The Journal of Open Source Software

BVPy: A FEniCS-based Python package to ease the expression and study of boundary value problems in Biology.

Florian Gacon[1], **Christophe Godin**[1], **and Olivier Ali**[*1]

1 Laboratoire Reproduction et Développement des Plantes, Université de Lyon, Ecole Normale Supérieure de Lyon, UCB Lyon 1, CNRS, INRAE, INRIA; 46, allée d'Italie, 69364 LYON Cedex 07, France.

DOI: 10.21105/joss.02831

Software

- Review
- Repository
- Archive

Editor: Kevin M. Moerman

Reviewers:

- @IgorBaratta
- @chennachaos
- @finsberg

Submitted: 28 October 2020
Published: 21 March 2021

Summary

BVPy is a python library to easily implement and study numerically Boundary Value Problems (*BVPs*) and Initial Boundary Value Problems (*IBVPs*) through the Finite Element Method (*FEM*). BVPy proposes an intuitive Application Programming Interface (*API*) to harness and combine the core functionalities of three powerful libraries: FEniCS (Logg et al., 2012) provides the core data structures and solving algorithms; Gmsh (Geuzaine & Remacle, 2009) defines the domains and their meshing; and Meshio (Schlömer et al., 2020) handles data reading and writing. Initially built in the context of developmental biology and morphomechanics, its purpose is to enable all users, even with little to none experience in *FEM*, to quickly and efficiently estimate the behavior of a wide variety of fields (scalars, vectors, tensors) on biologically relevant structures, inspired by biophysical and biochemical processes (morphogene patterning, active matter mechanics, active transports...). Despite this biological motivation, the BVPy library has been implemented in an *agnostic* manner that makes it suitable for many other scientific context.

Statement of need

FEMs are becoming an ubiquitous tool in many research areas and the need for a platform accessible to a large audience of non-specialists is growing. Such platforms should: *(i)* provide an easy access to top-tier, open source, existing libraries, usually restricted to expert users; *(ii)* serve as a means for experienced researchers to communicate with and educate novices. BVPy aims to fulfil these needs through a twofold strategy:

- Provide a high-level *API* to soften the learning curve of *FEMs*. So users with little to none knowledge can parametrize, run and monitor simulations based on built-in templates.
- Enable users experienced in *FEM*-based modeling to develop and fully customize *de novo* templates that could, in turn, be used by non-specialists.

*Corresponding Author

State of the field

Although an exhaustive inventory of the existing *FEM* solutions is far beyond the reach of this paper, it is worth mentioning a few existing alternatives; especially in the context of computational morphomechanics, in which BVPy has been developed.

Some *FEM*-frameworks provide *GUI* such as Sofa (Faure et al., 2012), FEBio (Maas et al., 2012) or MorphoMechanX (see (Sapala et al., 2018) for example of use) the yet-to-be published add-on to the MorphoGraphX software (Reuille et al., 2015). While such userfriendly solutions are accessible to a wide range of users; their "monolithic" construction reduces versatility and prevents more experienced users to tune them at will. Moreover, their code sources are not always freely available.

Besides these *GUI*-based solutions, some "lighter" and more open frameworks, such as FREEFEM (Hecht, 2012) or FEniCS (Logg et al., 2012) are also available. Such solutions appear more versatile and opensource. But their use is usually restricted to people already familiar with the theory of Finite Elements.

A gap currently exists between "closed," userfriendly, softwares on one hand and "open," technical frameworks on the other. BVPy attempts to fill this gap. Our strategy was to harness FEniCS richness into an intuitive and evolutive *API*. To that end, the library architecture maps the conceptual mathematical components of *BVPs* to abstract classes built on FEniCS and Gmsh core components (see the library general description below). From these abstract classes, experienced users can implement *concrete* classes, dedicated to their very specific need. Once implemented, these *concrete* classes can easily be instantiated and combined in a manner that do not require specific *FEM*-related skills (see the Example of use below). Although the library comes with a few of these *concrete* classes, covering *classic* equations (*e.g.* Poisson's and Helmholtz's equations, linear elasticity and hyperelasticity, reaction-diffusion equations...); the full potential of the library resides in its ability to integrate *de novo* classes addressing genuine equations and problems.

