Available online at www.sciencedirect.com

ScienceDirect

IFAC PapersOnLine 58-6 (2024) 119–124

Simulation and control of interactions in multi-physics, a Python package for port-Hamiltonian systems [⋆]

Giuseppe Ferraro [*] Michel Fournié [*] Ghislain Haine [*]

[*] *Fédération ENAC ISAE-SUPAERO ONERA Université de Toulouse, France*
e-mail: {giuseppe.ferraro, michel.fournie, ghislain.haine}@isae.fr.

Abstract: The Python package SCRIMP (Simulation and ContRol of Interactions in Multi-Physics) is presented through a collection of port-Hamiltonian systems (pHs) of increasing complexity, stemming from mechanics and thermodynamics. A focus is made on the syntax of SCRIMP allowing the user to easily describe a distributed pHs and its discretization method using the Partitioned Finite Element Method (PFEM) in space, together with the Differential Algebraic Equation (DAE) solver to use. A Graphical User Interface (GUI) is presented.

Keywords: Port-Hamiltonian systems; Structure-preserving discretization; Python package.

1. INTRODUCTION

SCRIMP (Simulation and ContRol of Interactions in Multi-Physics) is a Python package that aims to facilitate the structure-preserving discretization of distributed port-Hamiltonian systems (pHs) (van der Schaft and Maschke, 2002) in any dimension. The chosen discretization is the Partitioned Finite Element Method (PFEM) (Cardoso-Ribeiro et al., 2021), based on the well-known and robust Mixed Finite Element Method. The package is built on up-to-date libraries, well-established in the scientific computing community, such as GMSH (Geuzaine and Remacle, 2009), GetFEM (Renard and Poulios, 2020), PETSc (Abhyankar et al., 2018), and ParaView (Ayachit, 2015), with a transparent interface. In particular, advanced users keep access to all the capabilities of each library (e.g., matrices may be exported for MATLAB use, user-defined or GetFEM time schemes may be used, solutions may be saved in many formats for post-processing, etc.). A Graphical User Interface (GUI) is available and may be used to generate a Python script, either for a direct run inside the GUI, or for Python console use. The user is guided all along the different steps of discretization, resolution and post-processing, following the port-Hamiltonian formalism. The presentation will focus on already published studies, namely the wave equation (Haine et al., 2023), the heat equation (Serhani et al., 2019b) and the heat-wave coupling (Haine et al., 2022). These examples are available on GitHub [1]: `https://g-haine.github.io/scrimp/examples.html`

The paper is organized as follows: in Section 2, a definition for pHs is given. In Section 3, the PFEM is presented. Section 4 is concerned with two examples: the wave equation and the heat equation, together with their boundary interconnection. Section 5 presents the Python package itself and its syntax on a step-by-step heat example.

2. PORT-HAMILTONIAN SYSTEMS

For a review about pHs, we refer the reader to Rashad et al. (2020) and the many references therein. The chosen definition of pHs should allow for an application of the PFEM. Seeing distributed pHs as constrained open systems seems to be an easy way to achieve this goal.

In order to simplify the presentation, this work will focus on constant Stokes-Dirac structure examples: nonlinearities could only occur in the closure relations. Nevertheless, modulated Stokes-Dirac structures are not a limitation, see for instance how to tackle fluid dynamics in Cardoso-Ribeiro et al. (2021).

Definition 1. (State, Hamiltonian, and co-state). Let $\mathcal{X}$ be a Hilbert space and $\mathcal{H} : \mathcal{X} \to \mathbb{R}^+, \alpha \mapsto \mathcal{H}(\alpha)$. Then α is called a *state variable* and $\mathcal{H}$ a *Hamiltonian*. The variational derivative $e_\alpha := \delta_\alpha \mathcal{H}$ (depending on the inner product of $\mathcal{X}$) is called the *co-state variable.*

Definition 2. A *port-Hamiltonian system* is an open system of the form:

$$\begin{pmatrix} \partial_t \alpha \\ f_d \end{pmatrix} = J \begin{pmatrix} e_\alpha \\ e_d \end{pmatrix}, \tag{1}$$

under the constraints:

$$e_\alpha = \delta_\alpha \mathcal{H}, \quad \mathcal{R}_d(f_d, e_d) = 0, \tag{2}$$

and with control and observation:

$$B \left(e_\alpha \;\, e_d\right)^\top = u, \qquad y = C \left(e_\alpha \;\, e_d\right)^\top, \tag{3}$$

where $J \in \mathcal{L}(D(J), \mathcal{Z})$ is a (non-necessarily bounded) formally [2] skew-symmetric operator on $\mathcal{Z} := \mathcal{X} \times \mathcal{X}_d$, with $\mathcal{X}_d$ another Hilbert space; $\mathcal{R}_d$ is a (non-necessarily linear) relation modeling a constitutive law linking the flow f_d and the effort e_d on $\mathcal{X}_d$; $B \in \mathcal{L}(D(J), \mathcal{U})$ is a (non-necessarily bounded) control operator, with control space $\mathcal{U}$; and $C \in \mathcal{L}(D(J), \mathcal{Y})$ is a (non-necessarily bounded)

[⋆] This work was partially funded by the Agence Nationale de la Recherche (France), IMPACTS, grant no. ANR-21-CE48-0018, and by the FAMAS project, supported by the AID (Agence de l'Innovation de Défense) from the French Ministry of the Armed Forces (Ministère des Armées).

[1] `https://github.com/g-haine/scrimp`

[2] i.e., considering J on compactly supported C^∞ functions.

observation operator, with observation space $\mathcal{Y} := \mathcal{U}'$; u is the *control*, and y is the observation. The couple (u, y) is called the *control port*. The choice of B (hence of C) is called a *causality*.

Remark 1. Definition 2 requires (many) additional assumptions to derive well-posedness or regularity results, see Brugnoli et al. (2023) for the undamped linear case. Nevertheless, the present study focuses on a general framework that permits discretization using the PFEM.

