

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

February 3, 2026



Junior Professorship
Computational
Life Science

RWTHAACHEN
UNIVERSITY

Acknowledgements

Abstract

Contents

Acknowledgements	i
Abstract	ii
List of Code-Blocks	vi
1 Introduction	1
2 Material and Methods	2
2.1 Models	2
2.1.1 Bellasio2019	2
2.1.2 Fuente2024	2
2.1.3 Li2021	3
2.1.4 Matuszynska2016	3
2.1.5 Saadat2021	3
2.2 Model Demonstrations	3
2.2.1 Daylight Simulation	3
2.2.2 FvCB Add-on	4
2.2.3 Standard PAM Simulation	4
2.2.4 MCA of Photosynthesis	5
2.2.5 Fitting of NPQ	5
2.3 GreenSlothUtils	6
2.3.1 Documentation	6
2.4 Website	13
2.5 Additional Methods	13
2.5.1 Usage of AI	13
3 Results	14
4 Discussion	15
References	vii

Acronyms

AI Artificial Intelligence.

ATP Adenosine Triphosphate.

CBB Calvin-Benson-Bassham.

CLI Command-line Interface.

Cytb₆f Cytochrome b₆f.

FvCB Farquhar, von Caemmerer, and Berry.

KEGG Kyoto Encyclopedia of Genes and Genomes.

LLM Large Language Model.

MCA Metabolic Control Analysis.

NADPH Nicotinamide Adenine Dinucleotide Phosphate.

NEON National Ecological Observatory Network.

NPQ Non-Photochemical Quenching.

ODE Ordinary Differential Equation.

PAM Pulse Amplitude Modulation.

PC_{ox} oxidised plastocyanin.

PETC Photosynthetic Electron Transport Chain.

PGA Phosphoglycerate.

PPFD Photosynthetic Photon Flux Density.

PQ_{ox} oxidised plastoquinone.

PSI Photosystem I.

PSII Photosystem II.

RuBisCO Ribulose-1,5-bisphosphate carboxylase-oxygenase.

RuBP Ribulose-1,5-bisphosphate.

List of Variables

CO_2 carbon dioxide.

A carbon assimilation.

C_i intercellular carbon dioxide (CO_2) concentration.

F_m maximal fluorescence.

F fluorescence.

R_{light} respiration in the light.

Γ^* CO_2 compensation point in the absence of non-photorespiratory CO_2 release.

v_{ATPSynth} Adenosine Triphosphate (ATP) synthase rate.

v_{PSII} Photosystem II (PSII) rate.

v_{PSI} Photosystem I (PSI) rate.

v_{RuBisCO} Ribulose-1,5-bisphosphate carboxylase-oxygenase (RuBisCO) carboxylation rate.

v_{b6f} Cytochrome b₆f (Cytb₆f) rate.

List of Code-Blocks

1	The help command of the CLI of GreenSlothUtils	7
2	The initialize command of the CLI of GreenSlothUtils	7
3	The from-model-to-gloss command of the CLI of GreenSlothUtils	9
4	The update-glosses-from-main command of the CLI of GreenSlothUtils	10
5	The python-from-gloss command of the CLI of GreenSlothUtils	12
6	The latex-from-model command of the CLI of GreenSlothUtils	12

1 Introduction

2 Material and Methods

2.1 Models

To make a good demonstration of the capabilities of **GreenSloth**, five kinetic Ordinary Differential Equation (ODE) models of photosynthesis have been chosen to take part in this thesis. These five models, while all showing photosynthesis, vary in their complexity and execution. Some are based on each other, while others stand alone. In the following, a brief description of each model is given. These short summaries are the ones used on the **GreenSloth** website.

2.1.1 Bellasio2019

The Bellasio2019 [1] model is a generalized C₃ leaf-photosynthesis model, that includes simplified representations of the light and dark reactions and a stomatal behavior submodule. A lot of its implementation is based on past work by the same author and is mainly inspired by the common Farquhar, von Caemmerer, and Berry (FvCB) model. The light reactions are modified from Yin et al. (2004) and include the potential rates of ATP and NADPH production based on light intensity. This model has been created with the simple user in mind, and the author has made an effort to show its simplicity, by giving access to a Microsoft Excel Workbook containing the entire model. To showcase the model's capabilities, the author creates common steady-state carbon assimilation curves, against intercellular CO₂ concentration and light intensity, and compares them to experimental data from the literature. As many models of photosynthesis rely on purely steady-state assumptions, this model is also validated in dynamic conditions, showing for example the response of the model to a fluctuation of ambient oxygen concentration.

