
BMSpy Documentation

Release 0.0.7

Steven Masfarau

Jun 02, 2020

Contents

1	Getting Started	3
1.1	What is BMS for?	3
1.2	Installation	4
2	Development	5
2.1	Release Notes	5
2.2	Roadmap	6
3	Reference	7
3.1	Core	7
3.2	Signals	9
3.3	Blocks	10
	Python Module Index	19
	Index	21

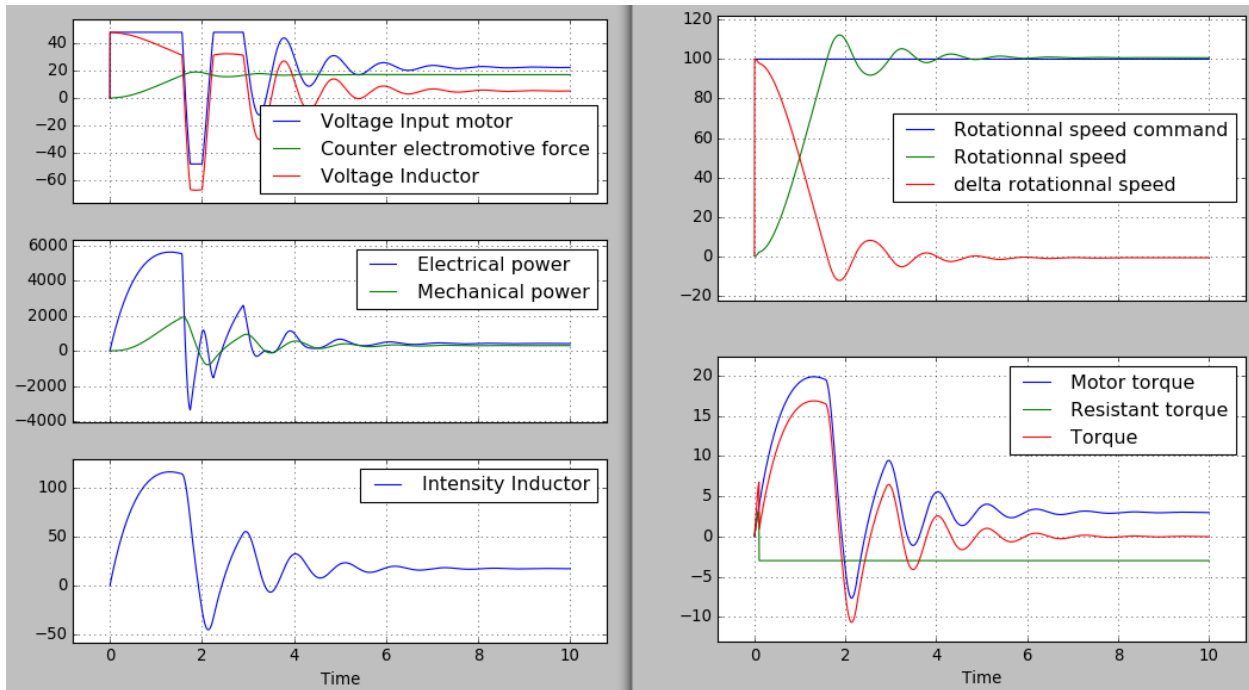
Contents:

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
```


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 rapidly, 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)

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 instantiated 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, occurrence_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.

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

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.

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

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.

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

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 differentiation 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*)

b

`bms.blocks`, 10
`bms.blocks.continuous`, 10
`bms.blocks.nonlinear`, 15
`bms.core`, 7
`bms.signals`, 9
`bms.signals.functions`, 9
`bms.signals.wltp`, 10

A

AddBlock() (*bms.core.DynamicSystem method*), 7
 AddCommandBlock() (*bms.core.PhysicalSystem method*), 8
 AddPhysicalBlock() (*bms.core.PhysicalSystem method*), 8
 args (*bms.core.ModelError attribute*), 8

B

Block (*class in bms.core*), 7
 bms.blocks (*module*), 10
 bms.blocks.continuous (*module*), 10
 bms.blocks.nonlinear (*module*), 15
 bms.core (*module*), 7
 bms.signals (*module*), 9
 bms.signals.functions (*module*), 9
 bms.signals.wltp (*module*), 10

C

Coulomb (*class in bms.blocks.nonlinear*), 15
 CoulombVariableValue (*class in bms.blocks.nonlinear*), 15

D

Delay (*class in bms.blocks.nonlinear*), 15
 DifferentiationBlock (*class in bms.blocks.continuous*), 10
 Division (*class in bms.blocks.continuous*), 11
 DrawModel() (*bms.core.DynamicSystem method*), 7
 dynamic_system (*bms.core.PhysicalSystem attribute*), 8
 DynamicSystem (*class in bms.core*), 7

E

Evaluate() (*bms.blocks.continuous.DifferentiationBlock method*), 11
 Evaluate() (*bms.blocks.continuous.Division method*), 11

Evaluate() (*bms.blocks.continuous.FunctionBlock method*), 12
 Evaluate() (*bms.blocks.continuous.Gain method*), 12
 Evaluate() (*bms.blocks.continuous.IntegrationBlock method*), 12
 Evaluate() (*bms.blocks.continuous.ODE method*), 13
 Evaluate() (*bms.blocks.continuous.Product method*), 13
 Evaluate() (*bms.blocks.continuous.Subtraction method*), 14
 Evaluate() (*bms.blocks.continuous.Sum method*), 14
 Evaluate() (*bms.blocks.continuous.WeightedSum method*), 15
 Evaluate() (*bms.blocks.nonlinear.Coulomb method*), 15
 Evaluate() (*bms.blocks.nonlinear.CoulombVariableValue method*), 15
 Evaluate() (*bms.blocks.nonlinear.Delay method*), 16
 Evaluate() (*bms.blocks.nonlinear.RegCoulombVariableValue method*), 16
 Evaluate() (*bms.blocks.nonlinear.Saturation method*), 16
 Evaluate() (*bms.blocks.nonlinear.Sign method*), 17

F

FunctionBlock (*class in bms.blocks.continuous*), 11

G

Gain (*class in bms.blocks.continuous*), 12
 GenerateDynamicSystem() (*bms.core.PhysicalSystem method*), 8
 graph (*bms.core.DynamicSystem attribute*), 8

I

InputValues() (*bms.blocks.continuous.DifferentiationBlock method*), 11
 InputValues() (*bms.blocks.continuous.Division method*), 11
 InputValues() (*bms.blocks.continuous.FunctionBlock method*), 12

InputValues () (bms.blocks.continuous.Gain method), 12
 InputValues () (bms.blocks.continuous.IntegrationBlock method), 12
 InputValues () (bms.blocks.continuous.ODE method), 13
 InputValues () (bms.blocks.continuous.Product method), 13
 InputValues () (bms.blocks.continuous.Subtraction method), 14
 InputValues () (bms.blocks.continuous.Sum method), 14
 InputValues () (bms.blocks.continuous.WeightedSum method), 15
 InputValues () (bms.blocks.nonlinear.Coulomb method), 15
 InputValues () (bms.blocks.nonlinear.CoulombVariableValue method), 15
 InputValues () (bms.blocks.nonlinear.Delay method), 16
 InputValues () (bms.blocks.nonlinear.RegCoulombVariableValue method), 16
 InputValues () (bms.blocks.nonlinear.Saturation method), 16
 InputValues () (bms.blocks.nonlinear.Sign method), 17
 InputValues () (bms.core.Block method), 7
 IntegrationBlock (class in bms.blocks.continuous), 12

L

Label () (bms.blocks.nonlinear.Delay method), 16
 LabelBlock () (bms.blocks.continuous.DifferentiationBlock method), 11
 LabelBlock () (bms.blocks.continuous.Division method), 11
 LabelBlock () (bms.blocks.continuous.FunctionBlock method), 12
 LabelBlock () (bms.blocks.continuous.Gain method), 12
 LabelBlock () (bms.blocks.continuous.IntegrationBlock method), 12
 LabelBlock () (bms.blocks.continuous.ODE method), 13
 LabelBlock () (bms.blocks.continuous.Product method), 13
 LabelBlock () (bms.blocks.continuous.Subtraction method), 14
 LabelBlock () (bms.blocks.continuous.Sum method), 14
 LabelBlock () (bms.blocks.continuous.WeightedSum method), 15
 LabelBlock () (bms.blocks.nonlinear.Coulomb method), 15