A first requirement is the existence of a Stokes-Dirac structure for a system as in Definition 2, which occurs when an appropriate relation between J, B and C holds. More precisely, we follow Brugnoli et al. (2023) and assume:

Assumption 1. The space $\mathcal{Z}$ admits a decomposition $\mathcal{X}_1 \times \mathcal{X}_2$ (not necessarily identical to $\mathcal{X} \times \mathcal{X}_d$) such that:

$$J = \begin{bmatrix} 0 & -K \\ L & 0 \end{bmatrix}, \quad \begin{array}{c} K \in \mathcal{L}(D(K), \mathcal{X}_1), \\ L \in \mathcal{L}(D(L), \mathcal{X}_2) \end{array}.$$

Assumption 2. The control and observation operators split accordingly: $\mathcal{U} = \mathcal{U}_1 \times \mathcal{U}_2$, and,

$$B = \begin{bmatrix} \gamma_1 & 0 \\ 0 & \gamma_2 \end{bmatrix}, \quad C = \begin{bmatrix} 0 & \beta_1 \\ \beta_2 & 0 \end{bmatrix},$$

with $\gamma_1 \in \mathcal{L}(D(L), \mathcal{U}_1)$, $\gamma_2 \in \mathcal{L}(D(K), \mathcal{U}_2)$, $\beta_1 \in \mathcal{L}(D(K), \mathcal{Y}_1)$, $\beta_2 \in \mathcal{L}(D(L), \mathcal{Y}_2)$, $\mathcal{Y}_i' = \mathcal{U}_i$ for $i = 1, 2$.

Assumption 3. For all $\varphi \in D(L)$ and all $\psi \in D(K)$:

$$\begin{aligned} (L\varphi, \psi)_{\mathcal{X}_2} &= (\varphi, K\psi)_{\mathcal{X}_1} \\ &\quad + \langle \gamma_1 \varphi, \beta_1 \psi \rangle_{\mathcal{U}_1, \mathcal{Y}_1} + \langle \gamma_2 \psi, \beta_2 \varphi \rangle_{\mathcal{U}_2, \mathcal{Y}_2}. \end{aligned} \quad (4)$$

Providing this abstract Green's identity, the following power balance holds along the solutions:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathcal{H} = -\left(e_d, f_d\right)_{\mathcal{X}_d} + \langle u_1, y_1 \rangle_{\mathcal{U}_1, \mathcal{Y}_1} + \langle u_2, y_2 \rangle_{\mathcal{U}_2, \mathcal{Y}_2}. \quad (5)$$

The term $-\left(e_d, f_d\right)_{\mathcal{X}_d}$ in (5) will be negative if the relation $\mathcal{R}_d$ models a dissipative law, leading to a *lossy pHs*, null for a *lossless pHs*, and positive for an *accretive pHs*.

3. STRUCTURE-PRESERVING DISCRETIZATION

The structure-preserving discretization of pHs is an active topic of research. The interested reader may refer to Kotyczka (2019); Egger (2019); Brugnoli et al. (2022) and the many references therein for the spatial approximation. Regarding time discretization, we refer to, e.g., Egger et al. (2021); Schulze (2023).

3.1 A mixed Galerkin method

The PFEM is a mixed Galerkin method:

(i) Write the variational formulations of (1)–(2)–(3);
(ii) Make use of (4);
(iii) Project the weak system on finite dimensional bases.

For the sake of simplicity, we restrict ourselves to one of the two following cases, although this is not mandatory:

(I) either $\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2$ and $\mathcal{X}_d = \emptyset$, hence, $\alpha = (\alpha_1 \ \alpha_2)^\top$;
(II) or, $\mathcal{X} = \mathcal{X}_1$ and $\mathcal{X}_d = \mathcal{X}_2$.

In (I), we note $f_1 := \partial_t \alpha_1$ and $f_2 := \partial_t \alpha_2$; in (II), $f_1 := \partial_t \alpha$ and $f_2 := f_d$. The efforts are e_1 and e_2 in both cases.

Let $\varphi \in D(L)$ and $\psi \in D(K)$, the weak form of (1) reads:

$$\begin{cases} (f_1, \varphi)_{\mathcal{X}_1} = (-K e_2, \varphi)_{\mathcal{X}_1}, \\ (f_2, \psi)_{\mathcal{X}_2} = (L e_1, \psi)_{\mathcal{X}_2}, \end{cases}$$

Using (4) on, either the first or the second line, makes appear (y_1, u_2) or (u_1, y_2) resp.; e.g., the latter case reads:

$$(f_2, \psi)_{\mathcal{X}_2} = (e_1, K\psi)_{\mathcal{X}_1} + \langle u_1, \beta_1 \psi \rangle_{\mathcal{U}_1, \mathcal{Y}_1} + \langle \gamma_2 \psi, y_2 \rangle_{\mathcal{U}_2, \mathcal{Y}_2}.$$

The weak form for the other control u_2 is obtained thanks to test functions $\zeta_2 \in \mathcal{U}_2 \cap \mathcal{Y}_2$:

$$\langle u_2, \zeta_2 \rangle_{\mathcal{U}_2, \mathcal{Y}_2} = \langle \gamma_2 e_2, \zeta_2 \rangle_{\mathcal{U}_2, \mathcal{Y}_2},$$

and similarly for the other observation y_1 with test functions $\zeta_1 \in \mathcal{U}_1 \cap \mathcal{Y}_1$:

$$\langle \zeta_1, y_1 \rangle_{\mathcal{U}_1, \mathcal{Y}_1} = \langle \zeta_1, \beta_1 e_2 \rangle_{\mathcal{U}_1, \mathcal{Y}_1}.$$

Given four finite-dimensional families (e.g., finite elements bases), $(\varphi^i)_{1 \le i \le N_1}$ to approximate (in space) f_1 and e_1; $(\psi^k)_{1 \le k \le N_2}$ for f_2 and e_2; $(\zeta_1^{m_1})_{1 \le m_1 \le N_\partial^1}$ for u_1 and y_1; and $(\zeta_2^{m_2})_{1 \le m_2 \le N_\partial^2}$ for u_2 and y_2, leads to the system:

$$\mathrm{Diag}\begin{bmatrix} \boldsymbol{M}_1 \\ \boldsymbol{M}_2 \\ \boldsymbol{M}_\partial^1 \\ \boldsymbol{M}_\partial^2 \end{bmatrix} \begin{pmatrix} \underline{f_1} \\ \underline{f_2} \\ -\underline{y_1} \\ \underline{u_2} \end{pmatrix} = \begin{bmatrix} 0 & -\boldsymbol{D}^\top & 0 & 0 \\ \boldsymbol{D} & 0 & \boldsymbol{B}_1 & -\boldsymbol{B}_2^\top \\ 0 & -\boldsymbol{B}_1^\top & 0 & 0 \\ 0 & \boldsymbol{B}_2 & 0 & 0 \end{bmatrix} \begin{pmatrix} \underline{e_1} \\ \underline{e_2} \\ \underline{u_1} \\ -\underline{y_2} \end{pmatrix} \quad (6)$$

