RWTH Aachen

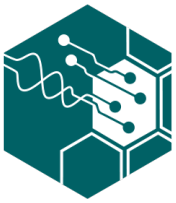
Master Thesis

GreenSloth: A Curated Web Resource for Validating and Comparing Peer-Reviewed Computational Models of Photosynthesis

Author:
Elouën Corvest

First Supervisor:
Prof. Anna Matuszyńska
Second Supervisor:
Prof. Oliver Ebenhöh

January 29, 2026



Junior Professorship
Computational
Life Science

RWTH AACHEN
UNIVERSITY

Acknowledgements

Abstract

Contents

List of Code-Blocks

1 Introduction

2 Material and Methods

2.1 Model Packaging

2.2 Model Demonstrations

2.2.1 Daylight Simulation

2.2.2 FvCB Addon

2.2.3 Standard PAM Simulation

2.2.4 MCA of Photosynthesis

2.2.5 Fitting of Flourescence and NPQ

2.3 GreenSlothUtils

The **GreenSlothUtils** package, written in Python, contains several utility functions to assist in packaging an already written kinetic Ordinary Differential Equation (ODE) model into a more code-readable and ready for uploading to the **GreenSloth** website format. The functions included in this package vary in their complexity, from simple directory creation to rewriting model files. While **GreenSlothUtils** has been written in a packge format, there is no intention to upload it to package sharing services such as PyPI. However, the package is intended to be used as a local package to assist in contribution to the **GreenSloth** website, therefore the GitHub repository is publically available at `https://github.com/ElouenCorvest/GreenSlothUtils.git`.

2.3.1 Documentation

All the functions included in the package are documented in the GitHub repository, however some key functions are also documented below to showcase their utility in this project. This biggest upside of this package is the ability to quickly create a new model directory with all the necessary files and format for uploading to **GreenSloth**. For ease of access, the functions required for this are combined inside a Command-line Interface (CLI).

Easily the most important part of the CLI of **GreenSlothUtils** is the `--help` command (see Code-Block 1), which gives an overview of all the possible commands and options available in the package. Each of the other commands also include a `--help` option to give more specific information about the specific command.

Using these commands following a specific order allows for a quick and easy creation of a new model directory.

1. Create the directory

First, the `initialize` command is used to create a new and complete model directory with the correct name and structure (see Code-Block 2). One required argument is the name of the model, of which the directory should be named after. In the **GreenSloth** ecosystem, the name of the model is the first author's last name followed by the year of publication, e.g., `Corvest2000`. This command also includes one optional argument, `--path`, which allows the user to specify where the new model directory should be created. If no path is given, the directory is created in the current working directory. The directory created contains various different sub-directories and files, all preformatted to the **GreenSloth** website standards (see Directory Tree 1).

```
Corvest2000/
├── figures/
│   ├── demonstrations.ipynb
│   └── paper_figures.ipynb
├── model/
│   ├── basic_funcs.py
│   ├── derived_quantities.py
│   ├── __init__.py
│   └── rates.py
├── model_info/
│   ├── comps.csv
│   ├── derived_comps.csv
│   ├── derived_params.csv
│   ├── params.csv
│   └── rates.csv
└── README_script.py
```

Directory Tree 1: Example directory structure created by the `initialize` command of GreenSlothUtils.

The `figures` directory consists of two Jupyter Notebooks. The `demonstrations.ipynb` includes all Model Demonstrations for the model, already written out, with each demonstration having its own cell. The creator of the model simply has to replace the `None`

```
$ GreenSloth-init --help

Usage: GreenSloth-init [OPTIONS] COMMAND [ARGS]...

Options:
  --help  Show this message and exit.

Commands:
  compare-gloss-to-model    Compares glosses to model
  from-model-to-gloss       Generate temporary Glosses from model info.
  initialize                Create '<model-name>' directory.
  latex-from-model          Write LaTex from Model
  python-from-gloss         Write Python Variables from Glossaries
  update-glosses-from-main  Update glosses from main
```

Code-Block 1: The help command of the CLI of GreenSlothUtils

```
$ GreenSloth-init initialize --help

Usage: GreenSloth-init initialize [OPTIONS] <model-name>

  Create '<model-name>' directory.

Options:
  -p, --path TEXT  Path to create model directory. Defaults to path here.
  --help           Show this message and exit.
```

Code-Block 2: The initialize command of the CLI of GreenSlothUtils

values of each string representation of model parts needed for the demonstration. For more information on what the demonstrations entail, please refer to Section 2.2. The `paper_figures.ipynb` is a blank notebook intended to be used to recreate the figures of the model's publication. Both notebooks already include necessary imports and helper functions to quickly get started. If the model is correctly implemented inside the `model` directory, the importing of the model should work.

