

PYPHS DOCUMENTATION

Version 0.1.9b2

Antoine Falaize

18 novembre 2016

1 Introduction

The python package `pyphs` is dedicated to the treatment of *passive multiphysical systems* in the *Port-Hamiltonian Systems* (PHS) formalism. This formalism structures physical systems into

- energy conserving parts,
- power dissipating parts and
- source parts.

This guarantees a *power balance* is fulfilled, including for numerical *simulations* based on an adapted *numerical method*.

1. Systems are described by *directed multi-graphs* (`networkx.MultiDiGraph`).
2. The time-continuous port-Hamiltonian structure is build from an *automated graph analysis*.
3. The discrete-time port-Hamiltonian structure is derived from a *structure preserving numerical method*.
4. \LaTeX description code and C++ simulation code are automatically generated.

1.1 Installation

Notice only python 2.7 is supported.

It is recommended to install `pyphs` using PYPI (the PYTHON PACKAGE INDEX). In terminal :

```
pip install pyphs
```

Mac OSX only : An installation for *Anaconda* users is also available. In terminal :

```
conda install -c afalaize pyphs
```

1.2 The PHS formalism

Below is a recall of the Port-Hamiltonian Systems (PHS) formalism. For details, the reader is referred to the *e.g.* the academic reference [Falaize and H  lie, 2016].

We consider systems that can be described by the following time-continuous non-linear state-space representation :

$$\underbrace{\begin{pmatrix} \frac{dx}{dt} \\ \mathbf{w} \\ \mathbf{y} \end{pmatrix}}_{\mathbf{b}} = \underbrace{\begin{pmatrix} \mathbf{M}_{xx} & \mathbf{M}_{xw} & \mathbf{M}_{xy} \\ \mathbf{M}_{wx} & \mathbf{M}_{ww} & \mathbf{M}_{wy} \\ \mathbf{M}_{yx} & \mathbf{M}_{yw} & \mathbf{M}_{yy} \end{pmatrix}}_{\mathbf{M}} \cdot \underbrace{\begin{pmatrix} \nabla H(\mathbf{x}) \\ \mathbf{z}(\mathbf{w}) \\ \mathbf{u} \end{pmatrix}}_{\mathbf{a}} \quad (1)$$

where

$$\mathbf{M} = \underbrace{\begin{pmatrix} \mathbf{J}_{xx} & \mathbf{J}_{xw} & \mathbf{J}_{xy} \\ \mathbf{J}_{wx} & \mathbf{J}_{ww} & \mathbf{J}_{wy} \\ \mathbf{J}_{yx} & \mathbf{J}_{yw} & \mathbf{J}_{yy} \end{pmatrix}}_{\mathbf{J}} - \underbrace{\begin{pmatrix} \mathbf{R}_{xx} & \mathbf{R}_{xw} & \mathbf{R}_{xy} \\ \mathbf{R}_{wx} & \mathbf{R}_{ww} & \mathbf{R}_{wy} \\ \mathbf{R}_{yx} & \mathbf{R}_{yw} & \mathbf{R}_{yy} \end{pmatrix}}_{\mathbf{R}} \quad (2)$$

and

- $\mathbf{J} : \mathbf{x} \mapsto \mathbf{J}(\mathbf{x})$ is a skew-symmetric matrix :

$$\mathbf{J}_{\alpha\beta} = -\mathbf{J}_{\beta\alpha}^T \text{ for } (\alpha, \beta) \in \{\mathbf{x}, \mathbf{w}, \mathbf{y}\}^2,$$

- $\mathbf{R} : \mathbf{x} \mapsto \mathbf{R}(\mathbf{x}) \succeq 0$ is a positive definite matrix,
- $\mathbf{x} : t \mapsto \mathbf{x}(t) \in \mathbb{R}^{n_x}$ is the *state vector*,
- $H : \mathbf{x} \mapsto H(\mathbf{x}) \in \mathbb{R}_+$ is a *storage function* (convex and positive-definite scalar function with $H(0) = 0$),
- $\nabla H : \mathbf{x} \mapsto \nabla H(\mathbf{x}) \in \mathbb{R}^{n_x}$ denote the gradient of the storage function with the *storage power*

