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

Antoine Falaize
IRCAM seminar - Research and Technology
04/12/2017

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 approaches

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

\[ \frac{dE}{dt} + P_D + P_S = 0 \]

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 inherit from this property.
Encoding of passivity in PyPHS

Port-Hamiltonian System (PHS)

\[ \mathcal{D} : \frac{\delta E}{\delta t} + P_D + P_S = 0 \]

Conservative interconnection

Network

Components

Multi-physical system

Storage \( \frac{dE}{dt} \)

Dissipation

Structure preserving numerical method

Discrete PHS
PyPHS : everything is formal

Network 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
- Graph theory

2012→2016

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

2016→...

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


5. see https://hamecmopsys.ens2m.fr/

1. Network
PyPHS inputs: Graph and Netlist.

1. 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
System representation paradigm: Power graphs

Directed graphs with self loops

- Set of nodes $N = \{N_1, \cdots, N_n\}$.
- Set of edges $B = \{B_1, \cdots, B_n\}$ with $B_i = (n, m) \in N^2$.
- Direction: $B_i \equiv n \to m$

Receiver convention

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

Connection $\equiv$ Nodes identification

Connection graph with self loops.
Physical quantities

Flow = Current (A), Effort = Voltage (V), $\epsilon = $ 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

Graph nodes = Circuit nodes
Ground = Reference node #

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

Flow = Force (N), Effort = Velocity (m/s), $\epsilon = \text{point velocity (m/s)}$

Example system

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

Graph nodes = unique velocities
Reference velocity = node #

Graph edges = components
Mechanical graphs (dual)

Physical quantities

Flow = Velocity (m/s), Effort = force (N), $\epsilon = $ 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), $\epsilon$ = 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), $\epsilon = \text{temperature (K)}$

Example system

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

Nodes

Graph nodes = temperatures
Reference temperature = node #

Graph edges = components
Note R = 2 edges (irreversibility)
Multiphysical graphs: connectors

Transformer

\[ e_{3\rightarrow4} = \frac{1}{\alpha} e_{1\rightarrow2}, \]
\[ f_{3\rightarrow4} = -\alpha f_{1\rightarrow2}, \]
\[ [\alpha] = \frac{[f_{3\rightarrow4}]}{[f_{1\rightarrow2}]} . \]

Gyrator

\[ e_{3\rightarrow4} = \alpha f_{1\rightarrow2}, \]
\[ f_{3\rightarrow4} = -\frac{1}{\alpha} e_{1\rightarrow2}, \]
\[ [\alpha] = \frac{[e_{3\rightarrow4}]}{[f_{1\rightarrow2}]} . \]

Conserving connection

In each case: \( P_{3\rightarrow4} = -P_{1\rightarrow2} \)
Kirchhoff laws on graphs

**Incidence Matrix**

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

\[
\Gamma = \begin{pmatrix} 
B_R & B_L & B_C & B_I \\
0 & 0 & +1 & -1 \\
-1 & 0 & 0 & 0 \\
+1 & -1 & 0 & 0 \\
0 & +1 & -1 & +1
\end{pmatrix}
\]

**Reduced incidence Matrix**

Arbitrary reference node \# 

\[
\Gamma = \begin{pmatrix} 
\gamma_0 \\
\gamma \\
\vdots \\
\gamma_{n_N}
\end{pmatrix}
\]

\[
\begin{pmatrix} 
B_1 & \cdots & B_{n_B} \\
\gamma_0 \\
\vdots \\
\gamma_{n_N}
\end{pmatrix}
\]

**Generalized Kirchhoff’s laws**

- Efforts \( \mathbf{e} \in \mathbb{R}^{n_B} \), flows \( \mathbf{f} \in \mathbb{R}^{n_B} \).
- Node quantities \( \mathbf{p} \in \mathbb{R}^{n_N} \).
- \( \gamma^T \mathbf{p} = \mathbf{e} \), (KVL).
- \( \gamma \mathbf{f} = 0 \), (KCL).
Dirac structure $\mathcal{D} = \text{Kirchhoff laws on graphs}$

**Edges splitting**

Depends on the components

**Flow controlled** $\mathbf{f} \rightarrow \text{edge} \rightarrow \mathbf{e}$.

**Effort controlled** $\mathbf{e} \rightarrow \text{edge} \rightarrow \mathbf{f}$.

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

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

**RLC example**

$B_L$ is $\mathbf{e}$-controlled, $B_R, B_C, B_I$ are $\mathbf{f}$-controlled.