The `model` directory consists of the actual ODE model implementation in Python. For ease of use, the **GreenSloth** ecosystem sets using the `mxlpy` [1] package as a standard for writing kinetic ODE models in Python. This has been chosen due to its ease of use, speed, and continuous support from the maintainers. The most important file of the `model` directory is the `__init__.py` file, which contains the main model initialization. As a strong recommendation, the actual parts of the model, such as the rates, basic functions, and derived quantities, should be written in separate files, as shown in Directory Tree 1. This allows for better code readability and easier debugging. To help with formatting a model written using `mxlpy`, the **GreenSlothUtils** package contains a subpackage, `greenslothutils.mxlpy_formatter`, which contains several functions to assist in rewriting a model into the correct format for **GreenSloth**.

The `model_info` directory contains all the necessary information about the model in a `.csv` format. This includes the variables, parameters, derived variables and parameters, and rates of the model. Each of these files follow a specific format, where a short description, the mathematical depiction in the publication and in **GreenSloth**, and the variable used in Python code. Additionally, the variables and rates also include a column of Kyoto Encyclopedia of Genes and Genomes (KEGG) IDs and Glossary Indices to assist in linking the model to existing databases and the **GreenSloth** overarching glossary. The parameters, on the other hand, also include a column for units, values and sources given in the publication. These `.csv` files

are used to automatically generate the glossary entries of the model when uploading to the **GreenSloth** website, therefore it is important that they are correctly filled out. To assist in this process, functions have been created, that extract the valuable information from an existing model implementation in `mxlpy` into the tables, already separating them into the correct columns. But more on those functions later.

Finally, the `README_script.py` file is a template script intended to be used to write the model's README file for the **GreenSloth** website and in general the model's documentation. This script includes several prefilled sections, such as tables extracted from the prior mentioned `model_info` directory, a generic installation guide, and the assumptions and brief description of the model demonstrations. The user simply has to fill in the specific details of the model, such as a short summary, LaTeX version of the equations used, the recreation of the publication figures, and notes to the demonstrations. Writing the README in this Python script format allows for easier accessing of repeating information, such as specific variables or parameters, without having to manually find them and replace them in a written Markdown file. Make a few changes in the script, run it, and the README file is ready to go.

2. Extract Information from the Model

Once the directory is created and the model is completely implemented, the next step is to extract necessary information from the model into the `model_info` tables. This is done using several commands of the **GreenSlothUtils** CLI.

The `from-model-to-gloss` function extracts all the necessary information from an `mxlpy` model into temporary Glossaries as a `.csv` format (see Code-Block 3). The options `--model-dir`, `--modelinfo-dir`, and `--modelgloss-dir` allow the user to specify where the model is located, where the model info tables are located, and where to store the generated Glosses, respectively. If no paths are given, the function assumes that the model directory is in the current working directory, the model info directory is in the model directory under `model_info/`, and the generated Glosses should be stored in a new directory called `model_glosses/` inside the `model_info/` directory. The option `--extract-option` allows the user to specify which parts of the model should be extracted. The possibilities are `all`, `variables`, `parameters`, `derived_variables`, `derived_parameters`, and `reactions`. By default, all parts are extracted. Finally, the `--check` option allows the user to check for inconsistencies between the generated Glossaries and the existing model info tables using the `compare-gloss-to-model` command. It is recommended to point the terminal inside the overarching directory of the model and run the command with the default values. This is the easiest way to ensure that all paths are correctly set and still follow the custom directory structure of the **GreenSloth** Directory Tree 1. With the automatic extraction of the model information filling out the actual `model_info` tables is made significantly easier. The user can then be on the safe side, that no variables, parameters, or rates have been forgotten. However, only the python variable is extracted and the other columns still have to be filled out manually by the user.