where an underlined variable denotes the vector of coefficients in the corresponding basis, the mass matrices are:

$$(\boldsymbol{M}_1)_{ij} := \left(\varphi^j, \varphi^i\right)_{\mathcal{X}_1}, \quad (\boldsymbol{M}_2)_{k\ell} := \left(\psi^\ell, \psi^k\right)_{\mathcal{X}_2},$$
$$(\boldsymbol{M}_\partial^\kappa)_{mn} := \langle \zeta_\kappa^n, \zeta_\kappa^m \rangle_{\mathcal{U}_\kappa, \mathcal{Y}_\kappa}, \quad \text{for } \kappa = 1, 2;$$

the *rectangular* stiffness matrix $\boldsymbol{D}$ is $(\boldsymbol{D})_{kj} := \left(\varphi^j, K\psi^k\right)_{\mathcal{X}_2}$ and the control matrices are:

$$(\boldsymbol{B}_1)_{kn} := \left\langle \zeta_1^n, \beta_1 \psi^k \right\rangle_{\mathcal{U}_1, \mathcal{Y}_1}, \quad (\boldsymbol{B}_2)_{m\ell} := \left\langle \gamma_2 \psi^\ell, \zeta_2^m \right\rangle_{\mathcal{U}_2, \mathcal{Y}_2}.$$

Remark 2. With the choice to apply (4) on the second line, a strong regularity is implicitly assumed for the family $(\psi^k)_k$ to be *conform*, namely, $(\psi^k)_k \subset D(K)$, as often in mixed formulations; the other choice would imply alternatively: $(\varphi^i)_i \subset D(L)$.

3.2 Dirac structure and discrete Hamiltonian

Theorem 1. System (6) generates a Dirac structure.

Clearly, the proof relies on (van der Schaft and Jeltsema, 2014, Section 5.1), given that the l.h.s. of (6) is symmetric positive-definite; while the r.h.s. is skew-symmetric.

To obtain the discrete power balance, it remains to define the discrete Hamiltonian and discretize the closure relations (2) accordingly. The next theorem is a straightforward consequence of Theorem 1.

Theorem 2.

Case (I) Let $\mathcal{H}^d(\underline{\alpha_1}, \underline{\alpha_2}) := \mathcal{H}(\alpha_1^d, \alpha_2^d)$, where α_κ^d is the spatial approximation of the energy variables α_κ, $\kappa = 1, 2$.

If $\boldsymbol{M}_1 \underline{e_1} = \boldsymbol{H}_1[\alpha_1^d, \alpha_2^d]\, \underline{\alpha_1}$ and $\boldsymbol{M}_2 \underline{e_2} = \boldsymbol{H}_2[\alpha_1^d, \alpha_2^d]\, \underline{\alpha_2}$, with:

$$\boldsymbol{H}_\kappa[\alpha_1^d, \alpha_2^d] := \nabla_{\underline{\alpha_\kappa}}^{\boldsymbol{M}_\kappa} \mathcal{H}^d, \quad \text{for } \kappa = 1, 2;$$

where $\nabla^{\boldsymbol{M}}$ denotes the gradient taken in the metric induced by the matrix $\boldsymbol{M}$, then:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathcal{H}^d = \underline{u_1}^\top \boldsymbol{M}_\partial^1 \, \underline{y_1} + \underline{y_2}^\top \boldsymbol{M}_\partial^2 \, \underline{u_2}. \quad (7)$$

Case (II) Let $\mathcal{H}^d(\underline{\alpha}) := \mathcal{H}(\alpha^d)$, where α^d is the spatial approximation of the energy variable α.

If $\boldsymbol{M}_1\underline{e_1} = \boldsymbol{H}[\alpha^d]\,\underline{\alpha}$, with $\boldsymbol{H}[\alpha^d] := \nabla^{\boldsymbol{M}_1}_{\underline{\alpha}}\mathcal{H}^d$, then:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathcal{H}^d = -\underline{e_2}\,\boldsymbol{M}_2\,\underline{f_2} + \underline{u_1}^\top \boldsymbol{M}^1_\partial\,\underline{y_1} + \underline{y_2}^\top \boldsymbol{M}^2_\partial\,\underline{u_2}. \qquad (8)$$

Remark 3. In case (II), also the second closure relation in (2), $\mathcal{R}_d(f_d, e_d) = 0$, has to be discretized accordingly.

Remark 4. When the Hamiltonian is polynomial, the gradient conditions of Theorem 2 are equivalent to the discrete weak forms of the constitutive relations $e_\alpha := \delta_\alpha\mathcal{H}$.

4. EXAMPLES

From now on, $\Omega \in \mathbb{R}^d$ is a domain, with $d = 1, 2$ or 3 and with $\boldsymbol{n}$ the outward unit vector at the boundary. The Dirichlet trace operator is denoted γ_0, and the normal trace operator is $\gamma_\perp := \boldsymbol{n}\cdot\gamma_0$. For convenience, solutions are assumed regular enough to reduce the duality bracket at the boundary to a boundary integral. Furthermore, the boundary $\Gamma := \partial\Omega$ may be decomposed in two regions Γ_1, and Γ_2, such that $\Gamma_1 \cap \Gamma_2 = \emptyset$ and $\Gamma_1 \cup \Gamma_2 = \Gamma$.

