BMSpy Documentation

Release 0.0.7

Steven Masfaraud

Jun 02, 2020

Contents

Contents:

CHAPTER 1

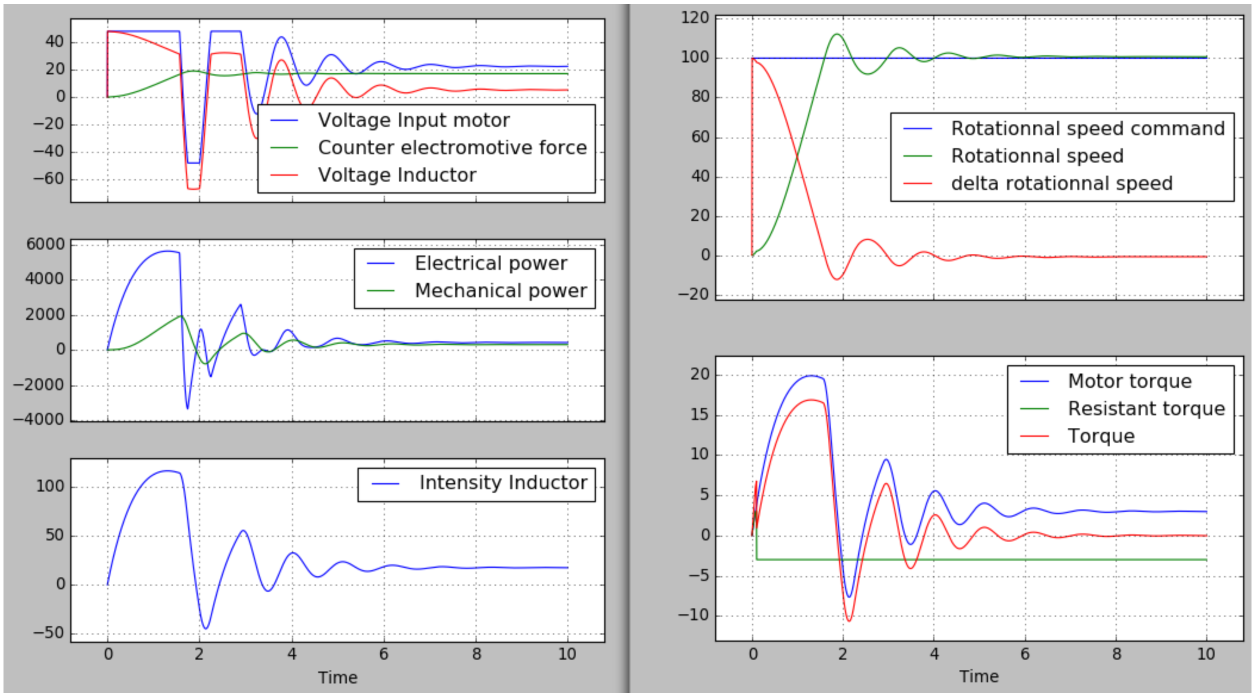
Getting Started

1.1 What is BMS for?

BMS stands for "Block Model Simulator". It helps defining a DynamicalSystem, a collection of variables linked by equations or behaviors.

The values of the model's variables are computed by the model and can be displayed or post-treated.

Here is for example the output of an electric motor model:



1.2 Installation

The easy way:

```
pip install bms
```

or, if you are running python3:

```
pip3 install bms
```

Alternatively, you can download the source at: https://pypi.python.org/pypi/bms/ After extracting, execute:

```
python setup.py install
```

If you are running python3:

```
python3 setup.py install
```

CHAPTER 2

Development

BMS is beeing actively developed! Feel free to interact!

Questions and bugs can be reported on github: https://github.com/masfaraud/BMSpy/issues

2.1 Release Notes

see also releases on github: https://github.com/masfaraud/BMSpy/releases

2.1.1 Version 0.0

This is the alpha version. Code standards change rapidely, and compatibility is this version is not guarentied.

Version 0.0.8

- Physical modeling in order to generate automaticaly dynamic systems from physical components layout.
- Solver major improvement: loops in dynamic system are solved as a system of equations with an optimizer.