$$\mathbf{P}_x = \frac{d\mathbf{x}}{dt} \cdot \nabla H(\mathbf{x}),$$

- $\mathbf{w} : t \mapsto \mathbf{w}(t) \in \mathbb{R}^{n_w}$ is the *dissipation vector variable*,
- $\mathbf{z} : \mathbf{w} \mapsto \mathbf{z}(\mathbf{w}) \in \mathbb{R}^{n_w}$ is a *dissipation function* (with positive definite jacobian matrix and $\mathbf{z}(0) = 0$) for the *dissipated power*

$$\mathbf{P}_w = \mathbf{w} \cdot \mathbf{z}(\mathbf{w}) + \mathbf{a} \cdot \mathbf{R} \cdot \mathbf{a},$$

- $\mathbf{u} : t \mapsto \mathbf{u}(t) \in \mathbb{R}^{n_y}$ is the *input vector*,
- $\mathbf{y} : t \mapsto \mathbf{y}(t) \in \mathbb{R}^{n_y}$ is the *output vector*,
- **the power received *by* the sources *from* the system is**

$$\mathbf{P} = \mathbf{u} \cdot \mathbf{y}.$$

The state is split according to $\mathbf{x} = (\mathbf{x}_1^T, \mathbf{x}_{n1}^T)^T$ with

$\mathbf{x}_1 = (x_1, \dots, x_{n_{x1}})^T$ the states associated with the quadratic components of the storage function $H_1(\mathbf{x}_1) = \frac{\mathbf{x}_1 \cdot \mathbf{Q} \cdot \mathbf{x}_1}{2}$

$\mathbf{x}_{n1} = (x_{n_{x1}+1}, \dots, x_{n_x})^T$ the states associated with the non-quadratic components of the storage function with $n_x = n_{x1} + n_{xn1}$ and

$$H(\mathbf{x}) = H_1(\mathbf{x}_1) + H_{n1}(\mathbf{x}_{n1})$$

.

The set of dissipative variables is split according to $\mathbf{w} = (\mathbf{w}_1^T, \mathbf{w}_{n1}^T)^T$ with

$\mathbf{w}_1 = (w_1, \dots, w_{n_w})^\top$ the variables associated with the linear components of the dissipative relation $\mathbf{z}_1(\mathbf{w}_1) = \mathbf{Z}_1 \mathbf{w}_1$

$\mathbf{w}_{n1} = (w_{n_w+1}, \dots, w_{n_w})^\top$ the variables associated with the nonlinear components of the dissipative relation $\mathbf{z}_{n1} : \mathbf{w}_{n1} \mapsto \mathbf{z}_{n1}(\mathbf{w}_{n1}) \in \mathbb{R}^{n_{n1}}$ with $n_w = n_{w1} + n_{wn1}$ and

$$\mathbf{z}(\mathbf{w}) = \begin{pmatrix} \mathbf{Z}_1 \mathbf{w}_1 \\ \mathbf{z}_{n1}(\mathbf{w}_{n1}) \end{pmatrix}.$$

Accordingly, the structure matrices are split as

$$\underbrace{\begin{pmatrix} \frac{dx_1}{dt} \\ \frac{dx_{n1}}{dt} \\ \mathbf{w}_1 \\ \mathbf{w}_{n1} \\ \mathbf{y} \end{pmatrix}}_{\mathbf{b}} = \underbrace{\begin{pmatrix} \mathbf{M}_{x1x1} & \mathbf{M}_{x1xn1} & \mathbf{M}_{x1w1} & \mathbf{M}_{x1wn1} & \mathbf{M}_{x1y} \\ \mathbf{M}_{xn1x1} & \mathbf{M}_{xn1xn1} & \mathbf{M}_{xn1w1} & \mathbf{M}_{xn1wn1} & \mathbf{M}_{xn1y} \\ \mathbf{M}_{w1x1} & \mathbf{M}_{w1xn1} & \mathbf{M}_{w1w1} & \mathbf{M}_{w1wn1} & \mathbf{M}_{w1y} \\ \mathbf{M}_{wn1x1} & \mathbf{M}_{wn1xn1} & \mathbf{M}_{wn1w1} & \mathbf{M}_{wn1wn1} & \mathbf{M}_{wn1y} \\ \mathbf{M}_{yx1} & \mathbf{M}_{yxn1} & \mathbf{M}_{yw1} & \mathbf{M}_{ywn1} & \mathbf{M}_{yy} \end{pmatrix}}_{\mathbf{M}} \cdot \underbrace{\begin{pmatrix} \mathbf{Q} \cdot \mathbf{x} \\ \nabla H_{n1}(\mathbf{x}_{n1}) \\ \mathbf{Z}_1 \cdot \mathbf{w}_1 \\ \mathbf{z}_{n1}(\mathbf{w}_{n1}) \\ \mathbf{u} \end{pmatrix}}_{\mathbf{a}} \quad (3)$$

