
Subcellular workflow

Release 1.0

João Pedro Gomes dos Santos, Kadri Pajo, Daniel Trpevski, Andr

2021/07/23

CONTENTS

1	Sections of the workflow in external repositories	3
2	Implemented models	5
3	Compatibility	7
4	References	123

(Under construction - last updated 2021/07/23)

This workflow has been developed to tackle the challenge of building and analyzing biochemical pathway models, combining pre-existing tools and custom-made software. (Santos et al. 2020) (Preprint)

At the root of our implementation is the SBtab format (Lubitz et al. 2016), a file that can store biochemical models and associated data in an easily readable and expandable way.

We have also developed tools to convert the SBtab format into several formats that can be used in MATLAB®, NEURON, STEPS and COPASI.

Using MATLAB® we have developed custom scripts for parameter estimation, and global sensitivities analysis, as well as diagnostics tools that can be used for model development. The global sensitivity analysis algorithm is modified from Halmes et al 2009.

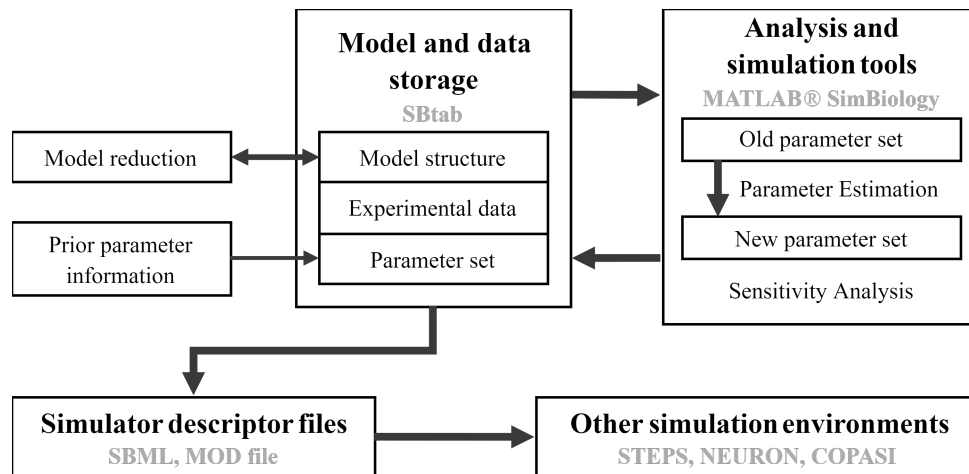
We demonstrate all these features using three example models, the main one being a modified version of the D1 MSN subcellular cascade model from Nair et al. 2016.

Code and files to run these models in different simulators:

- For MATLAB®
Matlab/; Model_Nair_2016/Matlab; Model_Fujita_2010/Matlab; Model_Viswan_2018/Matlab
- For Neuron
Model_Nair_2016; Model_Viswan_2018
- For the Subcellular application (STEPS)
Model_Nair_2016/BioNetGen and STEPS/; Model_Viswan_2018/BioNetGen and STEPS/
- Copasi
Model_Nair_2016; Model_Viswan_2018

Features:

- Wrapper for model simulation in MATLAB® (Matlab/Run_main.m)
- Analysis of selected parameter sets, using MATLAB® (Matlab/Run_main.m)
- Parameter optimization, using MATLAB® (Matlab/Run_main.m)
- Global Sensitivity analysis, using MATLAB® (Matlab/Run_main.m)
- Conversion tools:
 - SBtab (.xlsx,.xls) to SBtab (.tsv), using MATLAB® (Matlab/Run_main.m)
 - SBtab (.xlsx) to MATLAB® SimBiology® (.m, .sbproj), using MATLAB® (Matlab/Run_main.m)
 - MATLAB® SimBiology® to SBML (.xml), using MATLAB® (Matlab/Run_main.m)
Needs to be fixed with our R script (<https://github.com/a-kramer/simbiology-sbml-fix>)
 - SBtab (.tsv) to VFGEN (.vf), using R (<https://github.com/a-kramer/SBtabVFGEN>)
 - SBtab (.tsv) to Mod (.mod), using R (<https://github.com/a-kramer/SBtabVFGEN>)
 - SBtab (.tsv) to SBML (.xml), using R (<https://github.com/a-kramer/SBtabVFGEN>)



SECTIONS OF THE WORKFLOW IN EXTERNAL REPOSITORIES

Conversion tools:

- <https://github.com/a-kramer/SBtabVFGEN>
- <https://github.com/a-kramer/simbiology-sbml-fix>

IMPLEMENTED MODELS

- https://github.com/jpgsantos/Model_Nair_2016
- https://github.com/jpgsantos/Model_Fujita_2010
- https://github.com/jpgsantos/Model_Viswan_2018

COMPATIBILITY

Subcellular workflow MATLAB® code is compatible with MATLAB® 2020a or above running on Microsoft Windows, macOS and Linux.

Matlab® packages needed:

- Optimization Toolbox™
- Statistics and Machine Learning Toolbox™
- Fuzzy Logic Toolbox™
- Financial Toolbox™
- Global Optimization Toolbox
- SimBiology®
- Parallel Computing Toolbox™

3.1 SBtab

This is an overview of the SBtab syntax that is used in our workflow. This contains a list of the sheet names and subfields that are read by our software. The second column in the first row of each sheet must include “TableName=’sheet name’”. Fields that are not mentioned here are not used in the latest workflow and are not imported but might be added to future releases. Additional information on how to use SBtab can be found in <https://www.sbtabs.net/sbtabs/default/documentation.html>.

3.1.1 Defaults

- **!ID** - Identifier code for the entries, it should consist of the letter “D” followed by an integer, should start at 1.
- **!Name** - Name of the default variable being defined, we have used time, volume, substance, length, and area in our files.
- **!Unit** - Unit of the default variable, eg. time - second.

3.1.2 Compartment

- **!ID** - Identifier code for the entries, it should consist of the letter “V” followed by an integer, should start at 1.
- **!Name** - Name of the compartment.
- **!Unit** - Compartment volume unit, usually liter.
- **!Size** - Size of the compartment in units defined in the unit column.

3.1.3 Compound

- **!ID** - Identifier code for the entries, it should consist of the letter “S” followed by an integer, should start at 0.
- **!Name** - Name of the compound. We advise to use recognizable compound names and separate complexes of multiple compounds with ‘_’ (e.g. A_B). Must start with a letter.
- **!Unit** - Unit for the compound, usually nanomole or nanomole/liter. If it is not a default MATLAB [Reg] unit it should be added to it before trying to run the scripts.
- **!InitialValue** - Default initial values for the compounds before the equilibration step. These are usually overridden by the values *Si in the experiments sheet*
- **!IsConstant** - assigns a binary value, either TRUE or FALSE, depending on whether the value of a particular compound should stay constant throughout the simulations. Note that the input compound should remain constant.
- **!Assignment** - assigns a binary value, either TRUE or FALSE.
- **!Interpolation** -
- **!Type** - Type of the reaction, e.g. “kinetic”.
- **!Location** - Compartment in which a particular compound is present.

3.1.4 Reaction

- **!ID** - Identifier code for the entries, it should consist of the letter “R” followed by an integer, should start at 0.
- **!Name** - Reaction name. Must start with a letter and it is advisable to include the reaction index, e.g. Reaction-Flux0.
- **!KineticLaw** - Reaction kinetic law. Needs to include the precise mass action kinetic law with compound and parameter names from corresponding SBtab table. For a reaction ‘A + B <=> A_B’ with a forward reaction rate of kf_R0 and backward reaction rate of kr_R0 the formula would look like ‘kf_R0*A*B-kr_R0*A_B’.
- **!IsReversible** - Boolean identifying the reversibility of the reaction, it is TRUE for reversible and FALSE for irreversible reactions.
- **!Location** - Compartment in which a particular reaction is taking place.
- **!ReactionFormula** - Chemical formula of the reaction, should be written in the form ‘A + B <=> A_B’.

3.1.5 Parameter

- **!ID** - Identifier code for the entries, it should consist of the letter “K” followed by an integer, should start at 0.
- **!Name** - Name of the parameter. We followed the convention of using ‘kf’ for forward reactions rates and ‘kr’ for reverse rates, followed by the *reaction !ID*, e.g. ‘kf_R0’. These names can be arbitrary but they need to coincide with whatever is defined in the *Reaction kinetic law*
- **!Unit** - The units of the parameter.
- **!DefaultValue** - Parameter value in linear space.
- **!Value:lnspace** - Parameter value in linear space.
- **!Value:log2** - Parameter value in log base 2.
- **!Value:log10** - Parameter value in log base 10.
- **!Location** - Compartment in which a particular parameter is governing a reaction.
- **!Comment** - Could be any plain text, we used it as a handy way of determining which reaction the parameter is involved in. We advise to use the following syntax ‘kf_AXB__A_B’ and ‘kr_AXB__A_B’ for respectively forward and backward rates of a reaction ‘A + B <=> A_B’.

3.1.6 Expression

Compounds which are defined by expressions.

- **!ID** - Identifier code for the entries, it should consist of the letters “Ex” followed by an integer, should start at 0.
- **!Name** - Name of the compound defined by the expression.
- **!Formula** - Formula assigned to the compound, it should use the compound names used in the compound sheet and, if needed, constant names from the constant sheet.
- **!Unit** - Concentration unit for the Compound.
- **!Location** - Compartment in which a particular compound is located.

3.1.7 Output

Compounds used as outputs in experimental data.

- **!ID** - Identifier code for the entries, it should consist of the letter “Y” followed by an integer, should start at 0.
- **!Name** - Name used to identify the output compound, when an existing compound needs to be measured we usually use “compound_name”_out.
- **!ErrorName** - Name of the error of the output compound. It should start with ‘SD’ (referring to standard deviation) followed by the output ID, e.g. SD_Y0.
- **!ErrorType** - Type of error for the output compound, we have used ‘abs+rel random noise (std)’.
- **!Unit** - Concentration unit for the output compound.
- **!ProbDist** - Probability distribution type of the measured output in an experimental setting, e.g. ‘normal’.
- **!Location** - Compartment in which a particular output compound is located.
- **!Formula** - Formula that links the experimental measured output to the compounds in the model. Usually the experimental measurement corresponds to a sum of compounds existing in the model but ratios are also common.

3.1.8 Experiments

Each column corresponds to one experiment for which there is a separate sheet.

- **!ID** - Identifier code for the entries, it should consist of the letter “E” followed by an integer, should start at 0.
- **!Name** - Name used to identify the experiment, we advise using the the word ‘Experiment’ followed by the experiment index.
- **>Output** - Should list all the output *!ID*’s, i.e. Y’s followed by their indices and separated by commas.
- **>S_i**- List of the various compounds of the model that have starting amounts other than 0, the same *!ID* as in the compound table should be used.

In a model with 2 experiments and 5 compounds A,B,C,D,E with IDs S0,S1,S2,S3,S4 respectively

- Experiment1 with compounds starting amounts A=0,B=1,C=2,D=0,E=3
- Experiment2 with compounds starting amounts A=1,B=0,C=1,D=0,E=4

Four entries should be included as exemplified bellow:

	>S0	>S1	>S2	>S4
Experiment1	0	1	2	3
Experiment2	1	0	1	4

Note that D/S3 is omitted because the starting value is 0 in all experiments.

- **!SimTime** - Simulation time for a particular experiment.
- **!Normalize** - Normalizations to be performed to the outputs are defined here.

3.1.9 E_i

Corresponds to individual experiments. It should have the name of the experiment IDs used in the experiments sheet.

- **!ID** - Identifier code for the entries, it should consist of the letters “E_iT” followed by an integer, should start at 0.
- **!Time** - Time series data, this should include a list of all the time points during which the corresponding output data points were sampled.
- **`>Y_i**- Compound to be measured, corresponds to the *!ID* of the output sheet. It should have the experimental (or simulated from another model) data.
- **`SD_Y_i**- Error of the compound to be measured, corresponds to the *!ErrorName* of the output sheet. It should have the experimental (or simulated from another model) data.

3.1.10 E_iI

Corresponds to individual experiments. It should have the name of the experiment IDs used in the experiments sheet, with an i in between E and the experiment number.

- **!ID** - Identifier code for the entries, it should consist of the letters “E_iIT” followed by an integer, should start at 0.
- **!Input_Time_S_i**- Time series of the inputs to the model, this should include a list of all the time points during which the corresponding input data points were sampled. To produce simple step inputs, only the time points during which a change in concentration is happening can be included. To produce more complicated input curves, more time points are needed to represent the shape of the curve.

- S_i - Compound that is being changed as input to the model, corresponds to the *ID* in the compound table. This column should represent the sampled concentration data points corresponding to each time point.

3.2 MATLAB®

The MATLAB® section of this workflow has been developed to facilitate model building and rapid iteration between different versions of a model. In this workflow we use one main script, “Run_main.m”, that calls all the relevant functions to be used. To run the MATLAB® code the Subcellular Workflow repository should be added to the MATLAB® path. Running the script “Run_main.m” generates prompts in the MATLAB® terminal window with a request to the user, to choose between a number of different options. These prompts allow the user to choose:

- The model to use (from all the models that are in the “Matlab/model” folder).

Note that the very first time you run a model you have to add the folder for that specific model from its home repository into the “Matlab/model” folder(e.g. copy the folder “Model_nair_2016” from its repository to “Matlab/model”).

For implemented models so far go to the following links:

- [Fujita_2010 model](#)
- [Nair_2016 model](#)
- [Viswan_2018 model](#)

- The analysis to be performed, with the following options:

1. *Diagnostics*
2. *Parameter Estimation*
3. *Global Sensitivity Analysis*
4. Reproduction of a previous analysis

This option can be used to re-do an analysis that has previously been performed. This is useful for reproducibility and in the case of the code getting updated with extra functionalities. The user should specify the analysis file that they want to use, examples are provided in the each model repository.

5. Reproduction of the plots of a previous analysis

Similar to the previous option but here only the plots are re-done.

6. Import model files

Creation of the model files and folder that are needed to run the model in Matlab, the creation of a folder with the model in .tsv format (one tsv file for each excel sheet of the original SBtab), as well as the conversion of the model to the SBML format (.xml).

- The *settings file* to use on the model.

These settings files can be found either in the respective model repository in the directory “Matlab/Settings”, or in the example model from our main repository in the directory “Matlab/model/Model_Example/Matlab/Settings”, or by following these links:

- [Example model settings files](#)
- [Fujita_2010 model settings files](#)
- [Nair_2016 model settings files](#)
- [Viswan_2018 model settings files](#)

Examples of the output received when the different models are run through the workflow can be found on the respective model repository in the directory “Matlab/Results/Results/Examples”, or by following these links:

- [Fujita_2010 model example results](#)
- [Nair_2016 model example results](#)
- [Viswan_2018 model example results](#)

In order to gain a better understanding on how the code works, there are detailed pages for the following:

- *Scripts* - The script that we use;
- *Functions* - All the custom functions we have built; this is directed to anyone that wants to develop or iterate the code;
- *Settings file* - The master configuration file, where we describe everything that can be modified by the user without changing any code;
- *Results* - Explanation of all the files containing relevant results that are generated after running the built-in analysis of the code;
- *Model files and folders* - Description of all the files and folders that are generated when the model is imported from SBtab into relevant files, used by the rest of the MATLAB® code.

3.2.1 Diagnostics

The specifications of the diagnostics operations require user input in the *settings file*. The toolkit imports the model stored in *SBtab format* (with the name specified by *stg.sbtab_excel_name* variable from a folder chosen by the user terminal prompts) to a .sbproj file, and saves it in the subfolder called Data along with the imported data and inputs in a .mat format. Once the model has been imported, the import function can be disabled in the *import section of the settings file* for further procedures.

The parameter sets that are specified in the *diagnostics section of the settings file* are then used in model simulations with the input specifications (e.g. time series data for relevant input species) in the Experiment Input (EI) tables in the SBtab file, calculate the error between the simulation results and the experimental data, and plot the error scores as well as the comparative traces of the simulation results and the experimental data. At least one parameter set is currently required in the settings file. Experiments of interest can be specified by the *stg.exprun* variable in the *analysis section of the settings file*.

3.2.2 Parameter Estimation

Parameter estimation is performed by various MATLAB optimization algorithms. The number of parameters to estimate, possible thermodynamic constraints (can be determined by a standalone *script*), and upper and lower bounds can be specified in the *model section of the settings file*. In addition, the parameter indices and the best available parameter sets can be specified in the *diagnostics section*. Optimization algorithm and optimization settings can be found in the *optimization section of the settings file* and simulation settings in the *simulation section*.

3.2.3 Global Sensitivity Analysis

Global Sensitivity Analysis can be performed when a user is interested in parameter distribution rather than single point values, and how sensitive a specific output is towards perturbations in different parameters. Instructions on what settings are required to be specified can be found in the *Global Sensitivity Analysis section of the settings file*.

3.2.4 Scripts

This is the entry point for our code, it calls all other relevant functions.

Run_main

Code

```

1  % Script to import shtab and run the analysis
2  % clear functions
3
4  %Get the date and time
5  date_stamp = string(year(datetime)) + "_" + ...
6      string(month(datetime)) + "_" + string(day(datetime))...
7      + "_" + string(hour(datetime)) + "_" + string(minute(datetime))...
8      + "_" + string(round(second(datetime)));
9
10 % Get the folder where the MATLAB code and models are located
11 Matlab_main_folder = fileparts(mfilename('fullpath'))+"/";
12 Matlab_main_folder = strrep(Matlab_main_folder, "\", "/");
13 addpath(genpath(Matlab_main_folder));
14 mmf.main = Matlab_main_folder;
15
16 % Name of the various analysis that can be run with this code
17 analysis_options = ["Diagnostics", "Parameter Estimation", ...
18     "Global Sensitivity Analysis", "Reproduce a previous analysis", ...
19     "Reproduce the plots of a previous analysis", "Import model files"];
20
21 % Code for choosing the model and loading the settings files
22 [stg, rst, sb] = f_user_input(mmf, analysis_options);
23
24 % Get the folder structure used for the model files
25 [mmf] = default_folders(stg, mmf, date_stamp);
26
27 % Runs the import scripts if chosen in settings
28 if stg.import
29     [stg, sb] = f_import(stg, mmf);
30 else
31     % Creates a struct based on the shtab that is used elsewhere in the code
32     % and also adds the number of experiments and outputs to the settings
33     % variable
34     if isempty(sb) % check needed for plot reproduction
35         [stg, sb] = f_generate_shtab_struct(stg, mmf);
36     end
37 end
38

```

(continues on next page)

(continued from previous page)

```

39 % Runs the Analysis chosen in settings
40 if any(contains(analysis_options(1:3),stg.analysis))
41     rst = f_analysis(stg,stg.analysis,mmf,analysis_options);
42 end
43 % Save Analysis results if chosen in settings
44 if stg.save_results
45     f_save_analysis(stg,sb,rst,mmf)
46 end
47
48 % Plots the results of the analysis, this can be done independently after
49 % loading the results of a previously run analysis
50 if stg.plot
51     f_plot(rst,stg,mmf)
52     % Save plots results if chosen in settings
53     if stg.save_results
54         f_save_plots(mmf)
55     end
56 end

```

This is the main script from the MATLAB® portion of the workflow. Depending on the configurations on the *settings file* and choices on the user facing prompts it can call functions to:

- *Perform conversions of the SBtab:*
 - SBtab(.xlsx) to SBtab (.tsv)
 - SBtab (.xlsx) to MATLAB® SimBiology® (.m, .sbproj)
 - MATLAB® SimBiology® to SBML (.xml)
- *Perform analysis on the model:*
 - Diagnostics
 - Parameter Estimation
 - Global Sensitivity Analysis
- *Saving results from analysis*
- *Plotting relevant results*
- *Saving plots*

It can also reproduce a the calculations of a previous analysis or just its plots.

3.2.5 Functions

The MATLAB® functions used in this workflow are divided according to their role, we have:

- *Setup and Import functions*

Functions to import the model to MATLAB® and generate model specific files and functions.
- *Simulation and scoring functions*

Functions relating to the simulation and scoring of the model.
- *Analysis functions*

Functions for the analysis that we can perform on the model.

- *Plotting unctions*

Functions to plot the result of the different analyses.

- *General purpose functions*

General purpose functions that are usually used by other functions.

Setup and Import

f_load_settings

Code

```

1  function [stg,rst,sb] = f_user_input(mmf,analysis_options)
2
3  persistent last_SBtab_date
4  persistent last_model_folder
5  persistent last_settings_file_text
6  persistent last_settings_file_date
7  persistent last_analysis_text
8
9  Matlab_main_folder = mmf.main;
10
11  rst = [];
12  sb = [];
13  functions_cleared = false;
14
15  % Get the folder of the model
16  model_folder_general = Matlab_main_folder + "Model/";
17  last_choice = last_model_folder;
18  prompt = "What model folder should be used?\n";
19  [model_folder,last_model_folder] =...
20      choose_options(model_folder_general,prompt,last_choice);
21
22  model_name_specific = string(model_folder);
23  folder_model_specific = model_folder_general + "/" + model_name_specific;
24
25  prompt = "\nWhat analysis should be performed?\n";
26  last_choice = last_analysis_text;
27  [analysis_text,last_analysis_text] =...
28      parse_choices(prompt,analysis_options,last_choice);
29
30  % analysis_n = find(contains(analysis_options,analysis_text));
31
32  % Check if an analysis was chosen
33  if any(contains(analysis_options([1:3,6]),...
34      analysis_text))
35
36      %Get the Setting file to be used
37      settings_folder = folder_model_specific + "/Matlab/Settings";
38      last_choice = last_settings_file_text;
39      prompt = "\nWhat file should be used as settings?\n";
40      [settings_file_text,last_settings_file_text] =...