Version 0.0.7

Minor changes

Version 0.0.6

- Sphinx Documentation
- Variables accessible at time value by DynamicSystem method

Version 0.0.5

- Version number standard change
- Model Saving/Loading from file
- New version of model drawing
- Inputs renamed Signals
- Drag & Drop Model drawer

Version 0.04

- Reorganisation into subpackages of blocks and inputs

Version 0.03

- Bug correction for float time step
- Redefinition of number of steps

Version 0.02

- New blocks such as saturation or coulomb
- Bug fixes

Version 0.01

Initial release

2.2 Roadmap

- Implement computation of derivatives at t=0 for inputs
- Implement indicator of convergence when solving at a time step
- Nice model drawing (upgrade existing drag & drop interface)

CHAPTER 3

Reference

3.1 Core

Core of BMS. All content of this file is imported by bms, and is therefore in bms

This file defines the base of BMS.

class bms.core.**Block**(*inputs*, *outputs*, *max_input_order*, *max_output_order*)
Bases: object

Abstract class of block: this class should not be instanciate directly

InputValues(*it*, *nsteps=None*)
Returns the input values at a given iteration for solving the block outputs

OutputValues(*it*, *nsteps=None*)

Solve(*it*, *ts*)

class bms.core.**DynamicSystem**(*te*, *ns*, *blocks=[]*)
Bases: object

Defines a dynamic system that can simulate itself

Parameters

- **te** – time of simulation's end
- **ns** – number of steps
- **blocks** – (optional) list of blocks defining the model

AddBlock(*block*)
Add the given block to the model and also its input/output variables

DrawModel()

PlotVariables(*subplots_variables=None*)

Save(*name_file*)
name_file: name of the file without extension. The extension .bms is added by function

Simulate(*variables_to_solve=None*)

VariablesValues(*variables*, *t*)
Returns the value of given variables at time t. Linear interpolation is performed between two time steps.

Parameters

- **variables** – one variable or a list of variables
- **t** – time of evaluation

graph

bms.core.**Load**(*file*)
Loads a model from specified file

exception bms.core.**ModelError**(*message*)
Bases: Exception

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

class bms.core.**PhysicalBlock**(*physical_nodes*, *nodes_with_fluxes*, *occurence_matrix*, *commands*, *name*)
Bases: object

Abstract class to inherit when coding a physical block

class bms.core.**PhysicalNode**(*cl_solves_potential*, *cl_solves_fluxes*, *node_name*, *potential_variable_name*, *flux_variable_name*)
Bases: object

Abstract class

class bms.core.**PhysicalSystem**(*te*, *ns*, *physical_blocks*, *command_blocks*)
Bases: object

Defines a physical system

AddCommandBlock(*block*)

AddPhysicalBlock(*block*)

GenerateDynamicSystem()

Simulate()

dynamic_system

class bms.core.**Signal**(*names*)
Bases: *bms.core.Variable*

Abstract class of signal

values

class bms.core.**Variable**(*names='variable', initial_values=[0], hidden=False*)
Bases: object

Defines a variable

Parameters **names** – Defines full name and short name.

If names is a string the two names will be identical otherwise names should be a tuple of strings (full_name,short_name)

Parameters **hidden** – inner variable to hide in plots if true

values

3.2 Signals

3.2.1 Functions

Collection of mathematical function signals

class bms.signals.functions.**Ramp**(*name='Ramp'*, *amplitude=1*, *delay=0*, *offset=0*)

Bases: *bms.core.Signal*

Create a Ramp with a certain amplitude, time delay and offset.

$$f(t) = amplitude \times (t - delay) + offset$$

Parameters

- **name** (*str*) – The name of this signal.
- **amplitude** – The angular coefficient of the Ramp function.
- **delay** – The horizontal offset of the function.
- **offset** – The vertical offset of the function.

values

class bms.signals.functions.**SignalFunction**(*name*, *function*)

Bases: *bms.core.Signal*

Create a signal based on a function defined by the user.