4.1 The wave equation

Following Haine et al. (2023) and the references therein, the very first example of distributed linear pHs is the linear wave equation. Taking the Hamiltonian as the sum of the kinetic and potential energies, in function of the *linear momentum* $\alpha_p := \rho\partial_t w$ and the *strain* $\boldsymbol{\alpha_q} := \mathbf{grad}(w)$ respectively, where w stands for the deflection from the equilibrium position and ρ the mass density:

$$\mathcal{H}_w(\alpha_p, \boldsymbol{\alpha_q}) := \frac{1}{2}\int_\Omega \frac{|\alpha_p|^2}{\rho} + \frac{1}{2}\int_\Omega \boldsymbol{\alpha_q}^\top\,\overline{\boldsymbol{E}}\,\boldsymbol{\alpha_q}, \qquad (9)$$

with $\overline{\boldsymbol{E}}$ the Young's elasticity symmetric positive-definite second order tensor modulus. The co-state variables are:

$$e_p := \delta_{\alpha_p}\mathcal{H}_w = \frac{\alpha_p}{\rho}, \quad \boldsymbol{e_q} := \delta_{\boldsymbol{\alpha_q}}\mathcal{H}_w = \overline{\boldsymbol{E}}\,\boldsymbol{\alpha_q}, \qquad (10)$$

i.e., the *velocity* and the *stress* respectively. Hence, the pHs modeling the linear wave equation may be written as:

$$\begin{pmatrix}\partial_t\alpha_p\\ \partial_t\boldsymbol{\alpha_q}\end{pmatrix} = \begin{bmatrix}0 & \mathrm{div}\\ \mathbf{grad} & 0\end{bmatrix}\begin{pmatrix}e_p\\ \boldsymbol{e_q}\end{pmatrix}, \begin{pmatrix}e_p\\ \boldsymbol{e_q}\end{pmatrix} = \begin{bmatrix}\rho^{-1} & 0\\ 0 & \overline{\boldsymbol{E}}\end{bmatrix}\begin{pmatrix}\alpha_p\\ \boldsymbol{\alpha_q}\end{pmatrix}. \qquad (11)$$

Along the solution, the power balance computes:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathcal{H}_w = \int_{\partial\Omega}\gamma_0(e_p)\,\gamma_\perp(\boldsymbol{e_q}). \qquad (12)$$

Assume that we control the velocity, hence:

$$u_w = \gamma_0(e_p), \quad y_w = \gamma_\perp(\boldsymbol{e_q}). \qquad (13)$$

The discretized system with the PFEM may be obtained by integration by part of the second line of the first system of (11):

$$\mathrm{Diag}\begin{bmatrix}\boldsymbol{M}_p\\ \boldsymbol{M}_q\\ \boldsymbol{M}_\partial\end{bmatrix}\begin{pmatrix}\dot{\underline{\alpha_p}}\\ \dot{\underline{\alpha_q}}\\ -\underline{y_w}\end{pmatrix} = \begin{bmatrix}0 & -\boldsymbol{D}^\top & 0\\ \boldsymbol{D} & 0 & \boldsymbol{B}\\ 0 & -\boldsymbol{B}^\top & 0\end{bmatrix}\begin{pmatrix}\underline{e_p}\\ \underline{e_q}\\ \underline{u_w}\end{pmatrix},$$

where $(\boldsymbol{M}_p)_{ij} := \left(\varphi^j, \varphi^i\right)_{L^2}$, $(\boldsymbol{M}_q)_{k\ell} := \left(\psi^\ell, \psi^k\right)_{(L^2)^d}$, the stiffness matrix is $(\boldsymbol{D})_{kj} := -\left(\varphi^j, \mathrm{div}(\psi^k)\right)_{L^2}$, the control matrix is $(\boldsymbol{B})_{kn} := \left(\zeta^n, \gamma_\perp(\psi^k)\right)_{L^2(\partial\Omega)}$. Then, adding the constraints:

$$\begin{bmatrix}\boldsymbol{M}_p & 0\\ 0 & \boldsymbol{M}_q\end{bmatrix}\begin{pmatrix}\underline{e_p}\\ \underline{e_q}\end{pmatrix} = \begin{bmatrix}\boldsymbol{Q}_p & 0\\ 0 & \boldsymbol{Q}_q\end{bmatrix}\begin{pmatrix}\underline{\alpha_p}\\ \underline{\alpha_q}\end{pmatrix},$$

where $(\boldsymbol{Q}_p)_{ij} := \left(\varphi^j, \rho^{-1}\varphi^i\right)_{L^2}$, $(\boldsymbol{Q}_q)_{k\ell} := \left(\psi^\ell, \overline{\boldsymbol{E}}\psi^k\right)_{(L^2)^d}$, leads to the discrete power balance (7), i.e., in our case:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathcal{H}^d_w = \underline{u_w}^\top\,\boldsymbol{M}_\partial\,\underline{y_w},$$

which perfectly mimics the continuous version (12).

Remark 5. In this particular case where the Hamiltonian is quadratic, the constitutive relations are linear and may be *substituted*, leading to the *co-state formulation*:

$$\mathrm{Diag}\begin{bmatrix}\boldsymbol{P}_p\\ \boldsymbol{P}_q\\ \boldsymbol{M}_\partial\end{bmatrix}\begin{pmatrix}\dot{\underline{e_p}}\\ \dot{\underline{e_q}}\\ -\underline{y_w}\end{pmatrix} = \begin{bmatrix}0 & -\boldsymbol{D}^\top & 0\\ \boldsymbol{D} & 0 & \boldsymbol{B}\\ 0 & -\boldsymbol{B}^\top & 0\end{bmatrix}\begin{pmatrix}\underline{e_p}\\ \underline{e_q}\\ \underline{u_w}\end{pmatrix},$$

where $(\boldsymbol{P}_p)_{ij} := \left(\varphi^j, \rho\varphi^i\right)_{L^2}$, $(\boldsymbol{P}_q)_{k\ell} := \left(\psi^\ell, \overline{\boldsymbol{E}}^{-1}\psi^k\right)_{(L^2)^d}$.

4.2 The heat equation

Let us denote by T the temperature, and consider the first law of thermodynamics, which reads in a solid medium:

$$\rho\,\partial_t e = -\mathrm{div}\left(\boldsymbol{J}_Q\right),$$

where ρ is the mass density, e is the internal energy density, and $\boldsymbol{J}_Q$ is the heat flux. Furthermore, we assume Dulong-Petit's model, i.e., $e = c_v T$, where c_v is the specific heat capacity at constant volume (isochoric process).

Following Serhani et al. (2019a), several choices are possible for the Hamiltonian, leading to different pHs representations of the so-called heat equation.