Library general description

By definition, a *BVP* corresponds to a (set) differential equation(s) together with a set of constraints defined on the boundaries of the integration domain, see Equation 1.

$$\begin{cases} F(u, \boldsymbol{\nabla}(u) \dots) = 0 & \text{in} \quad \Omega \\ u(\boldsymbol{x}) = u_N(\boldsymbol{x}) & \text{on} \quad \partial\Omega_N \\ \boldsymbol{\nabla}(u(\boldsymbol{x})) = \boldsymbol{j}(\boldsymbol{x}) & \text{on} \quad \partial\Omega_D \end{cases} \tag{1}$$

The above formalization can be extended as follow in the case of first-order *IBVPs* (*i.e.* where time derivatives are limited to first order):

$$\begin{cases} \partial_t u = F(u, \boldsymbol{\nabla}(u) \dots) & \text{in} \quad \Omega \\ u(t_0, \boldsymbol{x}) = u_0(\boldsymbol{x}) & \text{on} \quad \Omega_{t_0} \\ u(t, \boldsymbol{x}) = u_N(t, \boldsymbol{x}) & \text{on} \quad \partial\Omega_N \\ \boldsymbol{\nabla}(u(t, \boldsymbol{x})) = \boldsymbol{j}(t, \boldsymbol{x}) & \text{on} \quad \partial\Omega_D, \end{cases} \tag{2}$$

the integration domain and its boundaries can be split into one-dimension *time*-line and orthogonal *spatial* hyperplanes: $\Omega = [t_0, t_\infty] \times \Omega_{t_0}$, $\partial\Omega_N = [t_0, t_\infty] \times \partial\Omega_{t_0 N}$ and $\partial\Omega_D = [t_0, t_\infty] \times \partial\Omega_{t_0 D}$. Assuming that the spatial hyperplane Ω_{t_0} does not evolve with time, such *IBVPs* can be implemented by coupling the resolution of time-dependent *BVPs* with time-integration schemes.

Scope and range

In terms of equations: In its current version (1.0.0), BVPy can handle non-homogeneous second order Partial Differential Equations (*PDEs*) with potentially non-linear first order terms but linear higher order ones. The unknowns within these *PDEs* can be scalar fields, vector fields or second-order tensor fields. Systems of coupled *PDEs* can be implemented. We extended the initial notion of *BVP* to encompass *IBVPs* featuring first-order time derivatives, as described by Equation 2.

In terms of domains: BVPy provides geometrical primitives to generate simple 2D and 3D domains (rectangles, cubes, ellipoids, torus, ...) as well as basic Constructive Solid Geometry (*CSG*) functionalities (addition, substraction and intersection) to combine these primitives into more complex geometries. The library can also handle triangulated, as well as piecewise-polygonal, meshes generated elsewhere. In the current version, `.txt` , `.ply` and `.obj` files are accepted as inputs. This versatility comes however with a drawback, for the domain generation procedure is not yet compatible with parallelization processes.

In terms of boundary conditions: Classic *Dirichlet* and *Neumann* boundary conditions can be implemented as well as combinations of these along the domain boundaries. Periodic boundary conditions can also be defined.

Organization

BVPy has been developed and organized as close as possible to the *BVP* and *IBVP* mathematical formulations given in Equation 1 and Equation 2. The library is built around the following key components, see Figure 1:

- Two main classes, named `BVP` and `IBVP`, that respectively encapsulate FEniCS implementations of Equation 1 & Equation 2.
- Three main modules, `bvpy.domains`, `bvpy.vforms` and `bvpy.boundary_conditions` emulating the corresponding mathematical components of a *bvp*.
- A `bvpy.solvers` module where a collection of FEniCS-based linear and non-linear solvers are available.

Besides these main components, the `bvpy.utils` module gathers some useful "housekeeping" functions regrouped thematically into sub-modules, *e.g.* `visu`, `io`... See online documentation.