Parameters

- **name** (*str*) – The name of this signal.
- **function** – A function that depends on time.

values

class bms.signals.functions.**Sinus**(*name='Sinus'*, *amplitude=1*, *w=1*, *phase=0*, *offset=0*)

Bases: *bms.core.Signal*

Create a Sine wave with a certain amplitude, angular velocity, phase and offset.

$$f(t) = amplitude \times sin(\omega \times t + phase) + offset$$

Parameters

- **name** (*str*) – The name of this signal.
- **amplitude** – The amplitude of the sine wave.
- **w** – The angular velocity of the sine wave (ω).
- **phase** – The phase of the sine wave.
- **offset** – The vertical offset of the function.

values

class bms.signals.functions.**Step**(*name='Step'*, *amplitude=1*, *delay=0*, *offset=0*)

Bases: *bms.core.Signal*

Create a Step with a certain amplitude, time delay and offset.

$$f(t) = amplitude \times u(t - delay) + offset$$

where

$$u(t) = \begin{cases} 0, & \text{if } t < 0 \\ 1, & \text{if } t \geq 0 \end{cases}$$

Parameters

- **name** (*str*) – The name of this signal.
- **amplitude** – The height of the step function.
- **delay** – The time to wait before the function stops being zero.
- **offset** – The vertical offset of the function.

values

3.2.2 WLTP signals

WLTP signals

class bms.signals.wltp.**WLTP1**(*name*)

Bases: *bms.core.Signal*

WLTP classe 1 cycle Caution! speed in m/s, not in km/h!

values

class bms.signals.wltp.**WLTP2**(*name*)

Bases: *bms.core.Signal*

WLTP classe 2 cycle Caution! speed in m/s, not in km/h!

values

class bms.signals.wltp.**WLTP3**(*name*)

Bases: *bms.core.Signal*

WLTP classe 3 cycle Caution! speed in m/s, not in km/h!

values

3.3 Blocks

3.3.1 Continuous Blocks

Collection of continuous blocks

class bms.blocks.continuous.**DifferentiationBlock**(*input_variable*, *output_variable*)

Bases: *bms.blocks.continuous.ODE*

Creates an ODE block that performs differentation of the input relative to time.

$$output = \frac{d[input]}{dt}$$

Parameters

- **input_variable** – This is the input or list of inputs of the block.
- **output_variable** (Variable) – This is the output of the block.

Evaluate(*it*, *ts*)

InputValues(*it*, *nsteps=None*)

Returns the input values at a given iteration for solving the block outputs

LabelBlock()

LabelConnections()

OutputMatrices(*delta_t*)

OutputValues(*it*, *nsteps=None*)

Solve(*it*, *ts*)

class bms.blocks.continuous.**Division**(*input_variable1*, *input_variable2*, *output_variable*)

Bases: *bms.core.Block*

Defines a division between its inputs.

$$output = \frac{input1}{input2}$$

Parameters

- **input_variable1** (Variable) – This is the first input of the block, the dividend.
- **input_variable2** (Variable) – This is the second input of the block, the divisor.
- **output_variable** (Variable) – This is the output of the block, the quotient.

Evaluate(*it*, *ts*)

InputValues(*it*, *nsteps=None*)

Returns the input values at a given iteration for solving the block outputs

LabelBlock()

LabelConnections()

OutputValues(*it*, *nsteps=None*)

Solve(*it*, *ts*)

class bms.blocks.continuous.**FunctionBlock**(*input_variable*, *output_variable*, *function*)

Bases: *bms.core.Block*

This defines a custom function over the input(s).

$$output = f(input)$$

Parameters

- **input_variable** – This is the input or list of inputs of the block.

- **output_variable** (Variable) – This is the output of the block.
- **function** – This is the function that takes the inputs and returns the output.

Evaluate (*it*, *ts*)

InputValues (*it*, *nsteps=None*)
Returns the input values at a given iteration for solving the block outputs

LabelBlock ()

LabelConnections ()

OutputValues (*it*, *nsteps=None*)

Solve (*it*, *ts*)

class bms.blocks.continuous.**Gain** (*input_variable*, *output_variable*, *value*, *offset=0*)
Bases: *bms.core.Block*

Defines a gain operation.

$$output = (value \times input) + offset$$

Parameters

- **input_variable** (Variable) – This is the input of the block.
- **output_variable** (Variable) – This is the output of the block.
- **gain** – This is what multiplies the input.
- **offset** – This is added to the input after being multiplied.