The easiest model consists in considering the squared L^2-weighted-norm of the temperature as Hamiltonian:

$$\mathcal{H}_h(T) := \frac{1}{2}\int_\Omega \rho c_v\,|T|^2. \qquad (14)$$

Then, the co-state variable is given by:

$$\delta_T\mathcal{H}_h = T, \qquad (15)$$

where the inner product in L^2 has been weighted by ρc_v. Finally, using Fourier's law: $\boldsymbol{J}_Q = -\overline{\boldsymbol{\Lambda}}\,\mathbf{grad}(T)$, with $\overline{\boldsymbol{\Lambda}}$ the symmetric positive-definite second order tensor heat conductivity, leads to:

$$\begin{pmatrix}\rho c_v\partial_t T\\ \boldsymbol{f}_Q\end{pmatrix} = \begin{bmatrix}0 & -\mathrm{div}\\ -\mathbf{grad} & 0\end{bmatrix}\begin{pmatrix}T\\ \boldsymbol{J}_Q\end{pmatrix}, \quad \boldsymbol{J}_Q = \overline{\boldsymbol{\Lambda}}\,\boldsymbol{f}_Q. \qquad (16)$$

Along the solutions, the power balance follows:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathcal{H}_h = -\int_\Omega\left\|\overline{\boldsymbol{\Lambda}}^{\frac{1}{2}}\boldsymbol{f}_Q\right\|^2 - \int_{\partial\Omega}\gamma_0(T)\,\gamma_\perp(\boldsymbol{J}_Q), \qquad (17)$$

which stands for the expected decay of a parabolic system.

We control the opposite of the normal heat flux on Γ_1:

$$u^1_h = -\gamma_\perp(\boldsymbol{J}_Q), \quad y^1_h = \gamma_0(T), \qquad (18)$$

and the temperature on the complement $\Gamma_2 := \partial\Omega\setminus\Gamma_1$:

$$u^2_h = \gamma_0(T), \quad y^2_h = -\gamma_\perp(\boldsymbol{J}_Q). \qquad (19)$$

Applying the PFEM by integrating by part the first line of the first system of (16) gives:

$$\text{Diag}\begin{bmatrix} \boldsymbol{P}_T \\ \boldsymbol{M}_Q \\ \boldsymbol{M}_\partial^1 \\ \boldsymbol{M}_\partial^2 \end{bmatrix}\begin{pmatrix} \dot{\underline{T}} \\ \underline{f_Q} \\ -\underline{y_h^1} \\ \underline{u_h^2} \end{pmatrix} = \begin{bmatrix} 0 & \boldsymbol{G} & \boldsymbol{B}_1 & -\boldsymbol{B}_2^\top \\ -\boldsymbol{G}^\top & 0 & 0 & 0 \\ -\boldsymbol{B}_1^\top & 0 & 0 & 0 \\ \boldsymbol{B}_2 & 0 & 0 & 0 \end{bmatrix}\begin{pmatrix} \underline{T} \\ \underline{J_Q} \\ \underline{u_h^1} \\ -\underline{y_h^2} \end{pmatrix}, \tag{20}$$

with $(\boldsymbol{P}_T)_{ij} := \left(\varphi^j, \rho c_v \varphi^i\right)_{L^2}$, $(\boldsymbol{M}_Q)_{k\ell} := \left(\psi^\ell, \psi^k\right)_{(L^2)^d}$, $(\boldsymbol{G})_{i\ell} := \left(\psi^\ell, \mathbf{grad}(\varphi^i)\right)_{(L^2)^d}$, $(\boldsymbol{B}_1)_{in} := \left(\zeta_1^n, \gamma_0(\varphi^i)\right)_{L^2(\Gamma_1)}$, $(\boldsymbol{B}_2)_{mj} := \left(\zeta_2^n, \gamma_0(\varphi^i)\right)_{L^2(\Gamma_2)}$. Furthermore, the Fourier's law in (16) writes:

$$\boldsymbol{M}_Q \underline{J_Q} := \boldsymbol{L}_Q \underline{f_Q}, \qquad (\boldsymbol{L}_Q)_{k\ell} := \left(\psi^\ell, \overline{\boldsymbol{\Lambda}}\psi^k\right)_{(L^2)^d}. \tag{21}$$

Then, the discrete Hamiltonian $\mathcal{H}_h^d$ satisfies:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathcal{H}_h^d = -\underline{f_Q}^\top \boldsymbol{L}_Q\, \underline{f_Q} + \underline{u_h^1}^\top \boldsymbol{M}_\partial^1\, \underline{y_h^1} + \underline{y_h^2}^\top \boldsymbol{M}_\partial^2\, \underline{u_h^2},$$

which perfectly mimics the continuous power balance (17).

4.3 Boundary interactions: the heat–wave system

Following Haine et al. (2022), the coupling of the wave equation in a domain Ω_w and the heat equation in a distinct domain Ω_h, at the interface $\Gamma_1 := \partial\Omega_w \cap \partial\Omega_h$ between the two domains, reveals to be as simple as to consider the appropriate interconnection between the two pHs (11) and (16), namely, taking a *gyrator* interconnection relating (13) and (18) at the interface Γ_1. Together with homogeneous Dirichlet boundary conditions, this reads:

$$\begin{aligned} u_h^1 &= -y_w^1, \quad \text{and} \quad u_w^1 = y_h^1, \qquad \text{on } \Gamma_1, \\ u_w^2 &= 0, \qquad \text{on } \Gamma_2^w := \partial\Omega_w \setminus \Gamma_1, \\ u_h^2 &= 0, \qquad \text{on } \Gamma_2^h := \partial\Omega_h \setminus \Gamma_1, \end{aligned} \tag{22}$$

where the superscripts ^{1} and ^{2} denote the boundary regions to consider Γ_1 and Γ_2 respectively. Provided the boundary discretizations $(\zeta_w^m)_m = (\zeta_h^m)_m = (\zeta^m)_m$ coincide at the interface Γ_1 (although this is not mandatory), these conditions read at the discrete level:

$$\begin{aligned} \boldsymbol{M}_\partial^1 \underline{u_h^1} &= -\boldsymbol{M}_\partial^1 \underline{y_w^1}, \quad \text{and} \quad \boldsymbol{M}_\partial^1 \underline{u_w^1} = \boldsymbol{M}_\partial^1 \underline{y_h^1}, \\ \boldsymbol{M}_\partial^w \underline{u_w^2} &= 0, \quad \text{and} \quad \boldsymbol{M}_\partial^h \underline{u_h^2} = 0, \end{aligned} \tag{23}$$

with boundary mass matrices: $(\boldsymbol{M}_\partial^1)_{mn} = (\zeta^n, \zeta^m)_{L^2(\Gamma_1)}$, and $(\boldsymbol{M}_\partial^k)_{mn} = (\zeta_k^n, \zeta_k^m)_{L^2(\Gamma_2^k)}$, for $k = w, h$.

5. SCRIMP

5.1 Generality

The Python package SCRIMP was developed to simplify the implementation of the structure-preserving discretization presented in Section 3.