\[
\begin{pmatrix}
\gamma_0 \\
\gamma_e \\
\gamma_f
\end{pmatrix}
= \begin{pmatrix}
0 & 0 & +1 & -1 \\
0 & -1 & 0 & 0 \\
-1 & +1 & 0 & 0 \\
+1 & 0 & -1 & +1
\end{pmatrix}
\]

\[
\begin{pmatrix}
\mathbf{e}_b \\
\mathbf{f}_b
\end{pmatrix}
= \begin{pmatrix}
0 & \gamma_e^T \\
-\gamma_e & 0
\end{pmatrix}
\begin{pmatrix}
\mathbf{f}_a \\
\mathbf{e}_b
\end{pmatrix}
\]

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

This is the Tellegen's theorem:

$\sum_{n}^{n_B} \mathbf{e}_n \mathbf{f}_n = \sum_{n}^{n_B} P_n = 0$.

**Realizability criterion**

If $\gamma_f$ is invertible, then $\exists! J$ s.t.

$\mathbf{b} = J \cdot \mathbf{a}$.

**Dirac structure**

1. $\mathbf{e}_b = \gamma_e^T \cdot \mathbf{p}$ and $\mathbf{e}_a = \gamma_f^T \cdot \mathbf{p}$,
2. $\gamma_e \mathbf{f}_a = -\gamma_f \cdot \mathbf{f}_b$,
3. $\gamma_e \mathbf{f} = \gamma_f^{-1} \cdot \gamma_e$,
Automated construction of the Dirac structure

**Algorithm**\(^8\)

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

**Résult**

- If realizable:
  1. partition \( B = [B_e, B_f] \),
  2. structure \( b = J \cdot a \).

- Else: Realizability fault detection → the user correct the netlist.

---

Components
Storage components (definitions)

Constitutive relation for component $s$

Storage function (Hamiltonian) $H_s$ of the state $x_s$.

- **Stored energy** $E_s(t) = H_s(x_s(t)) \geq 0$.
- **Received power** \( \frac{dE_s}{dt} = H'_s(x_s) \frac{dx_s}{dt} \).

Power variables for component $s$

- **Received power** \( \frac{dE_s}{dt} = \varepsilon_s f_s \).
  - $\varepsilon$-controlled $\varepsilon_s = \frac{dx_s}{dt} \Rightarrow f_s = H'_s(x_s)$.
  - $f$-controlled $f_s = \frac{dx_s}{dt} \Rightarrow \varepsilon_s = H'_s(x_s)$.

Total energy stored in $n_E$ storage edges

- $x = (x_1, \cdots, x_{n_E})$.
- $E = H(x) = \sum_{s=1}^{n_E} H_s(x_s) \geq 0$.
- \( \frac{dE}{dt} = \nabla H^T \frac{dx}{dt} = \sum_{s=1}^{n_E} \frac{dH_s}{dx_s} \frac{dx_s}{dt} \).
Storage components (examples)

Mass (flow=velocity, effort=force)

- **State** momentum $x_m = m v_m$.
- **Hamiltonian** kinetic energy $H_m(x_m) = \frac{x_m^2}{2m}$.
- **Flow** mass velocity $f_m = H'_m(x_m) = \frac{x_m}{m}$.
- **Effort** inertial force $e_m = \frac{dx_m}{dt} = m \frac{dv_m}{dt}$.

Capacitor

- **State** charge $q_C$.
- **Hamiltonian** electrostatic energy $H_C(x_C) = \frac{x_C^2}{2C}$.
- **Flow** current $f_C = \frac{dx_C}{dt} = \frac{dq_C}{dt}$.
- **Effort** voltage $e_C = H'_C(x_C) = \frac{x_C}{C}$. 
Dissipative components (definitions)

Constitutive relation for component $d$

Dissipation function $z_d$ of the variable $w_d$.

Received (dissipated) power $P_{Dd}(t) = z_d(w_d(t)) \geq 0$.

Power variables for component $d$

Received power $P_{Dd}(t) = \varepsilon_d \dot{f}_d \geq 0$

- $\varepsilon$-controlled $\varepsilon_d = w_d \implies \dot{f}_d = z_d(w_d)$.
- $f$-controlled $\dot{f}_d = w_d \implies \varepsilon_d = z_d(w_d)$.

