
autofmu

Release 0.1.0

Afonso Cerejeira

May 08, 2021

USER GUIDE

1	Installation	3
1.1	Compilers	3
2	Usage	5
3	Contributing	7
4	License	9
5	Table of contents	11
5.1	Installation	11
5.2	Usage	11
5.3	License	12
5.4	API Reference	12
5.5	Command line reference	15
6	Indices and tables	17
	Python Module Index	19
	Index	21

Automatic FMU approximation tool.

INSTALLATION

1.1 Compilers

To correctly build an FMU this program needs to compile the generated C source into a shared library, therefore it requires the installation of C compilers.

If you are using the provided `docker` image to run the program then you are already able to cross compile the generated FMU to `linux32`, `linux64`, `win32` and `win64` platforms.

Otherwise if you are using a Linux distribution, you probably already have `gcc` installed, so you should be able to compile FMUs for your system. If you want to share the generated FMU it is advisable to also install a cross compiler to produce the binaries for Windows platforms (like [MinGW](#)). Below are the instructions to install with `apt` and `dnf`:

Debian/Ubuntu:

```
sudo apt install gcc-x86-64-linux-gnu gcc-i686-linux-gnu gcc-mingw-w64 gcc-mingw-w64-  
↳ i686
```

Fedora:

```
sudo dnf install gcc-x86_64-linux-gnu mingw64-gcc mingw32-gcc
```


USAGE

autofmu process a dataset

```
autofmu "dataset.csv" --inputs "x" "y" --outputs "z" -o "My Awesome Model.fmu"
```

This will read the `dataset.csv` file, select the `x`, `y` and `z` columns and find an approximation of the relation between the inputs and the outputs. Based on this relation, the sources files for the FMU will be generated and compiled, resulting in the `My Awesome Model.fmu` file ready to be used for simulations.

CONTRIBUTING

CHAPTER
FOUR

LICENSE

This project is licensed under MIT license.

TABLE OF CONTENTS

5.1 Installation

5.1.1 Compilers

To correctly build an FMU this program needs to compile the generated C source into a shared library, therefore it requires the installation of C compilers.

If you are using the provided `docker` image to run the program then you are already able to cross compile the generated FMU to `linux32`, `linux64`, `win32` and `win64` platforms.

Otherwise if you are using a Linux distribution, you probably already have `gcc` installed, so you should be able to compile FMUs for your system. If you want to share the generated FMU it is advisable to also install a cross compiler to produce the binaries for Windows platforms (like [MinGW](#)). Below are the instructions to install with `apt` and `dnf`:

Debian/Ubuntu:

```
sudo apt install gcc-x86-64-linux-gnu gcc-i686-linux-gnu gcc-mingw-w64 gcc-mingw-w64-  
↳ i686
```

Fedora:

```
sudo dnf install gcc-x86_64-linux-gnu mingw64-gcc mingw32-gcc
```

5.2 Usage

`autofmu` process a dataset

```
autofmu "dataset.csv" --inputs "x" "y" --outputs "z" -o "My Awesome Model.fmu"
```

This will read the `dataset.csv` file, select the `x`, `y` and `z` columns and find an approximation of the relation between the inputs and the outputs. Based on this relation, the sources files for the FMU will be generated and compiled, resulting in the `My Awesome Model.fmu` file ready to be used for simulations.

5.3 License

MIT License

Copyright (c) 2020 Afonso Cerejeira

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

5.4 API Reference

- *autofmu*
 - *autofmu.main*
 - *autofmu.cli*
 - *autofmu.generator*
 - *autofmu.utils*

5.4.1 autofmu

Automatic FMU approximation tool.

autofmu.main

Main entry point for running the program from the command line.

`autofmu.main.main (args=None)`

Execute the program in a command line environment.

Parameters `args` (`Optional[Sequence[str]]`) – sequence of command line arguments

Return type `None`

autofmu.cli

Utilities for exposing a command line interface of the program.

`autofmu.cli.create_argument_parser()`

Create an argument parser object to process command line arguments.

Return type `ArgumentParser`

Returns An argument parser object

autofmu.generator

Utilities for generating valid Functional Mockup Units.

`autofmu.generator.generate_fmu(dataframe, model_name, inputs, outputs, outfile, strategy)`

Generate a valid FMU model.

