PyPHS: An open source Python library dedicated to the generation of passive guaranteed simulation code for multiphysical (audio) systems

Antoine Falaize[a]

IRCAM seminar - Research and Technology

04/12/2017

[a] Postdoc in the team M2N, LaSIE UMR CNRS 7356, ULR, La Rochelle, France

Introduction

Objective : Numerical simulation of multiphysical systems

- electronics, mechanics, magnetics, thermics.
- nonlinearities, non ideal behaviors.
- high complexity.

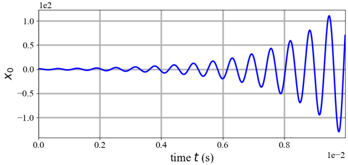
Standard approachs

1. Build a set of elementary physical models.
2. Build a system as the connection of these models.
3. Apply *ad-hoc* discretization methods.

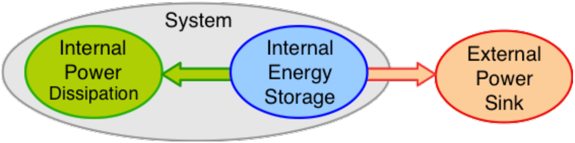
Difficulties

D1 The stability of a single model simulation is not guaranteed.

D2 This is even worst for the interconnected system.



But physical systems are passive systems



Power-balance $\frac{\mathrm{d}E}{\mathrm{d}t} + P_{\mathrm{D}} + P_{\mathrm{S}} = 0$

with

- Energy E (J),
- Dissipated power P_{D} (W),
- Sink Power P_{S} (W).

Our approach

1. **Structure physical models according to energy flows**

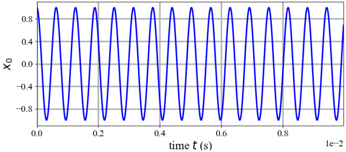
2. **Build a system as the structure preserving connection of these models**

3. **Apply a structure preserving discretization method**

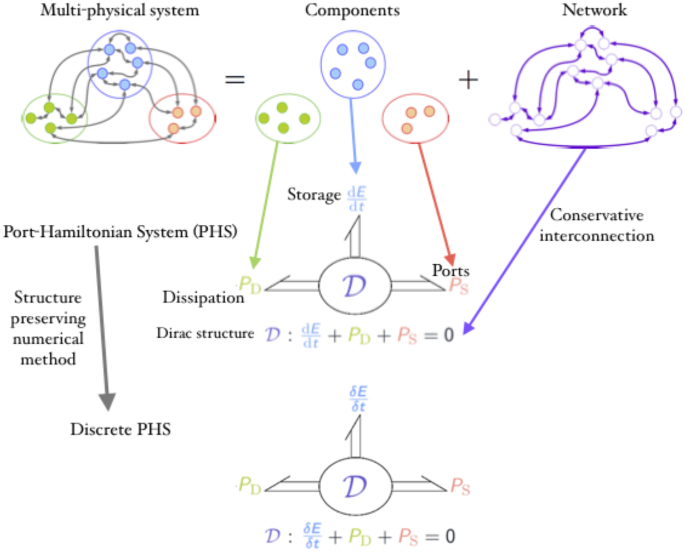
Result

D1 The stability of a single model simulation is guaranteed.

D2 The interconnected system inherits from this property.



Encoding of passivity in PyPHS



PyPHS : everything is formal

Networks are formal graph structures

- Use of NETWORKX[1] Python package.
- Creation and manipulation of graphs structures.

Model equations in symbolic form

- Use of SYMPY[2] Python package.
- A posteriori manipulation of system's equations.
- Automated generation of LaTeX documentation.

Numerical method is derived formally

- Also use SYMPY Python package.
- Symbolic optimization of the update equations.
- Easy analysis of the signal flow → Code generation.

---

1. see `https://networkx.github.io/`
2. see `http://www.sympy.org/en/index.html`

PyPHS background

Main tools

- Port-Hamiltonian Systems (PHS) formalism [3]
- Graph theory [4]

2012→2016

- ANR project HaMecMoPSys [5].
- PhD thesis of Antoine Falaize [6] in the team S3AM [7] at IRCAM - UMR STMS 9912 founded by EDITE.

2016→ ···

- Implementation of the scientific results obtained between 2012 and 2016.
- Further scientific developments.

---

3. Maschke, Van Der Schaft et Breedveld, "An intrinsic Hamiltonian formulation of network dynamics : Non-standard Poisson structures and gyrators", 1992.
4. Desoer et Kuh, Basic circuit theory, 2009.
5. see https://hamecmopsys.ens2m.fr/
6. Falaize, "Modélisation, simulation, génération de code et correction de systèmes multi-physiques audios : Approche par réseau de composants et formulation Hamiltonienne à Ports", 2016.
7. see http://s3am.ircam.fr/?lang=en

Table of contents

1. Network

PyPHS inputs : `Graph` and `Netlist`.

2. Components

PyPHS dictionary elements : `Graph` objects.

3. Port-Hamiltonian Systems

PyPHS `Core` object : Passive-guaranteed structure.