Total power dissipated in $n_D$ dissipative edges

- $w = (w_1, \cdots, w_{n_D})$.
- $z(w) = (z_1(w_1), \cdots, z_{n_D}(w_{n_D}))$.
- $P_D = z(w)^T \cdot w = \sum_{d=1}^{n_D} z_d(w_d) w_d \geq 0$. 

$D : \frac{dE}{dt} + P_D + P_S = 0$
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 \( P_D = f_D e_D = R v_D^2 \)

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 \( P_D = f_R e_R = R i_R^2 \)
Ports (definitions)

**Input and output on port** $p$

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

**Received Power** $P_{Sp}(t) = u_p(t)y_p(t)$.

**The power** $P_{Sp}$ is the power that goes out of the system on port $p$.

Ports are power sink.

**Power variables for port** $p$

**Received power** $P_{Sp}(t) = \epsilon_p \mathbf{f}_p$

- $\epsilon$-controlled $\epsilon_p = y_p \Rightarrow \mathbf{f}_p = u_p$ (flow source).
- $\mathbf{f}$-controlled $\mathbf{f}_p = y_p \Rightarrow \epsilon_p = u_p$ (effort source).

**Total power on** $n_S$ **port edges**

- $\mathbf{u} = (u_1, \cdots, u_{n_S})$.
- $\mathbf{y} = (y_1, \cdots, y_{n_S})$.
- $P_S = \mathbf{u}^T \cdot \mathbf{y} = \sum_{p=1}^{n_S} u_p y_p$. 

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 $P_S = f_U e_U = v_U i_U$

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 $P_S = f_U e_U = f_U v_U$. 
- **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

\[ \mathcal{D} : \frac{dE}{dt} + P_D + P_S = 0 \]
Putting all together

Components

Storage \( b_x = \frac{dx}{dt}, \quad a_x = \nabla H(x) \)

Dissipation \( b_w = w, \quad a_w = z(w) \)

Ports \( b_y = y, \quad b_y = u \)

This encodes the power balance

\[
0 = a^T \cdot b = \nabla H(x)^T \cdot \frac{dx}{dt} + z(w) \cdot w + u^T \cdot y
\]

Network (Dirac structure)

\[
b = \begin{pmatrix} b_x \\ b_w \\ b_y \end{pmatrix} \quad \text{and} \quad a = \begin{pmatrix} a_x \\ a_w \\ a_y \end{pmatrix} \]

with \( b = J \cdot a \) and \( J^T = -J \).

\[
\mathcal{D} : \frac{dE}{dt} + P_D + P_S = 0
\]
Port-Hamiltonian structure

\[
\begin{align*}
\frac{d}{dt}\begin{bmatrix} x \\ w \\ y \\ b \end{bmatrix} &= \begin{bmatrix}
+J_{xx} & +J_{xw} & +J_{xy} \\
-J_{xw}^T & +J_{ww} & +J_{wy} \\
-J_{xy}^T & -J_{wy}^T & +J_{yy}
\end{bmatrix}
\begin{bmatrix} \nabla H(x) \\ z(w) \\ u \end{bmatrix}
\end{align*}
\]
Splitting of \( z \)

\( Z_1 \) a diagonal matrix and \( z_{nl} \) a collection of nonlinear functions

\[
\begin{align*}
  w &= \begin{pmatrix} w_1 \\ w_{nl} \end{pmatrix}, \\
  z(w) &= \begin{pmatrix} Z_1 \cdot w_1 \\ z_{nl}(w_{nl}) \end{pmatrix},
\end{align*}
\]

New Port-Hamiltonian structure

\[
\begin{pmatrix}
  \frac{dx}{dt} \\
  \frac{w_{nl}}{y} \\
  \hat{b}
\end{pmatrix} = \begin{pmatrix} \hat{J} - R \\ M \end{pmatrix} \cdot \begin{pmatrix}
  \nabla H(x) \\
  z_{nl}(w_{nl}) \\
  \hat{a}
\end{pmatrix},
\]

Interpretation

- \( \hat{J} \rightarrow \) reduced conservative interconnection,
- \( R \geq 0 \rightarrow \) resistive interconnection (includes the coefficients from \( Z_1 \)).