Essential milestones of the library are illustrated in Fig. 1. The workflow relies on well-known third-party libraries, namely: GMSH, GetFEM, PETSc and ParaView.

In Section 5.2, we describe the calls to do in a Python script to solve a 2D heat equation, described in Section 4.2. In Section 5.3, we present the Graphical User Interface (GUI) for creating interactively the same kind of code.

5.2 Example step-by-step

We now describe the language required to solve a 2D heat equation under Lyapunov Hamiltonian formulation,

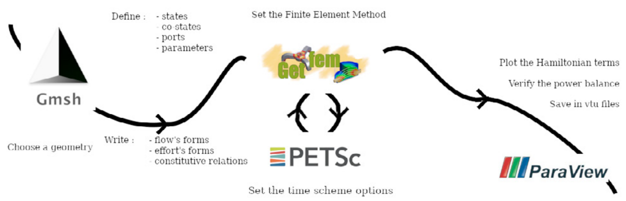


Fig. 1. SCRIMP milestones.

as proposed in Section 4.2; in particular, with opposite heat flux boundary control. We consider a manufactured solution on a rectangle domain $\Omega := (0,2) \times (0,1)$. We use $\mathbb{P}^2$ Lagrange finite element for the temperature T, $\mathbb{P}^1$ Lagrange finite element for $\boldsymbol{f}_Q$, $\boldsymbol{J}_Q$ and u_∂. The parameters are taken constant equal to one, i.e. $\rho = c_v = 1$, $\overline{\boldsymbol{\Lambda}} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$. Let $\boldsymbol{x} = (x\ y)^\top$ and $T(t,\boldsymbol{x}) = 4t + \|\boldsymbol{x}\|^2 + 3x - 5y$. Moreover, define $\boldsymbol{J}_Q(t,\boldsymbol{x}) = -\mathbf{grad}(T(t,\boldsymbol{x})) = -(2x+3\ 2y-5)^\top$. Let us define the following function as boundary control: $\forall t \geq 0,\ \boldsymbol{s} \in \partial\Omega$

$$u_\partial(t,\boldsymbol{s}) = \begin{cases} -5, & y = 0, \\ -7, & x = 2, \\ 3, & y = 1, \\ 3, & x = 0. \end{cases}$$

Then T is the solution of the following heat equation with boundary heat flux control:

$$\begin{cases} \partial_t T(t,\boldsymbol{x}) = \Delta T(t,\boldsymbol{x}), & \forall t \geq 0, \boldsymbol{x} \in \Omega, \\ T(0,\boldsymbol{x}) = \|\boldsymbol{x}\|^2 + 3x - 5y, & \forall \boldsymbol{x} \in \Omega, \\ -\mathbf{grad}(T(t,\boldsymbol{s})).\mathbf{n}(\boldsymbol{s}) = u_\partial(t,\boldsymbol{s}), & \forall t \geq 0, \boldsymbol{s} \in \partial\Omega. \end{cases}$$

The simulation is based on the weak formulation of the pHs (16), i.e., on (20)–(21).

The Python code of (20)–(21) corresponds to the following script. Some explanations are given after the code with circled integers as references to the different steps.

```
from scrimp import *                                                    01
heat = DPHS("real")
```

```
heat.set_domain(                                                        02
    Domain("Rectangle", {"L": 2.0,"l": 1.0, "h": 0.1}) )
```

```
states = [ State("T", "Temperature", "scalar-field") ]                  03
costates = [ CoState("T", "Temperature", states[0], substituted=
    True) ]
```

```
ports = [ Port("Heat flux", "f_Q", "J_Q", "vector-field") ]            04
control_ports = [
    Control_Port("B.C.(bottom)", "U_B", "Normal heat flux", "Y_B"
        , "Temperature", "scalar-field", 10),
    Control_Port("B.C.(right)", "U_R", "Normal heat flux", "Y_R",
        "Temperature", "scalar-field", 11),
    Control_Port("B.C.(top)", "U_T", "Normal heat flux", "Y_T", "
        Temperature", "scalar-field", 12),
    Control_Port("B.C.(left)", "U_L", "Normal heat flux", "Y_L",
        "Temperature", "scalar-field", 13), ]
```

```
FEMs = [                                                                05
    FEM("T", 2, "CG"),
    FEM("Heat flux", 1, "CG"),
    FEM("B.C.(bottom)", 1, "CG"),
    FEM("B.C.(right)", 1, "CG"),
    FEM("B.C.(top)", 1, "CG"),
    FEM("B.C.(left)", 1, "CG"), ]
```

```
parameters = [                                                          06
    Parameter("rho", "(Mass density)x(Heat capacity)", "scalar-
        field", "1.", "T"),
    Parameter("cv", "heat capacity", "scalar-field", "1.", "T"),
    Parameter("lambda", "heat conductivity", "tensor-field", "
        [[1.,0.],[0.,1.]]", "Heat flux") ]
```

```
for (state,costate) in zip(states,costates):                           07
    heat.add_state(state)
    heat.add_costate(costate)
```

```
for port in ports:
    heat.add_port(port)
for control_port in control_ports:
    heat.add_control_port(control_port)
for fem in FEMs:
    heat.add_FEM(fem)
for param in parameters:
    heat.add_parameter(param)
```

08
```
heat.hamiltonian.set_name("Lyapunov formulation")
terms = [ Term("L^2-norm", "0.5*rho*cv*T*T", [1], 0) ]
for term in terms:
     heat.hamiltonian.add_term(term)
```

09
```
bricks = [
 Brick("P_T", "rho*cv*T*Test_T", [1], dt=True, position="flow"),
 Brick("M_Q", "f_Q.Test_f_Q", [1], position="flow"),
 Brick("M_Y_B", "Y_B*Test_Y_B", [10], position="flow"),
 Brick("M_Y_R", "Y_R*Test_Y_R", [11], position="flow"),
 Brick("M_Y_T", "Y_T*Test_Y_T", [12], position="flow"),
 Brick("M_Y_L", "Y_L*Test_Y_L", [13], position="flow"),
 Brick("G", "J_Q.Grad(Test_T)", [1], position="effort"),
 Brick("-G^T", "-Grad(T).Test_f_Q", [1], position="effort"),
 Brick("B_B", "U_B*Test_T", [10], position="effort"),
 Brick("B_R", "U_R*Test_T", [11], position="effort"),
 Brick("B_T", "U_T*Test_T", [12], position="effort"),
 Brick("B_L", "U_L*Test_T", [13], position="effort"),
 Brick("-B_B^T", "-T*Test_Y_B", [10], position="effort"),
 Brick("-B_R^T", "-T*Test_Y_R", [11], position="effort"),
 Brick("-B_T^T", "-T*Test_Y_T", [12], position="effort"),
 Brick("-B_L^T", "-T*Test_Y_L", [13], position="effort"),
 Brick("-M_Q","-J_Q.Test_J_Q", [1], position="constitutive"),
 Brick("L_Q","f_Q.lambda.Test_J_Q",[1], position="constitutive")]
for brick in bricks:
  heat.add_brick(brick)
```