This model was created to stay as simple as possible, while still being able to accurately represent the main features of C₃ photosynthesis. As such, it can be used as a base for more complex models, or as a starting block in larger models of plant physiology. While giving access to the entire model in an Excel Workbook format is transparent and great, the execution

of said practice has been inefficient in this instance. The entire mathematical description of the model is also given in the Appendix of the publication, however there are missing or different equations between the publication and the Excel Workbook, which can lead to confusion. On top of that, the simulation protocols used for each figure are only given in small details, which leads to further confusion when trying to reproduce the results and see which equations are correct or not.

2.1.2 Fuente2024

The Fuente2024 [2] model is a kinetic model of photosynthesis that is based on Occam's razor, aiming to provide the minimal complexity to describe the core processes of this model. In this case, the model focuses on the dynamic light oscillation and its responses on the photosynthetic machinery. It focuses only on the light-dependent reactions, including simplified versions of photosystem II, photosystem I, the Plastoquinone pool, and proton and ATP concentration in the lumen and stroma. On top of that, it shows the activation of non-photochemical quenching (NPQ), the dynamics of chlorophyll fluorescence, and the rate of oxygen evolution.

The model includes the oscillating light intensity as a sinusoidal function, where the amplitude and frequency are adjustable parameters. To allow for easier comparison to other models, that often see light intensity as a constant value, the oscillation is defined around a base light intensity. However, the strength of having light with a specific frequency lies in the additional information and therefore analysis possibilities that can be performed. In this case, the model is used to create Bode plots of the response of fluorescence to light oscillations and comparing these results to experimental data from *Chlamydomonas reinhardtii*.

This simple model stays true to its name and the authors aim to provide a base model that can be easily extended, while still showing a new approach to photosynthesis modelling. Their work shows that even with a simple model, new insights can be gained by using dynamic light protocols, which may have been

overlooked in traditional steady-state models. To further extend the usability of the model, the authors provide a detailed notebook written in the Wolfram language, which also shows how to recreate some of the publication's figures.

2.1.3 Li2021

2.1.4 Matuszynska2016

The Matuszynska2016 [3] model, a small kinetic model, was developed to delve deeper into the effect of light memory caused by non-photochemical quenching. The systematic investigation of the Xanthophyll cycle, a combination of the pigments of violaxanthin, antheraxanthin, and zeaxanthin, sparked a series of experiments to determine whether plant light memory can be detected in a timescale of minutes to hours through pulse amplitude modulated chlorophyll fluorescence. The model was then created based on these experimental results, providing a comprehensive description of Non-Photochemical Quenching (NPQ) dynamics and the short-term memory of the *Arabidopsis thaliana* plant.

To keep the model as simple as possible, several processes not directly linked to NPQ have been simplified to create a dynamic ODE system consisting only of 6 different compounds. With these simplifications, the authors could fulfil an additional goal: to make a general framework that is not specific to one model organism.

To demonstrate the adaptability of their model, the authors took their calibrated *A. thaliana* model and successfully applied it to the non-model organism *Epipremnum aureum*. This adaptation allowed them to simulate realistic fluorescence measurements and replicate all the key features of chlorophyll induction, showcasing the model's versatility and potential for use in a variety of organisms.

2.1.5 Saadat2021

The Saadat2021 [4] model builds upon previous models, particularly the Matuszynska2019 model, by incorporating and modifying various reactions and aspects

of photosynthesis. Overall, the model can be divided into three modules: the ascorbate-glutathione cycle, the Calvin-Benson-Bassham (CBB) cycle and thioredoxin reductase-regulated reactions, and the Photosynthetic Electron Transport Chain (PETC).

The model is primarily used to investigate the electron flows around PSI and their relevance to photosynthetic efficiency. Several different analyses have been conducted to validate the model in both steady-state and dynamic environment conditions. The most interesting is the direct comparison of a knockout mutant of the protein PGR5. This protein is known to catalyse the reduction of plastoquinone by ferredoxin. The results of this comparison align with experimental values, which are, however, not presented in the publication but are referenced. Additionally, it is noted that the results should not be interpreted as accurate quantitative data, but rather as a proof of concept for the model.