PyPHS Port-Hamiltonian structure

\[
\begin{pmatrix}
\frac{dx}{dt} \\
\dot{w}
\end{pmatrix}
= \begin{pmatrix}
M_{xx} & M_{xw} & M_{xy} \\
M_{wx} & M_{ww} & M_{wy} \\
M_{yx} & M_{yw} & M_{yy}
\end{pmatrix}
\cdot
\begin{pmatrix}
\nabla H(x) \\
z(w) \\
u
\end{pmatrix}
\]

with

\[
M = \begin{pmatrix}
+J_{xx} & +J_{xw} & +J_{xy} \\
-J_{xw}^T & +J_{ww} & +J_{wy} \\
-J_{xy}^T & -J_{wy}^T & +J_{yy}
\end{pmatrix}
- \begin{pmatrix}
R_{xx} & R_{xw} & R_{xy} \\
R_{xw}^T & R_{ww} & R_{wy} \\
R_{xy}^T & R_{wy}^T & R_{yy}
\end{pmatrix}
\]

\[
J = \begin{pmatrix}
R_{xx} & R_{xw} & R_{xy} \\
R_{xw}^T & R_{ww} & R_{wy} \\
R_{xy}^T & R_{wy}^T & R_{yy}
\end{pmatrix}
\]

\[
R = \begin{pmatrix}
+J_{xx} & +J_{xw} & +J_{xy} \\
-J_{xw}^T & +J_{ww} & +J_{wy} \\
-J_{xy}^T & -J_{wy}^T & +J_{yy}
\end{pmatrix}
\]
4. Numerical method

\[ \mathcal{D} : \frac{\delta E}{\delta t} + P_D + P_S = 0 \]
**Objective**

Discrete time power balance: \( \frac{\delta E}{\delta T}[k] + P_D[k] + P_S[k] = 0 \).

**Choice**

- \( \frac{\delta E[k]}{\delta T} = \frac{E[k+1] - E[k]}{\delta T} = \frac{H(x[k+1]) - H(x[k])}{\delta T} \)
- **Mono-variate case:**
  \[
  \frac{E[k + 1] - E[k]}{\delta T} = \sum_n \frac{H_n(x_n[k + 1]) - H_n(x_n[k])}{x_n[k + 1] - x_n[k]} \cdot \frac{x_n[k + 1] - x_n[k]}{\delta T}
  \]

**Solution:**

\[
\frac{dx}{dt} \rightarrow \frac{\delta x[k]}{\delta T} = \frac{x[k+1] - x[k]}{\delta T}
\]

\[
\nabla H(x) \rightarrow \nabla^d H(x[k], \delta x[k]) \triangleq \text{discrete gradient}^{10}
\]

with

\[
\left[ \nabla^d H(x, \delta x) \right]_n = \frac{H_n([x + \delta x]_n) - H_n([x]_n)}{[\delta x]_n} \xrightarrow{[\delta x]_n \to 0} \frac{dH_n(x_n)}{dx_n}(x_n).
\]

Solution
\[
\frac{dx}{dt} \rightarrow \frac{\delta x[k]}{\delta T} = \frac{x[k+1]-x[k]}{\delta T}
\]
\[
\nabla H(x) \rightarrow \nabla^d H(x[k], \delta x[k])
\]

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

PHS structure is preserved in discrete time \(\Rightarrow\) numerical passivity.
Relative error on the power balance (PyPHS in blue)

\[ f_e = 5000 \text{Hz} \]

\[ f_e = 500 \text{Hz} \]

\[ f_e = 50 \text{Hz} \]

\[ f_e = 5 \text{Hz} \]
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 Python functions.
3. Updates of internal variable 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.
Only for Juce audio FX

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

Yield AU/VST real-time audio plugins

- Can be used in most Digital Audio Workstations.

Only for FAUST audio FX

- Dedicated Method object: Symbolic pre-inversion of every matrices.
- Fixed number of nonlinear solvers iteration → 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.

Last word
PyPHS today (v0.2)

- Open source Library on a GitHub repository\textsuperscript{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.

\textsuperscript{13} https://pyphs.github.io/pyphs/
Scientific results to be implemented

- Anti-aliasing observer (PhD Remy Müller).
- PHS in scattering variables (⇝ Wave Digital PHS).
- Piecewise Linear constitutive laws (⇝ cope with realizability faults).
- Improve Nonlinear solver (≠ Newton-Raphson).
- Automated derivation of a command laws (feedforward + feedback).
- ...
Accelerate development
CALL FOR DEVELOPERS

Improve robustness
CALL FOR USERS
Thank you for your attention

Contact: antoine.falaize@gmail.com