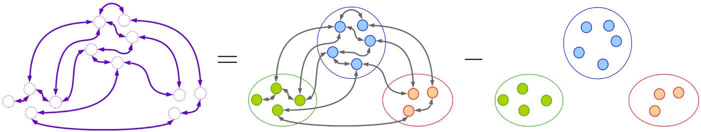
4. Numerical Method

PyPHS `Method` object : Structure preserving numerical scheme.

5. Code generation

PyPHS outputs : Python, C++, Juce and FAUST.

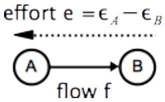
Network



Directed graphs with self loops

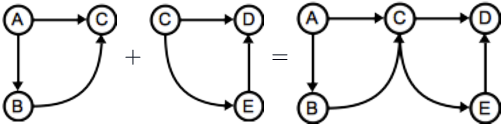
- Set of nodes $\mathtt{N} = \{\mathtt{N}_1, \cdots, \mathtt{N}_n\}$.
- Set of edges $\mathtt{B} = \{\mathtt{B}_1, \cdots, \mathtt{B}_n\}$ with $\mathtt{B}_i = (n, m) \in \mathtt{N}^2$.
- Direction : $\mathtt{B}_i \equiv n \rightarrow m$

Receiver convention



- Each edge $\equiv$ two power variables : Flow and Effort
- Flow $\mathfrak{f}$: defined through the edges.
- Effort $\mathfrak{e}$: defined across the edges as the difference of two quantities.
- Power received by the edge : $P = \mathfrak{f}\,\mathfrak{e}$ (W).

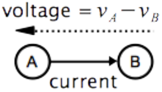
Connection $\equiv$ Nodes identification



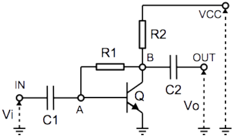
Electrical graphs

Physical quantities

Flow = Current (A), Effort = Voltage (V), ϵ = Potential (V)

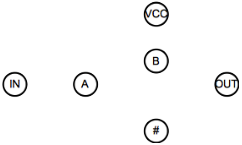


Example system



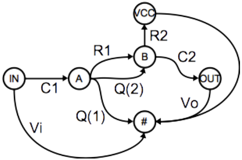
2 Capacitors C1 and C2,
2 Resistors R1 and R2,
1 BJ transistor Q,
3 Ports Vi, Vo and Vc.

Nodes



Graph nodes = Circuit nodes
Ground = Reference node #

Graph

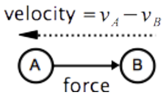


Graph edges = Circuit components
Note Q $\equiv$ 2 edges

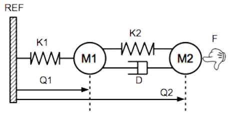
Mechanical graphs

Physical quantities

Flow = Force (N), Effort = Velocity (m/s), ϵ = point velocity (m/s)



Example system

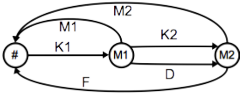


2 Masses M1 and M2,
2 Springs K1 and K2,
1 Damper,
1 Port F.

Nodes

Graph nodes = unique velocities
Reference velocity = node #

Graph

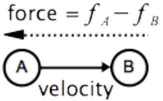


Graph edges = components

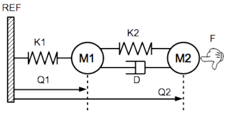
Mechanical graphs (dual)

Physical quantities

Flow = Velocity (m/s), Effort = force (N), ϵ = some force (N)

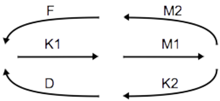


Example system



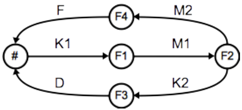
2 Masses M1 and M2,
2 Springs K1 and K2,
1 Damper,
1 Port F.

Edges



Serial edges = same velocity

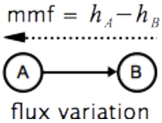
Graph



Add nodes to close the graph

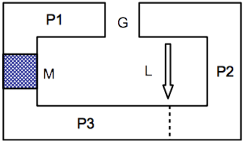
Magnetical graphs

Physical quantities



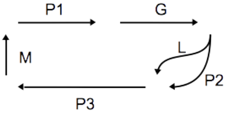
Flow = flux variation (V), Effort = magnetomotive force (A), ϵ = some mmf (A)

Example system



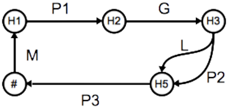
3 metal pieces P1, P2, P3,
1 Air gap G,
1 Flux leakage L,
1 Port M (magnet).

Edges



Serial = same magnetic flux

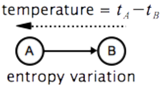
Graph



Add nodes to close the graph

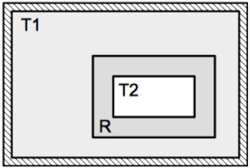
Thermal graphs

Physical quantities



Flow = entropy variation (W/K), Effort = temperature (K), ϵ = temperature (K)

Example system

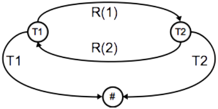


2 Heat capacities T1 and T2,
1 Heat transfer R,

Nodes

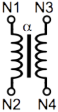
Graph nodes = temperatures
Reference temperature = node #

Graph



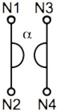
Graph edges = components
Note R = 2 edges (irreversibility)