To create a bridge between the different models, overarching glossaries for the variables and rates have been created in the **GreenSloth** ecosystem. With access to these glossaries, only the ID associated to that variable or rate has to be included in the model info tables, and the rest of the information is automatically filled out using the `update-glosses-from-main` command (see Code-Block 4). This command updates the existing model info tables with information from the main glossaries, such as the description, KEGG ID, and mathematical depiction in **GreenSloth**. The options `--model-dir`, `--modelinfo-dir`, and `--maingloss-dir` allow the user to specify where the model is located, where the model info tables are located, and where the main glossaries are located, respectively. If no paths are given, the function assumes that the model directory is in the current working directory, the model info directory is in the model directory under `model_info/`, and the main glossaries are in the parent directory of the current working di-

```
$ GreenSloth-init from-model-to-gloss --help

Usage: GreenSloth-init from-model-to-gloss [OPTIONS]

  Generate temporary Glosses from model info.

Options:
  -md, --model-dir TEXT         Path to model directory. Defaults to path here
  -mid, --modelinfo-dir TEXT    Path to model info directory. Defaults to
                                model-dir + 'model_info'
  -mgd, --modelgloss-dir TEXT   Path to where to store csvs. Defaults to model-
                                dir + 'model_info/model_glosses/'
  -eo, --extract-option TEXT    Parts of the model to extract. Possibilities:
                                   'all',
                                   'variables',
                                   'parameters',
                                   'derived_variables',
                                   'derived_parameters',
                                   'reactions',
                                   [default: 'all']
  --check / --no-check          Check for inconsistencies with
                                'compare_gloss_to_model'
  --help                        Show this message and exit.
```

Code-Block 3: The from-model-to-gloss command of the CLI of GreenSlothUtils

rectory. The `--add` option allows the user to add new entries to the main glossary if they do not already exist. By default, this option is set to `False`. Again, it is recommended to point the terminal inside the overarching directory of the model and run the command with the default values. This is the easiest way to ensure that all paths are correctly set and still follow the custom directory structure of the **GreenSloth** Directory Tree 1. If the model includes variables and rates that are not in the main glossaries, this command can also add them automatically, subsequently adding a new ID. While this is a very handy feature, it is recommended to critically evaluate the new entries before running `update-glosses-from-main` with the `--add` option, to ensure that no duplicate or incorrect entries are added to the main glossaries.

Once the information has been added to the model info tables, the LaTeX depiction of each piece of the model will act as the main liaison between the info tables and the subsequent model documentation, as these depictions will not change. The rest of the information may always need to be changed due to updates to the naming convention or other problems.

3. Correct the Model

After updating the model info tables from the main glossaries or deciding on a different name for something, it is vital to ensure the `mxlpy` model implementation still holds the same information. For that, **GreenSlothUtils** includes the `compare-gloss-to-model`, which is already implemented in the `from-model-to-gloss` command as the `--check` option (see Code-Block 3). This command compares the existing model info tables to the actual `mxlpy` model implementation, checking for inconsistencies in variable names, parameter names, and rate names. If any inconsistencies are found, they are printed in the terminal, allowing the user to quickly find and correct them in the model implementation. This step is crucial to ensure that the model implementation and the model info tables are in sync, as correct documentation is a key aspect of the **GreenSloth** ecosystem. It has to be noted, that the category of derived variables and parameters is separated by using the logic of `mxlpy`, meaning that if something is derived from at least one variable it is classified as a derived variable, and something that is

```
$ GreenSloth-init update-glosses-from-main --help

Usage: GreenSloth-init update-glosses-from-main [OPTIONS]

  Update glosses from main

Options:
  -magd, --maingloss-dir TEXT  Path to directory with main gloss. Defaults to
                               parent of here.
  -md, --model-dir TEXT        Path to model directory. Defaults to path here
  -mid, --modelinfo-dir TEXT   Path to model info directory. Defaults to
                               model-dir + 'model_info'
  --add / --no-add             Add new entries to main gloss. Defaults to
                               False.
  --help                       Show this message and exit.
```

Code-Block 4: The update-glosses-from-main command of the CLI of GreenSlothUtils

only derived from parameters is a derived parameter. Therefore, if the user manually wishes to change the category of a derived quantity, they may do so in the model info tables, but have to remember, that the `compare-gloss-to-model` command will not check for inconsistencies in this regard.