```

(continues on next page)

(continued from previous page)

```

41     choose_options(settings_folder,prompt,last_choice);
42
43     settings_file = strrep(settings_file_text,".m","");
44
45     % Add the default settings to the struct
46     stg_add_default = eval("default_settings()");
47
48     f = fieldnames(stg_add_default);
49     for i = 1:length(f)
50         stg.(f{i}) = stg_add_default.(f{i});
51     end
52
53     % Add chosen settings to the struct overwriting defaults when
54     % appropriate
55     [stg_add] = eval(settings_file + "()");
56
57     f = fieldnames(stg_add);
58     for i = 1:length(f)
59         stg.(f{i}) = stg_add.(f{i});
60     end
61
62     % Check if the date of the settings file changed, if so clear functions
63     listing = dir(settings_folder);
64     for n = 1:size(listing,1)
65         if matches(settings_file_text,listing(n).name,"IgnoreCase",true)
66             settings_file_date = listing(n).date;
67         end
68     end
69
70     [last_settings_file_date,functions_cleared] =...
71         compare_last(settings_file_date,last_settings_file_date,...
72             functions_cleared);
73
74     % Check if the name of the settings file changed, if so clear functions
75     [last_settings_file_text,functions_cleared] =...
76         compare_last(settings_file_text,last_settings_file_text,...
77             functions_cleared);
78
79     % Check if the date of the SBTAB changed, if so clear functions
80     listing = dir(folder_model_specific);
81
82     for n = 1:size(listing,1)
83         if matches(stg.sbtabs_excel_name,listing(n).name,"IgnoreCase",true)
84             sbtab_date = listing(n).date;
85         end
86     end
87     [last_SBTAB_date,~] =...
88         compare_last(sbtabs_date,last_SBTAB_date,functions_cleared);
89
90     % Store the name of the chosen analysis in the settings struct
91     stg.analysis = analysis_text;
92

```

(continues on next page)

(continued from previous page)

```

93     if contains(analysis_options(6),analysis_text)
94         stg.import = true;
95         stg.save_results = false;
96         stg.plot = false;
97     end
98 elseif any(contains(analysis_options(4:5),analysis_text))
99
100     % Get the folder of the Analysis that should be reproduced
101     folder_results = folder_model_specific + "/Matlab/Results";
102
103     last_choice = [];
104     prompt = "\nWhat analysis should be reproduced?\n";
105
106     [r_analysis_text,~] =...
107         choose_options(folder_results,prompt,last_choice);
108
109     folder_results_specific = folder_results + "/" + ...
110         r_analysis_text;
111
112     last_choice = [];
113     prompt = "\nWhen was this analysis run originally?\n";
114
115     [r_analysis_date_text,~] =...
116         choose_options(folder_results_specific,prompt,last_choice);
117
118     folder_results_specific_date = folder_results_specific + "/" + ...
119         r_analysis_date_text;
120
121     % Load the settings file and the SBtab struct
122     load(folder_results_specific_date + "/Analysis.mat","stg","sb")
123
124     % Set inport to false since we don't want to overwrite anything
125     stg.import = false;
126
127     % If the reproduction of an analysis is chosen clear the functions
128     % because the settings most likely changed
129     if contains(analysis_options(4),analysis_text)
130         f_functions_to_clear()
131     end
132
133     % If the reproduction of the plots of an analysis is chosen make sure
134     % we tell the code to produce plots and also load the results that were
135     % previously obtained
136     if contains(analysis_options(5),analysis_text)
137         stg.plot = true;
138         load(folder_results_specific_date + "/Analysis.mat","rst")
139     end
140 end
141
142 % Set the chosen model folder in the settings struct
143 stg.folder_model = model_name_specific;
144 end

```

(continues on next page)

(continued from previous page)

```

145
146 function [choice,last_choice] = choose_options(folder,prompt,last_choice)
147
148 listing = dir(folder);
149
150 for n = size(listing,1):-1:1
151     if any(matches(listing(n).name,[".", "..", "Place models here.txt"]))
152         listing(n)= [];
153     end
154 end
155
156 for n = 1:size(listing,1)
157     options(n) = string(listing(n).name);
158 end
159
160 [choice,last_choice] = parse_choices(prompt,options,last_choice);
161 end
162
163 function [choice,last_choice] = parse_choices(prompt,options,last_choice)
164
165 for n = 1:size(options,2)
166     prompt = prompt + "\n" + n + ": " + options(n);
167 end
168
169 if ~isempty(last_choice)
170     if any(contains(options,last_choice))
171         prompt = prompt + "\n\nPress enter to use " + last_choice;
172     else
173         last_choice = [];
174     end
175 end
176
177 prompt = prompt + "\n";
178
179 i = input(prompt);
180
181 if isempty(i)
182     choice = [];
183 elseif i > 0 && i < size(options,2)+1
184     choice = options(i);
185     disp("The option chosen was: " + choice)
186 else
187     prompt = "Please choose from the provided options";
188     [choice,last_choice] = parse_choices(prompt,options,last_choice);
189 end
190
191 if isempty(choice)
192     if ~isempty(last_choice)
193         choice = last_choice;
194         disp("The option chosen was: " + last_choice)
195     else
196         prompt = "Please choose from the provided options";

```

(continues on next page)

(continued from previous page)

```

197     [choice,last_choice] = parse_choices(prompt,options,last_choice);
198     end
199 else
200     last_choice = choice;
201 end
202 end
203
204 function [previous,f_cleared] = compare_last(current,previous,f_cleared)
205
206 if ~isempty(previous)
207     if ~contains(current,previous)
208         if f_cleared == false
209             disp("Settings file changed, clearing functions")
210             f_functions_to_clear()
211             f_cleared = true;
212         end
213     end
214 end
215 previous = current;
216 end

```

It prompts the user to choose the model to run, the settings file to use, and the Analysis to perform.

- **Outputs** - *stg*, *rst*, sb, Analysis_n

f_import

Code

```

1 function [stg,sb] = f_import(stg,mmf)
2
3 Model_folder = mmf.model.main;
4
5 disp("Generating model files and folder from SBtab")
6
7 % Create needed folders
8 [~,~] = mkdir(mmf.model.data.main);
9 [~,~] = mkdir(mmf.model.input_functions.main);
10 [~,~] = mkdir(mmf.model.tsv.model_name);
11 [~,~] = mkdir(mmf.model.data.model_exp.main);
12
13 % Creates a .mat and a tsvs from the SBtab file
14 f_excel_sbtabs_importer(mmf);
15
16 addpath(genpath(Model_folder));
17
18 % Creates a struct based on the SBtab that is used elsewhere in the code and
19 % also adds the number of experiments and outputs to the settings struct
20 [stg,sb] = f_generate_sbtabs_struct(stg,mmf);
21
22 %Create the model and input output structure from sbtab.mat
23

```

(continues on next page)

(continued from previous page)

```

24 % Saves the model in .mat, .sbproj and .xml format, while also creating a
25 % file with the data to run the model in all different experimental
26 % settings defined in the SBtab
27 f_sbtabs_to_model(stg,sb,mmf)
28
29 % Creates code that loads the inputs of each experiment into a .mat file,
30 % and creates the code to read this inputs at runtime when the experiments
31 % are being simulated, all this generated code is stored on the Input
32 % functions folder
33 f_setup_input(stg,mmf)
34
35 %Creates three .mat files for each experiment, with all the added rules,
36 %species and parameters needed depending on the inputs and outputs
37 %specified on the SBtab, one for the equilibrium simulation run, one for
38 %the default run, and one for a more detailed run.
39 f_build_model_exp(stg,sb,mmf)
40 disp("Model files and folders generated successfully")
41 end

```

Creates the necessary folders inside the model folder. Calls subfunctions that convert the SBtab from an Excel into MATLAB® files useful for the workflow, TSVs and a SBML.

f_excel_sbtab_importer

Code

```

1 function f_excel_sbtab_importer(mm)
2
3 Source_sbtab = mm.model.sbtab;
4 Matlab_sbtab = mm.model.data.sbtab;
5
6 % Get the total number of sheets in the SBTAB
7 sheets = sheetnames(Source_sbtab);
8
9 % Try to run the import the sheets in multicore, depending on the version
10 % of excel this might not work
11 try
12     parfor i = 1:size(sheets,1)
13         sbtab_excel{i} = impexp (i,mm);
14     end
15 catch
16     for i = 1:size(sheets,1)
17         sbtab_excel{i} = impexp (i,mm);
18     end
19 end
20
21 % Save the SBTAB tables in .mat format
22 save(Matlab_sbtab,'sbtab_excel');
23 disp("SBtab with " + size(sheets,1) + " sheets parsed successfully")
24 end
25

```

(continues on next page)

(continued from previous page)

```

26 function shtab_excel = impexp (i,mmf)
27
28 Source_shtab = mmf.model.shtab;
29 tsv_name_folder = mmf.model.tsv.model_name;
30
31 % Import the SHTAB to a cell with a sheet per cell
32 shtab_excel = readcell(Source_shtab,'sheet',i);
33
34 % Replace "ismissing" values with empty spaces
35 mask = cellfun(@ismissing, shtab_excel,'UniformOutput',false);
36 mask = cellfun(@min, mask);
37 mask = logical(mask);
38 shtab_excel(mask) = {[]};
39
40 % Get name for tsv that is going to be exported
41 field = regexp(shtab_excel{1,2},"TableName='[^']*'",'match');
42 field = string(replace(field,["TableName='", "'", " "],["'", "'", "_"]));
43
44 %Export the tsv
45 cell_write_tsv(tsv_name_folder + field + ".tsv",shtab_excel)
46 end
47
48 function cell_write_tsv(filename,origCell)
49
50 % save a new version of the cell for reference
51 modCell = origCell;
52 % assume some cells are numeric, in which case set to char
53 iNum = cellfun(@isnumeric,origCell);
54
55 % Replace numeric cells with cell strings
56 for n = 1:size(iNum,1)
57     for m = 1:size(iNum,2)
58         modCell(n,m) = cellstr(num2str(origCell{n,m}));
59     end
60 end
61
62 %Save the file that only as strings in each cell
63 modCell = transpose(modCell);
64
65 [rNum,cNum] = size(origCell);
66 frmt = repmat([repmat('%s\t',1,cNum-1), '%s\n'],1,rNum);
67 fid = fopen(filename,'wt');
68 fprintf(fid,frmt,modCell{:});
69 fclose(fid);
70 end

```

Loads the information in the SHTab and creates a .mat file that contains the shtab and *TSVs* corresponding to all the SHTab tabs.

- Inputs - *stg*
- Saves
 - .mat file containing the SHTab in the “Model/Data” folder

- TSVs containing *the SBT* in the “Model/tsv” folder

f_generate_sbt_struct

Code

```

1 function [stg,sb] = f_generate_sbt_struct(stg,mmf)
2
3 Matlab_sbt = mmf.model.data.sbt;
4
5 if isfile(Matlab_sbt)
6
7     load(Matlab_sbt,'sbt_excel');
8
9     sb = f_get_sbt_fields(sbt_excel);
10
11     stg.expn = size(sb.Experiments.ID,1);
12     stg.outn = size(sb.Output.ID,1);
13 end
14 end
15
16 function sb = f_get_sbt_fields(sbt_excel)
17 for n = 1:size(sbt_excel,2)
18
19     if ~isempty(sbt_excel{1,n}{1,2})
20
21         field = regexp(sbt_excel{1,n}{1,2},"TableName='[^']*'", 'match');
22         field = string(replace(field,["TableName='","'",'" '],["","","_"]));
23
24         for k = 1:size(sbt_excel{1,n},2)
25
26             if ~isempty(sbt_excel{1,n}{2,k})
27                 subfield = sbt_excel{1,n}{2,k};
28                 subfield = ...
29                     string(replace(subfield,["!", ">", ":", " "],["", "", "_", "_
30                     ↪"]));
31
32                 sb.(field).(subfield)(:,1) = sbt_excel{1,n}(3:end,k)';
33
34                 sb.(field).(subfield) = sb.(field).(subfield)...
35                     (~cellfun('isempty', sb.(field).(subfield)));
36             end
37         end
38     end
39 end

```

Loads the SBT saved in the *.mat file* and creates a MATLAB® struct that can be more easily parsed.

- **Inputs** - *stg*
- **Outputs** - *sb*, *stg.expn*, *stg.outn*.

f_sbtabs_to_model

Code

```

1 function f_sbtabs_to_model(stg,sb,mmf)
2 % Saves the model in .mat, .sbproj and .xml format, while also
3 % ↪ creating a
4 % file with the data to run the model in all different experimental
5 % settings defined in the sbtab
6
7 modelobj = sbiomodel(stg.name);
8 compObj = [];
9
10 sbtab.species = cat(2,sb.Compound.Name,sb.Compound.InitialValue,...
11     sb.Compound.IsConstant,sb.Compound.Unit,sb.Compound.Location);
12
13 sbtab.defpar = cat(2,sb.Parameter.Comment,sb.Parameter.Value_linspace,...
14     ↪ ...
15     sb.Parameter.Unit);
16
17 for n = 1:size(sb.Compartment.ID,2)
18     compObj{n} = addcompartment(modelobj, sb.Compartment.Name{n});
19     set(compObj{n}, 'CapacityUnits', sb.Compartment.Unit{n});
20     set(compObj{n}, 'Value', sb.Compartment.Size{n});
21 end
22
23 for n = 1:size(sbtab.species,1)
24     for m = 1:size(compObj,2)
25         if string(compObj{m}.Name) == string(sb.Compound.Location{n})
26             compartment_number_match = m;
27         end
28     end
29     addspecies (compObj{compartment_number_match}, sb.Compound.Name{n},
30     ↪ sb.Compound.InitialValue{n}...
31     , 'InitialAmountUnits', sb.Compound.Unit{n});
32 end
33
34 for n = 1:size(sbtab.defpar,1)
35     addparameter(modelobj,sb.Parameter.Name{n},...
36     sb.Parameter.Value_linspace{n}, 'ValueUnits', sb.Parameter.Unit
37     ↪ {n}, 'Notes', sb.Parameter.Comment{n});
38 end
39
40 for n = 1:size(sb.Reaction.ID,1)
41     if ischar(sb.Reaction.IsReversible{n})
42         if contains(convertCharsToStrings(sb.Reaction.IsReversible{n}),
43     ↪ "true")
44             reaction_name = strrep(sb.Reaction.ReactionFormula{n}, '<=>'
45     ↪ ', ' <-> ');
46         else

```

(continues on next page)

(continued from previous page)

```

44         reaction_name = strrep(sb.Reaction.ReactionFormula{n}, '<=>')
45     end
46     else
47         if sb.Reaction.IsReversible{n}
48             reaction_name = strrep(sb.Reaction.ReactionFormula{n}, '<=>')
49         else
50             reaction_name = strrep(sb.Reaction.ReactionFormula{n}, '<=>')
51         end
52     end
53
54     reaction_name_compartment = reaction_name;
55
56     for m = 1:size(sb.Compound.Name,1)
57         reaction_name_compartment = ...
58             insertBefore(string(reaction_name_compartment), " " + ...
59                 string(sb.Compound.Name{m}), " " + string(sb.Reaction.
60 Location{n}));
61     end
62
63     while contains(reaction_name_compartment, ...
64         string(sb.Reaction.Location{n}) + " " + string(sb.Reaction.
65 Location{n}))
66         reaction_name_compartment = ...
67             strrep(reaction_name_compartment, string(sb.Reaction.
68 Location{n}) + ...
69                 " " + string(sb.Reaction.Location{n}), " " + sb.Reaction.
70 Location{n});
71     end
72
73     while contains(reaction_name_compartment, " ")
74         reaction_name_compartment = strrep(reaction_name_compartment, " "
75         , " ");
76     end
77
78     reaction_name_compartment = strrep(reaction_name_compartment, ...
79         sb.Reaction.Location{n} + " ", sb.Reaction.Location{n} + ".");
80     reaction_name_compartment = string(sb.Reaction.Location{n}) + ".
81     "+reaction_name_compartment;
82
83     reactionObj = addreaction(modelObj, reaction_name_compartment);
84     set(reactionObj, 'ReactionRate', sb.Reaction.KineticLaw{n});
85 end
86
87 for n = 1:size(sb.Compound.ID,1)
88     if ischar(sb.Compound.Assignment{n})
89         if contains(convertCharsToStrings(sb.Compound.Assignment{n}), [
90 "true", "True"])
91             modelObj.species(n).BoundaryCondition = 1;
92         end
93     end

```

(continues on next page)

(continued from previous page)

```

86     else
87         if sb.Compound.Assignment{n} == 1
88             modelobj.species(n).BoundaryCondition = 1;
89         end
90     end
91     if ischar(sb.Compound.Interpolation{n})
92         if contains(convertCharsToStrings(sb.Compound.Interpolation{n}
↪),["true","True"])
93             modelobj.species(n).BoundaryCondition = 1;
94         end
95     else
96         if sb.Compound.Interpolation{n} == 1
97             modelobj.species(n).BoundaryCondition = 1;
98         end
99     end
100    if ischar(sb.Compound.IsConstant{n})
101        if contains(convertCharsToStrings(sb.Compound.IsConstant{n}),[
↪"true","True"])
102            modelobj.species(n).BoundaryCondition = 1;
103        end
104    else
105        if sb.Compound.IsConstant{n} == 1
106            modelobj.species(n).BoundaryCondition = 1;
107        end
108    end
109 end
110
111 shtab.sim_time = [sb.Experiments.Sim_Time{:}];
112
113 species_INP_matcher = {};
114 for n = 1:size(sb.Compound.ID,1)
115     if isfield(sb.Experiments,"S"+(n-1))
116         species_INP_matcher{size(species_INP_matcher,1)+1,1} = n;
117     end
118 end
119
120 for n = 1:size(sb.Experiments.ID,1)
121     startamount = cell(1,size(species_INP_matcher,1));
122     nstartamount = 0;
123     nInputTime = 0;
124     nInput = 0;
125     nOutput = 0;
126
127     for m = 1:size(sb.Compound.ID,1)
128         if isfield(sb.Experiments,"S"+(m-1))
129             nstartamount = nstartamount+1;
130             startamount{nstartamount} = eval("sb.Experiments.S"+(m-1)+
↪"(n)");
131             startAmountName(nstartamount) = sb.Compound.Name(m);
132         end
133     end
134

```

(continues on next page)

(continued from previous page)

```

135     if isfield(sb.Experiments,"Normalize")
136         sbtab.datasets(n).Normalize = sb.Experiments.Normalize{n};
137     else
138         sbtab.datasets(n).Normalize = [];
139     end
140
141     if isfield(eval(("sb.E")+(n-1)), "Time")
142         Data(n).Experiment.t = transpose(eval("[sb.E"+(n-1)+".Time{:}]
↪"));
143     end
144
145     for m = 1:size(sb.Compound.ID,1)
146         if isfield(eval(("sb.E")+(n-1)+"I"), "Input_Time_S"+(m-1))
147             nInputTime = nInputTime + 1;
148             sbtab.datasets(n).input_time{1,nInputTime} = ...
149                 eval("[sb.E" + (n-1) + "I.Input_Time_S" + (m-1) + "
↪{:}]");
150         end
151         if isfield(eval(("sb.E")+(n-1)+"I"), "S"+(m-1))
152             nInput = nInput + 1;
153             sbtab.datasets(n).input_value{1,nInput} = ...
154                 eval("[sb.E" + (n-1) + "I.S" + (m-1) + "{:}]");
155             sbtab.datasets(n).input{nInput} = char("S" + (m-1));
156         end
157     end
158
159     for m = 1:size(sb.Output.ID,1)
160         if isfield(eval(("sb.E")+(n-1)), "Y"+(m-1))
161             nOutput = nOutput+1;
162             Data(n).Experiment.x(:,nOutput) = eval("[sb.E" + ...
163                 (n-1) + ".Y" + (m-1) + "{:}]");
164             Data(n).Experiment.x_SD(:,nOutput) = eval("[sb.E" + ...
165                 (n-1) + ".SD_Y" + (m-1) + "{:}]");
166             sbtab.datasets(n).output{nOutput} = sb.Output.Name(m);
167             sbtab.datasets(n).output_value{nOutput} = ...
168                 {convertStringsToChars(...
169                     strrep(string(sb.Output.Location{m}) + "." + ...
170                     string(sb.Output.Name{m}) + " = " + ...
171                     string(sb.Output.Formula{m}), 'eps', '0.0001'))};
172             sbtab.datasets(n).output_name{nOutput} = ...
173                 sb.Output.Name(m);
174             sbtab.datasets(n).output_ID{nOutput} = ...
175                 sb.Output.ID(m);
176             sbtab.datasets(n).output_location{nOutput} = ...
177                 sb.Output.Location(m);
178         end
179     end
180     sbtab.datasets(n).stg.outnumber = nOutput;
181     sbtab.datasets(n).start_amount = cat(2, startAmountName(:)...
182         , transpose([startamount{:}], species_INP_matcher);
183 end
184

```

(continues on next page)

(continued from previous page)

```

185 if isfield(sb,"Expression")
186     for m = 1:size(sb.Expression.ID,1)
187         if isfield(sb.Expression,'Formula')
188             if isa(sb.Expression.Formula{m},'double')
189                 addspecies (modelobj, char(sb.Expression.Name(m)),...
190                     str2double(string(sb.Expression.Formula{m})),...
191                     'InitialAmountUnits',sb.Expression.Unit{m});
192             else
193                 try
194                     addspecies (modelobj, char(sb.Expression.Name(m)),
195                         0,...
196                         'InitialAmountUnits',sb.Expression.Unit{m});
197                 catch
198                     end
199                 addrule(modelobj, char({convertStringsToChars(...
200                     string(sb.Expression.Location{m}) + "." +...
201                     string(sb.Expression.Name{m}) + " = " +...
202                     string(sb.Expression.Formula{m}))),...
203                     'repeatedAssignment');
204             end
205         else
206             addparameter(modelobj,char(sb.Expression.Name(m)),...
207                 str2double(string(sb.Expression.DefaultValue{m})),...
208                 'ValueUnits',sb.Expression.Unit{m});
209         end
210     end
211
212 if isfield(sb,"Input")
213     for m = 1:size(sb.Input.ID,1)
214         if isfield(sb.Input,'Formula')
215             if isa(sb.Input.Formula{m},'double')
216                 addspecies (modelobj, char(sb.Input.Name(m)),...
217                     str2double(string(sb.Input.DefaultValue{m})),...
218                     'InitialAmountUnits',sb.Input.Unit{m});
219             else
220                 try
221                     addspecies (modelobj, char(sb.Input.Name(m)),0,...
222                         'InitialAmountUnits',sb.Input.Unit{m});
223                 catch
224                     end
225                 addrule(modelobj, char({convertStringsToChars(...
226                     string(sb.Input.Location{m}) + "." +...
227                     string(sb.Input.Name{m}) + " = " +...
228                     string(sb.Input.DefaultValue{m}))),...
229                     'repeatedAssignment');
230             end
231         else
232             addparameter(modelobj,char(sb.Input.Name(m)),...
233                 str2double(string(sb.Input.DefaultValue{m})),...
234                 'ValueUnits',sb.Input.Unit{m});
235         end
236     end
237

```

(continues on next page)

(continued from previous page)

```

236     end
237 end
238
239 if isfield(sb,"Constant")
240     for m = 1:size(sb.Constant.ID,1)
241         addparameter(modelobj,char(sb.Constant.Name(m)),...
242             str2double(string(sb.Constant.Value{m})),...
243             'ValueUnits',sb.Constant.Unit{m});
244     end
245 end
246
247 sbproj_model = mmf.model.data.sbproj_model;
248 matlab_model = mmf.model.data.mat_model;
249 data_model = mmf.model.data.data_model;
250 xml_model = mmf.model.data.xml_model;
251
252 sbiosaveproject(sbproj_model,'modelobj')
253
254 save(matlab_model,'modelobj')
255
256 save(data_model,'Data','sbtabs','sb')
257
258 sbmllexport(modelobj,xml_model)
259
260 end

```

Reads information from the SBtab and saves the model in MATLAB (*.mat*, *.sbproj*) and SBML(*.xml*) format, while also creating a *file* with the data to run the model in all different experimental settings defined in the SBtab.