10
```
expressions = ["-5", "-7", "3", "3"]
for control_port, expression in zip(control_ports, expressions):
     heat.set_control(control_port.get_name(), expression)
heat.set_initial_value("T", "x*x+y*y+3*x-5*y")
```

11
```
heat.set_time_scheme(t_f=1., dt=0.01, init_step=False)
heat.solve()
```

12
```
heat.plot_Hamiltonian()
err_L2 = heat.get_quantity("pow(T-4*t-x*x-y*y-3*x+5*y,2)")
from math import sqrt
print("Maximal L^2-error over time:", sqrt(max(err_L2)))
```

01 To start, we import SCRIMP and initialize a real distributed pHs object as an instantiation of the class `DPHS`. For our model, we choose the name `heat` for the model. This model has to be completed before calling the time iteration solver, and we will show how to do it step by step. For convenience, at each step, we are going to create a list of specific objects, and we will show how to add those objects to the model in step 07.

02 The domain $\Omega = [0; 2] \times [0; 1]$ is set directly over the pHs model by passing a specific instantiation of the class Domain. Here, in particular, the domain is meshed using a characteristic mesh size parameter $h = 0.1$ using the built-in geometry `Rectangle`. For a given mesh, the domain is decomposed into regions, referenced by integers : `1` is linked to the full domain, and `10`, `11`, `12`, `13` are linked to the bottom, the right, the top and the left boundaries respectively. Information about available geometries and the flags of the regions can be found in the documentation. Notice that the GMSH library (Geuzaine and Remacle, 2009) is called to effectively create the mesh, and external GMSH-meshes (with affectation of the boundaries flags) can be imported manually.

03 The next step consists in the definition of the state and co-state variables. Again for ease, two lists of `State()` and `CoState()` objects are created. The variables of the model are defined at this stage by the choice of their names, `T` and of their dimensions (`scalar-field`, `vector-field`, `tensor-field`). A description of the nature of the variable is given (useful to generate the legends of output plots). The co-state is explicitly linked to the state variable using the object `states[0]`. The use of `substituted=True` specifies that the *co-state formulation* is chosen (Remark 5).

04 To complete the definition of the variables, we specify the ports in a same manner. Here, the variables are vector fields with the names `f_Q` and `J_Q`. The control ports require considering boundary terms and are defined by using `Control_Port` class. We specify the names `U_B`, `U_R`, `U_T`, `U_L` linked to the boundary regions `10`, `11`, `12`, `13`. The observations linked to each control terms are defined at this stage with the names `Y_B`, `Y_R`, `Y_T`, `Y_L`.

05 The selection of the Finite Element Method (FEM) for each variable is defined by the instances of the class `FEM`. We have access to the finite element defined in the GetFEM library (Renard and Poulios, 2020), such that the classical continuous and discontinuous methods with polynomials of degree k ($\mathbb{P}_k$), Raviart-Thomas of degree k ($\mathbb{RT}_k$), Brezzi-Douglas-Marini element of degree k ($\mathbb{BDM}_k$), just to name a few. All choices are possible for each variable, so for each one its name is recalled and the finite element method is chosen. We refer to the documentation for the list of possible choices.

06 The parameters of the model are defined with the instances of `Parameter` class. Here, we fix the names `rho`, `cv` and `lambda`, their dimensions and a word to provide some information for clarity. The values of the fixed parameters are specified at this stage.

07 At this step, all objects previously defined are added to the model with a call of the form `heat.add_*()`, where `*` is specific to the nature of the element to add.

08 For post-processing, it is interesting, for instance, to evaluate the Hamiltonian. This can be done by adding one or more `Term()` object to the model. Note that these calls can be postponed after the resolution.

09 The next step corresponds to the weak formulation defining the block matrices of the dpHs. We use the GetFEM formalism based on bricks that is embedded in our `Brick` class and defined with Generic Weak Form Language (GWFL), intended to be close to the structure of a standard weak formulation of boundary value problems. Test functions can be used w.r.t. any of the variables. They are identified by the prefix `Test_` followed by the corresponding variable name. All expressions appearing in that formalism must be defined previously as variables or parameters (Renard and Poulios, 2020). Each brick corresponds to an integral term, where the integration domain is specified with the flags of the different parts (regions) of the boundary. Extra information is given by the affectation of `position` that can be equals to `flow` or `effort` to specify (left- or right-hand side) where the block matrix must be added into the model. When `position` is affected to `constitutive` the resulting brick corresponds to the constitutive relations, for that latter case, all terms are added on the same side, it is the reason why a negative sign is used in `-M_Q`. The first brick differs from the others with the additional information `dt=True` that underlines the fact that the temperature `T` appears in the model with a time derivative.

10 The control and initial values still need to be defined. It is realized by linking given values to the control ports

already defined. The model `heat` is enriched by the call `set.control()`. The initial value of the state variable `T` is given using `set_initial_value()`.

⑪ Everything is ready for the time evolution solver. In the background, the PETSc/TS library (Abhyankar et al., 2018) is called. The final time is set with `t_f` and the time step with `dt`. Using `init_step` is for consistent first step, not needed here. The simulation runs with the `solve()` method of the pHs model.

⑫ Post-processing can be done, for example, to plot the time evolution of the Hamiltonian, see Fig.2. Furthermore, integral quantities may be computed for each time step, e.g., `get_quantity("pow(T-4*t-x*x-y*y-3*x+5*y,2)")` is used to compute the L^2-error $\int_\Omega |T-4t-\|\boldsymbol{x}\|^2-3x+5y|^2$ between the computed temperature and the exact solution. The max over time is equal to 7.7×10^{-13} in this example.