Transformer



$$\begin{aligned} e_{3\to 4} &= \frac{1}{\alpha}\, e_{1\to 2}, \\ f_{3\to 4} &= -\alpha\, f_{1\to 2}, \\ [\alpha] &= \frac{[f_{3\to 4}]}{[f_{1\to 2}]}. \end{aligned}$$

Gyrator



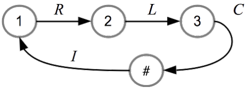
$$\begin{aligned} e_{3\to 4} &= \alpha\, f_{1\to 2}, \\ f_{3\to 4} &= -\frac{1}{\alpha}\, e_{1\to 2}, \\ [\alpha] &= \frac{[e_{3\to 4}]}{[f_{1\to 2}]}. \end{aligned}$$

Conserving connection

In each case : $P_{3\to 4} = -P_{1\to 2}$

Kirchhoff laws on graphs

Example : RLC



Incidence Matrix

$$[\Gamma]_{n,b} = \begin{cases} 1 & \text{if edge } b \text{ is ingoing node } n, \\ -1 & \text{if edge } b \text{ is outgoing node } n. \end{cases}$$

$$\Gamma = \begin{array}{c} \begin{array}{cccc} \mathtt{B}_R & \mathtt{B}_L & \mathtt{B}_C & \mathtt{B}_I \end{array} \\ \left(\begin{array}{cccc} 0 & 0 & +1 & -1 \\ \hline -1 & 0 & 0 & 0 \\ +1 & -1 & 0 & 0 \\ 0 & +1 & -1 & +1 \end{array}\right) \end{array} \begin{array}{c} \\ \# \\ \mathtt{N}_1 \\ \mathtt{N}_2 \\ \mathtt{N}_3 \end{array}$$

Reduced incidence Matrix

Arbitrary reference node $\#$

$$\Gamma = \begin{array}{c} \begin{array}{ccc} \mathtt{B}_1 & \cdots & \mathtt{B}_{n_\mathtt{B}} \end{array} \\ \left(\begin{array}{c} \gamma_0 \\ \hline \\ \gamma \\ \\ \end{array}\right) \end{array} \begin{array}{c} \\ \# \\ \mathtt{N}_1 \\ \vdots \\ \mathtt{N}_{n_\mathtt{N}} \end{array},$$

Generalized Kirchhoff's laws

- Efforts $\mathfrak{e} \in \mathbb{R}^{n_\mathtt{B}}$, flows $\mathfrak{f} \in \mathbb{R}^{n_\mathtt{B}}$.
- Node quantities $\mathbf{p} \in \mathbb{R}^{n_\mathtt{N}}$.
- $\gamma^\intercal \mathbf{p} = \mathfrak{e}$, (KVL).
- $\gamma \mathfrak{f} = 0$, (KCL).

Dirac structure $\mathcal{D}$ = Kirchhoff laws on graphs

Edges splitting

Depends on the components

Flow controlled $\mathfrak{f} \to$ edge $\to \mathfrak{e}$.

Effort controlled $\mathfrak{e} \to$ edge $\to \mathfrak{f}$.

Outputs $\mathfrak{a} \in \mathbb{R}^{n_B}$.

Inputs $\mathfrak{b} \in \mathbb{R}^{n_B}$.

Realizability criterion

If $\gamma_{\mathfrak{f}}$ is invertible, then $\exists! \mathbf{J}$ s.t. $\mathfrak{b} = \mathbf{J} \cdot \mathfrak{a}$.

Dirac structure

1. $\mathfrak{e}_{\mathfrak{b}} = \gamma_{\mathfrak{e}}^{\mathsf{T}} \cdot \mathbf{p}$ and $\mathfrak{e}_{\mathfrak{a}} = \gamma_{\mathfrak{f}}^{\mathsf{T}} \cdot \mathbf{p}$,
2. $\gamma_{\mathfrak{e}} \mathfrak{f}_{\mathfrak{a}} = -\gamma_{\mathfrak{f}} \cdot \mathfrak{f}_{\mathfrak{b}}$,
3. $\gamma_{\mathfrak{ef}} = \gamma_{\mathfrak{f}}^{-1} \cdot \gamma_{\mathfrak{e}}$,

RLC example

$\mathtt{B}_L$ is $\mathfrak{e}$-controlled, $\mathtt{B}_R, \mathtt{B}_C, \mathtt{B}_I$ are $\mathfrak{f}$-controlled.

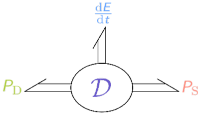
$$\left(\begin{array}{c|c} \multicolumn{2}{c}{\gamma_0} \\ \hline \gamma_{\mathfrak{e}} & \gamma_{\mathfrak{f}} \end{array}\right) = \begin{array}{c} \begin{array}{c|ccc} \mathtt{B}_L & \mathtt{B}_R & \mathtt{B}_C & \mathtt{B}_I \end{array} \\ \left(\begin{array}{c|ccc} 0 & 0 & +1 & -1 \\ \hline 0 & -1 & 0 & 0 \\ -1 & +1 & 0 & 0 \\ +1 & 0 & -1 & +1 \end{array}\right) \end{array} \begin{array}{c} \\ \mathtt{N}_0 \\ \mathtt{N}_1 \\ \mathtt{N}_2 \\ \mathtt{N}_3 \end{array} .$$

$$\underbrace{\begin{pmatrix} \mathfrak{e}_{\mathfrak{b}} \\ \mathfrak{f}_{\mathfrak{b}} \end{pmatrix}}_{\mathfrak{b}} = \underbrace{\begin{pmatrix} 0 & \gamma_{\mathfrak{ef}}^{\mathsf{T}} \\ -\gamma_{\mathfrak{ef}} & 0 \end{pmatrix}}_{\mathbf{J}} \underbrace{\begin{pmatrix} \mathfrak{f}_{\mathfrak{a}} \\ \mathfrak{e}_{\mathfrak{b}} \end{pmatrix}}_{\mathfrak{a}} .$$