- **Inputs** - *stg*, *sb*
- **Saves** - *model file .mat*, *model file .sbproj*, *model file .xml*, and *data file*

f_setup_input

Code

```

1  function f_setup_input(stg,mmf)
2  % Creates code that loads the inputs of each experiment into a .mat_
   ↪ file,
3  % and creates the code to read this inputs at runtime when the_
   ↪ experiments
4  % are being simulated, all this generated code is stored on the
5  % Input_functions folder
6
7  matlab_model = mmf.model.data.mat_model;
8  data_model = mmf.model.data.data_model;
9  inp_model_data = mmf.model.data.input_model_data;
10 Model_folder = mmf.model.main;
11 model_input = mmf.model.input_functions.input;
12
13 %Find correct path for loading depending on the platform

```

(continues on next page)

(continued from previous page)

```

14 load(data_model,'sbtabs')
15
16 load(matlab_model,'modelobj');
17
18 for Exp_n = 1:size(sbtabs.datasets,2)
19
20     for index = 1:size(sbtabs.datasets(Exp_n).input,2)
21         if size(sbtabs.datasets(Exp_n).input_value{index},2) > 100
22
23             input_name = strrep(modelobj.species(1+str2double(strrep(..
↵ .
24                 sbtabs.datasets(Exp_n).input(index),'S',''))).name,".",")
↵ ");
25
26             inpX = input_name + "X";
27             inpT = input_name + "T";
28             inph1 = input_name + "h1";
29             inph2 = input_name + "h2";
30
31             fullFileName = sprintf('%s.m',...
32                 model_input + Exp_n + "_" + input_name );
33
34             fileID = fopen(fullFileName, 'wt');
35
36             inp_str = template1();
37             inp_str = replace(inp_str,...
38                 ["SBtab_name","Exp_n","input_name",...
39                 "inpX", "inpT", "inph1", "inph2", "inp_model_data",...
40                 "sbtabs.sim_time(Exp_n)"],[stg.name,Exp_n,...
41                 input_name, inpX, inpT, inph1,...
42                 inph2, inp_model_data,...
43                 sbtabs.sim_time(Exp_n)]);
44
45             fprintf(fileID,inp_str);
46             fclose(fileID);
47         end
48     end
49 end
50
51 fullFileName = sprintf('%s.m',model_input + "_creator" );
52
53 fileID = fopen(fullFileName, 'wt');
54 fprintf(fileID, "function " + stg.name + "_input_creator(~)\n");
55 helper = 0;
56 for Exp_n = 1:size(sbtabs.datasets,2)
57     for index = 1:size(sbtabs.datasets(Exp_n).input,2)
58         if size(sbtabs.datasets(Exp_n).input_value{index},2) > 100
59             input_name = strrep(modelobj.species(1+str2double(strrep(..
↵ .
60                 sbtabs.datasets(Exp_n).input(index),'S',''))).name,".",")
↵ ");
61
62             if helper == 0
63                 helper = 1;

```

(continues on next page)

(continued from previous page)

```

62         helper2 = Exp_n;
63     end
64     if index == 1 && Exp_n == helper2
65         fprintf(fileID,"load('" + data_model + "','sbtabs');\n
↪");
66         inp_creator_str = template2();
67         inp_creator_str = replace(inp_creator_str,...
68             ["Exp_n", "input_name", "index", "inp_model_data"],
↪...
69             [Exp_n, input_name, index, inp_model_data]);
70         %                               fprintf(fileID,inp_creator_str);
71     else
72         inp_creator_str = template3();
73         inp_creator_str = replace(inp_creator_str,...
74             ["Exp_n", "input_name", "index", "inp_model_data"],
↪...
75             [Exp_n, input_name, index, inp_model_data]);
76     end
77     fprintf(fileID,inp_creator_str);
78 end
79 end
80 end
81 fprintf(fileID, "end\n");
82 fclose(fileID);
83
84 addpath(genpath(Model_folder));
85 eval(stg.name + "_input_creator()");
86 end
87
88 function inp_str = template1()
89 inp_str =...
90     "function thisAmp = SBtab_name_inputExp_n_input_name(times)\n"+...
91     "persistent inpX\n"+...
92     "persistent inpT\n"+...
93     "persistent inph1\n"+...
94     "persistent inph2\n"+...
95     "if isempty(inpX)\n"+...
96     "Data = coder.load('inp_model_data','expExp_n_input_name');\n
↪"+...
97     "inpX = Data.expExp_n_input_name(:,2);\n"+...
98     "inpT = Data.expExp_n_input_name(:,1);\n"+...
99     "inph1 = 1;\n"+...
100     "end\n"+...
101     "thisAmp = inpX(end);\n"+...
102     "if times ~= sbtab.sim_time(Exp_n)\n"+...
103     "if times == 0\n"+...
104     "inph1 = 1;\n"+...
105     "thisAmp = inpX(1);\n"+...
106     "else\n"+...
107     "while times > inpT(inph1)\n"+...
108     "inph1 = inph1 + 1;\n"+...
109     "end\n"+...

```

(continues on next page)

(continued from previous page)

```

110         "while times < inpT(inph1-1)\n"+...
111         "inph1 = inph1-1;\n"+...
112         "end\n"+...
113         "inph2 = (inpT(inph1)-times)*1/(inpT(inph1)-inpT(inph1-1));
↪\n"+...
114         "thisAmp = (inpX(inph1-1)*inph2 + inpX(inph1)*(1-inph2));\n
↪"+...
115         "end\n"+...
116         "end\n"+...
117         "end";
118     end
119
120
121     function inp_creator_str = template2()
122     inp_creator_str = ...
123         "expExp_n_input_name(:,1) = shtab.datasets(Exp_n).input_time{index}
↪;\n"+...
124         "expExp_n_input_name(:,2) = shtab.datasets(Exp_n).input_value
↪{index};\n"+...
125         "save('inp_model_data','expExp_n_input_name');\n";
126     end
127     function inp_creator_str = template3()
128     inp_creator_str = ...
129         "expExp_n_input_name(:,1) = shtab.datasets(Exp_n).input_time{index}
↪;\n"+...
130         "expExp_n_input_name(:,2) = shtab.datasets(Exp_n).input_value
↪{index};\n"+...
131         "save('inp_model_data','expExp_n_input_name','-append');\n";
132     end

```

Creates code that loads the inputs of each experiment into a *.mat file* and the code to read these inputs at runtime when the experiments are being simulated. All this generated code is stored on the “Model/«model folder name»/Formulas” folder.

- **Inputs** - *stg*
- **Saves** - *model-specific functions*

f_build_model_exp

Code

```

1     function f_build_model_exp(stg,sb,mmf)
2     %Creates two .mat files for each experiment, with all the added rules,
3     %species and parameters needed depending on the inputs and outputs
4     %specified on the shtab, one for the equilibrium simulation run and
↪one for
5     %the proper run
6
7     data_model = mmf.model.data.data_model;
8     mat_model = mmf.model.data.mat_model;
9     model_exp_eq = mmf.model.data.model_exp.equilibration;

```

(continues on next page)

(continued from previous page)

```

10 model_exp_default = mmf.model.data.model_exp.default;
11 model_exp_detail = mmf.model.data.model_exp.detail;
12
13 %Find correct path for loading depending on the platform
14 load(data_model,'Data','sbtabs')
15 load(mat_model,'modelobj');
16
17 model_run = cell(size(sb.Experiments.ID,1),1);
18 configsetObj = cell(size(sb.Experiments.ID,1),1);
19
20 for number_exp = 1:size(sb.Experiments.ID,1)
21
22     output_value = sbtab.datasets(number_exp).output_value;
23     output = sbtab.datasets(number_exp).output;
24     input_time = sbtab.datasets(number_exp).input_time;
25     input_value = sbtab.datasets(number_exp).input_value;
26     input_species = sbtab.datasets(number_exp).input;
27
28     model_run{number_exp} = copyobj(modelobj);
29     configsetObj{number_exp} = getconfigset(model_run{number_exp});
30
31     set(configsetObj{number_exp}, 'MaximumWallClock', stg.maxt);
32     set(configsetObj{number_exp}, 'StopTime', stg.eqt);
33     set(configsetObj{number_exp}.CompileOptions,...
34         'DimensionalAnalysis', stg.dimenanal);
35     set(configsetObj{number_exp}.CompileOptions,...
36         'UnitConversion', stg.UnitConversion);
37     set(configsetObj{number_exp}.SolverOptions,...
38         'AbsoluteToleranceScaling', stg.abstolscale);
39     set(configsetObj{number_exp}.SolverOptions,...
40         'RelativeTolerance', stg.reltol);
41     set(configsetObj{number_exp}.SolverOptions,...
42         'AbsoluteTolerance', stg.abstol);
43     set(configsetObj{number_exp}.SolverOptions, 'OutputTimes', stg.
44     ←eqt);
45     set(configsetObj{number_exp}, 'TimeUnits', stg.simtime);
46     set(configsetObj{number_exp}.SolverOptions,...
47         'MaxStep', stg.maxstepeq);
48
49     set(configsetObj{number_exp}.SolverOptions,...
50         'AbsoluteToleranceStepSize', stg.abstolstepsize_eq);
51
52     model_exp = model_run{number_exp};
53     config_exp = configsetObj{number_exp};
54
55     save(model_exp_eq + number_exp + ".mat", 'model_exp', 'config_exp')
56
57     sbiosaveproject(model_exp_eq + number_exp + ".sbproj", 'model_exp')
58
59     set(configsetObj{number_exp}, 'StopTime', sbtab.sim_time(number_
60     ←exp));

```

(continues on next page)

(continued from previous page)

```

60     set(configsetObj{number_exp}.SolverOptions, 'OutputTimes',...
61         Data(number_exp).Experiment.t);
62
63
64     set(configsetObj{number_exp}.SolverOptions, 'MaxStep', stg.
65     ↪maxstep);
66
67     for n = 1:size(output,2)
68
69         m = 0;
70         for k = 1:size(model_run{number_exp}.species,1)
71             if model_run{number_exp}.species(k).name == string(output
72             ↪{1,n})
73                 model_run{number_exp}.species(k).BoundaryCondition = 1;
74                 m = 1;
75             end
76         end
77
78         if m == 0
79             addspecies (model_run{number_exp}.Compartments(1),...
80                 char(output{1,n}),0,...
81                 'InitialAmountUnits',sb.Output.Unit{n});
82         end
83
84         addrule(model_run{number_exp}, char(output_value{1,n}),...
85             'repeatedAssignment');
86     end
87
88     for j = 1:size(input_species,2)
89
90         input_indexcode = str2double(strrep(input_species(j),'S',''));
91         input_name = string(model_run{number_exp}.species(1 +...
92             input_indexcode).name);
93
94         if size(input_time{j},2) < 100
95
96             model_run{number_exp}.species(...
97                 1 + input_indexcode).BoundaryCondition = 1;
98             for n = 1:size(input_time{j},2)
99                 if ~isnan(input_time{j}(n))
100                     addparameter(model_run{number_exp},char("time_
101                     ↪event_t_" + j + "_" + n),...
102                         str2double(string(input_time{j}(n))),...
103                         'ValueUnits',char(stg.simtime));
104
105                     addparameter(model_run{number_exp},char("time_
106                     ↪event_r_" + j + "_" + n),...
107                         str2double(string(input_value{j}(n))),...
108                         'ValueUnits',char(model_run{number_exp}.
109                     ↪species(1 +...
110                         input_indexcode).InitialAmountUnits));

```

(continues on next page)

(continued from previous page)

```

107         addevent(model_run{number_exp}, ...
108             char("time>=time_event_t_" + j + "_" + n),...
109             cellstr(sbtabs.datasets(number_exp).output_
110             ↪location{1} + ...
111             "." + input_name + " = time_event_r_" + j + "_"
112             ↪" + n));
113     end
114 end
115 else
116
117     addrule(model_run{number_exp}, char(sbtabs.datasets(...
118         number_exp).output_location{1} + "." + input_name + ...
119         "=" + string(model_run{number_exp}.name) + "_input" + ..
120         ↪number_exp + "_" + input_name + "(time)"),
121     ↪'repeatedAssignment');
122     end
123 end
124
125 model_exp = model_run{number_exp};
126 config_exp = configsetObj{number_exp};
127
128 save(model_exp_default + number_exp + ".mat", 'model_exp', 'config_
129 ↪exp')
130
131 sbiosaveproject(model_exp_default + number_exp + ".sbproj", 'model_
132 ↪exp')
133
134 set(configsetObj{number_exp}.SolverOptions, 'OutputTimes', []);
135 set(configsetObj{number_exp}.SolverOptions, 'MaxStep', stg.
136 ↪maxstepdetail);
137
138 model_exp = model_run{number_exp};
139 config_exp = configsetObj{number_exp};
140
141 save(model_exp_detail + number_exp + ".mat", 'model_exp', 'config_exp
142 ↪')
143
144 end
145 end

```

Creates two .mat files for each experiment, one for the *equilibrium simulation run* and one for the *proper simulation*. These files have all the added rules, species and parameters needed depending on the inputs and outputs specified on the SBtab.

- **Inputs** - *stg*, *sb*
- **Saves** - *Ready to run models*

Simulation and Scoring

f_sim_score

Code

```

1  function [score,rst,rst_not_simd] = f_sim_score(parameters,stg,mmf)
2
3  %Turn off Dimension analysis warning from simbiology
4  warning('off','SimBiology:DimAnalysisNotDone_MatlabFcn_Dimensionless')
5
6  % Call the function that simulates the model
7  rst = f_prep_sim(parameters,stg,mmf);
8
9  % Call the function that scores
10 rst = f_score(rst,stg,mmf);
11
12 % Get the total score explicitly for optimization functions
13 score = rst.st;
14
15 rst_not_simd = rmfield( rst , 'simd');
16 end

```

Calls the function that runs the simulations and the function that scores the output of the runs.

- **Inputs**

- *stg*
- *parameters* - (double) Set of parameters that we are working on

- **Outputs**

- *score* - *rst.st*
- *rst* - *rst.simd*, *rst.xfinal*, *rst.sd*, *rst.se*, and *rst.st*
- *rst_not_simd* - *rst.xfinal*, *rst.sd*, *rst.se*, and *rst.st*

- **Calls** - *f_prep_sim*, *f_score*

- **Loads**

f_prep_sim

Code

```

1  function rst = f_prep_sim(parameters,stg,mmf)
2
3  % Save variables that need to be maintained over multiple function calls
4  % persistent modelobj
5  persistent sbtab
6  persistent Data
7
8  data_model = mmf.model.data.data_model;
9
10 % Import the data on the first run

```

(continues on next page)

(continued from previous page)

```

11  if isempty(sbtabs)
12
13      %Find correct path for loading depending on the platform
14      load(data_model,'Data','sbtabs')
15  end
16
17  % Set the parameters that are going to be used for the simulation to
18  % the
19  % default ones as defined in the SBTAB
20  rt.par(:,1) = [sbtabs.defpar(:,2)];
21
22  % Check if the parameter needs to be set to the value relevant for
23  % Profile
24  % Likelihood
25  if isfield(stg,"PLind") && isfield(stg,"PLval")
26      parameters = [parameters(1:stg.PLind-1) stg.PLval parameters(stg.
27      % PLind:end)];
28  end
29
30  % Iterate over all the parameters of the model
31  for n = 1:size(rt.par,1)
32
33      % Check that a parameter should be changed from default
34      if stg.partest(n) > 0
35
36          % Set the parameters are being tested
37          rt.par(n) = 10.^(parameters(stg.partest(n,1)));
38      end
39
40      if isfield(stg,'tci')
41
42          % Check that there are thermodynamic constraints to implement
43          if ~isempty(stg.tci)
44
45              % Choose the parameters that need to be calculated with
46              % other
47              % parameters due to thermodynamic constraints
48              if ismember(n,stg.tci)
49
50                  % Check that a parameter should be changed from default
51                  if stg.partest(n) > 0
52
53                      % Iterate over the parameters that need to be
54                      % multiplied
55                      % for calculating the parameter that depends on the
56                      % thermodynamic constraints
57                      for m = 1:size(stg.tcm,2)
58
59                          % Check that the parameter that is going to be
60                          % used
61                          % to calculate the parameter dependent on
62                          % thermodynamic constraints is not the
63                          % default

```

(continues on next page)

(continued from previous page)

```

57         if stg.partest(stg.tcm(n,m),1) > 0
58
59             % Check if the parametrer needs to be set.
↪ to
60             % the value relevant for Profile Likelihood
61             if isfield(stg,"PLind")
62                 if stg.partest(stg.tcm(n,m),1) ==...
63                     stg.PLind
64                     parameters(stg.partest(...
65                         stg.tcm(n,m),1))...
66                         = stg.PLval;
67                 end
68             end
69
70             % Make the appropriate multiplications to.
↪ get
71             % the thermodynamicly constrained parameter
72             rt.par(n) = rt.par(n).*(10.^...
73                 (parameters(stg.partest(...
74                     stg.tcm(n,m),1))));
75         else
76
77             % Make the appropriate multiplications to.
↪ get
78             % the thermodynamicly constrained parameter
79             rt.par(n) = rt.par(n).*...
80                 (sbtabs.defpar{stg.tcm(n,m),2});
81         end
82     end
83
84     % Iterate over the parameters that need to be.
↪ divided
85     % for calculating the parameter that depends on the
86     % thermodynamic constraints
87     for m = 1:size(stg.tcd,2)
88
89         % Check that the parameter that is going to be.
↪ used
90         % to calculate the parameter dependent on
91         % thermodynamic constraints is not the.
↪ default
92         if stg.partest(stg.tcd(n,m),1) > 0
93
94             % Check if the parametrer needs to be set.
↪ to
95             % the value relevant for Profile Likelihood
96             if isfield(stg,"PLind")
97                 if stg.partest(stg.tcd(n,m),1) ==...
98                     stg.PLind
99                     parameters(stg.partest(...
100                         stg.tcd(n,m),1))...
101                         = stg.PLval;

```

(continues on next page)

(continued from previous page)

```

102         end
103     end
104     % Make the appropriate divisions to get the
105     % thermodynamicly constrained parameter
106     rt.par(n) = rt.par(n)./(10.^...
107         (parameters(stg.partest(...
108             stg.tcd(n,m),1))));
109     else
110
111         % Make the appropriate divisions to get the
112         % thermodynamicly constrained parameter
113         rt.par(n) = rt.par(n)./...
114             (sbtabs.defpar{stg.tcd(n,m),2});
115     end
116 end
117 end
118 end
119 end
120 end
121 end
122
123 % Set the start amount for the species in the model to 0
124 rt.ssa = zeros(size(sbtabs.species,1),max(stg.exprun));
125
126 % Initialize the results variable
127 rst = [];
128 rst.parameters = rt.par;
129 % Iterate over all the experiments that are being run
130 for n = stg.exprun
131
132     % Try catch used because iterations errors can happen unexectedly.
133     ↪and
134     % we want to be able to continue simulations
135     try
136
137         % If the correct setting is chosen display messages to console
138         if stg.simcsl
139             disp("Running dataset number " + n + ...
140                 " of " + stg.exprun(end))
141         end
142
143         % Check that this is not the first time the experiments are.
144         ↪being
145         % run and that the start values for the species are different.
146         ↪from
147         % the previous experiment
148         if n ~= stg.exprun(1) && ...
149             min([sbtabs.datasets(n).start_amount{:,2}] ==...
150                 [sbtabs.datasets(max(1,stg.exprun(...
151                     find(stg.exprun==n)-1))).start_amount{:,2}])

```

(continues on next page)

(continued from previous page)

```

151         % Set the values of the start amounts to the values_
↪obtained
152         % after the first equilibration
153         rt.ssa(:,n) =...
154             rt.ssa(:,stg.exprun(find(stg.exprun==n)-1));
155         if stg.simdetail
156             rt.ssa(:,n+2*stg.expn) =...
157                 rt.ssa(:,stg.exprun(find(stg.exprun==n)-1));
158         end
159     else
160
161         % Iterate over the numbre of species that need a starting_
↪value
162         % different than 0
163         for j = 1:size(sbtabs.datasets(n).start_amount,1)
164
165             % Set the start amount of the species to the number_
↪defined
166             % in the sbtab for each experiment
167             rt.ssa(sbtabs.datasets(n...
168                 ).start_amount{j,3},n+stg.expn) =...
169                 sbtabs.datasets(n).start_amount{j,2};
170         end
171
172         % Equilibrate the model
173         rst = f_sim(n+stg.expn,stg,rt,rst,mmf);
174
175         for j = 1:size(sbtabs.species,1)
176
177             % Set the starting amount for species that after
178             % equilibrium have very low values to zero
179             if rst.simd{n+stg.expn}.Data(end,j) < 1.0e-15
180                 rt.ssa(j,n) = 0;
181
182             % Set the starting amount for the rest of the_
↪species
183             else
184                 rt.ssa(j,n) =...
185                     rst.simd{n+stg.expn}.Data(end,j);
186                 if stg.simdetail
187                     rt.ssa(j,n+2*stg.expn) =...
188                         rst.simd{n+stg.expn}.Data(end,j);
189                 end
190             end
191         end
192     end
193
194     % Simulate the model
195     rst = f_sim(n,stg,rt,rst,mmf);
196
197     try
198         if stg.simdetail

```

(continues on next page)

(continued from previous page)

```

199         rst = f_sim(n+2*stg.expn,stg,rt,rst,mmf);
200     end
201     catch
202     end
203
204     % Check If the times of the simultaion output and the
↳simulation
205     % data from SBTAB match, if they don't it means that the
↳simulator
206     % didn't had enough time to run the model (happens in some
207     % unfavorable configurations of parameters, controlled by stg.
↳maxt
208     if size(Data(n).Experiment.t,1) ~=...
209         size(rst.simd{n}.Data(:,end),1)
210
211         % Set the simulation output to be 0, this is a non function
212         % value that the score function expects in simulations
↳that did
213         % not worked properly
214         rst.simd{n} = 0;
215     end
216     catch
217
218     % Set the simulation output to be 0, this is a non function
↳value
219     % that the score function expects in simulations that did not
220     % worked properly
221     rst.simd{n} = 0;
222 end
223 end
224 end

```

Prepares the simulation making sure that an equilibration is preformed when necessary before running the main simulation.