Table des matières

1	Introduction	1
1.1	Installation	1
1.2	The PHS formalism	1
2	Structure of the <code>pyphs.PortHamiltonianObject</code>	3
2.1	The <code>syms</code> module	4
2.2	The <code>exprs</code> module	4
2.3	The <code>dims</code> module	5
3	Algorithms	5
3.1	Graph analysis	6
3.2	Simulation methods	6
3.2.1	Split of the linear part from the nonlinear part	6
3.2.2	Presolve numerical nonlinear subsystem	6
3.2.3	Algorithm	7
3.3	Realizability solver	7

2 Structure of the `pyphs.PortHamiltonianObject`

Below is a list of each module of practical use in the object `pyphs.PortHamiltonianObject`, along with a short description. We consider the following instantiation :

```
# import of (pre-installed) pyphs package:
import pyphs

# instantiate the PortHamiltonianObject:
phs = pyphs.PortHamiltonianObject(label='mylabel')
```

2.1 The syms module

Container for all the SYMPY symbolic variables (`sympy.Symbol`).

Attributes are ordered *list of symbols* associated with the system's vectors components.

`phs.syms.x` : state vector symbols $\mathbf{x} \in \mathbb{R}^{n_x}$,
`phs.syms.w` : dissipative vector variable symbols $\mathbf{w} \in \mathbb{R}^{n_w}$,
`phs.syms.u` : input vector symbols $\mathbf{u} \in \mathbb{R}^{n_u}$,
`phs.syms.y` : output vector symbols $\mathbf{y} \in \mathbb{R}^{n_y}$,
`phs.syms.cu` : input vector symbols for connectors $\mathbf{c}_u \in \mathbb{R}^{n_y}$,
`phs.syms.cy` : output vector symbols for connectors $\mathbf{c}_y \in \mathbb{R}^{n_y}$,
`phs.syms.p` : Time-varying parameters symbols $\mathbf{p} \in \mathbb{R}^{n_p}$.

Methods :

`phs.syms.dx()` : Returns the symbols associated with the state differential $d\mathbf{x}$ formed by appending the prefix d to each symbol in \mathbf{x} .
`phs.syms.args()` : Return the list of symbols associated with the vector of all arguments of the symbolic expressions (`expr` module).

2.2 The exprs module

Container for all the SYMPY symbolic expressions `sympy.Expr` associated with the system's functions.

Attributes : For scalar function (e.g. the storage function H), arguments of `phs.exprs` are SYMPY expressions (`sympy.Expr`); for vector functions (e.g. the dissipative function \mathbf{z}), arguments are ordered lists of SYMPY expressions; for matrix functions (e.g. the Jacobian matrix of dissipative function \mathbf{z}), arguments are `sympy.Matrix` objects. Notice the expressions arguments¹ must belong either to (i) the elements of `phs.syms.args()`, or (ii) the keys of the dictionary `phs.syms.subs`.

`phs.exprs.H` : storage function $H \in \mathbb{R}$,
`phs.exprs.z` : dissipative function $\mathbf{z} \in \mathbb{R}^{n_z}$,
`phs.exprs.g` : input/output gains vector function $\mathbf{g} \in \mathbb{R}^{n_g}$,

The following expression are computed from the `exprs.build()` method (see below) :

`phs.exprs.dxDH` : the continuous gradient vector of storage scalar function $\nabla H(\mathbf{x}) \in \mathbb{R}^{n_x}$,
`phs.exprs.dxDHd` : the discrete gradient vector of storage scalar function $\overline{\nabla} H(\mathbf{x}, \delta\mathbf{x}) \in \mathbb{R}^{n_x}$,
`phs.exprs.hessH` : the continuous hessian matrix of storage scalar function (computed as $\nabla \nabla H(\mathbf{x}) \in \mathbb{R}^{n_x \times n_x}$),
`phs.exprs.jacz` : the continuous jacobian matrix of dissipative vector function $\nabla \mathbf{z}(\mathbf{w}) \in \mathbb{R}^{n_z \times n_w}$.