$\mathbf{J}$ is skew-symmetric $\Rightarrow \mathfrak{a}^{\mathsf{T}} \cdot \mathfrak{b} = \mathfrak{a}^{\mathsf{T}} \cdot \mathbf{J} \cdot \mathfrak{a} = 0$.

This is the Tellegen's theorem :

$\sum_n^{n_B} \mathfrak{e}_n \mathfrak{f}_n = \sum_n^{n_B} P_n = 0$.



Automated construction of the Dirac structure

Algorithme [8]

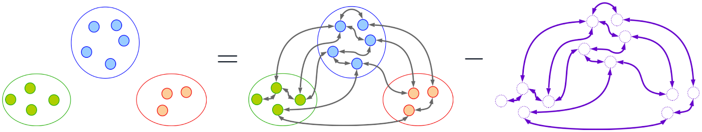
Data A **netlist** and a **dictionary** of components.

Résult

- If realizable :
  1. partition $\mathtt{B} = [\mathtt{B}_{\mathfrak{e}}, \mathtt{B}_{\mathfrak{f}}]$,
  2. structure $\mathfrak{b} = \mathbf{J} \cdot \mathfrak{a}$.
- Else : Realizability fault detection $\rightarrow$ the user correct the netlist.

8. Falaize et Hélie, "Passive guaranteed simulation of analog audio circuits : A port-Hamiltonian approach", 2016.

Components



Storage components (definitions)

Constitutive relation for component s

Storage function (Hamiltonian) H_s of the state x_s.

Stored energy $\mathrm{E}_s(t) = \mathrm{H}_s\big(x_s(t)\big) \geq 0$.

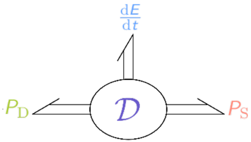
Received power $\frac{\mathrm{dE}_s}{\mathrm{d}t} = \mathrm{H}_s'(x_s)\,\frac{\mathrm{d}x_s}{\mathrm{d}t}$

Power variables for component s

Received power $\frac{\mathrm{dE}_s}{\mathrm{d}t} = \mathfrak{e}_s\,\mathfrak{f}_s$.

$\mathfrak{e}$-controlled $\mathfrak{e}_s = \frac{\mathrm{d}x_s}{\mathrm{d}t} \implies \mathfrak{f}_s = \mathrm{H}_s'(x_s)$.

$\mathfrak{f}$-controlled $\mathfrak{f}_s = \frac{\mathrm{d}x_s}{\mathrm{d}t} \implies \mathfrak{e}_s = \mathrm{H}_s'(x_s)$.



$\mathcal{D} : \frac{\mathrm{d}E}{\mathrm{d}t} + P_{\mathrm{D}} + P_{\mathrm{S}} = 0$

Total energy stored in n_{E} storage edges

- $\mathbf{x} = (x_1, \cdots, x_{n_{\mathrm{E}}})$.
- $\mathrm{E} = \mathrm{H}(\mathbf{x}) = \sum_{s=1}^{n_{\mathrm{E}}} \mathrm{H}_s(x_s) \geq 0$.
- $\frac{\mathrm{dE}}{\mathrm{d}t} = \nabla \mathrm{H}^{\mathsf{T}}\,\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \sum_{s=1}^{n_{\mathrm{E}}} \frac{\mathrm{dH}_s}{\mathrm{d}x_s}\,\frac{\mathrm{d}x_s}{\mathrm{d}t}$.

Storage components (examples)

Mass (flow=velocity, effort=force)

State momentum $x_m = m\, v_m$.

Hamiltonian kinetic energy $\mathrm{H}_m(x_m) = \frac{x_m^2}{2\,m}$.

Flow mass velocity $f_m = \mathrm{H}_m'(x_m) = \frac{x_m}{m}$.

Effort inertial force $e_m = \frac{\mathrm{d}x_m}{\mathrm{d}t} = m\frac{\mathrm{d}v_m}{\mathrm{d}t}$.

Capacitor

State charge q_C.

Hamiltonian electrostatic energy $\mathrm{H}_C(x_C) = \frac{x_C^2}{2\,C}$.

Flow current $f_C = \frac{\mathrm{d}x_C}{\mathrm{d}t} = \frac{\mathrm{d}q_C}{\mathrm{d}t}$.

Effort voltage $e_C = \mathrm{H}_C'(x_C) = \frac{x_C}{C}$.

Dissipative components (definitions)

Constitutive relation for component d

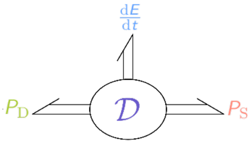
Dissipation function z_d of the variable w_d.