- **Inputs**

- *stg* - stg.folder_model, *stg.name*, *stg.partest*, *stg.tci*, *stg.tcm*, *stg.tcd*, *stg.exprun*, *stg.simcsl*, *stg.expn*
- parameters - (double) Set of parameters that we are working on

- **Created Variables**

- *rt*
 - * *rt.ssa* - (double) steady state amounts
 - * *rt.par* - (double) All parameters of the model, takes the default ones from SBtab and then replaces the ones being worked on.

- **Outputs**

- *rst* - *rst.simd*

- **Calls** - *f_sim*

- **Loads** - *data.mat*, *model.mat*

f_sim

Code

```

1 function rst = f_sim(exp_n,stg,rt,rst,mmf)
2
3 % Save variables that need to be maintained over multiple function calls
4 persistent model_run
5 persistent config_run
6
7 % If the function is called for the first time, load the appropriate
8 % and compile the code for simulation run
9 if isempty(model_run)
10
11     %Generate an empty array to be populated with the model suited for
12     %equilibration and experiment%
13     model_run = cell(1,stg.expn*2);
14     config_run = cell(1,stg.expn*2);
15
16     model_exp_default = mmf.model.data.model_exp.default;
17     model_exp_eq = mmf.model.data.model_exp.equilibration;
18     model_exp_detail = mmf.model.data.model_exp.detail;
19
20     % Iterate over the experiments thar are being run
21     for n = stg.exprun
22
23         if stg.simdetail
24             % Load the models for equilibrium
25             load(model_exp_detail + n + ".mat",'model_exp','config_exp
26
27             % Place the loaded models in the correct place in the
28             % models for equilibrium are set to be on the second half
29             % of
30             % the array
31             model_run{n+2*stg.expn} = model_exp;
32             config_run{n+2*stg.expn} = config_exp;
33
34             % Compile the matlab code that is going to simulate the
35             % using matlab built in function if the option is selected
36             % settings
37             if stg.sbioacc
38                 sbioaccelerate(model_run{n+2*stg.expn},config_run
39                 {n+2*stg.expn});
40             end
41             % Load the models for equilibrium
42             load(model_exp_eq + n + ".mat",'model_exp','config_exp')

```

(continues on next page)

(continued from previous page)

```

42
43     % Place the loaded models in the correct place in the array,
↪models
44     % for equilibrium are set to be on the second half of the array
45     model_run{n+stg.expn} = model_exp;
46     config_run{n+stg.expn} = config_exp;
47
48     % Compile the matlab code that is going to simulate the model,
↪using
49     % matlab built in function if the option is selected in,
↪settings
50     if stg.sbioacc
51         sbioaccelerate(model_run{n+stg.expn},config_run{n+stg.expn}
↪);
52     end
53
54     % Load the models for main run
55     load(model_exp_default + n + ".mat",'model_exp','config_exp')
56
57     % Place the loaded models in the correct place in the array,
↪models
58     % for main run are set to be on the first half of the array
59     model_run{n} = model_exp;
60     config_run{n} = config_exp;
61
62     % Compile the matlab code that is going to simulate the model,
↪using
63     % matlab built in function if the option is selected in,
↪settings
64     if stg.sbioacc
65         sbioaccelerate(model_run{n},config_run{n});
66     end
67 end
68 end
69
70 % substitute the start amount of the species in the model with the,
↪correct
71 % ones for simulations
72 set(model_run{exp_n}.species(1:size(rt.ssa(:,exp_n),1)),{'InitialAmount'
↪'},num2cell(rt.ssa(:,exp_n)))
73
74 % Substitute the values of the parameters in the model for the correct,
↪one
75 % for simultaions
76 set(model_run{exp_n}.parameters(1:size(rt.par,1)),{'Value'},
↪num2cell(rt.par))
77
78 %simulate the model using matlab built in function
79 rst.simd{exp_n} = sbiosimulate(model_run{exp_n},config_run{exp_n});
80 end

```

Simulates the model with the provided configurations. The first time it is run it loads a representation of the model and the simulation, and compiles this information to C code.

- **Inputs**

- `exp_n` - (double) Unique number to identify the model for each experiment or equilibrium reaction (it needs a new model object for each one)
- `stg` - `stg.expn`, `stg.folder_model`, `stg.name`, `stg.sbioacc`
- `rt`
 - * `rt.ssa` - (double) steady state amounts
 - * `rt.par` - (double) All parameters of the model, takes the default ones from SBtab and then replaces the ones being worked on.
- `rst` - `rst.simd`

- **Outputs**

- `rst` - `rst.simd`

- **Calls** - `Sbioaccelerate`, `Sbiosimulate`

- **Loads** - *Ready to run model*, *Ready to run model equilibration*

f_score**Code**

```

1 function rst = f_score(rst,stg,mmf)
2
3 persistent sbtab
4 persistent Data
5
6 data_model = mmf.model.data.data_model;
7
8 % Import the data on the first run
9 if isempty(Data)
10     load(data_model,'Data','sbtab')
11 end
12
13 % Iterate over the number of experiments
14 for n = stg.exprun
15
16     % Iterate over the number of datasets per experiment
17     for j = 1:size(sbtab.datasets(n).output,2)
18
19         % Calculate score per dataset if there are no errors
20         if rst.simd{n} ~= 0
21
22             data = Data(n).Experiment.x(:,j);
23             data_sd = Data(n).Experiment.x_SD(:,j);
24             number_points = size(Data(n).Experiment.x(:,j),1);
25             sim_results = f_normalize(rst,stg,n,j,mmf);
26             rst.xfinal{n,1}(j) = sim_results(end);
27
28             % Calculate score using formula that accounts for
29             ↪ normalization

```

(continues on next page)

(continued from previous page)

```

29         % with the starting point of the result
30         if stg.useLog == 4
31             rst.sd{n,1}(j) = sum(((data-sim_results)./...
32                 (data_sd*sqrt(number_points))).^2);
33         else
34             rst.sd{n,1}(j) = sum(((data-sim_results)./...
35                 (data_sd)).^2)/(number_points);
36         end
37
38         % If there are errors output a very high score value (10^
↪ 10)
39         elseif rst.simd{n} == 0 || rst.sd{n,1}(j) == inf
40
41             rst.sd{n,1}(j) = stg.errorscore;
42             rst.xfinal{n,1}(j) = 0;
43         end
44
45         % Calculate the log10 of dataset score if option selected
46         if stg.useLog == 1
47             rst.sd{n,1}(j) = max(0,log10(rst.sd{n,1}(j)));
48         end
49     end
50
51     % Calculate score per experiment
52     rst.se(n,1) = sum(rst.sd{n,1});
53
54     % Calculate the log10 of experiment score if option selected
55     if stg.useLog == 2
56         rst.se(n,1) = log10(rst.se(n,1));
57     end
58 end
59
60 % Calculate score per experiment
61 rst.st = sum(rst.se);
62
63 % Calculate the log10 of total score if option selected
64 if stg.useLog == 3
65     rst.st = log10(rst.st);
66 end
67 end

```

Uses the results from the simulation of the model and the Data provided via the SBTAB to calculate a score for a given parameter set.

- **Inputs**

- *rst* - *rst.simd*
- *stg* - *stg.folder_model*, *stg.name*, *stg.exprun*, *stg.useLog*

- **Outputs**

- *rst.st* - *rst.xfinal*, *rst.sd*, *rst.se*, *rst.st*

- **Calls**

- Loads - *data.mat*

Analysis

f_analysis

Code

```

1 function rst = f_analysis(stg,analysis,mmf,analysis_options)
2
3 if contains(analysis,analysis_options(1))
4     disp("Starting " + analysis_options(1))
5     rst.diag = f_diagnostics(stg,mmf);
6     disp(analysis_options(1) + " completed successfully")
7 end
8
9 if contains(analysis,analysis_options(2))
10    disp("Starting " + analysis_options(2))
11    rst.opt = f_opt(stg,mmf);
12    disp(analysis_options(2) + " completed successfully")
13 end
14
15 if contains(analysis,analysis_options(3))
16    disp("Starting " + analysis_options(3))
17    rst.gsa = f_gsa(stg,mmf);
18    disp(analysis_options(3) + " completed successfully")
19 end
20 end

```

Calls the proper analysis functions depending on the analysis that was chosen on the settings file. The supported analysis right now are:

- *Model diagnostics functions*
- *Optimization*
- *Global sensitivity analysis*
- **Inputs**
 - *stg*
 - analysis - (string) analysis being run (*stg.analysis*)
- **Outputs** - *rst*

Diagnostics

f_diagnostics

Code

```

1 function rst = f_diagnostics(stg,mmf)
2
3 % Run the model and obtain scores for fitness Multi Core

```

(continues on next page)

(continued from previous page)

```

4  if stg.optmc
5      disp("Running the model and obtaining Scores (Multicore)")
6
7      pa = stg.pa;
8      % Iterate over the parameter arrays to be tested
9      parfor n = stg.pat
10         [~,rst(n),~] = f_sim_score(pa(n,:),stg,mmf);
11     end
12
13     % Run the model and obtain scores for fitness single Core
14 else
15     disp("Running the model and obtaining Scores (Singlecore)")
16
17     % Iterate over the parameter arrays to be tested
18     for n = stg.pat
19         [~,rst(n),~] = f_sim_score(stg.pa(n,:),stg,mmf);
20     end
21 end
22 end

```

Used to understand the effects of different parameters sets on model behaviour or in comparing different parameters sets.

It loads the user defined configurations, performs all the needed simulations, and calculates scores of the error functions either per experimental output, per experiment, or in total (*check results*).

- **Inputs**

- *stg* - *stg.optmc* , *stg.pat*

- **Outputs** - *rst* (*diagnostics results*)

Optimization

f_opt

Code

```

1  function rst = f_opt(stg,mmf)
2  % Call function to run fmincon optimization algorithm if chosen in
   ↪ settings
3
4  if stg.fmincon
5      rst(1) = f_opt_fmincon(stg,mmf);
6  end
7
8  % Call function to run simulated annealing optimization algorithm if
   ↪ chosen
9  % in settings
10 if stg.sa
11     rst(2) = f_opt_sa(stg,mmf);

```

(continues on next page)

(continued from previous page)

```

12 end
13
14 % Call function to run pattern search optimization algorithm if chosen.
15 ↪ in
16 % settings
17 if stg.psearch
18     rst(3) = f_opt_psearch(stg,mmf);
19 end
20 % Call function to run genetic algorithm optimization if chosen in.
21 ↪ settings
22 if stg.ga
23     rst(4) = f_opt_ga(stg,mmf);
24 end
25 % Call function to run Particle swarm optimization algorithm if chosen.
26 ↪ in
27 % settings
28 if stg.pswarm
29     rst(5) = f_opt_pswarm(stg,mmf);
30 end
31 % Call function to run Surrogate optimization algorithm if chosen in
32 % settings
33 if stg.sopt
34     rst(6) = f_opt_sopt(stg,mmf);
35 end
36 end

```

Calls the correct optimizer or optimizers that have been chosen in the settings file.

- **Inputs**
 - *stg* - *stg.fmincon*, *stg.sa*, *stg.psearch*, *stg.ga*, *stg.pswarm*, *stg.sopt*
- **Outputs** - *rst* (*optimization results*)

f_opt_start

Code

```

1 function [spoint,spop] = f_opt_start(stg)
2
3 % Set the random seed for reproducibility
4 rng(stg.rseed);
5
6 % Optimization Start method 1
7 if stg.osm == 1
8
9     % Get a random starting point or group of starting points, if using
10    % multistart, inside the bounds
11    spoint = lhsdesign(stg.msts,stg.parnum).*(stg.ub-stg.lb)+stg.lb;
12

```

(continues on next page)

(continued from previous page)

```

13     % Get a group of random starting points inside the bounds
14     spop = lhsdesign(stg.popsiz, stg.parnum).*(stg.ub-stg.lb)+stg.lb;
15
16     % Optimization Start method 2
17 elseif stg.osm == 2
18
19     % Get a random starting point or group of starting points, if using
20     % multistart, near the best point
21     spoint = stg.bestpa - stg.dbpa +...
22             (stg.dbpa*2*lhsdesign(stg.msts, stg.parnum));
23
24     % Get a group of random starting points near the best point
25     spop = stg.bestpa - stg.dbpa +...
26           (stg.dbpa*2*lhsdesign(stg.popsiz, stg.parnum));
27 end
28 end

```

Creates the starting parameter set or sets of the optimizations, if single or multistart selected in settings file. It supports two different random distributions for the starting points.

- **Inputs**

- *stg* - *stg.rseed*, *stg.osm*, *stg.msts*, *stg.parnum*, *stg.ub*, *stg.lb*, *stg.popsiz*, *stg.bestpa*, *stg.dbpa*

- **Outputs**

- *spoint* - (double) starting parameter set for the optimization
- *spop* - (double) Starting parameter sets for multiple start optimizations

f_opt_fmincon/sa/psearch/ga/pswarm/sopt

Code

f_opt_fmincon

```

1 function rst = f_opt_fmincon(stg, mmf)
2
3 % Set the random seed for reproducibility
4 rng(stg.rseed);
5
6 % Set the starting point for the optimization
7 [startpoint, ~] = f_opt_start(stg);
8
9 % Get the optimization options from settings
10 options = stg.fm_options;
11 options.UseParallel = stg.optmc;
12
13 % Display console messages if chosen in settings
14 if stg.optcsl
15     options.Display = 'iter-detailed';

```

(continues on next page)

(continued from previous page)

```

16 end
17
18 % Display plots if chosen in settings
19 if stg.optplots
20     options.PlotFcn = ...
21         {@optimplotx,@optimplotfuncount,@optimplotfval,...
22         @optimplotstepsize,@optimplotfirstorderopt};
23 end
24
25 % Optimize the model with multiple starting points if chosen in
↳ settings
26 if stg.mst
27     parfor n = 1:stg.msts
28         disp(string(n))
29         [x(n,:),fval(n),exitflag(n),output(n)] =...
30             fmincon(@(x)f_sim_score(x,stg,mmf),startpoint(n,:),...
31             [],[],[],[],stg.lb,stg.ub,[],options);
32     end
33
34     % Optimize the model
35 else
36     [x(1,:),fval(1),exitflag(1),output(1)] =...
37         fmincon(@(x)f_sim_score(x,stg,mmf),startpoint(1,:),...
38         [],[],[],[],stg.lb,stg.ub,[],options);
39 end
40
41 % Save results
42 rst.name = 'fmincon';
43 rst.x = x;
44 rst.fval = fval;
45 rst.exitflag = exitflag;
46 rst.output = output;
47 end

```

f_opt_sa

```

1 function rst = f_opt_sa(stg,mmf)
2
3 % Set the random seed for reproducibility
4 rng(stg.rseed);
5
6 % Set the starting point for the optimization
7 [startpoint,~] = f_opt_start(stg);
8
9 % Get the optimization options from settings
10 options = stg.sa_options;
11 options.MaxTime = stg.optt;
12 options.InitialTemperature = ones(1,stg.parnum)*2;
13
14 % Display console messages if chosen in settings
15 if stg.optcsl
16     options.Display = 'iter';

```

(continues on next page)

(continued from previous page)

```

17 end
18
19 % Display plots if chosen in settings
20 if stg.optplots
21     options.PlotFcn =...
22         {@saplotbestf,@saplottemperature,@saplotf,@saplotstopping,
23         ↪ @saplotx};
24 end
25 % Optimize the model with multiple starting points if chosen in_
26 ↪ settings
27 if stg.mst
28     parfor n = 1:stg.msts
29         disp(string(n))
30         [x(n,:),fval(n),exitflag(n),output(n)] =...
31             simulannealbnd(@(x)f_sim_score(x,stg,mmf),startpoint(n,:),.
32             ↪ ...
33             stg.lb,stg.ub,options);
34     end
35     % Optimize the model
36 else
37     [x(1,:),fval(1),exitflag(1),output(1)] =...
38         simulannealbnd(@(x)f_sim_score(x,stg,mmf),startpoint(1,:),...
39         stg.lb,stg.ub,options);
40 end
41 % Save results
42 rst.name = 'Simulated annealing';
43 rst.x = x;
44 rst.fval = fval;
45 rst.exitflag = exitflag;
46 rst.output = output;
47 end

```

f_opt_psearch

```

1 function rst = f_opt_psearch(stg,mmf)
2
3 % Set the random seed for reproducibility
4 rng(stg.rseed);
5
6 % Set the starting point for the optimization
7 [startpoint,~] = f_opt_start(stg);
8
9 % Get the optimization options from settings
10 options = stg.psearch_options;
11 options.MaxTime = stg.optt;
12 options.UseParallel = stg.optmc;
13
14 % Display console messages if chosen in settings
15 if stg.optcsl

```

(continues on next page)

(continued from previous page)

```

16     options.Display = 'iter';
17 end
18
19 % Display plots if chosen in settings
20 if stg.optplots
21     options.PlotFcn = ...
22         {@psplotbestf,@psplotfuncount,@psplotmeshsize,@psplotbestx};
23 end
24
25 % Optimize the model with multiple starting points if chosen in_
↪ settings
26 if stg.mst
27     parfor n = 1:stg.msts
28         disp(string(n))
29         [x(n,:),fval(n),exitflag(n),output(n)] =...
30             patternsearch(@(x)f_sim_score(x,stg,mmf),startpoint(n,:),
↪ [],[], ...
31                 [],[],stg.lb,stg.ub,[],options);
32     end
33
34     % Optimize the model
35 else
36     [x(1,:),fval(1),exitflag(1),output(1)] =...
37         patternsearch(@(x)f_sim_score(x,stg,mmf),startpoint(1,:),[],[],
↪ ...
38             [],[],stg.lb,stg.ub,[],options);
39 end
40
41 % Save results
42 rst.name = 'Pattern search';
43 rst.x = x;
44 rst.fval = fval;
45 rst.exitflag = exitflag;
46 rst.output = output;
47 end

```

f_opt_ga

```

1  function rst = f_opt_ga(stg,mmf)
2
3  % Set the random seed for reproducibility
4  rng(stg.rseed);
5
6  % Get the optimization options from settings
7  options = stg.ga_options;
8  options.MaxTime = stg.optt;
9  options.UseParallel = stg.optmc;
10 options.PopulationSize = stg.popsiz;
11
12 % Display console messages if chosen in settings
13 if stg.optcsl
14     options.Display = 'iter';

```

(continues on next page)

(continued from previous page)

```

15 end
16
17 % Display plots if chosen in settings
18 if stg.optplots
19     options.PlotFcn = {@gaplotbestf,...
20         @gaplotexpectation,@gaplotrange,@gaplotscorediversity,...
21         @gaplotstopping,@gaplotscores,@gaplotdistance,...
22         @gaplotselection};
23 end
24
25 % Set the starting population for the optimization
26 [~,startpop] = f_opt_start(stg);
27 options.InitialPopulationMatrix = startpop;
28
29 % Optimize the model with multiple starting points if chosen in_
↪ settings
30 if stg.mst
31     parfor n = 1:stg.msts
32         disp(string(n))
33         [x(n,:),fval(n)] = ga(@(x)f_sim_score(x,stg,mmf),stg.parnum,...
34             [],[],[],[],stg.lb,stg.ub,[],options);
35     end
36
37     % Optimize the model
38 else
39     [x(1,:),fval(1)] = ga(@(x)f_sim_score(x,stg,mmf),stg.parnum,...
40         [],[],[],[],stg.lb,stg.ub,[],options);
41 end
42
43 % Save results
44 rst.name = 'Genetic algorithm';
45 rst.x = x;
46 rst.fval = fval;
47 rst.exitflag = [];
48 rst.output = [];
49 end

```

f_opt_pswarm

```

1 function rst = f_opt_pswarm(stg,mmf)
2
3 % Set the random seed for reproducibility
4 rng(stg.rseed);
5
6 % Get the optimization options from settings
7 options = stg.pswarm_options;
8 options.MaxTime = stg.optt;
9 options.UseParallel = stg.optmc;
10 options.SwarmSize = stg.popsiz;
11
12 % Display console messages if chosen in settings
13 if stg.optcsl

```

(continues on next page)

(continued from previous page)

```

14     options.Display = 'iter';
15 end
16
17 % Display plots if chosen in settings
18 if stg.optplots
19     options.PlotFcn = @pswplotbestf;
20 end
21
22 % Set the starting population for the optimization
23 [~,startpop] = f_opt_start(stg);
24 options.InitialSwarmMatrix = startpop;
25
26 % Optimize the model with multiple starting points if chosen in
  ↪ settings
27 if stg.mst
28     parfor n = 1:stg.msts
29         disp(string(n))
30         [x(n,:),fval(n),exitflag(n),output(n)] =...
31             particleswarm(@(x)f_sim_score(x,stg,mmf),...
32                 stg.parnum,stg.lb,stg.ub,options);
33     end
34
35     % Optimize the model
36 else
37     [x(1,:),fval(1),exitflag(1),output(1)] =...
38         particleswarm(@(x)f_sim_score(x,stg,mmf),...
39             stg.parnum,stg.lb,stg.ub,options);
40 end
41
42 % Save results
43 rst.name = 'Particle swarm';
44 rst.x = x;
45 rst.fval = fval;
46 rst.exitflag = exitflag;
47 rst.output = output;
48 end

```

f_opt_sopt

```

1  function rst = f_opt_sopt(stg,mmf)
2
3  % Set the random seed for reproducibility
4  rng(stg.rseed);
5
6  % Get the optimization options from settings
7  options = stg.sopt_options;
8  options.MaxTime = stg.optt;
9  options.UseParallel = stg.optmc;
10
11 % Display console messages if chosen in settings
12 if stg.optcsl
13     options.Display = 'iter';

```

(continues on next page)