4. Create Pointer to Python Variables

Once the model info tables and the model implementation are done, the documentation of the model can be tackled. One big feature that is important to **GreenSloth** is consistency. To ensure that all the variables, parameters, and rates mentioned in the documentation are consistent along the entire documentation file, a pointer in the `README_script.py` file can be created. To do this easily, the `python-from-gloss` command of **GreenSlothUtils** can be used (see Code-Block 5). This command creates Python variable assignments for all the variables, parameters, derived quantities, and rates in the model info tables, and writes them in separated `.txt` files. On top of that, these files are also managed as a log and shows the last update done with `python-from-gloss`. The options `--model-dir` and `--modelinfo-dir` allow the user to specify where the model is located and where the model info tables are located, respectively. If no paths are given, the function assumes that the model directory is in the current working directory and the model info directory is in the `model_info/` directory. The option `--glosstopython-dir` allows the user to specify where to store the generated Python variables. By default, this is set to the `python_written/gloss_to_python` directory in the path given to `--modelinfo-dir`. Once again, it is recommended to point the terminal inside the overarching directory of the model and run the command with the default values. This is the easiest way to ensure that all paths are correctly set and still follow the custom directory structure of the **GreenSloth** Directory Tree 1.

Once this command is run, the user can simply copy over the generated Python variable assignments into the `README_script.py` file, creating a direct pointer to the actual variables used in the model info tables. This ensures that any changes made to the variable names in the model info tables are automatically reflected in the documentation, maintaining consistency throughout. This last step of manual copying is to ensure that the user critically evaluates the changes they have made and with the log feature, shows the last update.

5. Generate LaTeX equations

A big part of kinetic ODE model documentation is the correct depiction of the equations used in the implementation. To assist in this process, the `latex-from-model` command of **GreenSlothUtils** can be used to automatically generate LaTeX equations from the `mxlpy` model implementation (see Code-Block 6). This command extracts the mathemat-

```
$ GreenSloth-init python-from-gloss --help

Usage: GreenSloth-init python-from-gloss [OPTIONS]

  Write Python Variables from Glossaries

Options:
  -md, --model-dir TEXT           Path to model directory. Defaults to path
                                  here
  -mid, --modelinfo-dir TEXT      Path to model info directory. Defaults to
                                  -md + 'model_info'
  -gpd, --glosstopython-dir TEXT  Path to gloss to python directory. Defaults
                                  to -mid + 'python_written/gloss_to_python'
  --help                          Show this message and exit.
```

Code-Block 5: The python-from-gloss command of the CLI of GreenSlothUtils

ical depictions of the variables, parameters, derived quantities, and rates from the `mxlpy` model and writes them in a `.txt` file. The options `--model-dir` and `--modelinfo-dir` allow the user to specify where the model is located and where the model info tables are located, respectively. If no paths are given, the function assumes that the model directory is in the current working directory and the model info directory is the `model_info/` directory. The option `-- modeltolatex - dir` allows the user to specify where to store the generated LaTeX equations. By default, this is set to the `python_written / model_to_latex` directory in the path given to `--modelinfo-dir`. Once again, it is recommended to point the terminal inside the overarching directory of the model and run the command with the default values. This is the easiest way to ensure that all paths are correctly set and still follow the custom directory structure of the **GreenSloth** Directory Tree 1. To generate the LaTeX equations, **GreenSlothUtils** uses the `latexify` package [2], which is specifically designed to extract LaTeX equations from Python functions. However, sometimes the python functions are too complex for `latexify` to handle, therefore it is recommended to critically evaluate the generated LaTeX equations before using them in the documentation. Once generated, the user can simply copy over the LaTeX equations into the correct sections of the `README_script.py` file, taking note, that the equations already include the python variable names used in the pointer created in the previous step.