Overall, the model has one advantage over other photosynthesis models, as it also highlights the importance of other electron flows, not just the PETC. Additionally, not only are the authors open about the model's flaws, but they are also insistent on making their code and analyses available on GitHub. Therefore, this model serves as a good stepping stone for more complex models that aim to incorporate aspects of photosynthesis, which are often simplified in other models.

2.2 Model Demonstrations

2.2.1 Daylight Simulation

The Photosynthetic Photon Flux Density (PPFD) data in a 1-minute resolution used for the daylight simulation demonstration is taken from National Ecological Observatory Network (NEON) at the KONZ site, located in Kansas, USA (39.10077, -96.56307) [5], using the `neonutilities` package [6]. As only a day is shown in the simulation, only data from the 19th of June 2023 is taken into account. Using a solar calculator package, `pysunnoaa` [7], the sunrise and sunset times can be calculated using the geographical location and date to pinpoint the actual solar day of

the location. The filtered data is then used as an input for the PPFD parameter in the daylight simulation demonstration for every model. The protocol is constructed by simulating each point of PPFD data for 60 seconds. Three different outputs are shown in this demonstration: the RuBisCO carboxylation rate (v_{RuBisCO}), the ATP and Nicotinamide Adenine Dinucleotide Phosphate (NADPH) ratio, and the fluorescence (F) yield. If these outputs are not available in the model at hand, the plot is still displayed with the corresponding y-axis label, struck through.

2.2.2 FvCB Add-on

The basis of the FvCB add-on demonstration is based on the min- W variant. If the supplied model already has a carbon assimilation (A), then a generic FvCB simulation is done and compared to the results from von Caemmerer (2013) [8]. If there is no available instance of A , the two main and mandatory connection points for the add-on are the v_{RuBisCO} and a quantity that represents CO₂. If these two connection points are not available in the model, the simulation will not run, and only the general FvCB output will be shown.

The v_{RuBisCO} needs to be in a unit of $\mu\text{mol m}^{-2} \text{s}^{-1}$. To convert from a concentration per time unit, the space that the stroma occupies in the chloroplast based on the leaf volume needs to be used. The value taken for this add-on was taken from Laisk et al. (2006) [9], where it was estimated and set to 0.0112 L m^{-2} .

As the FvCB model requires a partial pressure of CO₂, specifically the intercellular CO₂ concentration (C_i), this quantity can also be given to the add-on if available in the model at hand. If it is not given, but the CO₂ is, a conversion using Henry's law is done to convert the concentration of CO₂ into a partial pressure. The default Henry's law constant used for this conversion is $3.4 \times 10^{-5} \text{ mM } \mu\text{bar}^{-1}$ [10], however, the law constant can also be supplied, when it is a part of the model.

With these two quantities given in the correct format, the FvCB add-on can be added to any kinetic ODE model of photosynthesis. To finally cre-

ate the representation of A , the add-on includes the CO₂ compensation point in the absence of non-photorespiratory CO₂ release (Γ^*) and respiration in the light (R_{light}), $38.6 \mu\text{bar}$ and $1 \mu\text{mol m}^{-2} \text{s}^{-1}$, respectively, if not given by the model. With all of these quantities included in the model, the A can be calculated (see Equation 1) and shown in the demonstration plot with a generic FvCB simulation using parameters taken from von Caemmerer (2013) [8].

$$A = v_{\text{RuBisCO}} \cdot \left(1 - \frac{\Gamma^*}{C_i}\right) - R_{\text{light}} \quad (1)$$

2.2.3 Standard PAM Simulation

The standard Pulse Amplitude Modulation (PAM) simulation demonstration uses a generic protocol that does not reflect actual experiments, but are in the same spirit as true experimental work. Prior to the actual protocol, the simulation of the model is done under dark conditions for 30 min. Then the actual protocol consists of 22 periods of a length of 2 min, which start with the specific light intensity of that period and ends with a saturating pulse of $3000 \mu\text{mol m}^{-2} \text{s}^{-1}$ for 0.8 s. The first two periods are in dark conditions, followed by 10 periods of actinic light of $1000 \mu\text{mol m}^{-2} \text{s}^{-1}$ and finishes with 10 periods of dark conditions again. The dark condition is set to $40 \mu\text{mol m}^{-2} \text{s}^{-1}$, as it has been found that many models are not capable of simulating actual zero light conditions.