(continued from previous page)

```

14 end
15
16 % Display plots if chosen in settings
17 if stg.optplots
18     options.PlotFcn = @surrogateoptplot;
19 end
20
21 % Set the starting population for the optimization
22 [~,startpop] = f_opt_start(stg);
23 options.InitialPoints = startpop;
24
25 % Optimize the model with multiple starting points if chosen in
  ↪ settings
26 if stg.mst
27     parfor n = 1:stg.msts
28         disp(string(n))
29         [x(n,:),fval(n),exitflag(n),output(n)] =...
30             surrogateopt(@(x)f_sim_score(x,stg,mmf),stg.lb,stg.ub,
  ↪ options);
31     end
32
33     % Optimize the model
34 else
35     [x(1,:),fval(1),exitflag(1),output(1)] =...
36         surrogateopt(@(x)f_sim_score(x,stg,mmf),stg.lb,stg.ub,options);
37 end
38
39 % Save results
40 rst.name = 'Surrogate optimization';
41 rst.x = x;
42 rst.fval = fval;
43 rst.exitflag = exitflag;
44 rst.output = output;
45 end

```

These functions call built in MATLAB® functions that perform parameter optimization . For further information relating to how these optimizers work please follow the links to the MATLAB® documentation. Optimizers used:

- f_opt_fmincon - fmincon
- f_opt_sa - Simulated annealing
- f_opt_psearch - Pattern search
- f_opt_ga - Genetic algorithm
- f_opt_pswarm - Particle swarm
- f_opt_sopt - Surrogate optimization
- **Inputs** - *stg*
- **Outputs** - *Optimization results*

Global Sensitivity Analysis

f_gsa

Code

```

1 function rst = f_gsa(stg,mmf)
2
3 rst = f_make_par_samples(stg);
4
5 rst = f_make_output_sample(rst,stg,mmf);
6
7 rst = f_calc_sensitivities(rst,stg);
8 end

```

Calls the global sensitivity analysis functions in the correct order.

f_make_par_samples

Code

```

1 function rst= f_make_par_samples(stg)
2 % Code inspired by Geir Halnes et al. 2009 paper. (Halnes, Geir, et al.
3   ↪ J.
4   ↪ comp. neuroscience 27.3 (2009): 471.)
5
6 % MAKE SAMPLE MATRICES
7 M1 = zeros(stg.sansamples, stg.parnum); % Pre-allocate memory for data
8 M2 = zeros(stg.sansamples, stg.parnum);
9 N = zeros(stg.sansamples, stg.parnum, stg.parnum);
10 rng(stg.rseed)
11
12 % Create a distribution for each parameter according to settings(Note
13   ↪ that
14   ↪ the sampling is being performed in log space as the parameters and
15   ↪ its
16   ↪ bounds are in log space)
17 for i=1:stg.parnum
18     % Uniform distribution truncated at the parameter bounds
19     if stg.sasamplemode == 0
20         M1(:,i) = stg.lb(i) +...
21             (stg.ub(i)-stg.lb(i)).*rand(1,stg.sansamples);
22         M2(:,i) = stg.lb(i) +...
23             (stg.ub(i)-stg.lb(i)).*rand(1,stg.sansamples);
24         % Normal distribution with mu as the best value for a
25         ↪ parameter and
26         ↪ sigma as stg.sasamplesigma truncated at the parameter bounds
27         elseif stg.sasamplemode == 1
28             pd(i) = makedist('Normal','mu',stg.bestpa(i),...
29                 'sigma',stg.sasamplesigma);
30             t(i) = truncate(pd(i),stg.lb(i),stg.ub(i));
31             r{i} = random(t(i),stg.sansamples,1);

```

(continues on next page)

(continued from previous page)

```

28     r2{i} = random(t(i),stg.sansamples,1);
29     M1(:,i) = r{i};
30     M2(:,i) = r2{i};
31     % Same as 1 without truncation
32     elseif stg.sasamplemode == 2
33         pd(i) = makedist('Normal','mu',stg.bestpa(i),...
34             'sigma',stg.sasamplesigma);
35         r{i} = random(pd(i),stg.sansamples,1);
36         r2{i} = random(pd(i),stg.sansamples,1);
37         M1(:,i) = r{i};
38         M2(:,i) = r2{i};
39         % Normal distribution centered at the mean of the parameter.
↳ bounds
40         % and sigma as stg.sasamplesigma truncated at the parameter.
↳ bounds
41     elseif stg.sasamplemode == 3
42         pd(i) = makedist('Normal','mu',...
43             stg.lb(i) + (stg.ub(i)-stg.lb(i))/2,'sigma',stg.
↳ sasamplesigma);
44         t(i) = truncate(pd(i),stg.lb(i),stg.ub(i));
45         r{i} = random(t(i),stg.sansamples,1);
46         r2{i} = random(t(i),stg.sansamples,1);
47         M1(:,i) = r{i};
48         M2(:,i) = r2{i};
49         % Same as 3 without truncation.
50     elseif stg.sasamplemode == 4
51         pd(i) = makedist('Normal','mu',...
52             stg.lb(i) + (stg.ub(i)-stg.lb(i))/2,'sigma',stg.
↳ sasamplesigma);
53         r{i} = random(pd(i),stg.sansamples,1);
54         r2{i} = random(pd(i),stg.sansamples,1);
55         M1(:,i) = r{i};
56         M2(:,i) = r2{i};
57     end
58 end
59
60 for i=1:stg.parnum
61     % Replace the i:th column in M2 by the i:th column from M1 to.
↳ obtain Ni
62     N(:,i,i) = M2;
63     N(:,i,i) = M1(:,i);
64 end
65
66 rst.M1=M1;
67 rst.M2=M2;
68 rst.N=N;

```

Creates parameter sets samples with *specific parameter distributions* that are used to perform the global sensitivity analysis.

- **Inputs**

- stg - *stg.sansamples, stg.parnum, stg.sasamplemode, stg.ub, stg.lb*

- **Outputs** - *M1, M2, N*

Code inspired by Geir Halnes et al. 2009 paper.

f_make_output_sample

Code

```

1  function rst = f_make_output_sample(rst,stg,mmf)
2  % Code inspired by Geir Halnes et al. 2009 paper. (Halnes, Geir, et al.
   ↪ J.
3  % comp. neuroscience 27.3 (2009): 471.)
4
5  nSamples = stg.sansamples;
6  nPars = stg.parnum;
7  parameter_array = zeros(nSamples,nPars);
8  progress = 1;
9  time_begin = datetime;
10 D = parallel.pool.DataQueue;
11
12 afterEach(D, @progress_track);
13
14 for i=1:nSamples
15     parameter_array(i,:) = rst.M1(i,:);
16 end
17
18 parfor i=1:nSamples
19     [~,~,RM1(i)] = f_sim_score(parameter_array(i,:),stg,mmf);
20     send(D, "GSA M1 ");
21 end
22 disp("GSA M1 Runtime: " + string(datetime - time_begin) + ...
23      " All " + nSamples + " samples executed")
24
25
26 for i=1:nSamples
27     fM1.sd(i,:) = [RM1(i).sd{:}];
28     fM1.se(i,:) = RM1(i).se(:);
29     fM1.st(i,:) = RM1(i).st;
30     fM1.xfinal(i,:) = [RM1(i).xfinal{:}];
31 end
32
33 rst.fM1 = fM1;
34 clear a FM1
35
36 for i=1:nSamples
37     parameter_array(i,:)= rst.M2(i,:);
38 end
39
40 progress = 1;
41 parfor i=1:nSamples
42     [~,~,RM2(i)] = f_sim_score(parameter_array(i,:),stg,mmf);
43     send(D, "GSA M2 ");
44 end

```

(continues on next page)

(continued from previous page)

```

45 disp("GSA M2 Runtime: " + string(datetime - time_begin) +...
46      " All " + nSamples + " samples executed")
47
48 for i=1:nSamples
49     fM2.sd(i,:) = [RM2(i).sd{:}];
50     fM2.se(i,:) = RM2(i).se(:);
51     fM2.st(i,:) = RM2(i).st;
52     fM2.xfinal(i,:) = [RM2(i).xfinal{:}];
53 end
54
55 rst.fM2 = fM2;
56 clear b FM2
57
58 for i=1:nSamples
59     for j=1:nPars
60         parameter_array(i,:,j)= rst.N(i,:,j);
61     end
62 end
63
64 progress = 1;
65 parfor i=1:nSamples
66     for j=1:nPars
67         [~,~,RN{i,j}] = f_sim_score(parameter_array(i,:,j),stg,mmf);
68     end
69     send(D, "GSA N ");
70 end
71 disp("GSA N Runtime: " + string(datetime - time_begin) +...
72      " All " + nSamples + " samples executed")
73
74 for i=1:nSamples
75     for j=1:nPars
76         fN.sd(i,:,j) = [RN{i,j}.sd{:}];
77         fN.se(i,:,j) = RN{i,j}.se(:);
78         fN.st(i,:,j) = RN{i,j}.st;
79         fN.xfinal(i,:,j) = [RN{i,j}.xfinal{:}];
80     end
81 end
82
83 rst.fN = fN;
84 clear c FN
85
86 function progress_track(name)
87     progress = progress + 1;
88     if mod(progress,ceil(nSamples/10)) == 0 && progress ~= nSamples
89         disp(name + "Runtime: " + string(datetime - time_begin) +..
90             " Samples:" + progress + "/" + nSamples)
91     end
92 end
93 end

```

For each parameter set given in the matrices *M1*, *M2*, and *N* it runs the function *f_sim_score* generating new matrices *fM1*, *fM2*, and *fN* respectively.

- **Inputs** - $M1$, $M2$, N , $stg.sansamples$, $stg.parnum$,
- **Outputs** - $fM1$, $fM2$, fN

Code inspired by Geir Halnes et al. 2009 paper.

f_calc_sensitivities

Code

```

1  function rst = f_calc_sensitivities(rst,stg)
2
3  rst = remove_sim_error(rst,stg);
4
5  [rst.SiQ,rst.SiTQ,rst.Si,rst.SiT] = bootstrap(rst,stg);
6  end
7
8  function [SiQ,SiTQ,Si,SiT]=bootstrap(rst,stg)
9  % calculates confidence intervals.
10 fM1 = rst.fM1;
11 fM2 = rst.fM2;
12 fN = rst.fN;
13
14 scores_names_list = ["sd","se","st","xfinal"];
15
16 [Si,SiT] = bootstrap_h(fM1,fM2,fN,stg,scores_names_list);
17
18 if (isempty(stg.gsabootstrapsize))
19     stg.gsabootstrapsize=ceil(sqrt(size(fM1.sd)));
20 end%if
21
22 fM1q = cell(stg.gsabootstrapsize,1);
23 fM2q = cell(stg.gsabootstrapsize,1);
24 fNq = cell(stg.gsabootstrapsize,1);
25
26 parfor j=1:stg.gsabootstrapsize
27     [SiQh{j},SiTQh{j}] = bootstrap_hq(fM1,fM2,fN,stg,scores_names_list,
28     ↪j);
29 end%parfor
30
31 for n = 1:size(scores_names_list,2)
32     for j=1:stg.gsabootstrapsize
33         eval("SiQ." + scores_names_list(n) + "(j,:,:)= SiQh{j}." + ↪
34         ↪scores_names_list(n) + ";")
35         eval("SiTQ." + scores_names_list(n) + "(j,:,:)= SiTQh{j}." + ↪
36         ↪scores_names_list(n) + ";")
37     end
38 end
39 end%function
40
41 function [Si,SiT] = bootstrap_h(fM1,fM2,fN,stg,scores_names)
42 for n = 1:size(scores_names,2)
43     eval("fM1h=fM1." + scores_names(n) + ";");

```

(continues on next page)

(continued from previous page)

```

41     eval("fM2h=fM2." + scores_names(n) + ";");
42     eval("fNh=fN." + scores_names(n) + ";");
43
44     [Sih,SiTh] = calcSobolSaltelli(fM1h,fM2h,fNh,stg);
45
46     eval("Si." + scores_names(n) + "=Sih;");
47     eval("SiT." + scores_names(n) + "=SiTh;");
48 end
49 end
50
51 function [Si,SiT] = bootstrap_hq(fM1,fM2,fN,stg,scores_names,j)
52 for n = 1:size(scores_names,2)
53     eval("fM1h=fM1." + scores_names(n) + ";");
54     eval("fM2h=fM2." + scores_names(n) + ";");
55     eval("fNh=fN." + scores_names(n) + ";");
56
57     rng(j*stg.rseed)
58     I=ceil(rand(size(fM1h,1),1)*size(fM1h,1));
59     fM1q = fM1h(I,:);
60     fM2q = fM2h(I,:);
61     fNq = fNh(I,:,:);
62
63     [Sih,SiTh] = calcSobolSaltelli(fM1q,fM2q,fNq,stg);
64     eval("Si." + scores_names(n) + "=Sih;");
65     eval("SiT." + scores_names(n) + "=SiTh;");
66 end
67 end
68
69 function [Si,SiT] = calcSobolSaltelli(fM1,fM2,fN,stg)
70 %Code inspired by Geir Halnes et al. 2009 paper. (Halnes, Geir, et al. ↵
↵ J. comp. neuroscience 27.3 (2009): 471.)
71
72 [Nsamples,Nvars,Npars]=size(fN);
73
74 if(stg.sasubmean) % Makes the model more stable
75     fM1 = fM1 - mean(fM1,1);
76     fM2 = fM2 - mean(fM2,1);
77     for i=1:Npars
78         fN(:, :, i) = fN(:, :, i) - mean(fN(:, :, i),1);
79     end
80 end
81
82 EY2 = mean(fM1.*fM2); % Valid definition (see Halnes et. al. Appendix)
83 VY = sum(fM1.^2)/(Nsamples-1) - EY2;
84 VYT = sum(fM2.^2)/(Nsamples-1) - EY2;
85
86 Si = zeros(Nvars,Npars);
87 SiT= zeros(Nvars,Npars);
88
89 for i=1:Npars
90     Si(:,i) = (sum(fM1.*fN(:, :, i))/(Nsamples-1) - EY2)./VY;
91     SiT(:,i) = 1 - (sum(fM2.*fN(:, :, i))/(Nsamples-1) - EY2)./VYT;

```

(continues on next page)

(continued from previous page)

```

92 end
93 end
94
95 function rst = remove_sim_error(rst,stg)
96 error=[];
97 error_helper=[];
98
99 for n = 1:size(rst.fM1.sd,1)
100     if max(rst.fM1.sd(n,:)) == stg.errorscore
101         error = [error,n];
102     end
103     if max(rst.fM2.sd(n,:)) == stg.errorscore
104         error = [error,n];
105     end
106     for m = 1:stg.parnum
107         if max(rst.fN.sd(n,:,m)) == stg.errorscore
108             error_helper = 1;
109         end
110     end
111     if error_helper == 1
112         error = [error,n];
113     end
114     error_helper = 0;
115 end
116
117 error = unique(error);
118
119 if ~isempty(error)
120     for n = size(error,2):-1:1
121         rst.fM1.sd(error(n),:) = [];
122         rst.fM2.sd(error(n),:) = [];
123         rst.fN.sd(error(n),:,:) = [];
124
125         rst.fM1.se(error(n),:) = [];
126         rst.fM2.se(error(n),:) = [];
127         rst.fN.se(error(n),:,:) = [];
128
129         rst.fM1.st(error(n),:) = [];
130         rst.fM2.st(error(n),:) = [];
131         rst.fN.st(error(n),:,:) = [];
132
133         rst.fM1.xfinal(error(n),:) = [];
134         rst.fM2.xfinal(error(n),:) = [];
135         rst.fN.xfinal(error(n),:,:) = [];
136     end
137 end
138 end

```

Takes the matrices *fM1*, *fM2*, and *fN* and calculates sensitivity indexes. It calculates indexes based on the following *outputs* of the *f_sim_score* function:

- The scores of each experimental output
- The scores of each experiment

- *The total score*
- *The value of each experimental outputs at the end of the simulation*
- **Inputs** - *fM1, fM2, fN, stg.sasubmean*
- **Outputs** - *SI, STI*

Code modified from the Geir Haldnes et al. 2009 paper.

References

Haldnes, G., Ulfhielm, E., Ljunggren, E.E., Kotaleski, J.H. and Rospars, J.P., 2009. Modelling and sensitivity analysis of the reactions involving receptor, G-protein and effector in vertebrate olfactory receptor neurons. *Journal of Computational Neuroscience*, 27(3), p.471.

Plots

f_plot

Code

```
1 function f_plot(rst,stg,mmf)
2
3 % Inform the user that the plots are being done
4 disp("Plotting ...")
5
6 data_model = mmf.model.data.data_model;
7
8 % Import the data on the first run
9 load(data_model,'Data','sbtabs')
10
11 % Generate figure with Scores
12 if isfield(rst,'diag')
13     f_plot_scores(rst.diag,stg,sbtabs)
14 end
15
16 % Generate figure with Inputs
17 if isfield(rst,'diag')
18     f_plot_inputs(rst.diag,stg,sbtabs)
19 end
20
21 % Generate figure with Outputs
22 if isfield(rst,'diag')
23     f_plot_outputs(rst.diag,stg,sbtabs,Data,mmf)
24 end
25
26 % Generate figure with input and Output of all experiments
27 if isfield(rst,'diag')
28     f_plot_in_out(rst.diag,stg,sbtabs,Data)
29 end
30
31 % Generate figure with optimization results
```

(continues on next page)

(continued from previous page)

```

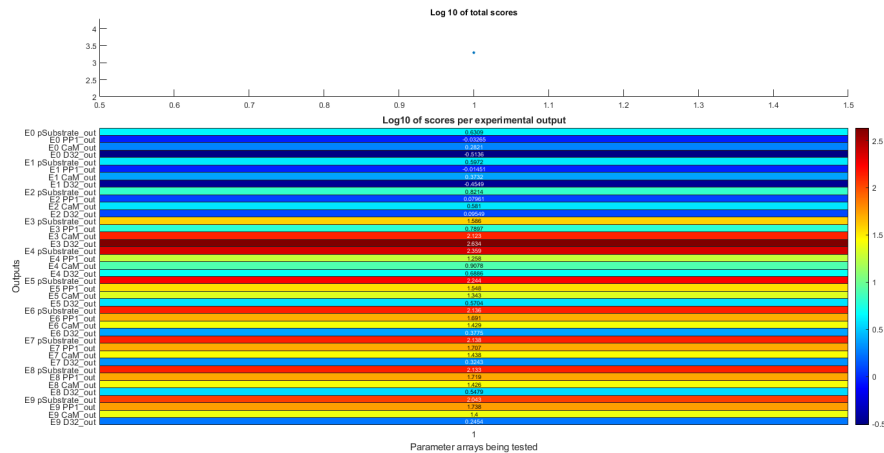
32 if isfield(rst,'opt')
33     f_plot_opt(rst,stg)
34 end
35
36 % Generate figures for Sensitivity Analysis
37 if isfield(rst,'gsa')
38     f_plot_gsa_sensitivities(rst.gsa,stg,sbtabs);
39 end
40 end

```

The function that calls all the custom plot functions when appropriate Plots diagnosis that are important to understand if everything is working as it was supposed, it , expected outputs, observed outputs and scores for the models and conditions specified.