Received (dissipated) power $P_{\mathrm{D}\,d}(t) = z_d\big(w_d(t)\big) \geq 0$.

Power variables for component d

Received power $P_{\mathrm{D}\,d}(t) = \mathfrak{e}_d\,\mathfrak{f}_d \geq 0$

- **$\mathfrak{e}$-controlled** $\mathfrak{e}_d = w_d \implies \mathfrak{f}_d = z_d\big(w_d\big)$.
- **$\mathfrak{f}$-controlled** $\mathfrak{f}_d = w_d \implies \mathfrak{e}_d = z_d\big(w_d\big)$.



$\mathcal{D}$: $\frac{\mathrm{d}E}{\mathrm{d}t} + P_{\mathrm{D}} + P_{\mathrm{S}} = 0$

Total power dissipated in n_{D} dissipative edges

- $\mathbf{w} = (w_1, \cdots, w_{n_{\mathrm{D}}})$.
- $\mathbf{z}(\mathbf{w}) = (z_1(w_1), \cdots, z_{n_{\mathrm{D}}}(w_{n_{\mathrm{D}}}))$.
- $P_{\mathrm{D}} = \mathbf{z}(\mathbf{w})^{\mathsf{T}} \cdot \mathbf{w} = \sum_{d=1}^{n_{\mathrm{D}}} z_d(w_d)\, w_d \geq 0$.

Dissipative components (examples)

Dashpot (flow=force, effort=velocity)

Variable elongation velocity $w_D = v_D$.

Function resistance force $z_D(w_D) = D\, w_D$, with D $>$ 0.

Flow force $f_D = z_D(w_D) = D\, v_D$.

Effort velocity $e_D = w_D = v_D$.

Dissipated Power $P_{\mathrm{D}} = f_D\, e_D = R\, v_D^2$

Resistor

Variable current $w_R = i_R$.

Function resistance voltage $z_R(w_R) = R\, i_R$, with R $>$ 0.

Flow current $f_R = w_R = i_R$.

Effort velocity $e_R = z_R(w_R) = R\, i_R$.

Dissipated Power $P_{\mathrm{D}} = f_R\, e_R = R\, i_R^2$

Ports (definitions)

Input and output on port p

Actuated quantity u (input) and Observed quantity y (output).

Received Power $P_{\mathrm{S}p}(t) = u_p(t)\, y_p(t)$.

The power $P_{\mathrm{S}p}$ is the power that goes out of the system on port p.

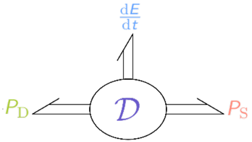
Ports are power sink.

Power variables for port p

Received power $P_{\mathrm{S}p}(t) = \mathfrak{e}_p\, \mathfrak{f}_p$

$\mathfrak{e}$-controlled $\mathfrak{e}_p = y_p \implies \mathfrak{f}_p = u_p$ (flow source).

$\mathfrak{f}$-controlled $\mathfrak{f}_p = y_p \implies \mathfrak{e}_p = u_p$ (effort source).



$\mathcal{D} : \frac{\mathrm{d}E}{\mathrm{d}t} + P_{\mathrm{D}} + P_{\mathrm{S}} = 0$

Total power on n_{S} port edges

- $\mathbf{u} = (u_1, \cdots, u_{n_{\mathrm{S}}})$.
- $\mathbf{y} = (y_1, \cdots, y_{n_{\mathrm{S}}})$.
- $P_{\mathrm{S}} = \mathbf{u}^{\mathsf{T}} \cdot \mathbf{y} = \sum_{p=1}^{n_{\mathrm{S}}} u_p\, y_p$.

Ports (examples)

Voltage source

Input voltage $u_U = v_U$.

Output current $y_U = i_U$.

Flow current $f_U = y_U$.

Effort voltage $e_U = u_U$.

Received Power $P_{\mathrm{S}} = f_U\, e_U = v_U\, i_U$.

Imposed force (flow=force, effort=velocity)

Input force$u_U = f_U$.

Output velocity $y_U = v_U$.

Flow force $f_U = u_U$.

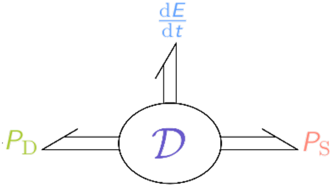
Effort velocity $e_U = y_U$.

Received Power $P_{\mathrm{S}} = f_U\, e_U = f_U\, v_U$.

PyPHS Dictionary (v0.2)

- **Mechanics (1D) :** masses, springs lin./nonlin. (cubic, saturating, etc.), lin./nonlin. damping, visco-elastic (fractional derivatives).
- **Electronics :** batteries, coils and lin./nonlin. capacitors, resistors, transistors, diodes, triodes.
- **Magnetics :** Magnets, lin./nonlin capacitors, resisto-inductor (fractional integrators).
- **Thermics :** heat sources, capacitors.
- **Connections :** electromagnetic couplings, electromechanic coupling, irreversible transfers, gyrators, transformers.