1. Accessed through the `sympy.Expr.free_symbols` (e.g. `phs.exprs.H.free_symbols` to recover the arguments of the Storage function H).

`phs.exprs.y` : the expression of the continuous output vector function
 $\mathbf{y}(\nabla H, \mathbf{z}, \mathbf{u}) \in \mathbb{R}^{n_y}$,

`phs.exprs.yd` : the expression of the discrete output vector function
 $\bar{\mathbf{y}}(\nabla H, \mathbf{z}, \mathbf{u}) \in \mathbb{R}^{n_y}$,

Methods :

`phs.exprs.build()` : Build the following system functions as SYMPY expressions and append them as attributes to the `phs.exprs` module :
`phs.exprs.dxDH`, `phs.exprs.dxDd`, `phs.exprs.hessH`, `phs.exprs.jacz`, `phs.exprs.y`,
and `phs.exprs.yd`.

`phs.exprs.setexpr(name, expr)` : Add the SYMPY expression `expr` to the `phs.exprs` module, with argument `name`, and add `name` to the set of `phs.exprs._names`.

`phs.exprs.freesymbols()` : Return a python set of all the free symbols (`sympy.Symbol`) that appear at least once in all expressions with names in `phs.exprs._names`.

2.3 The dims module

Container for accessors to the system's dimensions. No attributes should be changed manually. To split the system into its linear and nonlinear part, use `phs.split_linear()` which organize the system vectors as

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_{nl} \end{pmatrix}, \quad \dim(\mathbf{x}_1) = \quad (4)$$

Attributes : `phs.dims.xl` : Number of state vector components associated with a quadratic storage function : $H_1(\mathbf{x}_1) = \mathbf{x}_1^T \cdot \frac{\mathbf{Q}}{2} \cdot \mathbf{x}_1$, and `phs.dims.x()` is equal to `phs.dims.xl + phs.dims.xnl()`.

`phs.dims.wl` : Number of dissipative vector variable components associated with a linear dissipative function : $\mathbf{z}_1(\mathbf{w}_1) = \mathbf{Z}_1 \cdot \mathbf{w}_1$, and `phs.dims.w()` is equal to `phs.dims.wl + phs.dims.wnl()`.

Methods :

`phs.dims.x()` : Return the dimension of state vector `len(phs.syms.x)`.

`phs.dims.xnl()` : Return the number of state vector components associated with a nonlinear storage function

`len(phs.syms.x)`.

`setexpr(name, expr)` : Add the SYMPY expression `expr` to the `exprs` module, with argument `name`, and add `name` to the set of `exprs._names`.

`freesymbols()` : Return a python set of all the free symbols (`sympy.symbols`) that appear at least once in all expressions with names in `exprs._names`.

3 Algorithms

This section details the algorithms actually implemented for

1. the graph analysis and

2. the different simulation methods

3.1 Graph analysis

The graph analysis method that derives the port-Hamiltonian system's differential-algebraic equations from with a given netlist is detailed in the reference [Falaize and H  lie, 2016]. The algorithm implemented in PyPHS is exactly that in [Falaize and H  lie, 2016, algorithm 1].

3.2 Simulation methods

The discrete gradient method is used in conjunction with the port-Hamiltonian structure to produce a passive-guaranteed numerical scheme (see [Falaize and H  lie, 2016] for details). In the sequel, quantities are defined on the current time step $\mathbf{x} \equiv \mathbf{x}(t_k)$, with $k \in \mathbb{N}_+^*$.