- Inputs - *rst*, *stg*
- Outputs

Figure Scores



Total scores and scores per dataset given the parameters specified in *stg.pa*

Code Figure Scores

```

1 function f_plot_scores(rst,stg,sbtabs)
2 set(0,'defaultTextFontName', 'Helvetica')
3 set(0,'defaultAxesFontName', 'Helvetica')
4
5 % Generates a figure with Scores
6
7 % Inform the user that fig1 is being plotted
8 disp("Plotting Scores")
9
10 %Closes previous instances of the figure and generates a new one
11 figHandles = findobj('type', 'figure', 'name', 'Scores');
12 close(figHandles);
13 figure('WindowStyle', 'docked','Name','Scores','NumberTitle', 'off');
14

```

(continues on next page)

(continued from previous page)

```

15 % Generate top plot of figure 1
16 subplot(4,1,1)
17
18 % Plot the total scores of each parameter array to test
19 scatter(stg.pat,[rst(stg.pat).st],20,'filled')
20 ylabel('Total Score')
21 set(gca,'xtick',[])
22 set(gca,'FontSize',10,'Fontweight','bold')
23
24 % Choose correct title according to settings
25 if stg.useLog == 1
26     title("Sum of the Log base 10 of the Score of each Experimental_
↪Output")
27 elseif stg.useLog == 2
28     title("Sum of the Log base 10 of the Score of each Experiment")
29 elseif stg.useLog == 3
30     title("Log base 10 of sum of the Score of all Experiments")
31 elseif stg.useLog == 4 || stg.useLog == 0
32     title("Sum of the Score of all Experiments")
33 end
34
35 % Choose the bounds for the x axis so it aligns with the bottom plot
36 xlim([min(stg.pat)-0.5 max(stg.pat)+0.5])
37
38 % Generate bottom plot of figure 1
39 subplot(4,1,[2,3,4])
40
41 % Generate labels for left of heatmap (Experiment number Dataset_
↪number)
42 label = [];
43
44 % Iterate over the number of experiments
45 for n = stg.exprun
46
47     % Iterate over the number of datasets in each experiment
48     for j = 1:size(sbtabs.datasets(n).output,2)
49         label{size(label,2)+1} = {strrep("E" + (n-1) + " " + ...
50             string(rst(max(stg.pat)).simd{1,n}.DataNames(...
51                 end-size(sbtabs.datasets(n).output,2)+j)), "_", "\_")};
52     end
53 end
54
55 % Choose whether to use the score of each dataset or its log base 10
56 % according to settings
57 % if stg.useLog == 1 || stg.useLog == 4
58 headline = [];
59
60 % Iterate over the number of parameter arrays to test
61 for k = stg.pat
62     heatpoint{k} = [];
63
64     % Iterate over the number of experiments

```

(continues on next page)

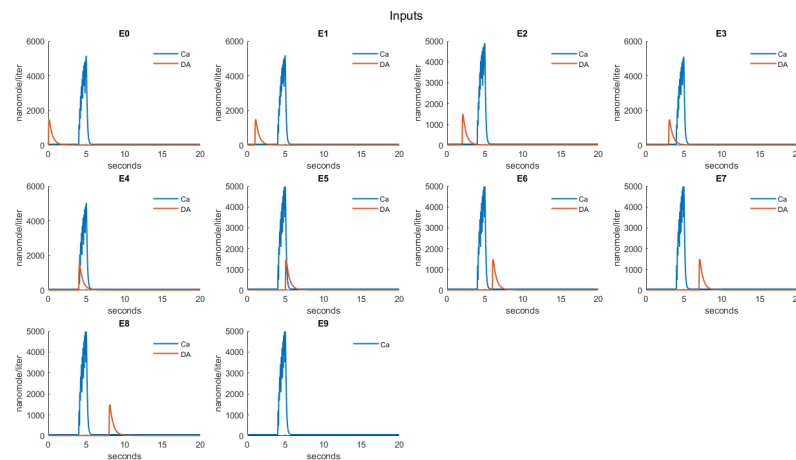
(continued from previous page)

```

65     for n = stg.exprun
66         % Get the score of each dataset
67         heatpoint{k} = [heatpoint{k},rst(k).sd{n,1}];
68     end
69
70     % Combine heatpoints in order to correctly display heatmap
71     heatline = [heatline;heatpoint{k}];
72 end
73
74 % Plot the heatmap
75 h = heatmap(transpose(heatline),'Colormap',turbo,'YDisplayLabels',
76     ↪label,...
77     'GridVisible','off','FontSize',10);
78 h.CellLabelFormat = '%.2e';
79
80 title("Score of each Experimental Output")
81 h.XLabel = 'Parameter arrays being tested';
82 h.YLabel = 'Experimental Outputs';
83
84 end

```

Figure Inputs



Checks inputs to the model

Code Figure Inputs

```

1 function f_plot_inputs(rst,stg,sbtab)
2 % Generates a figure with Inputs, one subplot per experiment
3
4 % Inform the user that fig2 is being plotted

```

(continues on next page)

(continued from previous page)

```

5  disp("Plotting Inputs")
6
7  plot_n = 1;
8  fig_n = 0;
9  % Iterate over the number of experiments
10 for n = stg.exprun
11
12     % Generate the right amount of figures for all plots and
    ↳ calculates
13     % proper subplotting position
14     fig_n = f_get_subplot(size(stg.exprun,2),plot_n,fig_n,"Inputs");
15
16     plot_n = plot_n +1;
17
18     hold on
19
20     % Iterate over the number of inputs in each experiment
21     for j = 1:size(sbtabs.datasets(n).input,2)
22
23         % Iterate over the number of parameter arrays to test
24         for m = stg.pat
25
26             % (Until a non broken simulation is found)
27             if rst(m).simd{1,n} ~= 0
28
29                 % Plot the inputs to each experiment
30                 plot(rst(m).simd{1,n}.Time,rst(m).simd{1,n}.
    ↳ Data(1:end,...
31                     str2double(strrep(sbtabs.datasets(n).input(j),'S',
    ↳ '))+1),'LineWidth',1.5)
32
33                 % Get the correct label for each input of the
    ↳ experiment
34                 labelfig2(j) = rst(m).simd{1,n}.
    ↳ DataNames(str2double(...
35                     strrep(sbtabs.datasets(n).input(j),'S','')+1);
36
37                 ylabel(string(rst(m).simd{1,n}.DataInfo{...
38                     str2double(strrep(sbtabs.datasets(n).input(j),'S',
    ↳ '))+1,1}.Units))
39
40                 break
41             end
42         end
43     end
44
45     xlabel('seconds')
46     % Add a legend to each plot
47     legend(labelfig2)
48     legend boxoff
49     clear labelfig2
50

```

(continues on next page)

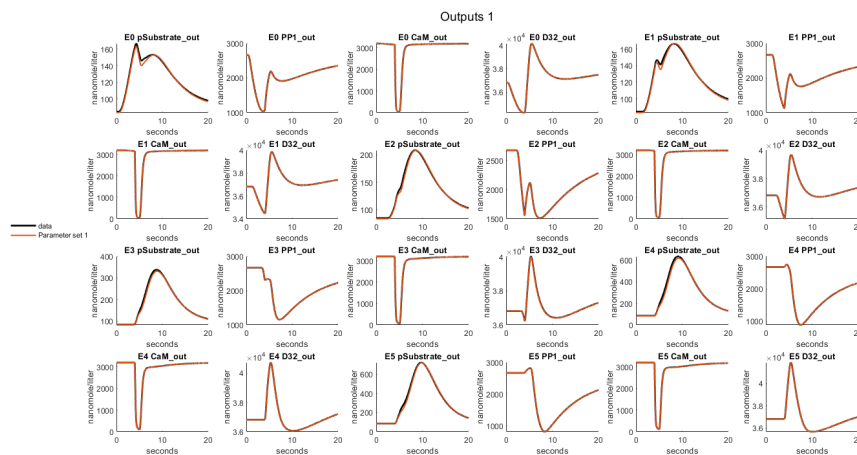
(continued from previous page)

```

51 ylim([0 inf])
52
53 % Add a title to each plot
54 title("E"+(n-1))
55
56 hold off
57 end
58
59 end

```

Figure Outputs



Expected outputs, observed outputs

Code Figure Outputs

```

1 function f_plot_outputs(rst,stg,sbtab,Data,mmf)
2 % Generates a figure with Outputs, one subplot per experimental output
3
4 % Inform the user that fig3 is being plotted
5 disp("Plotting Outputs")
6
7 % Get the total number of outputs to set the total number of plots
8 [plot_tn,~] = f_get_outputs(stg,sbtab);
9 plot_n = 1;
10 fig_n = 0;
11
12 % Iterate over the number of experiments
13 for n = stg.exprun
14
15     % Iterate over the number of datasets in each experiment
16     for j = 1:size(sbtab.datasets(n).output,2)

```

(continues on next page)

(continued from previous page)

```

17
18      % Generate the right amount of figures for all plots and calculates
19      % proper subplotting position
20      fig_n = f_get_subplot(plot_tn,plot_n,fig_n,"Outputs");
21
22      % Add a legend to the figure
23      if mod(plot_n,24) == 1
24          Lgnd = legend('show');
25          Lgnd.Position(1) = 0;
26          Lgnd.Position(2) = 0.5;
27          legend boxoff
28      end
29
30      plot_n = plot_n + 1;
31
32      hold on
33
34      % Iterate over the number of parameter arrays to test
35      for m = stg.pat
36          % (Until a non broken simulation is found)
37          if rst(m).simd{1,n} ~= 0
38
39              time = rst(m).simd{1,n}.Time;
40              data = Data(n).Experiment.x(:,j);
41
42              data_SD = Data(n).Experiment.x_SD(:,j);
43
44              % Plot the outputs to each dataset (new subplots) as they
45              % are given in the data provided in sbtab
46              scatter(time,data,'filled','k',...
47                  'DisplayName','data')
48
49              errorbar(time,data,data_SD, 'vertical', 'k',
50      ↪ 'LineStyle', 'none', 'LineWidth',1);
51
52              break
53          end
54      end
55
56      % Iterate over the number of parameter arrays to test
57      for m = stg.pat
58          % Plot only if the simulation was successful
59          if rst(m).simd{1,n} ~= 0
60
61              time = rst(m).simd{1,n}.Time;
62              [sim_results,~] = f_normalize(rst(m),stg,n,j,mmf);
63              if stg.simdetail
64                  time_detailed = rst(m).simd{1,n+2*stg.expn}.Time;
65                  [~,sim_results_detailed]= f_normalize(rst(m),stg,n,j,
66      ↪ mmf);
67              end

```

(continues on next page)

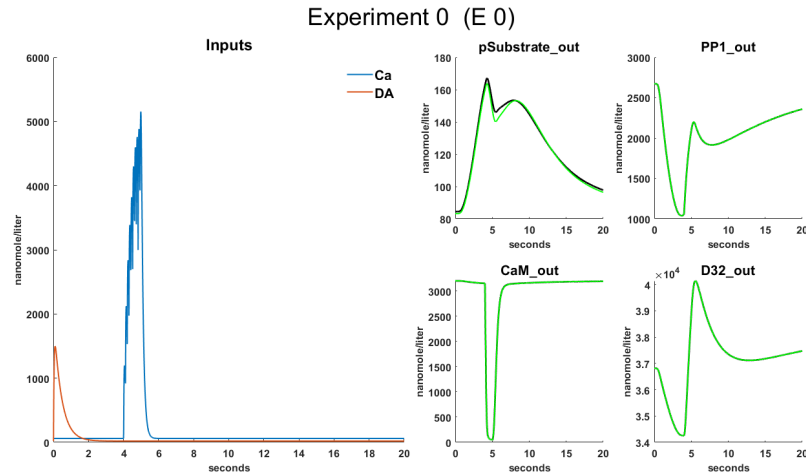
(continued from previous page)

```

67
68     % Plot the outputs to each dataset (new subplots) and
69     % parameter array to test that are simulated using
70     % Simbiology
71     if stg.simdetail
72         plot(time_detailed,...
73              sim_results_detailed,'DisplayName',...
74              string("Parameter set "+m),'LineWidth',1.5)
75     else
76
77         plot(time,...
78              sim_results,'DisplayName',...
79              string("Parameter set "+m),'LineWidth',1.5)
80     end
81
82     ylabel(string(rst(m).simd{1,n}.DataInfo{end-...
83                size(sbtabs.datasets(n).output,2)+j,1}.Units))
84
85     end
86
87     hold off
88
89     xlabel('seconds')
90
91     if stg.simdetail
92         ylim([min([0,min(sim_results_detailed),min(sim_results),
93     ↪ min(data-data_SD),min(data)]) inf])
94     else
95         ylim([min([0,min(sim_results),min(data-data_SD),min(data)])
96     ↪ inf])
97     end
98
99     % Choose correct title according to settings
100    if stg.plotln == 1
101        title("E" + (n-1) + " " +...
102              strrep(string(sbtabs.datasets(n).output_name{1,j}),'_','\_'))
103    else
104        title("E" + (n-1) + " " +...
105              string(sbtabs.datasets(n).output{1,j}))
106    end
107
108    % Choose number of decimal places for y axis
109    ytickformat('%0.2g')
110
111    end
112 end

```

Figure Input and Outputs per experiment



Combined figure of the inputs and outputs for each experiment, on the left side we have the inputs of the experiment and on the right side the outputs

Code Figure Input and Outputs

```

1 function f_plot_in_out(rst,stg,sbtabs,Data)
2 % Generates a figure with input and Output of all experiments on the left
3 % side it plots the inputs of the experiment and on the right side it plots
4 % the outputs
5
6 for n = stg.exprun
7
8     helper = 1;
9     f_plot_in_out_left(rst,stg,sbtabs,helper,...
10         size(sbtabs.datasets(n).output,2) > 4)
11
12     for j = 1:size(sbtabs.datasets(n).output,2)
13
14         if j/4 > helper
15             helper = helper +1;
16             f_plot_in_out_left(rst,stg,sbtabs,helper,...
17                 size(sbtabs.datasets(n).output,2) > 4)
18         end
19
20         if size(sbtabs.datasets(n).output,2) == 1
21             subplot(1,2,j+ceil(j/(2/2))*1)
22         elseif size(sbtabs.datasets(n).output,2) == 2
23             subplot(2,2,j+ceil(j/(2/2))*1)
24         elseif size(sbtabs.datasets(n).output,2) > 2 &&...
25             size(sbtabs.datasets(n).output,2) <= 4
26             subplot(2,4,j+ceil(j/(4/2))*2)
27         elseif size(sbtabs.datasets(n).output,2) > 4
28             subplot(2,4,j+ceil(j/(4/2))*2-helper*8+8)
29         end
30
31         hold on
32
33         % Iterate over the number of parameter arrays to test

```

(continues on next page)

(continued from previous page)

```

34     for m = stg.pat
35         % (Until a non broken simulation is found)
36         if rst(m).simd{1,n} ~= 0
37
38             % Plot the outputs to each dataset (new subplots) as they
39             % are given in the data provided in sbtab
40
41             time = rst(m).simd{1,n}.Time;
42             data = Data(n).Experiment.x(:,j);
43             data_SD = Data(n).Experiment.x_SD(:,j);
44
45             scatter(time,data,'filled','k',...
46                 'DisplayName','data')
47
48             errorbar(time,data,data_SD, 'vertical', 'k',
49 ← 'LineStyle', 'none', 'LineWidth',1);
50             break
51         end
52     end
53     % Iterate over the number of parameter arrays to test
54     for m = stg.pat
55
56         % Plot only if the simulation was successful
57         if rst(m).simd{1,n} ~= 0
58
59             time = rst(m).simd{1,n}.Time;
60             [sim_results] = f_normalize(rst(m),stg,n,j);
61
62             if stg.simdetail
63                 time_detailed = rst(m).simd{1,n+2*stg.expn}.Time;
64                 [~,sim_results_detailed]= f_normalize(rst(m),stg,n,j);
65             end
66
67             % Plot the outputs to each dataset (new subplots) and
68             % parameter array to test that are simulated using
69             % Simbiology
70             if stg.simdetail
71                 plot(time_detailed,...
72                     sim_results_detailed,'DisplayName',...
73                     string("Parameter set "+m),'LineWidth',1.5)
74             else
75                 plot(time,...
76                     sim_results,...
77                     'DisplayName',string("Parameter set "+m),...
78                     'LineWidth',1.5)
79             end
80
81             ylabel(string(rst(m).simd{1,n}.DataInfo{end-...
82                 size(sbtabs.datasets(n).output,2)+j,1}.Units),...
83                 'FontSize', 12,'Fontweight','bold')
84         end

```

(continues on next page)

(continued from previous page)

```

85     end
86
87     hold off
88
89     set(gca,'FontSize',12,'Fontweight','bold')
90
91     if stg.simdetail
92         ylim([min([0,min(sim_results_detailed),min(sim_results),
93 ←min(data-data_SD),min(data)]) inf])
94     else
95         ylim([min([0,min(sim_results),min(data-data_SD),min(data)]) ←
96 ←inf])
97     end
98
99     xlabel('seconds','FontSize', 12,'Fontweight','bold')
100
101     % Choose correct title according to settings
102     if stg.plotoln == 1
103         title(strrep(string(sbtabs.datasets(n).output_name{1,j}),'_','...
104         '\_'),'FontSize', 16,'Fontweight','bold')
105     else
106         title(string(sbtabs.datasets(n).output{1,j}),'FontSize', 16,...
107         'Fontweight','bold')
108     end
109
110     % Choose number of decimal places for y axis
111     ytickformat('%0.2g')
112 end
113
114 function f_plot_in_out_left(rst,stg,sbtabs,helper,reuse)
115     if reuse
116         figHandles = findobj('type','figure','name',"E " + (n-1) + ...
117         " " + helper);
118         close(figHandles);
119         figure('WindowStyle','docked','Name',"E " + (n-1) + " " + ...
120         helper,'NumberTitle','off');
121         sgtitle("Experiment " + (n-1) + " " + helper + " (E " + ...
122         (n-1) + " " + helper + ")", 'FontSize', 28);
123     else
124         figHandles = findobj('type','figure','name',"E " + (n-1));
125         close(figHandles);
126         figure('WindowStyle','docked','Name',"E " + (n-1),...
127         'NumberTitle','off');
128         sgtitle("Experiment " + (n-1) + " (E " + (n-1) + ...
129         ")", 'FontSize', 28);
130     end
131
132     subplot(2,4,[1,2,5,6])
133
134     hold on
135     for o = 1:size(sbtabs.datasets(n).input,2)

```

(continues on next page)

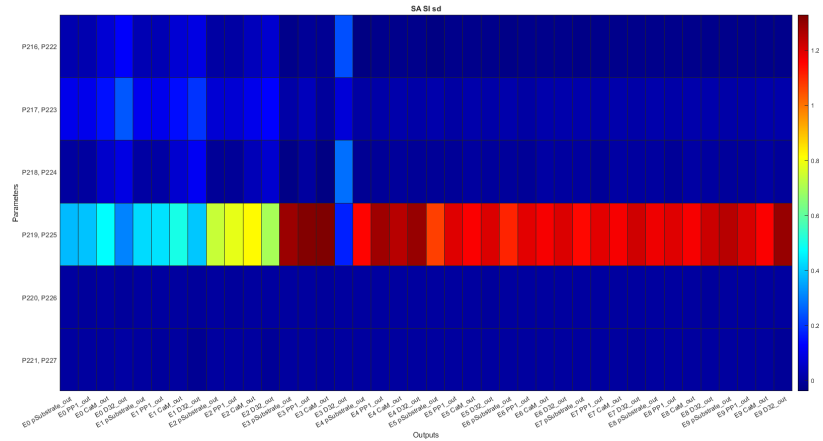
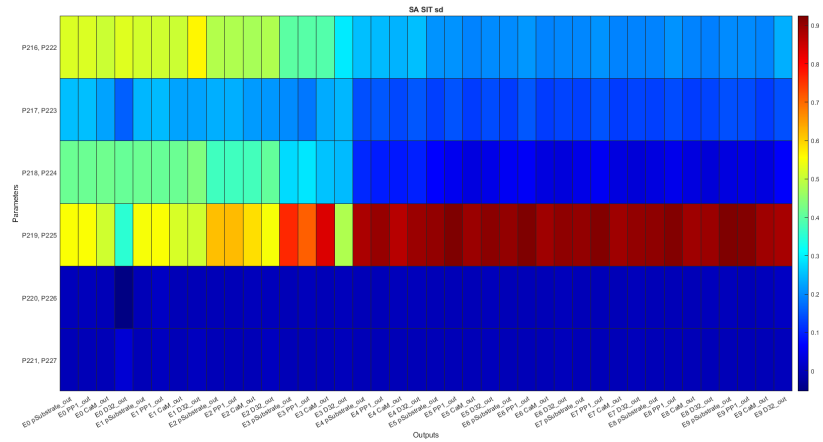
(continued from previous page)

```

135     for p = stg.pat
136
137         % (Until a non broken simulation is found)
138         if rst(p).simd{1,n} ~= 0
139             % Plot the inputs to each experiment
140             plot(rst(p).simd{1,n}.Time,rst(p).simd{1,n}.Data(1:end,
141 ↪ ..
142             str2double(strrep(sbtabs.datasets(n).input(o), 'S', '
143 ↪ ')...
144             )+1), 'LineWidth', 1.5)
145
146             % Get the correct label for each input of the
147             % experiment
148             labelfig2(o) = rst(p).simd{1,n}.DataNames(str2double(...
149             strrep(sbtabs.datasets(n).input(o), 'S', ''')+1);
150
151             ylabel(string(rst(p).simd{1,n}.DataInfo{...
152             str2double(strrep(sbtabs.datasets(n).input(o), 'S', '
153 ↪ '))+1,1}.Units),...
154             'FontSize', 12, 'Fontweight', 'bold')
155
156             break
157         end
158     end
159
160     set(gca, 'FontSize', 12, 'Fontweight', 'bold')
161
162     xlabel('seconds', 'FontSize', 12, 'Fontweight', 'bold')
163     % Add a legend to each plot
164     legend(labelfig2, 'FontSize', 16, 'Fontweight', 'bold')
165     legend boxoff
166     clear labelfig2
167
168     ylim([0 inf])
169
170     % Add a title to each plot
171     title("Inputs", 'FontSize', 18, 'Fontweight', 'bold')
172
173     hold off
174 end
175 end

```

Figure Sensitivity Analysis S_i

Figure Sensitivity Analysis S_{Ti} 

Code figures SA

```

1 function f_plot_gsa_sensitivities(rst,stg,sbtav)
2 % Generates figures for Sensitivity Analysis
3
4 % Get the total number of outputs
5 [~,outputNames.sd] = f_get_outputs(stg,sbtav);
6
7 for n = 1:size(outputNames.sd,2)
8     outputNames.sd{n}{:} = strrep(outputNames.sd{n}{:}, "_", "\_");
9 end
10 for n = stg.exprun
11     outputNames.se{n} = "E " + string(n-1);
12 end
13
14 outputNames.xfinal = outputNames.sd;
15
16 parNames = cell(1,stg.parnum);
17 parNames2 = cell(1,stg.parnum);
18
19 for n = 1:stg.parnum
20     parNames{n} = char("P" + find(stg.partest==n));

```

(continues on next page)

(continued from previous page)

```

21 end
22
23 for n = 1:size(parNames,2)
24     parNames2{n} = string(parNames{n}(1,:));
25     for m = 2:size(parNames{n},1)
26         parNames2{n} = string(parNames2{n}) + ", " +...
27             string(parNames{n}(m,:));
28     end
29 end
30
31 % Bootstrapping quartile mean of first order Sensitivity index for score
32 % per Experimental Output
33 f_generate_plot(rst,stg,outputNames,parNames2,...
34     "Si seo bm",...
35     "First order Sensitivities calculated using the Score of each_
↳ Experimental Output (Bootstrapping Mean)",...
36     "outputNames.sd",...
37     "transpose(reshape(mean(rst.SiQ.sd(:,:,1:stg.parnum)),[size(rst.SiQ.sd,
↳ 2),size(rst.SiQ.sd,3)]))")
38
39 % Bootstrapping quartile mean of total order Sensitivity index for score
40 % per Experimental Output
41 f_generate_plot(rst,stg,outputNames,parNames2,"SiT seo bm",...
42     "Total order Sensitivities calculated using the Score of each_
↳ Experimental Output (Bootstrapping Mean)",...
43     "outputNames.sd",...
44     "transpose(reshape(mean(rst.SiTQ.sd(:,:,1:stg.parnum)),[size(rst.SiQ.sd,
↳ 2),size(rst.SiQ.sd,3)]))")
45
46 % Bootstrapping quartile mean of first order Sensitivity index for score
47 % per Experiments
48 f_generate_plot(rst,stg,outputNames,parNames2,"Si se bm",...
49     "First order Sensitivities calculated using the Score of each_
↳ Experiment (Bootstrapping Mean)",...
50     "outputNames.se",...
51     "transpose(reshape(mean(rst.SiQ.se(:,:,1:stg.parnum)),[size(rst.SiQ.se,
↳ 2),size(rst.SiQ.se,3)]))")
52
53 % Bootstrapping quartile mean of total order Sensitivity index for score
54 % per Experiments
55 f_generate_plot(rst,stg,outputNames,parNames2,"SiT se bm",...
56     "Total order Sensitivities calculated using the Score of each_
↳ Experiment (Bootstrapping Mean)",...
57     "outputNames.se",...
58     "transpose(reshape(mean(rst.SiTQ.se(:,:,1:stg.parnum)),[size(rst.SiQ.se,
↳ 2),size(rst.SiQ.se,3)]))")
59
60 % Bootstrapping quartile mean of first order Sensitivity index for the
61 % final points of the simulations for the output beeing measured
62 f_generate_plot(rst,stg,outputNames,parNames2,"Si xfinal bm",...
63     "First order Sensitivities calculated using the final value of each_
↳ Experimental Output (Bootstrapping Mean)",...