3. Port-Hamiltonian Systems



$$\mathcal{D} : \frac{\mathrm{d}E}{\mathrm{d}t} + P_{\mathrm{D}} + P_{\mathrm{S}} = 0$$

Putting all together

Components

Storage	$\mathbf{b_x} = \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t}$, $\mathbf{a_x} = \nabla \mathrm{H}(\mathbf{x})$
Dissipation	$\mathbf{b_w} = \mathbf{w}$, $\mathbf{a_w} = \mathbf{z}(\mathbf{w})$
Ports	$\mathbf{b_y} = \mathbf{y}$, $\mathbf{b_y} = \mathbf{u}$

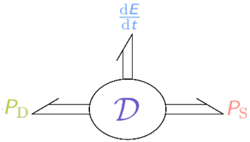
Network (Dirac structure)

$$\mathfrak{b} = \begin{pmatrix} \mathbf{b_x} \\ \mathbf{b_w} \\ \mathbf{b_y} \end{pmatrix} \text{ and } \mathfrak{a} = \begin{pmatrix} \mathbf{a_x} \\ \mathbf{a_w} \\ \mathbf{a_y} \end{pmatrix}$$

with $\mathfrak{b} = \mathbf{J} \cdot \mathfrak{a}$ and $\mathbf{J}^\intercal = -\mathbf{J}$.

This encodes the power balance

$$\begin{aligned} 0 &= \mathfrak{a}^\intercal \cdot \mathfrak{b} \\ &= \underbrace{\nabla \mathrm{H}(\mathbf{x})^\intercal \cdot \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t}}_{\frac{\mathrm{d}E}{\mathrm{d}t}} + \underbrace{\mathbf{z}(\mathbf{w}) \cdot \mathbf{w}}_{P_\mathrm{D}} + \underbrace{\mathbf{u}^\intercal \cdot \mathbf{y}}_{P_\mathrm{S}} \end{aligned}$$



$$\mathcal{D} : \frac{\mathrm{d}E}{\mathrm{d}t} + P_\mathrm{D} + P_\mathrm{S} = 0$$

Port-Hamiltonian structure

$$
\begin{array}{r}
\text{Storage} \\
\text{Dissipation} \\
\text{Ports}
\end{array}
\underbrace{\begin{pmatrix} \frac{\mathrm{d}x}{\mathrm{d}t} \\ \mathbf{w} \\ \mathbf{y} \end{pmatrix}}_{\mathfrak{b}}
=
\underbrace{\begin{pmatrix} +\mathbf{J}_{\mathbf{xx}} & +\mathbf{J}_{\mathbf{xw}} & +\mathbf{J}_{\mathbf{xy}} \\ -{\mathbf{J}_{\mathbf{xw}}}^{\mathsf{T}} & +\mathbf{J}_{\mathbf{ww}} & +\mathbf{J}_{\mathbf{wy}} \\ -{\mathbf{J}_{\mathbf{xy}}}^{\mathsf{T}} & -{\mathbf{J}_{\mathbf{wy}}}^{\mathsf{T}} & +\mathbf{J}_{\mathbf{yy}} \end{pmatrix}}_{\mathbf{J}}
\cdot
\underbrace{\begin{pmatrix} \nabla \mathrm{H}(\mathbf{x}) \\ \mathbf{z}(\mathbf{w}) \\ \mathbf{u} \end{pmatrix}}_{\mathfrak{a}}
$$

Reduction of the linear dissipative structure [9]

Splitting of $\mathbf{z}$

$\mathbf{Z}_{\mathrm{l}}$ a diagonal matrix and $\mathbf{z}_{\mathrm{nl}}$ a collection of nonlinear functions

$$\mathbf{w} = \begin{pmatrix} \mathbf{w}_{\mathrm{l}} \\ \mathbf{w}_{\mathrm{nl}} \end{pmatrix}, \quad \mathbf{z}(\mathbf{w}) = \begin{pmatrix} \mathbf{Z}_{\mathrm{l}} \cdot \mathbf{w}_{\mathrm{l}} \\ \mathbf{z}_{\mathrm{nl}}(\mathbf{w}_{\mathrm{nl}}) \end{pmatrix},$$

New Port-Hamiltonian structure

$$\underbrace{\begin{pmatrix} \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} \\ \hline \mathbf{w}_{\mathrm{nl}} \\ \hline \mathbf{y} \end{pmatrix}}_{\hat{\mathfrak{b}}} = \underbrace{\left(\hat{\mathbf{J}} - \mathbf{R}\right)}_{\mathbf{M}} \cdot \underbrace{\begin{pmatrix} \nabla \mathrm{H}(\mathbf{x}) \\ \hline \mathbf{z}_{\mathrm{nl}}(\mathbf{w}_{\mathrm{nl}}) \\ \hline \mathbf{u} \end{pmatrix}}_{\hat{\mathfrak{a}}}$$

Interpretation

- $\hat{\mathbf{J}} \to$ reduced conservative interconnection,
- $\mathbf{R} \succeq 0 \to$ resistive interconnection (includes the coefficients from $\mathbf{Z}_{\mathrm{l}}$).

