BMSpy Documentation

Release 0.0.6

Steven Masfaraud

February 04, 2016

CONTENTS

Contents:

CHAPTER

ONE

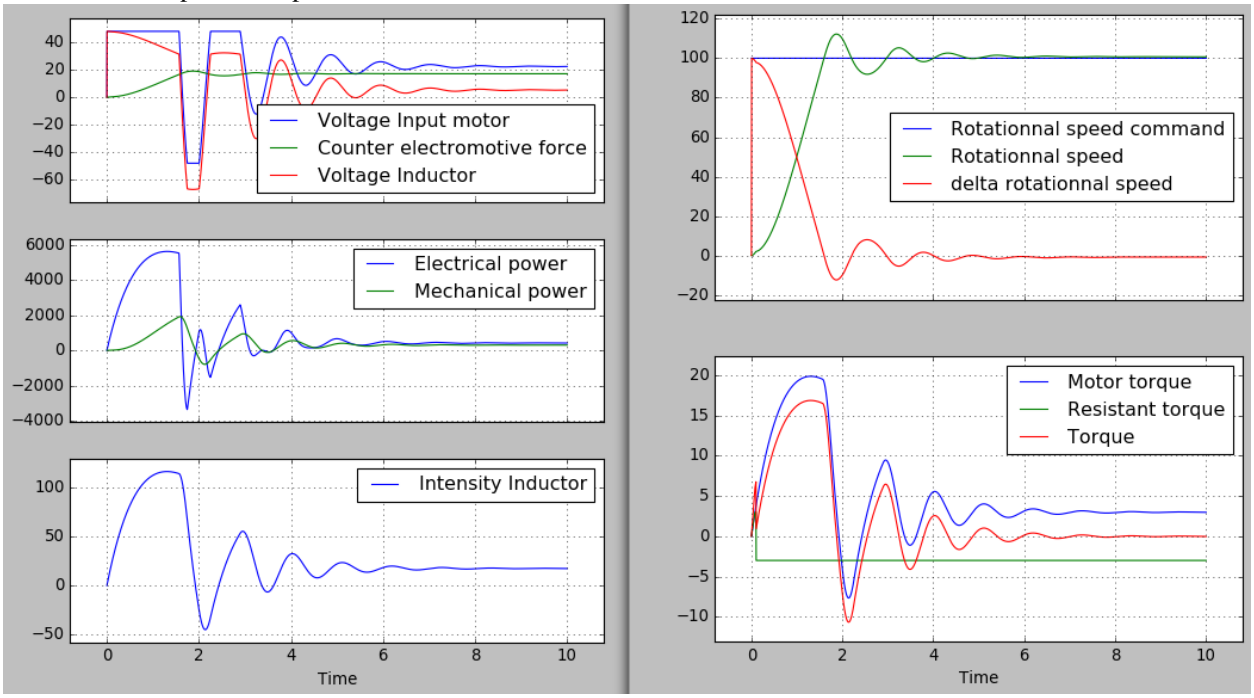
GETTING STARTED

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 of 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

TWO

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 0.0.6

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

2.1.2 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

2.1.3 Version 0.04

- Reorganisation into subpackages of blocks and inputs

2.1.4 Version 0.03

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

2.1.5 Version 0.02

- New blocks such as saturation or coulomb
- Bug fixes

2.1.6 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

THREE

TUTORIAL

The DynamicSystem Class is a python class defined by BMS core. It allows to define a complete model containing all the data for simulation.

3.1 Defining a model

3.1.1 Defining the inputs

Inputs are special variable in the model which are not computed. See the [inputs list](Inputs) Here we define a ramp named which name is e

```
e=Ramp('e',1.)
```

3.1.2 Defining Variables

Let's define a variable s which will be the output of a first order block

```
s=bms.Variable('s')
```

3.1.3 Defining Blocks

Let's define this block:

```
from bms.blocks.continuous import ODE
 block=ODE(e,s,[1],[1,3])
```

3.1.4 Defining the model

```
te=10# time of end in seconds
ns=2000 # number of time steps
model=bms.DynamicSystem(te,ns,[block])
```

The blocks are given in a list as third argument

3.2 Model methods

3.2.1 Simulating

```
model.Simulate()
```

3.2.2 Plotting variables

```
model.PlotVariables()
```

3.2.3 Accessing values

Values of variables at a given time t is accessible by:

```
ds.VariablesValues(t)
```

The time values vector of a variable is accessible via the values attribute:

```
import matplotlib.pyplot as plt
plt.plot(ds.t,e.values)
plt.plot(ds.t,s.values)
```