```

(continues on next page)

(continued from previous page)

```

64     "outputNames.xfinal",...
65     "transpose(reshape(mean(rst.SiQ.xfinal(:,:,1:stg.parnum)),[size(rst.SiQ.
↳ xfinal,2),size(rst.SiQ.xfinal,3)]))")
66
67 % Bootstrapping quartile mean of total order Sensitivity index for the
68 % final points of the simulations for the output beeing measured
69 f_generate_plot(rst,stg,outputNames,parNames2,"SiT xfinal bm",...
70     "Total order Sensitivities calculated using the final value of each_
↳ Experimental Output (Bootstrapping Mean)",...
71     "outputNames.xfinal",...
72     "transpose(reshape(mean(rst.SiTQ.xfinal(:,:,1:stg.parnum)),[size(rst.
↳ SiQ.xfinal,2),size(rst.SiQ.xfinal,3)]))")
73
74 figHandles = findobj('type', 'figure', 'name', 'Si,SiT');
75 close(figHandles);
76 figure('WindowStyle', 'docked','Name','Si,SiT', 'NumberTitle', 'off');
77
78 for n = 1:size(parNames2,2)
79     a{n} = char(parNames2{n});
80 end
81
82 a = categorical(a,a);
83
84 bar(a,[transpose(rst.Si.st(:,1:stg.parnum)),...
85     transpose(rst.SiT.st(:,1:stg.parnum))])
86 xlabel('Parameters');
87 ylabel('Sensitivity');
88 title('Sensitivities calculated using the sum of the Score of all_
↳ Experiments');
89 legend({'SI', 'SiT'});
90 legend boxoff
91
92 figHandles = findobj('type', 'figure', 'name', 'Si,SiT b');
93 close(figHandles);
94 figure('WindowStyle', 'docked','Name','Si,SiT b', 'NumberTitle', 'off');
95
96 T = [];
97
98 for n = 1:size(a,2)
99     for m = 1:size(rst.SiQ.st(:,n),1)
100         T = [T;table(rst.SiQ.st(m,n),a(n),"Si")];
101     end
102 end
103 for n = 1:size(a,2)
104     for m = 1:size(rst.SiTQ.st(:,n),1)
105         T = [T;table(rst.SiTQ.st(m,n),a(n),"SiT")];
106     end
107 end
108
109 boxchart(T.Var2,T.Var1,'GroupByColor',T.Var3,'MarkerStyle','.',
↳ 'JitterOutliers','on')
110 xlabel('Parameters');

```

(continues on next page)

(continued from previous page)

```

111 ylabel('Sensitivity');
112 title('Sensitivities calculated using the sum of the Score of all_
↳ Experiments (Bootstrapping)');
113 legend({'Si', 'SiT'}, 'Location', 'best');
114 legend boxoff
115 end
116
117 function f_generate_plot(rst,stg,outputNames,parNames2,name,title,...
118     helper1,helper2)
119
120 eval("figHandles = findobj('type', 'figure', 'name', '"' + name + "');")
121 close(figHandles);
122 eval("figure('WindowStyle', 'docked','Name','" + name + "...
123     "','NumberTitle', 'off');")
124
125 heatmap_fixer = eval(helper1);
126 heatmap_fixer=heatmap_fixer(~cellfun('isempty',heatmap_fixer));
127
128 heatmap_fixer2 = eval(helper2);
129 heatmap_fixer2 = heatmap_fixer2(:,all(~isnan(heatmap_fixer2)));
130
131 h = heatmap(heatmap_fixer,parNames2,heatmap_fixer2,'Colormap',turbo,...
132     'ColorLimits',[0 1],'GridVisible','off');
133 h.CellLabelFormat = '%.2f';
134
135 eval(" h.Title = "" + title + """);")
136 h.XLabel = 'Outputs';
137 h.YLabel = 'Parameters';
138 end

```

- Calls
- Loads - *data.mat*

f_get_subplot

Code

```

1 function fig_n = f_get_subplot(plot_tn,plot_n,fig_n,fig_name)
2
3 size_x = 4;
4 size_y = 6;
5 size_t = 24;
6 ratio_1 = 2;
7 ratio_2 = 3;
8
9 % If the amount of plots is bigger than the maximum amount of plots_
↳ per
10 % figure subdivide the plots to more than one figure
11 if plot_tn > 24
12
13     % Generate a new figure for the first plot and each time the_
↳ number of

```

(continues on next page)

(continued from previous page)

```

14      % plots is greater than figure number divided by max plot number.
↪ per
15      % figure
16      if mod(plot_n-1,24) == 0
17
18          fig_n = fig_n + 1;
19
20          %Close previous instances of the figure and generates a new one
21          figHandles = findobj('type', 'figure', 'name', fig_name + " "
↪ + fig_n);
22          close(figHandles);
23          figure('WindowStyle', 'docked', 'Name', fig_name + " " + fig_n,
↪ 'NumberTitle', 'off');
24          sgtitle(fig_name + " " + fig_n);
25      end
26
27      % Get the correct subplotting position for each plot
28      if plot_tn/24 < fig_n
29          subplot(ceil(sqrt((plot_tn-(fig_n-1)*24)/6)*2),ceil(sqrt((plot_
↪ tn-(fig_n-1)*24)/6)*3),plot_n-(fig_n-1)*24)
30      else
31          subplot(ceil(sqrt(24/6)*2),ceil(sqrt(24/6)*3),plot_n-(fig_n-
↪ 1)*24)
32      end
33
34  else
35
36      % Generate a new figure for the first plot
37      if mod(plot_n-1,24) == 0
38
39          %Close previous instances of the figure and generates a new one
40          figHandles = findobj('type', 'figure', 'name', fig_name);
41          close(figHandles);
42          figure('WindowStyle', 'docked', 'Name', fig_name, 'NumberTitle',
↪ 'off');
43          sgtitle(fig_name);
44
45      end
46
47      % Get the correct subplotting position for each plot
48      subplot(ceil(sqrt(plot_tn/6)*2),ceil(sqrt(plot_tn/6)*3),plot_n)
49
50  end
51  end

```

- Inputs
- Outputs
- Calls
- Loads

General purpose

f_save_analysis

Code

```

1  function f_save_analysis(stg,sb,rst,mmf)
2
3  Results_Folder = mmf.model.results.main;
4  Analysis_folder = mmf.model.results.analysis.main;
5  Analysis_date_folder = mmf.model.results.analysis.date.main;
6
7  [~,~] = mkdir(Results_Folder);
8  [~,~] = mkdir(Analysis_folder);
9  [~,~] = mkdir(Analysis_date_folder);
10 addpath(Analysis_date_folder)
11
12 save (Analysis_date_folder + "Analysis.mat",'stg','sb','rst');
13 end

```

- Inputs
- Outputs
- Calls
- Loads

f_save_plots

Code

```

1  function f_save_plots(mmf)
2
3  Analysis_date_folder = mmf.model.results.analysis.date.main;
4
5  FigList = findobj(allchild(0), 'flat', 'Type', 'figure');
6
7  [~,~] = mkdir(Analysis_date_folder);
8
9  savefig(FigList(end:-1:1),...
10         Analysis_date_folder + "All_figures.fig");
11
12 for iFig = 1:length(FigList)
13     FigHandle = FigList(iFig);
14     FigName    = get(FigHandle, 'Name');
15
16     saveas(FigHandle, Analysis_date_folder + FigName + ".png")
17 end
18 end

```

- Inputs
- Outputs
- Calls

- Loads

f_get_outputs

Code

```
1 function [nOutputs,outputNames] = f_get_outputs(stg,sbtab)
2
3 persistent n_out
4 persistent out_name
5
6 if isempty(n_out)
7     n_out = 0;
8     out_name = [];
9     for n = stg.exprun
10         for j = 1:size(sbtab.datasets(n).output,2)
11             n_out = n_out + 1;
12             out_name{n_out} = {"E" + (n-1) + " " + string(sbtab.
↳ datasets(n).output{1,j})};
13         end
14     end
15 end
16
17 nOutputs = n_out;
18 outputNames = out_name;
19 end
```

- Inputs
- Outputs
- Calls
- Loads

3.2.6 Settings file

A place for the user to define all the relevant properties of model simulation that are not stored in SBtab. This are usually things that need to change during optimizations or model development.

These settings files can be found can be found on the respective model repository in the directory “Matlab/Settings”, in the example model from our main repository in the directory “Matlab/model/Model_Example/Matlab/Settings”, or by following these links:

- [Example model settings files](#)
- [Fujita_2010 model settings files](#)
- [Nair_2016 model settings files](#)
- [Viswan_2018 model settings files](#)

Default settings code

```

1 function [stg] = default_settings()
2
3 %% Import
4
5 % True or false to decide whether to run import functions (Import)
6 stg.import = true;
7
8
9 % Name of the excel file with the shtab (SBtab excel name)
10 stg.shtab_excel_name = "SBTAB.xlsx";
11
12 % Name of the model (Name)
13 stg.name = "model_name";
14
15 % Name of the default model compartment (Compartment name)
16 stg.cname = "Compartment";
17
18 % Name of the shtab saved in .mat format (SBtab name)
19 stg.shtab_name = "SBtab_" + stg.name;
20
21 %% Analysis
22
23 % Experiments to run (example experiment 1 to 3 and experiment 6)
24 stg.exprun = [1:3,6];
25
26 % Choice between 0,1,2 and 3,4 to choose the scoring method (check
27 % documentation) (Use logarithm)
28 stg.useLog = 4;
29
30 % True or false to decide whether to use multicore everywhere it is
31 % available (Optimization Multicore)
32 stg.optmc = true;
33
34 % Choice of random seed (Random seed)
35 stg.rseed = 1;
36
37 % True or false to decide whether to use display simulation diagnostics in
38 % the console (Simulation Console)
39 stg.simcsl = false;
40
41 % True or false to decide whether to display optimization results on
42 % console (Optimization console)
43 stg.optcsl = false;
44
45 % True or false to decide whether to save results (Save results)
46 stg.save_results = true;
47
48 % True or false to decide whether to run detailed simulation for plots
49 stg.simdetail = true;
50
51 %% Simulation
52
53 % Maximum time for each individual function to run in seconds (Maximum

```

(continues on next page)

(continued from previous page)

```

54 % time)
55 stg.maxt = 10;
56
57 % Equilibration time (Equilibration time)
58 stg.eqt = 50000;
59
60 % True or false to decide whether to do Dimensional Analysis (Dimensional
61 % Analysis)
62 stg.dimenanal = false;
63
64 % True or false to decide whether to do Unit conversion (Unit conversion)
65 stg.UnitConversion = false;
66
67 % True or false to decide whether to do Absolute Tolerance Scaling
68 % (Absolute Tolerance Scaling)
69 stg.abstolscale = true;
70
71 % Value of Relative tolerance (Relative tolerance)
72 stg.reltol = 1.0E-4;
73
74 % Value of Absolute tolerance (Absolute tolerance)
75 stg.abstol = 1.0E-4;
76
77 % Time units for simulation (Simulation time)
78 stg.simtime = "second";
79
80 % True or false to decide whether to run sbioaccelerate (after changing
81 % this value you need to run "clear functions" to see an effect)
82 % (sbioaccelerate)
83 stg.sbioacc = true;
84
85 % (Absolute tolerance step size for equilibration)
86 stg.abstolstepsize_eq = [];
87
88 % Max step size in the simulation (if empty matlab decides whats best)
89 % (Maximum step)
90 stg.maxstep = [10];
91
92 % Max step size in the equilibration (if empty matlab decides whats best)
93 % (Maximum step)
94 stg.maxstepeq = [2];
95
96 % Max step size in the detailed plots (if empty matlab decides whats best)
97 % (Maximum step)
98 stg.maxstepdetail = [2];
99
100 % Default score when there is a simulation error, this is needed to keep
101 % the optimizations working.
102 % (error score)
103 stg.errorscore = 10^5;
104 %% Model
105

```

(continues on next page)

(continued from previous page)

```

106 % Number of parameters to optimize (Parameter number)
107 stg.parnum = 5;
108
109 original_parameter_set = zeros(1,10);
110
111 % Array with the lower bound of all parameters (Lower bound)
112 stg.lb = original_parameter_set-5;
113
114 % Array with the upper bound of all parameters (Upper bound)
115 stg.ub = original_parameter_set+5;
116
117 %% Diagnostics
118
119 % Choice of what parameters in the array to test, the indices correspond to
120 % the parameters in the model and the numbers correspond to the parameters
121 % in the optimization array, usually not all parameters are optimized so
122 % there needs to be a match between one and the other. (Parameters to test)
123 % In this example there are ten parameters in this imaginary model and we
124 % are only interested in parameter 2,4,8,9, and 10. Note that stg.parnum is
125 % five because of this and not ten
126 stg.partest(:,1) = [0,1,0,2,0,0,0,3,4,5];
127
128 % (Parameter array to test)
129 stg.pat = [1:2];
130
131 % All the parameter arrays, in this case there is only one (parameters here
132 % are in log10 space)(Parameter arrays)
133 stg.pa(1,:) = [1,1,1,1,1];
134 stg.pa(1,:) = [1,0,1,2,1];
135
136 % Best parameter array found so far for the model (Best parameter array)
137 stg.bestpa = stg.pa(1,:);
138
139 %% Plots
140
141 % True or false to decide whether to plot results (Plots)
142 stg.plot = true;
143
144 % True or false to decide whether to use long names in the title of the
145 % outputs plots in f_plot_outputs.m (Plot outputs long names)
146 stg.plotoln = true;
147
148 %% Sensitivity analysis
149
150 % Number of samples to use in SA (Sensitivity analysis number of samples)
151 stg.sansamples = 100;
152
153 % True or false to decide whether to subtract the mean before calculating
154 % SI and SIT (Sensitivity analysis subtract mean)
155 stg.sasubmean = true;
156
157 % Choose the way you want to obtain the samples of the parameters for

```

(continues on next page)

(continued from previous page)

```

158 % performing the SA; 0 Log uniform distribution truncated at the parameter
159 % bounds 1 Log normal distribution with mu as the best value for a
160 % parameter and sigma as stg.sasamplesigma truncated at the parameter
161 % bounds 2 same as 1 without truncation 3 Log normal distribution centered
162 % at the mean of the parameter bounds and sigma as stg.sasamplesigma
163 % truncated at the parameter bounds 4 same as 3 without truncation.
164 % (Sensitivity analysis sampling mode)
165 stg.sasamplemode = 2;
166
167 % Sigma for creating the normal distribution of parameters to perform
168 % sensitivity analysis (Sensitivity analysis sampling sigma)
169 stg.sasamplesigma = 0.1;
170
171 %% Optimization
172
173 % Time for the optimization in seconds (fmincon does not respect this
174 % time!!) (Optimization time)
175 stg.optt = 60*5;
176
177 % Population size for the algorithms that use populations (Population size)
178 stg.popsiz = 144;
179
180 % optimization start method, choose between: 1 Random starting point or
181 % group of starting points inside the bounds 2 Random starting point or
182 % group of starting points near the best point (Optimization start method)
183 stg.osm = 1;
184
185 % Distance from best parameter array to be used in stg.osm method 2
186 % (Distance from best parameter array)
187 stg.dbs = 0.1;
188
189 % True or false to decide whether to use Multistart (Multistart)
190 stg.mst = false;
191
192 % Multistart size
193 stg.msts = 1;
194
195 % True or false to decide whether to display Plots (Plots doesn't work if
196 % using multicore) (Optimization plots)
197 stg.optplots = true;
198
199 % True or false to decide whether to run fmincon (no gradient so this
200 % doesn't work very well, no max time!!)
201 stg.fmincon = false;
202
203 % Options for fmincon (fmincon options)
204 stg.fm_options = optimoptions('fmincon',...
205     'Algorithm','interior-point',...
206     'MaxIterations',2,'OptimalityTolerance',0,...
207     'StepTolerance',1e-6,'FiniteDifferenceType','central',...
208     'MaxFunctionEvaluations',10000);
209

```

(continues on next page)

(continued from previous page)

```

210 % True or false to decide whether to run simulated annealing (Simulated
211 % annealing)
212 stg.sa = false;
213
214 % Options for simulated annealing (Simulated annealing options)
215 stg.sa_options = optimoptions(@simulannealbnd, ...
216     'MaxTime',stg.optt,...
217     'ReannealInterval',40);
218
219 % True or false to decide whether to run Pattern search (Pattern search)
220 stg.psearch = false;
221
222 % Options for Pattern search (Pattern search options)
223 stg.psearch_options = optimoptions(@patternsearch,...
224     'MaxTime',stg.optt,'UseParallel',stg.optmc,...
225     'UseCompletePoll',true,'UseCompleteSearch',true,...
226     'MaxMeshSize',2,'MaxFunctionEvaluations',2000);
227
228 % True or false to decide whether to run Genetic algorithm (Genetic
229 % algorithm)
230 stg.ga = false;
231
232 % Options for Genetic algorithm (Genetic algorithm options)
233 stg.ga_options = optimoptions(@ga,'MaxGenerations',200,...
234     'MaxTime',stg.optt,'UseParallel',stg.optmc,...
235     'PopulationSize',stg.popsize,...
236     'MutationFcn','mutationadaptfeasible');
237
238 % True or false to decide whether to run Particle swarm (Particle swarm)
239 stg.pswarm = false;
240
241 % Options for Particle swarm (Particle swarm options)
242 stg.pswarm_options = optimoptions('particleswarm',...
243     'MaxTime',stg.optt,'UseParallel',stg.optmc,'MaxIterations',200,...
244     'SwarmSize',stg.popsize);
245
246 % True or false to decide whether to run Surrogate optimization (Surrogate
247 % optimization)
248 stg.sopt = false;
249
250 % Options for Surrogate optimization (Surrogate optimization options)
251 stg.sopt_options = optimoptions('surrogateopt',...
252     'MaxTime',stg.optt,'UseParallel',stg.optmc,...
253     'MaxFunctionEvaluations',5000,...
254     'MinSampleDistance',0.2,'MinSurrogatePoints',32*2+1);
255 end

```

Example settings code

```

1 function [stg] = Example_model()
2
3 %% Import

```

(continues on next page)

(continued from previous page)

```

4
5 % True or false to decide whether to run import functions (Import)
6 stg.import = true;
7
8 % Name of the excel file with the shtab (Shtab excel name)
9 stg.shtab_excel_name = "SHTAB example.xlsx";
10
11 % Name of the model (Name)
12 stg.name = "Example";
13
14 % Name of the default model compartment (Compartment name)
15 stg.cname = "Compartment";
16
17 %% Analysis
18
19 % Experiments to run
20 stg.exprun = [1,3];
21 % stg.exprun = [1,2,3];
22
23 % Choice between 0,1,2 and 3 to change either and how to apply log10 to the
24 % scores (check documentation) (Use logarithm)
25 stg.useLog = 0;
26
27 % True or false to decide whether to use multicore everywhere it is
28 % available (Optimization Multicore)
29 stg.optmc = false;
30
31 % Choice of random seed (Random seed)
32 stg.rseed = 1;
33
34 % True or false to decide whether to use display simulation diagnostics in
35 % the console (Simulation Console)
36 stg.simcsl = false;
37
38 % True or false to decide whether to display optimization results on
39 % console (Optimization console)
40 stg.optcsl = true;
41
42 % True or false to decide whether to display PLA results on console (PLA
43 % console)
44 stg.placsl = true;
45
46 % True or false to decide whether to save results (Save results)
47 stg.save_results = true;
48
49 % True or false to decide whether to run detailed simulation for plots
50 stg.simdetail = false;
51
52 %% Simulation
53
54 % Maximum time for each individual function to run in seconds (Maximum
55 % time)

```

(continues on next page)

(continued from previous page)

```

56 stg.maxt = 2;
57
58 % Equilibration time (Equilibration time)
59 stg.eqt = 50000;
60
61 % True or false to decide whether to do Dimensional Analysis (Dimensional
62 % Analysis)
63 stg.dimenanal = true;
64
65 % True or false to decide whether to do Unit conversion (Unit conversion)
66 stg.UnitConversion = true;
67
68 % True or false to decide whether to do Absolute Tolerance Scaling
69 % (Absolute Tolerance Scaling)
70 stg.abstolscale = true;
71
72 % Value of Relative tolerance (Relative tolerance)
73 stg.reltol = 1.0E-4;
74
75 % Value of Absolute tolerance (Absolute tolerance)
76 stg.abstol = 1.0E-7;
77
78 % Time units for simulation (Simulation time)
79 stg.simtime = "second";
80
81 % True or false to decide whether to run sbioaccelerate (after changing
82 % this value you need to run "clear functions" to see an effect)
83 % (sbioaccelerate)
84 stg.sbioacc = false;
85
86 % Max step size in the simulation (if empty matlab decides whats best)
87 % (Maximum step)
88 stg.maxstep = [];
89
90 % Max step size in the equilibration (if empty matlab decides whats best)
91 % (Maximum step)
92 stg.maxstepeq = [];
93
94 % Max step size in the detailed plots (if empty matlab decides whats best)
95 % (Maximum step)
96 stg.maxstepdetail = [0.001];
97
98 % Default score when there is a simulation error, this is needed to keep
99 % the optimizations working. (error score)
100 stg.errorscore = 10^5;
101
102 %% Model
103
104 % Number of parameters to optimize (Parameter number)
105 stg.parnum = 12;
106
107 % Index for the parameters that have thermodynamic constrains (Termodiamic

```

(continues on next page)