3.2.1 Split of the linear part from the nonlinear part

The discrete gradient for the quadratic part of the Hamiltonian is $\nabla H_1 = \frac{1}{2} \mathbf{Q} (2\mathbf{x}_1 + \delta\mathbf{x}_1)$ and the discrete linear subsystem is

$$\begin{aligned} \mathbf{D}_1^{-1} = \mathbf{iD}_1 &= \begin{pmatrix} \mathbf{I}_d & 0 \\ 0 & \mathbf{I}_d \end{pmatrix} - \begin{pmatrix} \mathbf{M}_{x1x1} & \mathbf{M}_{x1w1} \\ \mathbf{M}_{w1x1} & \mathbf{M}_{w1w1} \end{pmatrix} \begin{pmatrix} \frac{1}{2} \mathbf{Q} & 0 \\ 0 & \mathbf{Z}_1 \end{pmatrix}, \\ \underbrace{\begin{pmatrix} \delta\mathbf{x}_1 \\ \mathbf{w}_1 \end{pmatrix}}_{\mathbf{v}_1} &= \underbrace{\mathbf{D}_1 \begin{pmatrix} \mathbf{M}_{x1x1} \\ \mathbf{M}_{w1x1} \end{pmatrix}}_{\underbrace{\mathbf{N}_{1x1}}_{\mathbf{N}_{1x1}}} \mathbf{Q} \mathbf{x}_1 + \underbrace{\mathbf{D}_1 \begin{pmatrix} \mathbf{M}_{x1xn1} & \mathbf{M}_{x1wn1} \\ \mathbf{M}_{w1xn1} & \mathbf{M}_{w1wn1} \end{pmatrix}}_{\underbrace{\mathbf{N}_{1n1}}_{\mathbf{N}_{1n1}}} \underbrace{\begin{pmatrix} \nabla H_{n1} \\ \mathbf{z}_{n1} \end{pmatrix}}_{\mathbf{f}_{n1}} + \underbrace{\mathbf{D}_1 \begin{pmatrix} \mathbf{M}_{x1y} \\ \mathbf{M}_{w1y} \end{pmatrix}}_{\underbrace{\mathbf{N}_{1y}}_{\mathbf{N}_{1y}}} \mathbf{u} \end{aligned} \quad (5)$$

and the nonlinear subsystem is

$$\begin{aligned} \begin{pmatrix} \mathbf{I}_d & 0 \\ 0 & \mathbf{I}_d \end{pmatrix} \underbrace{\begin{pmatrix} \delta\mathbf{x}_{n1} \\ \mathbf{w}_{n1} \end{pmatrix}}_{\mathbf{v}_{n1}} &= \underbrace{\begin{pmatrix} \mathbf{M}_{xn1xn1} & \mathbf{M}_{xn1wn1} \\ \mathbf{M}_{wn1xn1} & \mathbf{M}_{wn1wn1} \end{pmatrix}}_{\mathbf{N}_{n1n1}} \mathbf{f}_{n1} + \underbrace{\begin{pmatrix} \mathbf{M}_{xn1y} \\ \mathbf{M}_{wn1y} \end{pmatrix}}_{\mathbf{N}_{n1y}} \mathbf{u} \\ &+ \underbrace{\begin{pmatrix} \mathbf{M}_{xn1x1} & \mathbf{M}_{xn1w1} \\ \mathbf{M}_{wn1x1} & \mathbf{M}_{wn1w1} \end{pmatrix}}_{\mathbf{N}_{n1l1}} \begin{pmatrix} \frac{1}{2} \mathbf{Q} & 0 \\ 0 & \mathbf{Z}_1 \end{pmatrix} \mathbf{v}_1 + \underbrace{\begin{pmatrix} \mathbf{M}_{xn1x1} \\ \mathbf{M}_{wn1x1} \end{pmatrix}}_{\mathbf{N}_{n1x1}} \mathbf{Q} \mathbf{x}_1 \end{aligned} \quad (6)$$

3.2.2 Presolve numerical nonlinear subsystem

$$\begin{aligned} \begin{pmatrix} \mathbf{I}_d & 0 \\ 0 & \mathbf{I}_d \end{pmatrix} \mathbf{v}_{n1} &= \underbrace{(\mathbf{N}_{n1x1} + \mathbf{N}_{n1l1} \mathbf{N}_{1x1})}_{\mathbf{N}_{n1x1}} \mathbf{x}_1 + \underbrace{(\mathbf{N}_{n1n1} + \mathbf{N}_{n1l1} \mathbf{N}_{1n1})}_{\mathbf{N}_{n1n1}} \mathbf{f}_{n1} \\ &\underbrace{(\mathbf{N}_{n1y} + \mathbf{N}_{n1l1} \mathbf{N}_{1y})}_{\mathbf{N}_{n1y}} \mathbf{u} \end{aligned} \quad (7)$$