LabelBlock () (bms.blocks.nonlinear.CoulombVariableValue method), 15
 LabelBlock () (bms.blocks.nonlinear.RegCoulombVariableValue method), 16
 LabelBlock () (bms.blocks.nonlinear.Saturation method), 16
 LabelBlock () (bms.blocks.nonlinear.Sign method), 17
 LabelConnections () (bms.blocks.continuous.DifferentiationBlock method), 11
 LabelConnections () (bms.blocks.continuous.Division method), 11
 LabelConnections () (bms.blocks.continuous.FunctionBlock method), 12
 LabelConnections () (bms.blocks.continuous.Gain method), 12
 LabelConnections () (bms.blocks.continuous.IntegrationBlock method), 12
 LabelConnections () (bms.blocks.continuous.ODE method), 13
 LabelConnections () (bms.blocks.continuous.Product method), 13
 LabelConnections () (bms.blocks.continuous.Subtraction method), 14
 LabelConnections () (bms.blocks.continuous.Sum method), 14
 LabelConnections () (bms.blocks.continuous.WeightedSum method), 15
 Load () (in module bms.core), 8

M

ModelError, 8

O

ODE (class in bms.blocks.continuous), 13
 OutputMatrices () (bms.blocks.continuous.DifferentiationBlock method), 11
 OutputMatrices () (bms.blocks.continuous.IntegrationBlock method), 13
 OutputMatrices () (bms.blocks.continuous.ODE method), 13
 OutputValues () (bms.blocks.continuous.DifferentiationBlock method), 11
 OutputValues () (bms.blocks.continuous.Division method), 11
 OutputValues () (bms.blocks.continuous.FunctionBlock method), 12

OutputValues() (*bms.blocks.continuous.Gain method*), 12
 OutputValues() (*bms.blocks.continuous.IntegrationBlock method*), 13
 OutputValues() (*bms.blocks.continuous.ODE method*), 13
 OutputValues() (*bms.blocks.continuous.Product method*), 14
 OutputValues() (*bms.blocks.continuous.Subtraction method*), 14
 OutputValues() (*bms.blocks.continuous.Sum method*), 14
 OutputValues() (*bms.blocks.continuous.WeightedSum method*), 15
 OutputValues() (*bms.blocks.nonlinear.Coulomb method*), 15
 OutputValues() (*bms.blocks.nonlinear.CoulombVariableValue method*), 15
 OutputValues() (*bms.blocks.nonlinear.Delay method*), 16
 OutputValues() (*bms.blocks.nonlinear.RegCoulombVariableValue method*), 16
 OutputValues() (*bms.blocks.nonlinear.Saturation method*), 16
 OutputValues() (*bms.blocks.nonlinear.Sign method*), 17
 OutputValues() (*bms.core.Block method*), 7

P

PhysicalBlock (*class in bms.core*), 8
 PhysicalNode (*class in bms.core*), 8
 PhysicalSystem (*class in bms.core*), 8
 PlotVariables() (*bms.core.DynamicSystem method*), 7
 Product (*class in bms.blocks.continuous*), 13

R

Ramp (*class in bms.signals.functions*), 9
 RegCoulombVariableValue (*class in bms.blocks.nonlinear*), 16

S

Saturation (*class in bms.blocks.nonlinear*), 16
 Save() (*bms.core.DynamicSystem method*), 7
 Sign (*class in bms.blocks.nonlinear*), 16
 Signal (*class in bms.core*), 8
 SignalFunction (*class in bms.signals.functions*), 9
 Simulate() (*bms.core.DynamicSystem method*), 8
 Simulate() (*bms.core.PhysicalSystem method*), 8
 Sinus (*class in bms.signals.functions*), 9
 Solve() (*bms.blocks.continuous.DifferentiationBlock method*), 11
 Solve() (*bms.blocks.continuous.Division method*), 11
 Solve() (*bms.blocks.continuous.FunctionBlock method*), 12
 Solve() (*bms.blocks.continuous.Gain method*), 12
 Solve() (*bms.blocks.continuous.IntegrationBlock method*), 13
 Solve() (*bms.blocks.continuous.ODE method*), 13
 Solve() (*bms.blocks.continuous.Product method*), 14
 Solve() (*bms.blocks.continuous.Subtraction method*), 14
 Solve() (*bms.blocks.continuous.Sum method*), 14
 Solve() (*bms.blocks.continuous.WeightedSum method*), 15
 Solve() (*bms.blocks.nonlinear.Coulomb method*), 15
 Solve() (*bms.blocks.nonlinear.CoulombVariableValue method*), 15
 Solve() (*bms.blocks.nonlinear.Delay method*), 16
 Solve() (*bms.blocks.nonlinear.RegCoulombVariableValue method*), 16
 Solve() (*bms.blocks.nonlinear.Saturation method*), 16
 Solve() (*bms.blocks.nonlinear.Sign method*), 17
 Solve() (*bms.core.Block method*), 7
 Step (*class in bms.signals.functions*), 10
 Subtraction (*class in bms.blocks.continuous*), 14
 Sum (*class in bms.blocks.continuous*), 14

V

values (*bms.core.Signal attribute*), 8
 values (*bms.core.Variable attribute*), 9
 values (*bms.signals.functions.Ramp attribute*), 9
 values (*bms.signals.functions.SignalFunction attribute*), 9
 values (*bms.signals.functions.Sinus attribute*), 9
 values (*bms.signals.functions.Step attribute*), 10
 values (*bms.signals.wltp.WLTP1 attribute*), 10
 values (*bms.signals.wltp.WLTP2 attribute*), 10
 values (*bms.signals.wltp.WLTP3 attribute*), 10
 Variable (*class in bms.core*), 8
 VariablesValues() (*bms.core.DynamicSystem method*), 8

W

WeightedSum (*class in bms.blocks.continuous*), 14
 with_traceback() (*bms.core.ModelError method*), 8
 WLTP1 (*class in bms.signals.wltp*), 10
 WLTP2 (*class in bms.signals.wltp*), 10
 WLTP3 (*class in bms.signals.wltp*), 10