Parameters

- **dataframe** (`DataFrame`) – dataframe that contains the data used for the approximation
- **model_name** (`str`) – name of the model as used in the modeling environment
- **inputs** (`Iterable[str]`) – variable input names
- **outputs** (`Iterable[str]`) – variable output names
- **outfile** (`Path`) – path to the file to write the FMU
- **strategy** (`str`) – strategy to use to find the approximation (e.g. “linear”)

Return type `None`

`autofmu.generator.generate_model_description(model_name, model_identifier, guid, inputs, outputs)`

Generate a valid FMI 2.0 model description XML document.

Parameters

- **model_name** (`str`) – name of the model as used in the modeling environment
- **model_identifier** (`str`) – short class name according to C syntax, for example, “A_B_C”
- **guid** (`str`) – globally unique identifier that identifies this model
- **inputs** (`Iterable[str]`) – variable input names
- **outputs** (`Iterable[str]`) – variable output names

Return type `ElementTree`

Returns Valid FMI 2.0 model description XML document

`autofmu.generator.generate_model_source(guid, inputs, outputs, strategy, result)`

Generate a valid FMI 2.0 C source code implementation.

Parameters

- **guid** (`str`) – globally unique identifier that identifies this model
- **inputs** (`Iterable[str]`) – variable input names
- **outputs** (`Iterable[str]`) – variable output names

- **result** (`Union[LinearRegressionResult, LogisticRegressionResult]`)
– a result from an approximation calculation

Return type `str`

Returns Valid C source code that implements the FMI

autofmu.utils

General utilities.

`autofmu.utils.compile_fmu(model_identifier, fmu_path)`

Compile the C sources files of an FMU.

Extracts the FMU into a temporary directory, calling cmake to build the FMU, copying the generated library back into the FMU file. If `MinGW` is installed, it also cross compiles the FMU for Linux and Windows.

Parameters

- **model_identifier** (`str`) – short class name according to C syntax, for example, “A_B_C”
- **fmu_path** (`Path`) – path to the FMU file

Return type `None`

`autofmu.utils.run_cmake(source_dir, build_dir, variables=None)`

Run cmake command and build the targets.

Roughly equivalent to running the following two commands:

```
cmake -S source_dir -B build_dir
cmake --build build_dir
```

Parameters

- **source_dir** (`Path`) – path to source directory
- **build_dir** (`Path`) – path to build directory
- **variables** (`Optional[Mapping[str, str]]`) – a mapping between variable names and their values, e.g. `{"CMAKE_PROJECT_NAME": "Unicorn"}` would be passed as `DCMAKE_PROJECT_NAME=Unicorn` in the command line

Return type `None`

`autofmu.utils.slugify(value, allow_unicode=False)`

Convert a string to a URL slug.

Convert to ASCII if ‘allow_unicode’ is False. Convert spaces or repeated dashes to single dashes. Remove characters that aren’t alphanumerics, underscores, or hyphens. Convert to lowercase. Also strip leading and trailing whitespace, dashes, and underscores.

See: <https://docs.djangoproject.com/en/3.1/ref/utils/#django.utils.text.slugify>

Return type `str`

5.5 Command line reference

```
usage: autofmu [-h] [-o FILE] [-v] [-V] --inputs VARIABLE [VARIABLE ...]
              --outputs VARIABLE [VARIABLE ...] [-s {linear,logistic}]
              FILE
```

5.5.1 Positional Arguments

FILE CSV files that contain the datasets for training the FMU model

5.5.2 Named Arguments

-o, --outfile file to output the generated FMU model (default 'model.fmu')
Default: model.fmu

-v, --verbose run the program in verbose mode
Default: False

-V, --version show program's version number and exit

--inputs list of names of the model input variables

--outputs list of names of the model output variables

-s, --strategy Possible choices: linear, logistic
strategy to use to deduce the approximation
Default: "linear"

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

a

`autofmu`, [12](#)
`autofmu.cli`, [13](#)
`autofmu.generator`, [13](#)
`autofmu.main`, [12](#)
`autofmu.utils`, [14](#)

A

autofmu
 module, 12
autofmu.cli
 module, 13
autofmu.generator
 module, 13
autofmu.main
 module, 12
autofmu.utils
 module, 14

C

compile_fmu() (*in module autofmu.utils*), 14
create_argument_parser() (*in module autofmu.cli*), 13

G

generate_fmu() (*in module autofmu.generator*), 13
generate_model_description() (*in module autofmu.generator*), 13
generate_model_source() (*in module autofmu.generator*), 13

M

main() (*in module autofmu.main*), 12
module
 autofmu, 12
 autofmu.cli, 13
 autofmu.generator, 13
 autofmu.main, 12
 autofmu.utils, 14

R

run_cmake() (*in module autofmu.utils*), 14

S

slugify() (*in module autofmu.utils*), 14