# GeneralSimulationLibrary

***Release 1.0.0***

**Aug 14, 2020**

# Contents:

1 Introduction 3

2 Installation 5

3 Examples 7

4 How to use it 9

5 SimulationInterface class 13

6 Acknowledgements 15

7 Links and References 17

Index 19

**GenericSimulationLibrary** is a package encapsulates a methodology and tools for reproducible simulations. The main idea is to use python and/or jupyter notebooks to provide a lightweight and for-dummies easy "Simulation as a Service". The framework puts emphasis on simplicity: for the client to install and use, for the programmer to distribute and update, and for everyone to store and reproduce results. The framework can be personalized and extended for a specific simulation need.

# CHAPTER 1

# Introduction

## 1.1 Why do we care so much?

In 2008, we started the development of numerical implementation of generic reaction diffusion equations of catalysts immobilized on small porous particles. Python was a great choice back then, as it provided a language that was simple and high level, but still fast enough. Nevertheless, the python installation, versioning and library updates were messy and required constant supervision and side by side work. We longed for a simpler method. Fast travel to 2020, where you can now run python code on your browser on a server. You can now really collaborate with anyone on the planet and make sure the simulation is being run exactly as supposed. We've been reflecting on the key elements that are required on a simulation framework that provides this kind of "SaaS behavior". How could you make things as easy as possible for the other party while, as a developer, still been able to have full control on the code and guarantee the reproducibility of results?

## 1.2 Objective

The objective is to provide a working answer for the following constrains:

- Dealing with installation and versioning of python, jupyter and libraries.
- Simplifying the deployment and versioning of a specific piece of code (the custom simulation library).
- Exposing a simple interface to the final user to hied a (complex) numerical implementation.
- Allowing to store and share simulation results, so that they can be reproduced and analyzed.
- Allowing to use external computational resources, so that the simulation doesn't take a toll on your cheap notebook.

## 1.3 Solution

We think the objetive can be obtained with a code with the following characteristics:

- **pip-installable library**: this allows for a flexible approach. You can install nothing at all and run everything on google colab, or install it locally. But it's your call, and you are not forced to run everything on local (thanks free cloud providers!). This addresses for installation and versioning.
- **git-versioning** and **library versioning**: smash those bugs and document the code increments. This addresses reproducibility and versioning.
- **simple interface**: hide the complexity of the code with some OO to make it simple for the end user.
- **simulation seed**: the library should create a "simulation seed" that contains all the information (inputs, system and libraries configuration, options and outputs). This "simulation seed" can be stored and shared on itself, or together with a jupyter notebooks. This addresses the reproducibility.

## 1.4 Limitations

- Does not contains any Monthy Python jokes. If you have a good one, please send it!
- It's just a framework with simplistic inputs and simulation code. You **will** need to personalize the class SimulationInterface and all its methods.

# CHAPTER 2

# Installation

The repository for the code is hosted at https://github.com/sebastiandres/GenericSimulationLibrary.

The current implementation has been developed in Python 3.

To use it the simulation code, you should consider using the standard libraries numpy and matplotlib. , sphinx and sphinx-rtd-theme.

To use this project's code and personalize it, you should consider the libraries numpy and matplotlib for simulation, sphinx and sphinx-rtd-theme for documentation, and twine.

## 2.1 Install from pypi

You can install the library from pypi. This is good for testing the library, but will **not** allow you to personalize the code.

```
pip install GenericSimulationLibrary
```

## 2.2 Install from repository

You can install the library from github. This is good for testing the library, but will **not** allow you to personalize the code.

```
pip install git+https://github.com/sebastiandres/GenericSimulationLibrary.git
```

## 2.3 Clone repository and install

You can clone the library from github. This is good for testing the library **and** will allow you to personalize the code.

```
git clone https://github.com/sebastiandres/GenericSimulationLibrary.git
cd GenericSimulationLibrary
python setup.py install
```

## 2.4 Clone repository and install, but record installation files

Optionally, record the files installed, in case you want to remove them later:

```
python setup.py install --record installation_files.txt
```

To uninstall removing the files:

```
cat installation_files.txt | xargs sudo rm -rf
```

# CHAPTER 3

# Examples

## 3.1 Example in Google colab

Here is an executable example using Google Colab. Requires a google account (but it's worth it :).

## 3.2 Example in mybinder

Here is an executable example using MyBinder. Does not requires any account, but it will not store results.

## 3.3 Code example

To run it, you need to install the library.