With this protocol, the simulation of the model is run and the F and NPQ is shown in the demonstration plot. If F is not present in the model, the corresponding plot is still displayed with the y-axis label struck through. If NPQ is not available, but F is, it is calculated by using maximal fluorescence (F_m) (see Equation 2).

$$NPQ(t) = \frac{F_m(t=0) - F_m(t=t)}{F_m(t=t)} \quad (2)$$

F_m is extracted by using the simulated F values at the time points where the saturating pulses are applied. To be sure that the actual F_m is found, an interval between the two saturating pulses is taken,

and the maximum value in that interval is used as F_m . Most of the time this value is correct, but for full transparency, these values are plotted as triangles on the F plot, additionally to the NPQ.

2.2.4 MCA of Photosynthesis

The Metabolic Control Analysis (MCA) demonstration limits its analysis to variables and fluxes that are deemed integral parts of photosynthesis. The variables to be given should account for a representation of Phosphoglycerate (PGA), Ribulose-1,5-bisphosphate (RuBP), oxidised plastoquinone (PQ_{ox}), oxidised plastocyanin (PC_{ox}), ATP, and NADPH. The rates should include the $v_{RuBisCO}$, PSII rate (v_{PSII}), PSI rate (v_{PSI}), Cytb₆f rate (v_{b6f}), and ATP synthase rate ($v_{ATPSynth}$). The parameters analyzed are supposed to be the control coefficients of each prior mentioned rate, and therefore the MCA simulations are run to steady-state. Each parameter is displaced by $\pm 0.01\%$ and the results are put into two separate heatmaps, one for the variables and one for the fluxes. If any of the prior mentioned variables or rates are not available in the model, the corresponding row and column in the heatmaps are still displayed, but white and with less opacity.

2.2.5 Fitting of NPQ

The fitting demonstration uses experimental data taken from von Bismarck (2022) [11]. The data consists of measurements of F , F_m , and NPQ calculated (see Equation 2) under different light intensities, following a PAM protocol after a dark adaptation period of 5 min. The data was taken with Maxi Imaging-PAM (Walz, Germany) using Col-0 *A. thaliana* plants and as no details were given, the default settings of the machine are taken. That means, that each saturating pulse is $5000 \mu\text{mol m}^{-2} \text{s}^{-1}$ for 720 ms. Each period consists of a simulation of a specific light intensity and finishes with a saturating pulse, together lasting approximately 1 min. There are four different sections during this PAM protocol. Starting with one dark period with a light intensity of $0 \mu\text{mol m}^{-2} \text{s}^{-1}$, followed by 10 high actinic light periods of $903 \mu\text{mol m}^{-2} \text{s}^{-1}$

then dropping the actinic light to $90 \mu\text{mol m}^{-2} \text{s}^{-1}$, also for 10 periods, then again 5 dark periods to finish the protocol.

The protocol used for the simulation is based on the actual data provided. The difference of each timestamp recorded and the PPFD value at that time is taken. With this simulation protocol, the fitting procedure can be created. The parameters to be fitted are given to the fitting routine with only a bound of zero. If the default value of the parameter is lower than zero, then that bound will be considered the upper bound, else it is set as the lower bound. With each variation of the parameters, the simulation is run with the prior mentioned protocol. If the model includes NPQ, that output will be fitted to the experimental data. If NPQ is not available, but F is, then the F output will be used to calculate the NPQ using (see Equation 2) and then fitted to the experimental data. If neither are available, the fitting will not be done and the demonstration plot will only show the experimental data.

To actually fit the simulation output to the experimental data, the Levenberg-Marquardt method is used. This method is the most common and basic non-linear fitting algorithm, which is why it was chosen for this demonstration. On top of that, the standard scale method is implemented by default to help the fitting process. As the fitting is done using the `Model` instance from the package `lmfit` [12], the inclusion of standard scaling is done by setting the `weights` argument to the reciprocal fraction of the standard deviation of the experimental data and having both the NPQ results of the experimental data and simulation results be centered around the mean of the data (see Equation 3). As sometimes no results for the simulation occurs depending on the parameter set, a large penalty value is returned to the fitting routine to avoid these parameter sets.