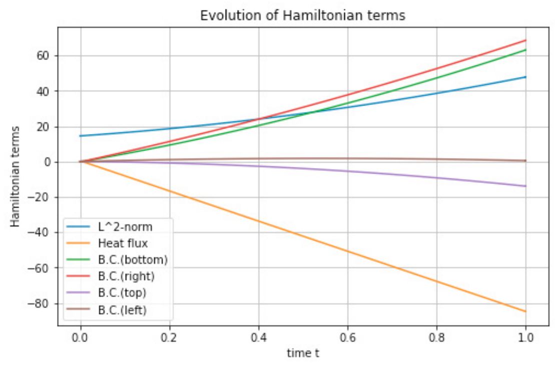
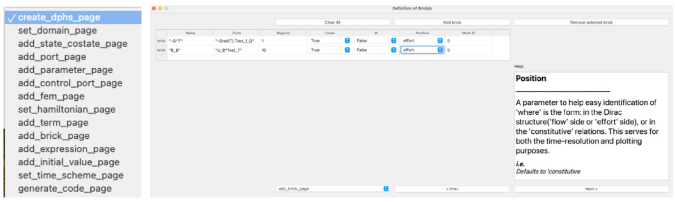


Fig. 2. Plot generated for the Hamiltonian.

5.3 *Graphical User Interface*

A Graphical User Interface (GUI) was developed to reduce the repetitive task required to program new pHs formulations and to avoid learning in detail the syntax of the library. Each step described in the previous section can be informed, see the dedicated pages available in Fig. 3 (left). It is also possible to fill dedicated fields, see for example Fig. 3 (right), to reproduce the action realized in step ⑨. Interactive access to the associated documentation is possible. From scratch, it is possible to construct the full model interactively. The generated script can be modified into any text-editor or executed on the fly.



	Name	Form	Regions	Linear	dt	Position	Mesh ID
brick	"-G*T"	"-Grad(T).Test_f_Q"	1	True	False	effort	0
brick	"B_B"	"U_B*Test_T"	10	True	False	effort	0

Help:

Position

A parameter to help easy identification of 'where' is the form: in the Dirac structure('flow' side or 'effort' side), or in the 'constitutive' relations. This serves for both the time-resolution and plotting purposes.

i.e.
Defaults to 'constitutive

Fig. 3. GUI pages (left), one example of interface (right).

6. CONCLUSION

We have presented the Python package SCRIMP useful to easily start simulations on distributed port-Hamiltonian systems, through examples stemming from mechanics and thermodynamics. In the future, SCRIMP aims at being a tool for the pHs community, both for rapid experiments on new models and more advanced simulations, including control facilities and system analysis. The examples are available on the GitHub of SCRIMP, among others.

ACKNOWLEDGEMENTS

The authors thank the former and present developers of SCRIMP, namely A. Bendimerad-Hohl, A. Brugnoli, F. Monteghetti, X. Vasseur and A. Serhani.

REFERENCES

Abhyankar, S., Brown, J., Constantinescu, E.M., Ghosh, D., Smith, B.F., and Zhang, H. (2018). PETSc/TS: A Modern Scalable ODE/DAE Solver Library. Technical report.

Ayachit, U. (2015). *The ParaView Guide: A Parallel Visualization Application*. Kitware, Inc., Clifton Park, NY, USA.

Brugnoli, A., Haine, G., and Matignon, D. (2023). Stokes-Dirac structures for distributed parameter port-Hamiltonian systems: An analytical viewpoint. *Communications in Analysis and Mechanics*, 15(3), 362–387.

Brugnoli, A., Rashad, R., and Stramigioli, S. (2022). Dual field structure-preserving discretization of port-Hamiltonian systems using finite element exterior calculus. *Journal of Computational Physics*, 471, 111601.

Cardoso-Ribeiro, F.L., Matignon, D., and Lefèvre, L. (2021). A Partitioned Finite-Element Method for power-preserving discretization of open systems of conservation laws. *IMA J. Mathematical Control and Information*, 38(2), 493–533.

Egger, H. (2019). Structure preserving approximation of dissipative evolution problems. *Numerische Mathematik*, 143(1), 85–106.

Egger, H., Habrich, O., and Shashkov, V. (2021). On the energy stable approximation of hamiltonian and gradient systems. *Computational Methods in Applied Mathematics*, 21(2), 335–349.

Geuzaine, C. and Remacle, J.F. (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.

Haine, G., Matignon, D., and Monteghetti, F. (2022). Long-time behavior of a coupled heat-wave system using a structure-preserving finite element method. *Math. Reports*, 24(74)(1-2), 187–215.

Haine, G., Matignon, D., and Serhani, A. (2023). Numerical analysis of a structure-preserving space-discretization for an anisotropic and heterogeneous boundary controlled N-dimensional wave equation as a port-Hamiltonian system. *International Journal of Numerical Analysis & Modeling*, 20(1), 92–133.

Kotyczka, P. (2019). *Numerical Methods for Distributed Parameter Port-Hamiltonian Systems*. TUM.University Press.

Rashad, R., Califano, F., van der Schaft, A.J., and Stramigioli, S. (2020). Twenty years of distributed port-Hamiltonian systems: a literature review. *IMA Journal of Mathematical Control and Information*, 37(4), 1400–1422.

Renard, Y. and Poulios, K. (2020). GetFEM: Automated FE modeling of multiphysics problems based on a generic weak form language. URL `https://hal.science/hal-02532422`.

Schulze, P. (2023). Structure-preserving time discretization of port-hamiltonian systems via discrete gradient pairs. ArXiv:2311.00403.

Serhani, A., Matignon, D., and Haine, G. (2019a). Anisotropic heterogeneous n-D heat equation with boundary control and observation: I. Modeling as port-Hamiltonian system. *IFAC-PapersOnLine*, 52(7), 51–56. (TFMST 2019).

Serhani, A., Matignon, D., and Haine, G. (2019b). Anisotropic heterogeneous n-D heat equation with boundary control and observation: II. Structure-preserving discretization. *IFAC-PapersOnLine*, 52(7), 57–62. (TFMST 2019).

van der Schaft, A.J. and Jeltsema, D. (2014). Port-Hamiltonian Systems Theory: An Introductory Overview. *Foundations and Trends® in Systems and Control*, 1(2), 173–378.

van der Schaft, A.J. and Maschke, B.M. (2002). Hamiltonian formulation of distributed-parameter systems with boundary energy flow. *Journal of Geometry and Physics*, 42(1–2), 166–194.