tensor([ [[-0.0],[-0.0],[-0.0],[-0.0]], [[-0.0],[-0.0],[-0.0],[-0.0]],
         ...,
         [[-0.0],[-0.0],[-0.0],[-0.0]], [[-0.0],[-0.0],[-0.0],[-0.0]],
         dtype=torch.float64)
```

3.4 Poisson Bivector Fields with Prescribed Casimirs

Consider the functions $K_1 = \frac{1}{2}x_4$ and $K_2 = -x_1^2 + x_2^2 + x_3^2$ on $\mathbf{R}^4$.

Example 17. We can construct a Poisson bivector field Π on $\mathbf{R}^4$ that admits K_1 and K_2 as Casimir functions using the `flaschka_ratiu_bivector` function:

```
>>> casimirs = ["1/2 * x4", "-x1**2 + x2**2 + x3**2"]
                                   # list containing string expressions for K1 and K2
>>> pg4.flaschka_ratiu_bivector(casimirs)
                                         # run flaschka_ratiu_bivector function

{(1,2): "x3", (1,3): "-x2", (2,3): "-x1"}
```

This yields $\Pi = x_3\frac{\partial}{\partial x_1}\wedge\frac{\partial}{\partial x_2} - x_2\frac{\partial}{\partial x_1}\wedge\frac{\partial}{\partial x_3} - x_1\frac{\partial}{\partial x_2}\wedge\frac{\partial}{\partial x_3}$.

Example 18. We can verify that K_1 and K_2 are Casimir functions of Π with the `is_casimir` method:

```
>>> P = {(1,2): "x3", (1,3): "-x2", (2,3): "-x1"}
                                        # dictionary encoding Π
>>> pg4.is_casimir(P, "1/2 * x4"),        # run is_casimir function
    pg4.is_casimir(P, "-x1**2 + x2**2 + x3**2")

True, True
```

Example 19. To evaluate Π at the 'corners' $Q^4 := \{0,1\}^4$ of the unit cube in $\mathbf{R}^4$ we can use the function `num_flaschka_ratiu_bivector`. In this example the output is a list of dictionaries (by default):

```
>>> npg4.num_flaschka_ratiu_bivector(functions, Qmesh_4)
        # run num_flaschka_ratiu_bivector function with Qmesh_4 a NumPy array encoding Q^4

[{(1, 2): 0.0, (1, 3):  0.0, (2, 3):  0.0},  {(1, 2): 0.0, (1, 3):  0.0, (2, 3):  0.0},
 {(1, 2): 1.0, (1, 3):  0.0, (2, 3):  0.0}, {(1, 2): 0.0, (1, 3): -1.0, (2, 3):  0.0},
 ...
 {(1, 2): 1.0, (1, 3): -1.0, (2, 3): -1.0},  {(1, 2): 1.0, (1, 3): -1.0, (2, 3): -1.0}]
```

Example 20. In the same way, we can evaluate the bivector Π from example 17 on Q^4 using our `num_bivector` method:

```
>>> npg4.num_bivector(P, Qmesh_4, tf_output=True)
     # run num_bivector method with Qmesh_4 a NumPy array encoding Q^4 with tf_output flag

tensor([[[-0.0, 1.0,-1.0,-0.0], [-1.0,-0.0,-0.0,-0.0],
          [ 1.0,-0.0,-0.0,-0.0], [-0.0,-0.0,-0.0,-0.0]],
         ...
         [[-0.0, 1.0,-1.0,-0.0], [-1.0,-0.0,-1.0,-0.0],
          [ 1.0, 1.0,-0.0,-0.0], [-0.0,-0.0,-0.0,-0.0]]],
         dtype=float64)
```

In this example the output is a TensorFlow tensor, invoked with the flag `>>> tf_output=True`.

3.5 Classification of Lie–Poisson Bivector Fields on $\mathbf{R}^3$

Consider the following Lie–Poisson bivector field defined on $\mathbf{R}^3$,

$$\Pi = -10x_3\frac{\partial}{\partial x_1}\wedge\frac{\partial}{\partial x_2} + 10x_2\frac{\partial}{\partial x_1}\wedge\frac{\partial}{\partial x_3} - 10x_1\frac{\partial}{\partial x_2}\wedge\frac{\partial}{\partial x_3}.$$

Example 21. To compute a normal form Π_0 of Π, run:

```
>>> P = {(1,2): "-10*x3", (1,3): "10*x2", (2,3): "-10*x1"}
                                        # dictionary encoding Π
```

```
>>> pg3.linear_normal_form_R3(P),          # run linear_normal_form_R3 function

{(1,2): "x3", (1,3): "-x2", (2,3): "x1"}
```

We thus obtain $\Pi_0 = x_3 \frac{\partial}{\partial x_1} \wedge \frac{\partial}{\partial x_2} - x_2 \frac{\partial}{\partial x_1} \wedge \frac{\partial}{\partial x_3} - x_1 \frac{\partial}{\partial x_2} \wedge \frac{\partial}{\partial x_3}$, which clearly simplifies Π.

Example 22. It is easy to verify that Π and Π_0 are isomorphic using our `isomorphic_lie_poisson_R3` function:

```
>>> P0 = {(1,2): "x3", (1,3): "-x2", (2,3): "x1"}
                                      # dictionary encoding Π_so(3)
>>> pg3.isomorphic_lie_poisson_R3(P, P_so3)
                                   # run isomorphic_lie_poisson function

True
```

Example 23. To evaluate Π_0 at points of $Q^3 := \{0,1\}^3$ we can use the method `num_linear_normal_form_R3`:

```
>>> npg4.num_linear_normal_form_R3(P, Qmesh_3)
          # run num_linear_normal_form_R3 method with Qmesh_3 a NumPy array encoding Q^3

[{(1, 2): 0.0, (1, 3): -1.0, (2, 3): 0.0},  {(1, 2): 0.0, (1, 3):  0.0, (2, 3): 1.0},
 {(1, 2): 1.0, (1, 3):  0.0, (2, 3): 0.0},  {(1, 2): 0.0, (1, 3): -1.0, (2, 3): 0.0},
 ...
 {(1, 2): 1.0, (1, 3): -1.0, (2, 3): -1.0},  {(1, 2): 1.0, (1, 3): -1.0, (2, 3): -1.0}]
```

References

1. Dufour, J.P., Zung, N.T.: *Poisson Structures and their Normal Forms.* 1st edition, Birkhäuser Basel, (2005).
2. Evangelista–Alvarado, M.A., Ruíz–Pantaleón, J.C., Suárez–Serrato, P.: *On computational Poisson geometry I: Symbolic foundations.* Preprint `arXiv:1912.01746`
3. Evangelista–Alvarado, M.A., Ruíz–Pantaleón, J.C., Suárez–Serrato, P.: *On computational Poisson geometry II: Numerical methods.* Preprint `arXiv:2010.09785`
4. Laurent–Gengoux, C., Pichereau, A., Vanhaecke, P.: *Poisson Structures.* 1st edition, Springer–Verlag, Berlin Heidelberg, (2013).
5. Ruíz–Pantaleón, J.C., García–Beltrán, D., Vorobiev, Yu.: *Infinitesimal Poisson algebras and linearization of Hamiltonian systems.* Ann. Glob. Anal. Geom. **58**, 415–431 (2020).
6. Weinstein, A.: *Poisson Geometry.* Diff. Geom. Appl. **9**, 213–238 (1998).