After the best "possible" fit is found, according to the performed simulations, the results are plotted in the demonstration plot, showing both the experimental data points and the fitted simulation line. The F and NPQ are both shown separately for the experimental data and the best fit, and top of that a rela-

$$\begin{aligned}
\text{Standard Scale : } x_{\text{new}} &= \frac{x_{\text{old}} - \mu_{\text{data}}}{\sigma_{\text{data}}} \\
\text{lmfit Calculation : Residual} &= \text{weights} \cdot (x_{\text{data}} - x_{\text{model}}) \\
\text{Insert Standard Scale : Residual} &= \frac{1}{\sigma_{\text{data}}} \cdot ((x_{\text{data|old}} - \mu_{\text{data}}) - (x_{\text{sim|old}} - \mu_{\text{data}}))
\end{aligned} \tag{3}$$

tive difference plot between the best fit or the original model parameter set is shown against the experimental data. To showcase the changes made in the fitting, a table of the changed parameters and their relative change is also displayed in the demonstration plot. The choosing of the parameters to be fitted is left to the user, as different models may have different parameters that influence NPQ more than others.

2.3 GreenSlothUtils

The **GreenSlothUtils** package, written in Python, contains several utility functions to assist in packaging an already written kinetic 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 pre-formatted to the **GreenSloth** website standards (see Directory Tree 1).

```

1 $ GreenSloth-init --help
2
3 Usage: GreenSloth-init [OPTIONS] COMMAND [ARGS]...
4
5 Options:
6   --help  Show this message and exit.
7
8 Commands:
9   compare-gloss-to-model      Compares glosses to model
10  from-model-to-gloss         Generate temporary Glosses from model info.
11  initialize                  Create '<model-name>' directory.
12  latex-from-model            Write LaTeX from Model
13  python-from-gloss          Write Python Variables from Glossaries
14  update-glosses-from-main   Update glosses from main

```

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

```

1 $ GreenSloth-init initialize --help
2
3 Usage: GreenSloth-init initialize [OPTIONS] <model-name>
4
5 Create '<model-name>' directory.
6
7 Options:
8   -p, --path TEXT  Path to create model directory. Defaults to path here.
9   --help           Show this message and exit.

```

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

```

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

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` [13] 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 sep-

arate 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, L^AT_EX 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

```

1 $ GreenSloth-init from-model-to-gloss --help
2
3 Usage: GreenSloth-init from-model-to-gloss [OPTIONS]
4
5   Generate temporary Glosses from model info.
6
7 Options:
8   -md, --model-dir TEXT          Path to model directory. Defaults to path here
9   -mid, --modelinfo-dir TEXT     Path to model info directory. Defaults to
10                           model-dir + 'model_info'
11   -mgd, --modelgloss-dir TEXT   Path to where to store csvs. Defaults to model-
12                           dir + 'model_info/model_glosses/'
13   -eo, --extract-option TEXT    Parts of the model to extract. Possibilities:
14                           'all',
15                           'variables',
16                           'parameters',
17                           'derived_variables',
18                           'derived_parameters',
19                           'reactions',
20                           [default: 'all']
21   --check / --no-check          Check for inconsistencies with
22                           'compare_gloss_to_model'
23   --help                         Show this message and exit.

```

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

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 directory. The `--add` option allows the user to add new entries to the main glossary if they do not already ex-

ist. 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 L^AT_EX 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

```

1 $ GreenSloth-init update-glosses-from-main --help
2
3 Usage: GreenSloth-init update-glosses-from-main [OPTIONS]
4
5     Update glosses from main
6
7 Options:
8     -magd, --maingloss-dir TEXT      Path to directory with main gloss. Defaults to
9                               parent of here.
10    -md, --model-dir TEXT          Path to model directory. Defaults to path here
11    -mid, --modelinfo-dir TEXT     Path to model info directory. Defaults to
12                               model-dir + 'model_info'
13    --add / --no-add               Add new entries to main gloss. Defaults to
14                               False.
15    --help                         Show this message and exit.

```

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

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 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 implemen-

tation 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 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 L^AT_EX 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 L^AT_EX equations from the `mxlpy` model implementation (see Code-Block 6). This command extracts the mathematical 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 L^AT_EX 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 L^AT_EX equations, **GreenSlothUtils** uses the `latexify` package [14], which is specifically designed to extract L^AT_EX equa-

tions from Python functions. However, sometimes the python functions are too complex for `latexify` to handle, therefore it is recommended to critically evaluate the generated L^AT_EX equations before using them in the documentation. Once generated, the user can simply copy over the L^AT_EX 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 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 publi-