Evaluate (*it*, *ts*)

InputValues (*it*, *nsteps=None*)
Returns the input values at a given iteration for solving the block outputs

LabelBlock ()

LabelConnections ()

OutputValues (*it*, *nsteps=None*)

Solve (*it*, *ts*)

class bms.blocks.continuous.**IntegrationBlock** (*input_variable*, *output_variable*)
Bases: *bms.blocks.continuous.ODE*

Creates an ODE block that performs integration of the input over time.

$$output = \int_0^t input\ dt$$

Parameters

- **input_variable** – This is the input or list of inputs of the block.
- **output_variable** (Variable) – This is the output of the block.

Evaluate (*it*, *ts*)

InputValues (*it*, *nsteps=None*)
Returns the input values at a given iteration for solving the block outputs

LabelBlock ()

LabelConnections ()

OutputMatrices (*delta_t*)

OutputValues (*it*, *nsteps=None*)

Solve (*it*, *ts*)

class bms.blocks.continuous.**ODE** (*input_variable*, *output_variable*, *a*, *b*)
Bases: *bms.core.Block*

Defines an ordinary differential equation based on the input.

a, b are vectors of coefficients so that H, the transfer function of the block, can be written as:

$$H(p) = \frac{a_i p^i}{b_j p^j}$$

with Einstein sum on i and j, and p is Laplace's variable.

For example, `a=[1], b=[0,1]` is an integration, and `a=[0,1], b=[1]` is a differentiation.

Parameters

- **input_variable** (Variable) – This is the input of the block.
- **output_variable** (Variable) – This is the output of the block.
- **a** – This is the a vector for the transfer function.
- **b** – This is the b vector for the transfer function.

Evaluate (*it*, *ts*)

InputValues (*it*, *nsteps=None*)
Returns the input values at a given iteration for solving the block outputs

LabelBlock ()

LabelConnections ()

OutputMatrices (*delta_t*)

OutputValues (*it*, *nsteps=None*)

Solve (*it*, *ts*)

class bms.blocks.continuous.**Product** (*input_variable1*, *input_variable2*, *output_variable*)
Bases: *bms.core.Block*

Defines a multiplication between its inputs.

$$output = input_1 \times input_2$$

Parameters

- **input_variable1** (Variable) – This is the first input of the block, one factor.
- **input_variable2** (Variable) – This is the second input of the block, another factor.
- **output_variable** (Variable) – This is the output of the block, the product.

Evaluate (*it*, *ts*)

InputValues (*it*, *nsteps=None*)
Returns the input values at a given iteration for solving the block outputs

LabelBlock ()

LabelConnections ()

OutputValues (*it*, *nsteps=None*)

Solve (*it*, *ts*)

class bms.blocks.continuous.**Subtraction** (*input_variable1*, *input_variable2*, *output_variable*)

Bases: `bms.core.Block`

Defines a subtraction between its two inputs.

$$output = input_1 - input_2$$

Parameters

- **input_variable1** (`Variable`) – This is the first input of the block, the minuend.
- **input_variable2** (`Variable`) – This is the second input of the block, the subtrahend.
- **output_variable** (`Variable`) – This is the output of the block, the difference.

Evaluate (*it*, *ts*)

InputValues (*it*, *nsteps=None*)

Returns the input values at a given iteration for solving the block outputs

LabelBlock ()

LabelConnections ()

OutputValues (*it*, *nsteps=None*)

Solve (*it*, *ts*)

class bms.blocks.continuous.**Sum** (*inputs*, *output_variable*)

Bases: `bms.core.Block`

Defines a sum over its inputs.

$$output = \sum input_i$$

Parameters

- **input_variable** (*list [Variables]*) – This is the list of inputs of the block.
- **output_variable** (`Variable`) – This is the output of the block.

Evaluate (*it*, *ts*)

InputValues (*it*, *nsteps=None*)

Returns the input values at a given iteration for solving the block outputs

LabelBlock ()

LabelConnections ()

OutputValues (*it*, *nsteps=None*)

Solve (*it*, *ts*)

class bms.blocks.continuous.**WeightedSum** (*inputs*, *output_variable*, *weights*, *offset=0*)

Bases: `bms.core.Block`

Defines a weighted sum over its inputs.

$$output = \sum w_i \times input_i$$

Parameters

- **input_variable** (*list[Variables]*) – This is the list of inputs of the block.
- **output_variable** (Variable) – This is the output of the block.
- **weights** – These are the weights that are multiplied by the elements of the input.
- **offset** – This offset is added to the final result.