3.2.3 Algorithm

Inputs

$$\begin{aligned}
\mathbf{iD}_1 &= \begin{pmatrix} \frac{\mathbf{I}_d}{\delta t} & 0 \\ 0 & \mathbf{I}_d \end{pmatrix} - \begin{pmatrix} \mathbf{M}_{x1x1} & \mathbf{M}_{x1w1} \\ \mathbf{M}_{w1x1} & \mathbf{M}_{w1w1} \end{pmatrix} \begin{pmatrix} \frac{1}{2} \mathbf{Q} & 0 \\ 0 & \mathbf{Z}_1 \end{pmatrix} \\
\overline{\mathbf{N}_{1x1}} &= \begin{pmatrix} \mathbf{M}_{x1x1} \\ \mathbf{M}_{w1x1} \end{pmatrix} \mathbf{Q} \\
\overline{\mathbf{N}_{1n1}} &= \begin{pmatrix} \mathbf{M}_{x1xn1} & \mathbf{M}_{x1wn1} \\ \mathbf{M}_{w1xn1} & \mathbf{M}_{w1wn1} \end{pmatrix} \\
\overline{\mathbf{N}_{1y}} &= \begin{pmatrix} \mathbf{M}_{x1y} \\ \mathbf{M}_{w1y} \end{pmatrix} \\
\overline{\mathbf{N}_{n1n1}} &= \begin{pmatrix} \mathbf{M}_{xn1xn1} & \mathbf{M}_{xn1wn1} \\ \mathbf{M}_{wn1xn1} & \mathbf{M}_{wn1wn1} \end{pmatrix} \\
\overline{\mathbf{N}_{n11}} &= \begin{pmatrix} \mathbf{M}_{xn1x1} & \mathbf{M}_{xn1w1} \\ \mathbf{M}_{wn1x1} & \mathbf{M}_{wn1w1} \end{pmatrix} \begin{pmatrix} \frac{1}{2} \mathbf{Q} & 0 \\ 0 & \mathbf{Z}_1 \end{pmatrix} \\
\overline{\mathbf{N}_{n1x1}} &= \begin{pmatrix} \mathbf{M}_{xn1x1} \\ \mathbf{M}_{wn1x1} \end{pmatrix} \mathbf{Q} \\
\overline{\mathbf{N}_{n1y}} &= \begin{pmatrix} \mathbf{M}_{xn1y} \\ \mathbf{M}_{wn1y} \end{pmatrix} \\
\mathcal{J}_{f_{n1}}(\mathbf{v}_{n1}) &= \begin{pmatrix} \mathcal{J}_{\nabla H_{n1}} & 0 \\ 0 & \mathcal{J}_{z_{n1}} \end{pmatrix} \\
\mathbf{I}_{n1} &= \begin{pmatrix} \frac{\mathbf{I}_d}{\delta t} & 0 \\ 0 & \mathbf{I}_d \end{pmatrix}
\end{aligned} \tag{8}$$