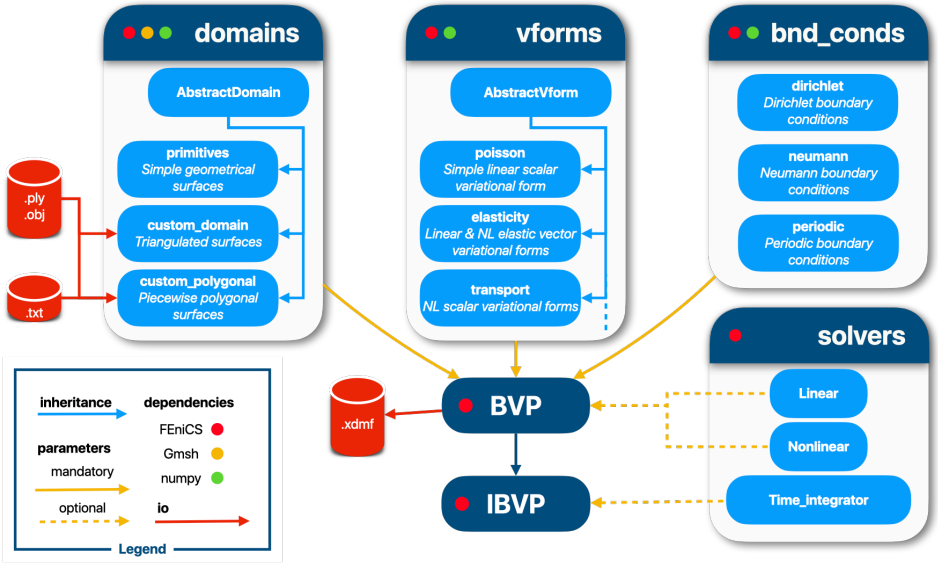

Figure 1: Representation of the main modules and submodules forming the BVPy library.

Example of use

The following example is loosely inspired by (Zhao et al., 2020). The idea is to estimate the mechanical stress distribution within a 2D cross section of a pressurized plant tissue with heterogeneous rigidity. The goal here is to illustrate how parsimonious, intuitive and yet insightful, BVPy simulations can be.

```
from bvpy import *

# domain definition and meshing
sepal = CustomPolygonalDomain.read("./sepal.ply", cell_size=.5, dim=2)
sepal.discretize()

# instanciation of an heterogeneous linear elastic model
tissue_rigidities = {0: 100, 1: 300}
young_moduli = HeterogeneousParameter(sepal.cdata, tissue_rigidities)
elastic_response = LinearElasticForm(young=young_moduli, source=[0, 0],
                                     plane_stress=True)

# loading forces implemented as Neumann boundary conditions
inner_pressure = 1
pressure_forces = NormalNeumann(val=inner_pressure, boundary='all')

# problem definition and resolution
prblm = BVP(sepal, elastic_response, pressure_forces)
prblm.solve(linear_solver='gmres', absolute_tolerance=1e-2)

# stress field computation
displacement = prblm.solution
stress = elastic_response.stress(displacement)
```

The recorded results can be processed by third-party software, *e.g.* Figure 2 shows the visualization of the corresponding stress field within the Paraview software.