9. Falaize et Hélie, "Passive guaranteed simulation of analog audio circuits : A port-Hamiltonian approach", 2016.

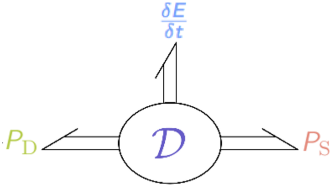
PyPHS Port-Hamiltonian structure

$$\underbrace{\begin{pmatrix} \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} \\ \mathbf{w} \\ \mathbf{y} \end{pmatrix}}_{\mathfrak{b}} = \underbrace{\begin{pmatrix} \mathbf{M_{xx}} & \mathbf{M_{xw}} & \mathbf{M_{xy}} \\ \mathbf{M_{wx}} & \mathbf{M_{ww}} & \mathbf{M_{wy}} \\ \mathbf{M_{yx}} & \mathbf{M_{yw}} & \mathbf{M_{yy}} \end{pmatrix}}_{\mathbf{M}} \cdot \underbrace{\begin{pmatrix} \nabla \mathrm{H}(\mathbf{x}) \\ \mathbf{z}(\mathbf{w}) \\ \mathbf{u} \end{pmatrix}}_{\mathfrak{a}}$$

with

$$\mathbf{M} = \underbrace{\begin{pmatrix} +\mathbf{J_{xx}} & +\mathbf{J_{xw}} & +\mathbf{J_{xy}} \\ -\mathbf{J_{xw}}^\intercal & +\mathbf{J_{ww}} & +\mathbf{J_{wy}} \\ -\mathbf{J_{xy}}^\intercal & -\mathbf{J_{wy}}^\intercal & +\mathbf{J_{yy}} \end{pmatrix}}_{\mathbf{J}} - \underbrace{\begin{pmatrix} \mathbf{R_{xx}} & \mathbf{R_{xw}} & \mathbf{R_{xy}} \\ \mathbf{R_{xw}}^\intercal & \mathbf{R_{ww}} & \mathbf{R_{wy}} \\ \mathbf{R_{xy}}^\intercal & \mathbf{R_{wy}}^\intercal & \mathbf{R_{yy}} \end{pmatrix}}_{\mathbf{R}}$$

4. Numerical method



$$\mathcal{D} : \frac{\delta E}{\delta t} + P_{\mathrm{D}} + P_{\mathrm{S}} = 0$$

Structure preserving numerical method 1

Objective

Discrete time power balance : $\frac{\delta \mathrm{E}}{\delta T}[k] + P_{\mathrm{D}}[k] + P_{\mathrm{S}}[k] = 0.$

Choice

- $\frac{\delta E[k]}{\delta T} = \frac{E[k+1]-E[k]}{\delta T} = \frac{\mathrm{H}(\mathbf{x}[k+1])-\mathrm{H}(\mathbf{x}[k])}{\delta T}$
- Mono-variate case :

$$\frac{\mathrm{E}[k+1]-\mathrm{E}[k]}{\delta T} = \sum_n \frac{\mathrm{H}_n(x_n[k+1]) - \mathrm{H}_n(x_n[k])}{x_n[k+1]-x_n[k]} \cdot \frac{x_n[k+1]-x_n[k]}{\delta T}$$

Solution :

$$\begin{array}{ccccc} \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} & \longrightarrow & \frac{\delta \mathbf{x}[k]}{\delta T} = \frac{\mathbf{x}[k+1]-\mathbf{x}[k]}{\delta T} & & \\ \nabla \mathrm{H}(\mathbf{x}) & \longrightarrow & \nabla^d \mathrm{H}\big(\mathbf{x}[k], \delta\mathbf{x}[k]\big) & \triangleq & \text{discrete gradient [10]} \end{array}$$

with

$$\Big[\nabla^d \mathrm{H}(\mathbf{x}, \delta\mathbf{x})\Big]_n = \frac{\mathrm{H}_n\big([\mathbf{x}+\delta\mathbf{x}]_n\big) - \mathrm{H}_n\big([\mathbf{x}]_n\big)}{[\delta\mathbf{x}]_n} \underset{[\delta\mathbf{x}]_n \to 0}{\longrightarrow} \frac{\mathrm{d}\mathrm{H}_n}{\mathrm{d}x_n}(x_n).$$

10. Itoh et Abe, “Hamiltonian-conserving discrete canonical equations based on variational difference quotients”, 1988.

Structure preserving numerical method 2

Solution

$$
\begin{array}{ccc}
\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} & \longrightarrow & \frac{\delta \mathbf{x}[k]}{\delta T} = \frac{\mathbf{x}[k+1]-\mathbf{x}[k]}{\delta T} \\
\nabla \mathrm{H}(\mathbf{x}) & \longrightarrow & \nabla^d \mathrm{H}\big(\mathbf{x}[k], \delta \mathbf{x}[k]\big)
\end{array}
$$

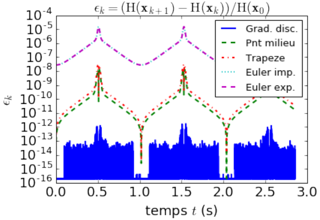
Discret PHS

$$
\begin{pmatrix} \frac{\delta \mathbf{x}[k]}{\delta T} \\ \hline \mathbf{w}[k] \\ \hline \mathbf{y}[k] \end{pmatrix} = \mathbf{M} \cdot \begin{pmatrix} \nabla^d \mathrm{H}\big(\mathbf{x}[k], \delta \mathbf{x}[\mathbf{k}]\big) \\ \hline \mathbf{z}(\mathbf{w}[k]) \\ \hline \mathbf{u}[k] \end{pmatrix}.
$$

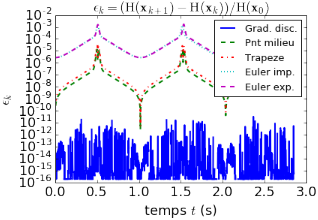
PHS structure is preserved in discrete time $\Rightarrow$ numerical passivity.