Process

$$\begin{aligned}
\mathbf{D}_1 &= \mathbf{iD}_1^{-1} \\
\mathbf{N}_{1x1} &= \mathbf{D}_1 \overline{\mathbf{N}_{1x1}} \\
\mathbf{N}_{1n1} &= \mathbf{D}_1 \overline{\mathbf{N}_{1n1}} \\
\mathbf{N}_{1y} &= \mathbf{D}_1 \overline{\mathbf{N}_{1y}} \\
\mathbf{N}_{n1x1} &= \overline{\mathbf{N}_{n1x1}} + \overline{\mathbf{N}_{n11}} \mathbf{N}_{1x1} \\
\mathbf{N}_{n1n1} &= \overline{\mathbf{N}_{n1n1}} + \overline{\mathbf{N}_{n11}} \mathbf{N}_{1n1} \\
\mathbf{N}_{n1y} &= \overline{\mathbf{N}_{n1y}} + \overline{\mathbf{N}_{n11}} \mathbf{N}_{1y} \\
\mathbf{c} &= \mathbf{N}_{n1x1} \mathbf{x}_1 + \mathbf{N}_{n1y} \mathbf{u} \\
\text{Iterate : } \mathbf{F}_{n1}(\mathbf{v}_{n1}) &= \mathbf{I}_{n1} \mathbf{v}_{n1} - \mathbf{N}_{n1n1} \mathbf{f}_{n1} - \mathbf{c} \\
\mathcal{J}_{\mathbf{F}_{n1}}(\mathbf{v}_{n1}) &= \mathbf{I}_{n1} - \mathbf{N}_{n1n1} \mathcal{J}_{f_{n1}}(\mathbf{v}_{n1}) \\
\mathbf{v}_{n1} &= \mathbf{v}_{n1} - \mathcal{J}_{\mathbf{F}_{n1}}^{-1}(\mathbf{v}_{n1}) \mathbf{F}_{n1}(\mathbf{v}_{n1}) \\
\mathbf{v}_1 &= \mathbf{N}_{1x1} \mathbf{x}_1 + \mathbf{N}_{1n1} \mathbf{f}_{n1} + \mathbf{N}_{1y} \mathbf{u} \\
\mathbf{y} &= \mathbf{M}_{yx1} \nabla H_1 + \mathbf{M}_{yxn1} \nabla H_{n1} \mathbf{M}_{yw1} \mathbf{Z}_1 \mathbf{w}_1 + \mathbf{M}_{ywn1} \mathbf{z}_{n1} + \mathbf{M}_{yy} \mathbf{u} \\
\mathbf{x} &= \mathbf{x} + \delta \mathbf{x}
\end{aligned} \tag{9}$$

$$\mathbf{y} = \mathbf{M}_{yx1} \nabla H_1 + \mathbf{M}_{yxn1} \nabla H_{n1} \mathbf{M}_{yw1} \mathbf{Z}_1 \mathbf{w}_1 + \mathbf{M}_{ywn1} \mathbf{z}_{n1} + \mathbf{M}_{yy} \mathbf{u} \tag{10}$$

$$= \mathbf{M}_{yx1} \nabla H_1 + \mathbf{M}_{yxn1} \nabla H_{n1} \mathbf{M}_{yw1} \mathbf{Z}_1 \mathbf{w}_1 + \mathbf{M}_{ywn1} \mathbf{z}_{n1} + \mathbf{M}_{yy} \mathbf{u} \tag{11}$$

$$\tag{12}$$

3.3 Realizability solver

Connections

Serial $\sum_{n=1}^N e_n = 0$,
 $f_1 = \dots = f_N = \phi$ (variable commune)

parallel $\sum_{n=1}^N f_n = 0$,
 $e_1 = \dots = e_N = \phi$ (variable commune)
Storage

Realizable

$$\begin{cases} \phi = u = \frac{d}{dt}x & = \frac{dx_1}{dt} = \dots = \frac{dx_N}{dt} \\ y = \nabla H(x) & = \sum_{i=1}^N \nabla H_i(x_i) \end{cases}$$

alors $x = x_1 = x_2$ et $H(x) = \left(\sum_{i=1}^N H_i \right) (x)$

Non-Realizable

$$\begin{cases} \phi = u = \nabla H(x) & = \nabla H_1(x_1) = \dots = \nabla H_N(x_N), \\ y = \frac{d}{dt}x & = \sum_{i=1}^N \frac{d}{dt}x_i \end{cases}$$

alors $x = \sum_{i=1}^N x_i$ et $H(x) = \left(\sum_{i=1}^N H_i \nabla H_i^{-1} G \right) (x)$

avec $G^{-1}(x) = \sum_{i=1}^N \nabla H_i(x_i)$

Dissipatives

Realizable

$$\begin{cases} \phi = u = w & = w_1 = \dots = w_N, \\ y = z(w) & = \sum_{i=1}^N z_i(x_i) \end{cases}$$

Non-Realizable

$$\begin{cases} \phi = u = z(w) & = z_1(w_1) = \dots = z_N(w_N), \\ y = w & = \sum_{i=1}^N w_i \end{cases}$$

alors $w = \sum_{i=1}^N w_i = \left(\sum_{i=1}^N z_i^{-1} \right) (\phi)$ et $z^{-1}(\phi) = w \Rightarrow z(w) = \left(\sum_{i=1}^N z_i \right)^{-1} (w)$

Références

[Falaize and H  lie, 2016] Falaize, A. and H  lie, T. (2016). Passive guaranteed simulation of analog audio circuits : A port-hamiltonian approach. *Applied Sciences*, 6(10) :273.