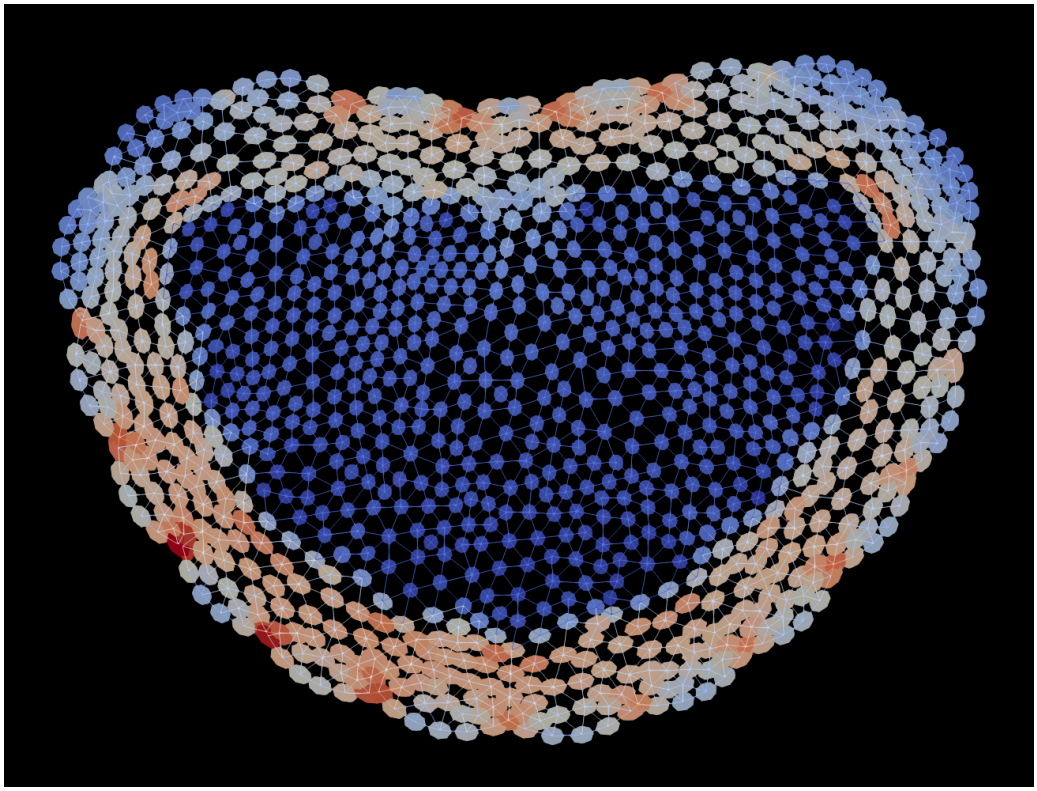

Figure 2: Example of a stress field computed from an heterogeneous linear elastic model applied to a piecewise polygonal domain. The epidermis layer has been assumed three times stiffer than the inner tissues. A corresponding distinction can be seen within the computed stress field.

More examples are available in the tutorial section of the online documentation.

Acknowledgements

The authors would like to thank Guillaume Cerutti and Jonathan Legrand for their numerous advices and their technical guidance during the maturation of this work.

This work was supported by the Inria grant *ADT Gnomon*.

References

Faure, F., Duriez, C., Delingette, H., Allard, J., Gilles, B., Marchesseau, S., Talbot, H., Courtecuisse, H., Bousquet, G., Peterlik, I., & Cotin, S. (2012). SOFA: A Multi-Model Framework for Interactive Physical Simulation. In Y. Payan (Ed.), *Soft Tissue Biomechanical Modeling for Computer Assisted Surgery* (Vol. 11, pp. 283–321). Springer. https://doi.org/10.1007/8415_2012_125

Geuzaine, C., & Remacle, J. (2009). Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, *79*(11), 1309–1331. https://doi.org/10.1002/nme.2579

Hecht, F. (2012). New development in FreeFem++. *J. Numer. Math.*, *20*(3-4), 251–265. https://freefem.org/

Logg, A., Mardal, K.-A., & Wells, G. (2012). *Automated solution of differential equations by the finite element method: The FEniCS book*. Springer Publishing Company, Incorporated. ISBN: 3642230989

Maas, S. A., Ellis, B. J., Ateshian, G. A., & Weiss, J. A. (2012). FEBio: Finite Elements for Biomechanics. *Journal of Biomechanical Engineering*, *134*(1), 011005. https://doi.org/10.1115/1.4005694

Reuille, P. B. de, Routier-Kierzkowska, A.-L., Kierzkowski, D., Bassel, G. W., Schuepbach, T., Tauriello, G., Bajpai, N., Strauss, S., Weber, A., Kiss, A., Burian, A., Hofhuis, H., Sapala, A., Lipowczan, M., Heimlicher, M. B., Robinson, S., Bayer, E. M., Basler, K., Koumoutsakos, P., … Smith, R. S. (2015). MorphoGraphX: A platform for quantifying morphogenesis in 4D. *eLife*, *4*. https://doi.org/10.7554/elife.05864

Sapala, A., Runions, A., Routier-Kierzkowska, A.-L., Gupta, M. D., Hong, L., Hofhuis, H., Verger, S., Mosca, G., Li, C.-B., Hay, A., Hamant, O., Roeder, A. H., Tsiantis, M., Prusinkiewicz, P., & Smith, R. S. (2018). Why plants make puzzle cells, and how their shape emerges. *eLife*, *7*, 2061. https://doi.org/10.7554/elife.32794

Schlömer, N., McBain, G. D., Luu, K., christos, Li, T., Keilegavlen, E., Ferrándiz, V. M., Barnes, C., Lukeš, V., Dalcin, L., eolianoe, Wagner, N., Jansen, M., Gupta, A., Müller, S., Woodsend, B., Krande, Schwarz, L., Blechta, J., … Harsch, J. (2020). *Nschloe/meshio v4.3.1* (Version v4.3.1) [Computer software]. Zenodo. https://doi.org/10.5281/zenodo.4090832

Zhao, F., Du, F., Oliveri, H., Zhou, L., Ali, O., Chen, W., Feng, S., Wang, Q., Lü, S., Long, M., Schneider, R., Sampathkumar, A., Godin, C., Traas, J., & Jiao, Y. (2020). Microtubule-Mediated Wall Anisotropy Contributes to Leaf Blade Flattening. *Current Biology*. https://doi.org/10.1016/j.cub.2020.07.076