Relative error on the power balance (PyPHS in blue)

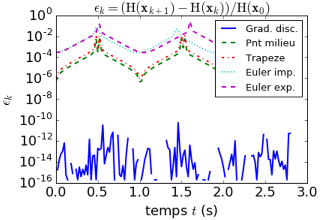
$f_e = 5000\textbf{Hz}$



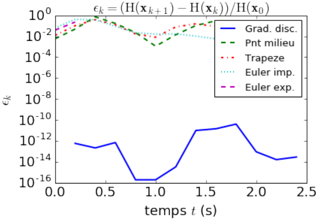
$f_e = 500\textbf{Hz}$



$f_e = 50\textbf{Hz}$

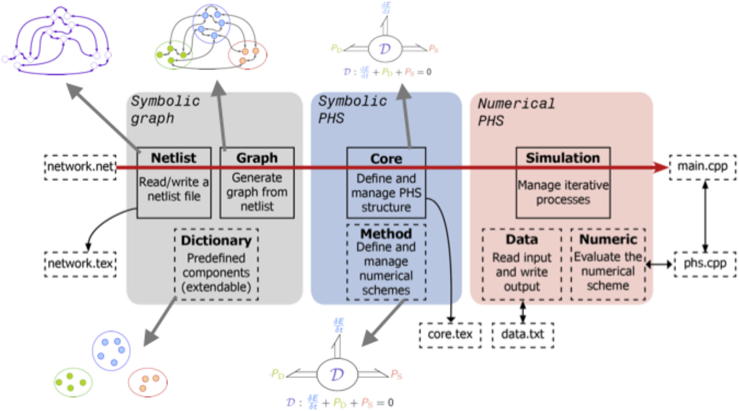


$f_e = 5\textbf{Hz}$



5. Code generation

PyPHS : an overview



Python simulation

Formal `Method` object to numerical `Simulation` object

1. Parameters are substituted in the discrete PHS.
2. Each symbolic expression is simplified and transformed into a Python function.
3. Updates of internal variables is defined by a message passing system.

Perform simulation

- Inputs are :
    1. A sequence of input values,
    2. A sequence of control parameters values.
- Apply each update sequentially.
- Results are stored on disk to avoid memory overload.

C++ code generation

Formal `Method` object to C++ code

1. Parameters are associated to pointers → can be changed after generation.
2. Each symbolic expression is simplified and transformed into a C++ function.
3. Same message passing system.

Perform simulation

- Inputs are :
    1. the sample rate,
    2. a sequence of input values,
    3. a sequence of control parameters values.
- Apply each update sequentially.
- Results are stored on disk → call back into Python for post processing.

Juce[11] C++ snippets generation for real-time audio plugins

Only for Juce audio FX

1. Call the generated C++ object into Juce Template.
2. Generation of a set of snippets $\rightarrow$ copy/past into Juce template.
3. The control parameters are automatically associate with sliders $\rightarrow$ real-time control.
4. Still experimental.

Yield AU/VST real-time audio plugins

- Can be used in most Digital Audio Workstations.

11. https ://juce.com/

FAUST [12] code generation for real-time audio plugins

Only for FAUST audio FX

- Dedicated `Method` object : Symbolic pre-inversion of every matrices.
- Fixed number of nonlinear solvers iteration $\rightarrow$ duplicate of a single iteration.
- A complete iteration is built and encompassed in a dedicated feedback system.
- Control parameters are associated with sliders.
- Still experimental.

Yield VST real-time audio plugins

- Automated optimization of the signal flow.
- Can be used in most Digital Audio Workstations.
- Several compilation targets.

12. `http://faust.grame.fr/`

Last word

PyPHS today (v0.2)

- Open source Library on a GITHUB repository[13].
- Licence CECILL (CEA-CNRS-INRIA Logiciels libres).
- PYTHON 2.7 & 3.5 supported, Mac OSX, Windows 10 and Linux.
- Multiphysical components dictionary.
- Automated graph analysis.
- Automated derivation of the PHS structure and LaTeX code generation.
- Passive guaranteed simulations.
- Automated generation of C++, JUCE and FAUST code.

13. https://pyphs.github.io/pyphs/

PyPHS tomorrow

Scientific results to be implemented

- Anti-aliasing observer (PhD Remy Müller).
- PHS in scattering variables ($\rightsquigarrow$ Wave Digital PHS).
- Piecewise Linear constitutive laws ($\rightsquigarrow$ cope with realizability faults).
- Improve Nonlinear solver ($\neq$ Newton-Raphson).
- Automated derivation of command laws (feedforward + feedback).
- ...

PyPHS tomorrow

Accelerate development

CALL FOR DEVELOPERS

Improve robustness

CALL FOR USERS

Thank you for your attention

Contact : antoine.falaize@gmail.com