Evaluate (*it*, *ts*)

InputValues (*it*, *nsteps=None*)
Returns the input values at a given iteration for solving the block outputs

LabelBlock ()

LabelConnections ()

OutputValues (*it*, *nsteps=None*)

Solve (*it*, *ts*)

3.3.2 Non-linear Blocks

Collection of non-linear blocks

class bms.blocks.nonlinear.**Coulomb** (*input_variable*, *speed_variable*, *output_variable*, *max_value*, *tolerance=0*)
Bases: *bms.core.Block*

Return coulomb force under condition of speed and sum of forces (input)

Evaluate (*it*, *ts*)

InputValues (*it*, *nsteps=None*)
Returns the input values at a given iteration for solving the block outputs

LabelBlock ()

OutputValues (*it*, *nsteps=None*)

Solve (*it*, *ts*)

class bms.blocks.nonlinear.**CoulombVariableValue** (*external_force*, *speed_variable*, *value_variable*, *output_variable*, *tolerance=0*)
Bases: *bms.core.Block*

Return coulomb force under condition of speed and sum of forces (input) The max value is driven by an input

Evaluate (*it*, *ts*)

InputValues (*it*, *nsteps=None*)
Returns the input values at a given iteration for solving the block outputs

LabelBlock ()

OutputValues (*it*, *nsteps=None*)

Solve (*it*, *ts*)

class bms.blocks.nonlinear.**Delay** (*input_variable*, *output_variable*, *delay*)
Bases: *bms.core.Block*

Simple block to delay output with respect to input.

Parameters **delay** – a delay in seconds

Evaluate(*it*, *ts*)

InputValues(*it*, *nsteps=None*)
Returns the input values at a given iteration for solving the block outputs

Label()

OutputValues(*it*, *nsteps=None*)

Solve(*it*, *ts*)

class bms.blocks.nonlinear.**RegCoulombVariableValue**(*external_force*, *speed_variable*, *value_variable*, *output_variable*, *tolerance=0*)
Bases: bms.core.Block

Return coulomb force under condition of speed and sum of forces (input) The max value is driven by an input

Evaluate(*it*, *ts*)

InputValues(*it*, *nsteps=None*)
Returns the input values at a given iteration for solving the block outputs

LabelBlock()

OutputValues(*it*, *nsteps=None*)

Solve(*it*, *ts*)

class bms.blocks.nonlinear.**Saturation**(*input_variable*, *output_variable*, *min_value*, *max_value*)
Bases: bms.core.Block

Defines a saturation block.

$$output = \begin{cases} min\_value, & \text{if } input < min\_value \\ max\_value, & \text{if } input > max\_value \\ input, & \text{if } min\_value \leq input \leq max\_value \end{cases}$$

Parameters

- **input_variable** (Variable) – This is the input of the block.
- **output_variable** (Variable) – This is the output of the block.
- **min_value** – This is the lower bound for the output.
- **max_value** – This is the upper bound for the output.

Evaluate(*it*, *ts*)

InputValues(*it*, *nsteps=None*)
Returns the input values at a given iteration for solving the block outputs

LabelBlock()

OutputValues(*it*, *nsteps=None*)

Solve(*it*, *ts*)

class bms.blocks.nonlinear.**Sign**(*input_variable*, *output_variable*)
Bases: bms.core.Block

Defines a sign operation on the input.

$$output = \begin{cases} -1, & \text{if } input < 0 \\ 0, & \text{if } input = 0 \\ 1, & \text{if } input > 0 \end{cases}$$

Evaluate (*it*, *ts*)

InputValues (*it*, *nsteps=None*)
    Returns the input values at a given iteration for solving the block outputs

LabelBlock ()

OutputValues (*it*, *nsteps=None*)

Solve (*it*, *ts*)

Python Module Index

b

Index

L

M

O

P

R

S

V

W