CHAPTER

FOUR

EXAMPLES

See the project examples folder on github: https://github.com/masfaraud/BMSpy/tree/master/bms/examples

CHAPTER

FIVE

REFERENCE

5.1 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*)
Abstract class of block: this class should not be instanciate directly

InputValues(*it*)
Returns the input values at a given iteration for solving the block outputs

OutputValues(*it*)

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

CheckModelConsistency()

Check for model consistency:

- an input variable can't be set as the output of a block
- a variable can't be the output of more than one block

DrawModel()

PlotVariables(*subplots_variables=None*)

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

Simulate()

VariablesValues(*variables*, *t*)
Returns the value of given variables at time t

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**
Bases: exceptions.Exception

args

message

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

Abstract class of signal

values

class bms.core.**Variable**(*names=''', initial_values=[0]*)
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)

values

5.2 Signals

5.2.1 Functions

Collection of mathematical function signals

class bms.signals.functions.**Ramp**(*name='Ramp'*, *amplitude=1*, *delay=0*, *initial_value=0*)
Bases: *bms.core.Signal*

Create a ramp such as : f(t)=(t-delay)*amplitude+initial_value

values

class bms.signals.functions.**SignalFunction**(*name*, *function*)
Bases: *bms.core.Signal*

User defined function for signal.

Parameters **function** – a function that will give the time values to the signal

values

class bms.signals.functions.**Sinus**(*name='Sinus'*, *amplitude=1*, *w=1*, *phase=0*, *initial_value=0*)
Bases: *bms.core.Signal*

values

class bms.signals.functions.**Step**(*name='Step'*, *amplitude=1*, *delay=0*, *initial_value=0*)
Bases: *bms.core.Signal*

Create a Step of amplitude beginning at time delay

values

5.2.2 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

5.3 Blocks

5.3.1 Continuous Blocks

Collection of continuous blocks

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

output=input1/input2

InputValues(*it*)
Returns the input values at a given iteration for solving the block outputs

LabelBlock()

LabelConnections()

OutputValues(*it*)

Solve(*it*, *ts*)

class bms.blocks.continuous.**FunctionBlock**(*input_variable*, *output_variable*, *function*)
Bases: *bms.core.Block*

output=f(input)

InputValues(*it*)
Returns the input values at a given iteration for solving the block outputs

LabelBlock()

LabelConnections()

OutputValues(*it*)

Solve(*it*, *ts*)

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

output=value* input

InputValues(*it*)
Returns the input values at a given iteration for solving the block outputs

LabelBlock()

LabelConnections()

OutputValues(*it*)

Solve(*it*, *ts*)

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

a,b are vectors of coefficients such as H, the transfert function of the block, may be written as: H(p)=(a[i]p**i)/(b[j]p**j) (Einstein sum on i,j) p is Laplace's variable

InputValues(*it*)
Returns the input values at a given iteration for solving the block outputs

LabelBlock()

LabelConnections()

OutputMatrices(*delta_t*)

OutputValues(*it*)

Solve(*it*, *ts*)

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

output=input1*input2

InputValues(*it*)
Returns the input values at a given iteration for solving the block outputs

LabelBlock()

LabelConnections()

OutputValues(*it*)

Solve(*it*, *ts*)

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

output=input1-input2

InputValues(*it*)
Returns the input values at a given iteration for solving the block outputs

LabelBlock()

LabelConnections()

OutputValues(*it*)

Solve(*it*, *ts*)

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

output=input1+input2

InputValues(*it*)
Returns the input values at a given iteration for solving the block outputs

LabelBlock()

LabelConnections()

OutputValues(*it*)

Solve(*it*, *ts*)

5.3.2 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)

InputValues(*it*)
Returns the input values at a given iteration for solving the block outputs

LabelBlock()

OutputValues(*it*)

Solve(*it*, *ts*)

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

InputValues(*it*)
Returns the input values at a given iteration for solving the block outputs

OutputValues(*it*)

Solve(*it*, *ts*)

class bms.blocks.nonlinear.**Hysteresis**(*input_variable*, *output_variable*, *zone_width*, *initial_value*)
Bases: *bms.core.Block*

InputValues(*it*)
Returns the input values at a given iteration for solving the block outputs

OutputValues(*it*)

Solve(*it*, *ts*)

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

output=min_value if input < min_value output=max_value if input > max_value output=input if min_value < input < max_value

InputValues(*it*)
Returns the input values at a given iteration for solving the block outputs

LabelBlock()

OutputValues(*it*)

Solve(*it*, *ts*)

b

A

B

C

D

F

G

H

I

L

M

O

P

R

S

V

W