6. Recreate the Publication figures

To validate the recreation of the `mxlpy` model implementation, it is important to recreate the figures of the original publication. Sadly, this step cannot be automated by **GreenSlothUtils**, as the figures vary significantly between different models. However, the `paper_figures.ipynb` notebook included in the `figures/` directory of the model structure (see Directory Tree 1) is intended to be used for this purpose. The user simply has to fill in the necessary code to recreate the figures, using the implemented `mxlpy` model. It is recommended to create a `str` dictionary at the top of the Notebook, that consists of the string representations of the different parts of the model, needed for the simulations. This allows for easier changing of variables and parameters, without having to search through the entire Notebook for the correct strings. Once the figures are recreated, a template in the `README_script.py` file is already included to integrate the generated figures into the documentation. Additionally, a brief description on how this figure was created and a short analysis on the recreation process is needed. A rule of thumb for the caption of a simulation figure, is to include as many details that are needed to recreate the simulation done in the figure, without having to refer back to the code. This rule of thumb is often missing in scientific publications, which is why some figures may be hard or even

```
$ GreenSloth-init latex-from-model --help

Usage: GreenSloth-init latex-from-model [OPTIONS]

  Write LaTex from Model

Options:
  -md, --model-dir TEXT          Path to model directory. Defaults to path
                                 here
  -mid, --modelinfo-dir TEXT     Path to model info directory. Defaults to -md
                                 + 'model_info'
  -mld, --modeltolatex-dir TEXT  Path to model to latex directory. Defaults to
                                 -mid + 'python_written/model_to_latex'
  --help                         Show this message and exit.
```

Code-Block 6: The latex-from-model command of the CLI of GreenSlothUtils

impossible to recreate. If that is the case in the model at hand, it is recommended to still include a brief note on what was missing to recreate the figure, to ensure full transparency. As this section is supposed to represent a comparison of the figures of the original publication and the `mxlpy` model implementation, showing the figures side by side would be ideal, however, due to possible copyright issues, it is best to only refer the reader to the publication for the original figures.

7. Create the Demonstrations

The last step in the documentation process is to fill out the `demonstrations.ipynb` Notebook included in the `figures/` directory of the model structure (see Directory Tree 1). This Notebook already includes all the necessary code to run the model demonstrations described in Section 2.2. The user simply has to fill in the string representations of the different parts of the model needed for each demonstration. Some demonstrations may need specific pre-work done beforehand, but more details can be found in Section 2.2 and the documentation of **GreenSlothUtils**. Once the demonstrations are filled out, a brief description of each demonstration is already included in the `README_script.py` file, where the user simply has to fill in a brief analyses of the demonstration, including how it looks and if something did not work, why.

8. Finishing touches

As the `README_script.py` is now filled out with all the necessary information, the last step is to run the script to generate the final `README.md` file for the model. It is recommended to critically evaluate the generated README file for any mistakes or inconsistencies, especially in the regard of any LaTeX representations, as errors in these are very prone to happen. Once the README file is finalised, the model is ready to be included on the **GreenSloth** website.

2.4 Website

2.5 Additional Methods

2.5.1 Usage of AI

To stay completely transparent during this thesis, it has to be noted that several Artificial Intelligence (AI) tools have been used to assist in the code writing, documentation, and text writing process. Several Large Language Models (LLMs) have been used, including Gemini (Google) to help with finding models, ideas and writing code snippets for visualisation. Additionally, GitHub Copilot has been used to assist in code writing, mainly to streamline the writing and debugging process. The exact parts of this thesis that have been assisted by AI tools cannot be exactly determined, as the suggestions are mainly used to inspire and speed up the writing process. However, every part of this thesis has been critically evaluated and edited by the author to ensure accuracy and quality.

3 Results

4 Discussion

References

[1] Marvin Van Aalst et al. *MxlPy - Python Package for Mechanistic Learning in Life Science*. 2025. DOI: 10.1101/2025.05.06.652335.

[2] *Latexify-Py: Generates LaTeX Math Description from Python Functions.*