To run and save a simulation:

```
from GenericSimulationLibrary import SimulationInterface

inputs = {
        "x_min":0,
        "x_max":10,
        "N_points":12,
        "m":3.0,
        "b":-2.0
}
plot_options = {
        "xlabel":"x [x_units]",
        "ylabel":"y [y_units]",
        "title":"My title",
        "data_x":[ 0.1, 2.1,  3.9,  6.1,  7.9,  9.9],
        "data_y":[-2.8, 3.6, 10.7, 13.6, 22.8, 27.1],  # -2 + 3*x + error
```

(continues on next page)

(continued from previous page)

```
        "data_kwargs": {'label':'exp', 'color':'red',
                        'marker':'s', 'markersize':6,
                        'linestyle':'none','linewidth':2,
        },
        "sim_kwargs": {'label':'sim', 'color':'black',
                        'marker':'o', 'markersize':6,
                        'linestyle':'dashed','linewidth':2,
        },
}
filepath = "/test.sim"
print("Sim file:", filepath)

SI = SimulationInterface()
SI.new(inputs, plot_options)
SI.simulate()
SI.save(filepath)
SI.status()
del SI
```

To load a simulation and plot the results:

```
from GenericSimulationLibrary import SimulationInterface
SI_2 = SimulationInterface()
SI_2.load(filepath)
SI_2.plot(filename="test.png")
SI_2.status()
```

As you can see, all you need is to define the inputs and plot options, and run the simulation. Libraries and outputs are silently handled. Saving, plotting or exporting the results is trivially easy for the user.

# CHAPTER 4

# How to use it

*Ok, I'm in. What should I do?*

## 4.1 Considerations

Through this explanation, let's asume you have the username *my-username*, that you will rename the project name from *GenericSimulationLibrary* to *MyProject* and that you will rename the class *SimulationInterface* to *MyInterface*.

## 4.2 Getting the code and versioning

You should start by getting a copy of the repo to play around: https://github.com/sebastiandres/GenericSimulationLibrary.

There are several ways to do this.

The first option: if you have a github account, you can fork the project to your github account. You can then clone **your** project to your local environment to start making changes:

```
git clone https://github.com/PUT-YOUR-USERNAME-HERE/GenericSimulationLibrary.git
```

The second option is to download the zip file of the repository from the repository github's webpage. This allows to start with no previous commits. But now you need to add the project to your github account.

In both cases, you'll have full control to version the project and push your changes. You should end up with a project stored at *https://github.com/my-username/MyProject*

Test it by making a small change of the README.md, making a commit and pushing it. If you don't know how to do any of those, look at a tutorial like https://try.github.io/.

It may look as a overkill to do **git-versioning** and **library versioning** for your small project. **It is not**. Learn the tools and you will save an HUGE amount of time and frustration. Smash those bugs and document the code increments.

## 4.3 Personalize the Simulation Code

The provided code already takes into account a **simple interface** and the creation of a **simulation seed**. Henceforth, code complexity is been hiden from the end user and a “simulation seed” can store all the information (inputs, system and libraries configuration, options and outputs) to guarantte reproducibility.

To personalize the code, you need to replace the mentions to **GenericSimulationLibrary** and **SimulationInterface** with the choosen names, in this example, MyProject and MyInterface. You should edit the files at *GenericSimulationLibrary/GenericSimulationLibrary/* (in your project shoul be at *MyProject/GenericSimulationLibrary/*), in particular *simulation_interface.py* and *__init__.py*. You can add more files if needed.

The code version is centralized and stored at *MyProject/GenericSimulationLibrary/version.py*. Follow some updating rule.

## 4.4 Personalize the Documentation

The documentation is stored at *GenericSimulationLibrary/docs/source/* (in your project shoul be at *MyProject/docs/source*) You should personalize all the rst files. The files should give you some pointers on how to use, but you can consult the rst specification

To rebuilt the documentation, execute the following at the path *GenericSimulationLibrary/docs/* (or *MyProject/docs/* at your project):

```
make clean
make html
```

You can check how the documentation turn out at *GenericSimulationLibrary/docs/build/* (or *MyProject/docs/build* at your project), the main file being *index.html*.

You can now go to read-the-docs and import your library documentation to make it public.

## 4.5 Distribution

The project is a **pip-installable library**. This has been taken care in the project structure and the file *setup.py*. You should edit *setup.py* and own it: make the changes of library name, version, author, packages, repository url, licence and description.

The project can be distributed from day one from the github repository.

If you have a stable version, that you would like to distribute through pypi. There’s a nice pypi tutorial you can follow.

You need to have installed twine (*pip intall twine*), and to have accounts at pypi and test.pypi. Note that that they required different accounts.