```

1 $ GreenSloth-init python-from-gloss --help
2
3 Usage: GreenSloth-init python-from-gloss [OPTIONS]
4
5     Write Python Variables from Glossaries
6
7 Options:
8     -md, --model-dir TEXT          Path to model directory. Defaults to path
9                      here
10    -mid, --modelinfo-dir TEXT    Path to model info directory. Defaults to
11                      -md + 'model_info'
12    -gpd, --glosstopython-dir TEXT Path to gloss to python directory. Defaults
13                      to -mid + 'python_written/gloss_to_python'
14    --help                         Show this message and exit.

```

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

```

1 $ GreenSloth-init latex-from-model --help
2
3 Usage: GreenSloth-init latex-from-model [OPTIONS]
4
5     Write LaTex from Model
6
7 Options:
8     -md, --model-dir TEXT          Path to model directory. Defaults to path
9                      here
10    -mid, --modelinfo-dir TEXT    Path to model info directory. Defaults to -md
11                      + 'model_info'
12    -mld, --modeltolatex-dir TEXT Path to model to latex directory. Defaults to
13                      -mid + 'python_written/model_to_latex'
14    --help                         Show this message and exit.

```

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

cation 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. Addition-

ally, 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] Chandra Bellasio. „A Generalised Dynamic Model of Leaf-Level C3 Photosynthesis Combining Light and Dark Reactions with Stomatal Behaviour“. In: *Photosynthesis Research* 141.1 (2019), pp. 99–118. ISSN: 0166-8595, 1573-5079. DOI: 10.1007/s11120-018-0601-1.
- [2] David Fuente et al. „A Mathematical Model to Simulate the Dynamics of Photosynthetic Light Reactions under Harmonically Oscillating Light“. In: *Plant Physiology and Biochemistry* 217 (2024), p. 109138. ISSN: 09819428. DOI: 10.1016/j.plaphy.2024.109138.
- [3] Anna Matuszyńska et al. „A Mathematical Model of Non-Photochemical Quenching to Study Short-Term Light Memory in Plants“. In: *Biochimica et Biophysica Acta (BBA) - Bioenergetics* 1857.12 (2016), pp. 1860–1869. ISSN: 00052728. DOI: 10.1016/j.bbabi.2016.09.003.
- [4] Nima P. Saadat et al. „Computational Analysis of Alternative Photosynthetic Electron Flows Linked With Oxidative Stress“. In: *Frontiers in Plant Science* 12 (2021), p. 750580. ISSN: 1664-462X. DOI: 10.3389/fpls.2021.750580.
- [5] National Ecological Observatory Network (NEON). *Photosynthetically Active Radiation (PAR) (DP1.00024.001): RELEASE-2023*. 2023. DOI: 10.48443/VZFH-7675.
- [6] Claire Lunch. *Neonutilities: A Package for Accessing and Wrangling Data Generated and Published by the National Ecological Observatory Network*.
- [7] Santosh Philip. *Pysunnoaa: Python Implementation of NOAA's Solar Position Calculators*.
- [8] Susanne Von Caemmerer. „Steady-state Models of Photosynthesis“. In: *Plant, Cell & Environment* 36.9 (2013), pp. 1617–1630. ISSN: 0140-7791, 1365-3040. DOI: 10.1111/pce.12098.
- [9] Agu Laisk, Hillar Eichelmann, and Vello Oja. „C3 Photosynthesis in Silico“. In: *Photosynthesis Research* 90.1 (2007), pp. 45–66. ISSN: 0166-8595, 1573-5079. DOI: 10.1007/s11120-006-9109-1.
- [10] Rolf Sander. „Compilation of Henry's Law Constants (Version 5.0.0) for Water as Solvent“. In: *Atmospheric Chemistry and Physics* 23.19 (2023), pp. 10901–12440. ISSN: 1680-7324. DOI: 10.5194/acp-23-10901-2023.
- [11] Thekla Von Bismarck et al. „Light Acclimation Interacts with Thylakoid Ion Transport to Govern the Dynamics of Photosynthesis in Arabidopsis“. In: *New Phytologist* 237.1 (2023), pp. 160–176. ISSN: 0028-646X, 1469-8137. DOI: 10.1111/nph.18534.
- [12] Matthew Newville et al. *LMFIT: Non-Linear Least-Squares Minimization and Curve-Fitting for Python*. Zenodo. 2025. DOI: 10.5281/ZENODO.598352.
- [13] Marvin Van Aalst et al. *MxlPy - Python Package for Mechanistic Learning in Life Science*. 2025. DOI: 10.1101/2025.05.06.652335.
- [14] *Latexify-Py: Generates LaTeX Math Description from Python Functions*.