First, at the main folder, test the distribution at testpypi:

```
python setup.py sdist bdist_wheel
python -m twine upload --repository testpypi dist/*
```

There’s a Makefile, so you can also just do *make test.pypi* at *GenericSimulationLibrary/* (in your project shoul be at *MyProject/*)

You can check how everything looks at https://test.pypi.org/. If everything is looking good, upload it to (real) pypi:

```
python setup.py sdist bdist_wheel
python -m twine upload --repository pypi dist/*
```

Using the Makefile, you can just do *make pypi* at *GenericSimulationLibrary/* (in your project shoul be at *MyProject/*)

CHAPTER 5

# SimulationInterface class

This is the proposed class and its methods, that you should personalize to your needs.

**class** simulation_interface.**SimulationInterface**

Bases: `object`

GenericSimulationLibrary is a package encapsulates a methodology and tools for reproducible simulations. The main idea is to use python and/or jupyter notebooks to provide a lightweight and for-dummies easy "Simulation as a Service". The framework puts emphasis on simplicity: for the client to install and use, for the programmer to distribute and update, and for everyone to store and reproduce results. The framework can be personalized and extended for a specific simulation need. Link: https://generalsimulationlibrary.readthedocs.io/

**download**(*filename*)

Utility to download file, using colab

**export_xlsx**(*filename*)

Creates an excel file and saves the plot data and simulation data. It helps providing a file format that final users might be more familiar with.

**Parameters** **filename** (*string*) – Name for the file.

**interactive_plot**()

[summary]

**load**(*filename*)

Loads a simulation from a simulation file generated with the *save* method to restore the simulation.

**Parameters** **filename** (*string*) – Name for the simulation file.

**new**(*inputs*, *plot_options=None*)

Associates inputs and plot options to the simulation.

**Parameters**

- **inputs** (*dict*) – The inputs that will be used in the simulation. This can be completely personalized.
- **plot_options** (*dict, optional*) – The plot options, defaults to None

**plot** (*filename=”*, *display=True*)

Conditionally imports the matplotlib library, and if possible, plots the experimental data given in plot_options, and the simulation data.

**Parameters**

- **filename** (*str, optional*) – Filename to save the graph. If not provided, figure is not saved. Defaults to ‘’.
- **display** (*bool, optional*) – Boolean to show (True) or not show (False) the graph. Defaults to False

**save** (*filename*)

Saves the current state of the simulation, with all the provided information. The created file can be used with the *load* method to restore the simulation.

**Parameters** **filename** (*string*) – Name for the simulation file.

**simulate** ()

Conditionally imports the numpy library.

**status** ()

Prints out the detected configuration: environment, python and library versions.

# CHAPTER 6

# Acknowledgements

We stand on the shoulder of giants.

We would like to thank:

- The python ecosystem: python and a large list of libraries (numpy, scipy, matplotlib, among many others).
- The Jupyter Notebooks: the game changer in using python interactive and documentable.
- Colab and Mybinder: for making easy to share and run jupyter notebooks.
- Read the docs: for making very easy to make and share good documentation.

# CHAPTER 7

# Links and References

## 7.1 Documentation

- reStructuredText (rst) Markup Specification: https://docutils.sourceforge.io/docs/ref/rst/restructuredtext.html
- Documentation with sphinx and read the Docs: https://sphinx-rtd-tutorial.readthedocs.io/en/latest/

## 7.2 Versioning

- Github Tutorial: https://try.github.io/.

## 7.3 Distribution

- How to pack a library for distribution with pypi: https://packaging.python.org/tutorials/packaging-projects/

## 7.4 Generic Simulation Library Repositories

- Code: the framework, a work in progress: https://github.com/sebastiandres/GenericSimulationLibrary

## 7.5 Reproducibility

- Post: Lorena Barba's group reproducibility syllabus: https://lorenabarba.com/blog/barbagroup-reproducibility-syllabus

# Index

## D

download() (*simulation_interface.SimulationInterface method*), 13

## E

export_xlsx() (*simulation_interface.SimulationInterface method*), 13

## I

interactive_plot() (*simulation_interface.SimulationInterface method*), 13

## L

load() (*simulation_interface.SimulationInterface method*), 13

## N

new() (*simulation_interface.SimulationInterface method*), 13

## P

plot() (*simulation_interface.SimulationInterface method*), 13

## S

save() (*simulation_interface.SimulationInterface method*), 14

simulate() (*simulation_interface.SimulationInterface method*), 14

SimulationInterface (*class in simulation_interface*), 13

status() (*simulation_interface.SimulationInterface method*), 14