(continued from previous page)

```

108 % Constrains Index)
109 stg.tci = [8];
110
111 % Parameters to multiply to the first parameter (in Stg.partest to get to
112 % the correct thermodynamic constrain formula) (Termodiamic Constrains
113 % multipliers)
114 stg.tcm([8],1) = [4];
115 stg.tcm([8],2) = [5];
116 stg.tcm([8],3) = [7];
117
118 % Parameters to divide to the first parameter (in Stg.partest to get to the
119 % correct thermodynamic constrain formula) (Termodiamic Constrains
120 % divisors)
121 stg.tcd([8],1) = [1];
122 stg.tcd([8],2) = [3];
123 stg.tcd([8],3) = [6];
124
125 % Array with the lower bound of all parameters (Lower bound)
126 stg.lb = zeros(1,stg.parnum)-4;
127
128 % Array with the upper bound of all parameters (Upper bound)
129 stg.ub = zeros(1,stg.parnum)+4;
130
131 %% Diagnostics
132
133 % Choice of what parameters in the array to test, the indices correspond to
134 % the parameters in the model and the numbers correspond to the parameters
135 % in the optimization array, usually not all parameters are optimized so
136 % there needs to be a match between one and the other. (Parameters to test)
137 stg.partest(:,1) = [1 ,2 ,3 ,4 ,5 ,6 ,7 ,2 ,8 ,9,...
138     10 ,11 ,12];
139
140 % (Parameter array to test)
141 stg.pat = 1:3;
142
143 % All the parameter arrays, in this case there is only one (Parameter
144 % arrays)
145 stg.pa(1,:) = [3.999,0.696,1.072,3.429,-0.751,-3.741,-0.569,0.831,3.068,0.921,-
146     ↪ 2.156,-1.970];
147 stg.pa(2,:) = stg.pa(1,)-1;
148 stg.pa(3,:) = stg.pa(1,)+1;
149
150 % Best parameter array found so far for the model (Best parameter array)
151 stg.bestpa = stg.pa(1,:);
152
153 %% Plots
154
155 % True or false to decide whether to plot results (Plots)
156 stg.plot = true;
157
158 % True or false to decide whether to use long names in the title of the
159 % outputs plots in f_plot_outputs.m (Plot outputs long names)

```

(continues on next page)

(continued from previous page)

```

159 stg.plotnames = true;
160
161 %% Sensitivity analysis
162
163 % Number of samples to use in SA (Sensitivity analysis number of samples)
164 stg.sansamples = 1000;
165
166 % True or false to decide whether to subtract the mean before calculating
167 % SI and SIT (Sensitivity analysis subtract mean)
168 stg.sasubmean = true;
169
170 % Choose the way you want to obtain the samples of the parameters for
171 % performing the SA; 0 Log uniform distribution truncated at the parameter
172 % bounds 1 Log normal distribution with mu as the best value for a
173 % parameter and sigma as stg.sasamplesigma truncated at the parameter
174 % bounds 2 same as 1 without truncation 3 Log normal distribution centered
175 % at the mean of the parameter bounds and sigma as stg.sasamplesigma
176 % truncated at the parameter bounds 4 same as 3 without truncation.
177 % (Sensitivity analysis sampling mode)
178 stg.sasamplemode = 2;
179
180 % Sigma for creating the normal distribution of parameters to perform
181 % sensitivity analysis (Sensitivity analysis sampling sigma)
182 stg.sasamplesigma = 0.1;
183
184 stg.gsabootstrapsize = ceil(sqrt(stg.sansamples));
185
186 %% Optimization
187
188 % Time for the optimization in seconds (fmincon does not respect this
189 % time!!) (Optimization time)
190 stg.optt = 60*1;
191
192 % Population size for the algorithms that use populations (Population size)
193 stg.popsiz = 10;
194
195 % optimization start method, choose between: 1 Random starting point or
196 % group of starting points inside the bounds 2 Random starting point or
197 % group of starting points near the best point (Optimization start method)
198 stg.osm = 1;
199
200 % Distance from best parameter array to be used in stg.osm method 2
201 % (Distance from best parameter array)
202 stg.dbs = 0.1;
203
204 % True or false to decide whether to use Multistart (Multistart)
205 stg.mst = false;
206
207 % Multistart size
208 stg.msts = 1;
209
210 % True or false to decide whether to display Plots (Plots doesn't work if

```

(continues on next page)

(continued from previous page)

```

211 % using multicore) (Optimization plots)
212 stg.optplots = true;
213
214 % True or false to decide whether to run fmincon (no gradient so this
215 % doesn't work very well, no max time!!)
216 stg.fmincon = false;
217
218 % Options for fmincon (fmincon options)
219 stg.fm_options = optimoptions('fmincon',...
220     'UseParallel',stg.optmc,...
221     'Algorithm','interior-point',...
222     'MaxIterations',2,'OptimalityTolerance',0,...
223     'StepTolerance',1e-6,'FiniteDifferenceType','central',...
224     'MaxFunctionEvaluations',10000);
225
226 % True or false to decide whether to run simulated annealing (Simulated
227 % annealing)
228 stg.sa = false;
229
230 % Options for simulated annealing (Simulated annealing options)
231 stg.sa_options = optimoptions(@simulannealbnd, ...
232     'MaxTime',stg.optt,...
233     'InitialTemperature',...
234     ones(1,stg.parnum)*2,'ReannealInterval',40);
235
236 % True or false to decide whether to run Pattern search (Pattern search)
237 stg.psearch = false;
238
239 % Options for Pattern search (Pattern search options)
240 stg.psearch_options = optimoptions(@patternsearch,...
241     'MaxTime',stg.optt,'UseParallel',stg.optmc,...
242     'UseCompletePoll',true,'UseCompleteSearch',true,...
243     'MaxMeshSize',2,'MaxFunctionEvaluations',2000);
244
245 % True or false to decide whether to run Genetic algorithm (Genetic
246 % algorithm)
247 stg.ga = true;
248
249 % Options for Genetic algorithm (Genetic algorithm options)
250 stg.ga_options = optimoptions(@ga,'MaxGenerations',200,...
251     'MaxTime',stg.optt,'UseParallel',stg.optmc,...
252     'PopulationSize',stg.popsize,...
253     'MutationFcn','mutationadaptfeasible','Display','diagnose');
254
255 % True or false to decide whether to run Particle swarm (Particle swarm)
256 stg.pswarm = false;
257
258 % Options for Particle swarm (Particle swarm options)
259 stg.pswarm_options = optimoptions('particleswarm',...
260     'MaxTime',stg.optt,'UseParallel',stg.optmc,...
261     'SwarmSize',stg.popsize);
262

```

(continues on next page)

(continued from previous page)

```

263 % True or false to decide whether to run Surrogate optimization (Surrogate
264 % optimization)
265 stg.sopt = false;
266
267 % Options for Surrogate optimization (Surrogate optimization options)
268 stg.sopt_options = optimoptions('surrogateopt',...
269     'MaxTime',stg.optt,'UseParallel',stg.optmc,...
270     'MaxFunctionEvaluations',5000,...
271     'MinSampleDistance',0.2,'MinSurrogatePoints',32*2+1);
272 end

```

Import

Default settings code

```

1 % True or false to decide whether to run import functions (Import)
2 stg.import = true;
3
4
5 % Name of the excel file with the shtab (SBtab excel name)
6 stg.shtab_excel_name = "SBTAB.xlsx";
7
8 % Name of the model (Name)
9 stg.name = "model_name";
10
11 % Name of the default model compartment (Compartment name)
12 stg.cname = "Compartment";
13
14 % Name of the shtab saved in .mat format (SBtab name)
15 stg.shtab_name = "SBtab_" + stg.name;

```

Example settings code

```

1 % True or false to decide whether to run import functions (Import)
2 stg.import = true;
3
4 % Name of the excel file with the shtab (SBtab excel name)
5 stg.shtab_excel_name = "SBTAB example.xlsx";
6
7 % Name of the model (Name)
8 stg.name = "Example";
9
10 % Name of the default model compartment (Compartment name)
11 stg.cname = "Compartment";

```

- **stg.import** - (logical) Decide whether to run import functions
- **stg.shtab_excel_name** - (string) Name of the Excel file with the SBtab
- **stg.name** - (string) Name of the model
- **stg.cname** - (string) Name of the default model compartment
- **stg.shtab_name** - (string) Name of the SBtab saved in .mat format

Analysis

Default settings code

```
1 % Experiments to run (example experiment 1 to 3 and experimet 6)
2 stg.exprun = [1:3,6];
3
4 % Choice between 0,1,2 and 3,4 to choose the scoring method (check
5 % documentation) (Use logarithm)
6 stg.useLog = 4;
7
8 % True or false to decide whether to use multicore everywhere it is
9 % available (Optimization Multicore)
10 stg.optmc = true;
11
12 % Choice of random seed (Ramdom seed)
13 stg.rseed = 1;
14
15 % True or false to decide whether to use display simulation diagnostics in
16 % the console (Simulation Console)
17 stg.simcsl = false;
18
19 % True or false to decide whether to display optimization results on
20 % console (Optimization console)
21 stg.optcsl = false;
22
23 % True or false to decide whether to save results (Save results)
24 stg.save_results = true;
25
26 % True or false to decide whether to run detailed simulation for plots
27 stg.simdetail = true;
```

Example settings code

```
1 % Experiments to run
2 stg.exprun = [1,3];
3 % stg.exprun = [1,2,3];
4
5 % Choice between 0,1,2 and 3 to change either and how to apply log10 to the
6 % scores (check documentation) (Use logarithm)
7 stg.useLog = 0;
8
9 % True or false to decide whether to use multicore everywhere it is
10 % available (Optimization Multicore)
11 stg.optmc = false;
12
13 % Choice of random seed (Ramdom seed)
14 stg.rseed = 1;
15
16 % True or false to decide whether to use display simulation diagnostics in
17 % the console (Simulation Console)
18 stg.simcsl = false;
19
20 % True or false to decide whether to display optimization results on
```

(continues on next page)

(continued from previous page)

```

21 % console (Optimization console)
22 stg.optcsl = true;
23
24 % True or false to decide whether to display PLA results on console (PLA
25 % console)
26 stg.placsl = true;
27
28 % True or false to decide whether to save results (Save results)
29 stg.save_results = true;
30
31 % True or false to decide whether to run detailed simulation for plots
32 stg.simdetail = false;

```

- **stg.exprun** - (double) Experiments to run
- **stg.useLog** - (double) Choice between 0,1,2 and 3 to change either and how to apply log10 to the scores, check *results*:
- **stg.optmc** - (logical) Decide whether to use multicore everywhere it is available
- **stg.rseed** - (double) Choice of random seed
- **stg.simcsl** - (logical) Decide whether to display simulation diagnostics in the console
- **stg.optcsl** - (logical) Decide whether to display optimization results on console
- **stg.save_results** - (logical) Decide whether to save results
- **stg.simdetail** - (logical) Decide whether to run detailed simulation for plots

Simulation

Default settings code

```

1 % Maximum time for each individual function to run in seconds (Maximum
2 % time)
3 stg.maxt = 10;
4
5 % Equilibration time (Equilibration time)
6 stg.eqt = 50000;
7
8 % True or false to decide whether to do Dimensional Analysis (Dimensional
9 % Analysis)
10 stg.dimenanal = false;
11
12 % True or false to decide whether to do Unit conversion (Unit conversion)
13 stg.UnitConversion = false;
14
15 % True or false to decide whether to do Absolute Tolerance Scaling
16 % (Absolute Tolerance Scaling)
17 stg.abstolscale = true;
18
19 % Value of Relative tolerance (Relative tolerance)
20 stg.reltol = 1.0E-4;
21

```

(continues on next page)

(continued from previous page)

```

22 % Value of Absolute tolerance (Absolute tolerance)
23 stg.abstol = 1.0E-4;
24
25 % Time units for simulation (Simulation time)
26 stg.simtime = "second";
27
28 % True or false to decide whether to run sbioaccelerate (after changing
29 % this value you need to run "clear functions" to see an effect)
30 % (sbioaccelerate)
31 stg.sbioacc = true;
32
33 % (Absolute tolerance step size for equilibration)
34 stg.abstolstepsize_eq = [];
35
36 % Max step size in the simulation (if empty matlab decides whats best)
37 % (Maximum step)
38 stg.maxstep = [10];
39
40 % Max step size in the equilibration (if empty matlab decides whats best)
41 % (Maximum step)
42 stg.maxstepeq = [2];
43
44 % Max step size in the detailed plots (if empty matlab decides whats best)
45 % (Maximum step)
46 stg.maxstepdetail = [2];
47
48 % Default score when there is a simulation error, this is needed to keep
49 % the optimizations working.
50 % (error score)
51 stg.errorscore = 10^5;

```

Example settings code

```

1 % Maximum time for each individual function to run in seconds (Maximum
2 % time)
3 stg.maxt = 2;
4
5 % Equilibration time (Equilibration time)
6 stg.eqt = 50000;
7
8 % True or false to decide whether to do Dimensional Analysis (Dimensional
9 % Analysis)
10 stg.dimenanal = true;
11
12 % True or false to decide whether to do Unit conversion (Unit conversion)
13 stg.UnitConversion = true;
14
15 % True or false to decide whether to do Absolute Tolerance Scaling
16 % (Absolute Tolerance Scaling)
17 stg.abstolscale = true;
18
19 % Value of Relative tolerance (Relative tolerance)

```

(continues on next page)

(continued from previous page)

```

20 stg.reltol = 1.0E-4;
21
22 % Value of Absolute tolerance (Absolute tolerance)
23 stg.abstol = 1.0E-7;
24
25 % Time units for simulation (Simulation time)
26 stg.simtime = "second";
27
28 % True or false to decide whether to run sbioaccelerate (after changing
29 % this value you need to run "clear functions" to see an effect)
30 % (sbioaccelerate)
31 stg.sbioacc = false;
32
33 % Max step size in the simulation (if empty matlab decides whats best)
34 % (Maximum step)
35 stg.maxstep = [];
36
37 % Max step size in the equilibration (if empty matlab decides whats best)
38 % (Maximum step)
39 stg.maxstepeq = [];
40
41 % Max step size in the detailed plots (if empty matlab decides whats best)
42 % (Maximum step)
43 stg.maxstepdetail = [0.001];
44
45 % Default score when there is a simulation error, this is needed to keep
46 % the optimizations working. (error score)
47 stg.errorscore = 10^5;

```

- **stg.maxt** - (double) Maximum time for each individual function has to run in seconds
- **stg.eqt** - (double) Equilibration time in seconds
- **stg.dimenanal** - (logical) Decide whether to do Dimensional Analysis
- **stg.UnitConversion** - (logical) Decide whether to do Unit conversion
- **stg.abstolscale** - (logical) Decide whether to do Absolute Tolerance Scaling
- **stg.reltol** - (double) Value of Relative tolerance
- **stg.abstol** - (double) Value of Absolute tolerance
- **stg.simtime** - (string) Time units for simulation
- **stg.sbioacc** - (logical) Decide whether to run **sbioaccelerate** (after changing this value you need to run “clear functions” to see an effect)
- **stg.abstolstepsize_eq** - (double) Absolute tolerance step size for equilibration (if empty MATLAB® decides whats best)
- **stg.maxstep** - (double) Max step size in the simulation (if empty MATLAB® decides what’s best)
- **stg.maxstepeq** - (double) Max step size in the equilibration (if empty MATLAB® decides whats best)
- **stg.maxstepdetail** - (double) Max step size in the detailed plots (if empty MATLAB® decides whats best)
- **stg.errorscore** - (double) Default score when there is a simulation error, this is needed to keep the optimizations working.

Model

Default settings code

```

1 % Number of parameters to optimize (Parameter number)
2 stg.parnum = 5;
3
4 original_parameter_set = zeros(1,10);
5
6 % Array with the lower bound of all parameters (Lower bound)
7 stg.lb = original_parameter_set-5;
8
9 % Array with the upper bound of all parameters (Upper bound)
10 stg.ub = original_parameter_set+5;

```

Example settings code

```

1 % Number of parameters to optimize (Parameter number)
2 stg.parnum = 12;
3
4 % Index for the parameters that have thermodynamic constraints (Termodiamic
5 % Constrains Index)
6 stg.tci = [8];
7
8 % Parameters to multiply to the first parameter (in Stg.partest to get to
9 % the correct thermodynamic constrain formula) (Termodiamic Constrains
10 % multipliers)
11 stg.tcm([8],1) = [4];
12 stg.tcm([8],2) = [5];
13 stg.tcm([8],3) = [7];
14
15 % Parameters to divide to the first parameter (in Stg.partest to get to the
16 % correct thermodynamic constrain formula) (Termodiamic Constrains
17 % divisors)
18 stg.tcd([8],1) = [1];
19 stg.tcd([8],2) = [3];
20 stg.tcd([8],3) = [6];
21
22 % Array with the lower bound of all parameters (Lower bound)
23 stg.lb = zeros(1,stg.parnum)-4;
24
25 % Array with the upper bound of all parameters (Upper bound)
26 stg.ub = zeros(1,stg.parnum)+4;

```

- **stg.parnum** - (double) Number of parameters to optimize
- **stg.tci** - (double) Index for the parameters that have thermodynamic constraints
- **stg.tcm** - (double) Parameters to multiply to the first parameter (in *stg.partest* to get to the correct thermodynamic constraint formula)
- **stg.tcd** - (double) Parameters to divide to the first parameter (in *stg.partest* to get to the correct thermodynamic constraint formula)
- **stg.lb** - (double) Lower bound of all parameters

$$stg.lb = [lb_1 \quad lb_2 \quad \dots \quad lb_i]$$

– i = Parameter index

- **stg.ub** - (double) Upper bound of all parameters

$$stg.ub = [ub_1 \quad ub_2 \quad \dots \quad ub_i]$$

– i = Parameter index

Diagnostics

Default settings code

```

1 % Choice of what parameters in the array to test, the indices correspond to
2 % the parameters in the model and the numbers correspond to the parameters
3 % in the optimization array, usually not all parameters are optimized so
4 % there needs to be a match between one and the other. (Parameters to test)
5 % In this example there are ten parameters in this imaginary model and we
6 % are only interested in parameter 2,4,8,9, and 10. Note that stg.parnum is
7 % five because of this and not ten
8 stg.partest(:,1) = [0,1,0,2,0,0,0,3,4,5];
9
10 % (Parameter array to test)
11 stg.pat = [1:2];
12
13 % All the parameter arrays, in this case there is only one (parameters here
14 % are in log10 space)(Parameter arrays)
15 stg.pa(1,:) = [1,1,1,1,1];
16 stg.pa(1,:) = [1,0,1,2,1];
17
18 % Best parameter array found so far for the model (Best parameter array)
19 stg.bestpa = stg.pa(1,:);

```

Example settings code

```

1 % Choice of what parameters in the array to test, the indices correspond to
2 % the parameters in the model and the numbers correspond to the parameters
3 % in the optimization array, usually not all parameters are optimized so
4 % there needs to be a match between one and the other. (Parameters to test)
5 stg.partest(:,1) = [1 ,2 ,3 ,4 ,5 ,6 ,7 ,2 ,8 ,9,...
6     10 ,11 ,12];
7
8 % (Parameter array to test)
9 stg.pat = 1:3;
10
11 % All the parameter arrays, in this case there is only one (Parameter
12 % arrays)
13 stg.pa(1,:) = [3.999,0.696,1.072,3.429,-0.751,-3.741,-0.569,0.831,3.068,0.921,-
14     ↪ 2.156,-1.970];
15 stg.pa(2,:) = stg.pa(1:)-1;
16 stg.pa(3,:) = stg.pa(1:)+1;
17
18 % Best parameter array found so far for the model (Best parameter array)
19 stg.bestpa = stg.pa(1,:);

```

- **stg.partest** - (double) Choice of which parameters to work on, since depending on the task, not all SBtab parameters are worked on. k indices correspond to the parameters in the SBtab and numbers up to i correspond to the parameters in the work set. This is the set that actually gets used for diagnostics, optimization, and sensitivity analysis.

$$stg.partest_k = [1_{k_1} \quad 2_{k_2} \quad \dots \quad i_{k_{end}}]$$

In our example model parameter 216 from the SBtab is parameter number 1 of the work set, parameter 217 from the SBtab is parameter number 2 of the work set, and successively.

$$stg.partest_{[216:227]} = [1_{216} \quad 2_{217} \quad \dots \quad 6_{221} \quad 1_{222} \quad 2_{223} \quad \dots \quad 6_{227}]$$

- **stg.pat** - (double) Index(j) of the parameter set to work on
- **stg.pa** - (double) All the parameter sets

$$stg.pa = \begin{bmatrix} x_{1,1} & x_{2,1} & \dots & x_{i,1} \\ x_{1,2} & x_{2,2} & \dots & x_{i,2} \\ \dots & \dots & \dots & \dots \\ x_{1,j} & x_{2,j} & \dots & x_{i,j} \end{bmatrix}$$

- **stg.bestpa** - (double) Best parameter set found so far during optimization

$$stg.bestx = [bestx_1 \quad bestx_2 \quad \dots \quad bestx_i]$$

- x = Parameters being worked on
- i = Index of Parameters being worked on
- k = Index of the parameters in SBtab
- j = Index of the Parameter set to work on

Plots

Default settings code

```
1 % True or false to decide whether to plot results (Plots)
2 stg.plot = true;
3
4 % True or false to decide whether to use long names in the title of the
5 % outputs plots in f_plot_outputs.m (Plot outputs long names)
6 stg.plotoln = true;
```

Example settings code

```
1 % True or false to decide whether to plot results (Plots)
2 stg.plot = true;
3
4 % True or false to decide whether to use long names in the title of the
5 % outputs plots in f_plot_outputs.m (Plot outputs long names)
6 stg.plotoln = true;
```

- **stg.plot** - (logical) Decide whether to plot results
- **stg.plotoln** - (logical) Decide whether to use long names in the title of the output plots in f_plot_outputs.m

Global Sensitivity Analysis (GSA)

Default settings code

```

1 % Number of samples to use in SA (Sensitivity analysis number of samples)
2 stg.sansamples = 100;
3
4 % True or false to decide whether to subtract the mean before calculating
5 % SI and SIT (Sensitivity analysis subtract mean)
6 stg.sasubmean = true;
7
8 % Choose the way you want to obtain the samples of the parameters for
9 % performing the SA; 0 Log uniform distribution truncated at the parameter
10 % bounds 1 Log normal distribution with mu as the best value for a
11 % parameter and sigma as stg.sasamplesigma truncated at the parameter
12 % bounds 2 same as 1 without truncation 3 Log normal distribution centered
13 % at the mean of the parameter bounds and sigma as stg.sasamplesigma
14 % truncated at the parameter bounds 4 same as 3 without truncation.
15 % (Sensitivity analysis sampling mode)
16 stg.sasamplemode = 2;
17
18 % Sigma for creating the normal distribution of parameters to perform
19 % sensitivity analysis (Sensitivity analysis sampling sigma)
20 stg.sasamplesigma = 0.1;

```

Example settings code

```

1 % Number of samples to use in SA (Sensitivity analysis number of samples)
2 stg.sansamples = 1000;
3
4 % True or false to decide whether to subtract the mean before calculating
5 % SI and SIT (Sensitivity analysis subtract mean)
6 stg.sasubmean = true;
7
8 % Choose the way you want to obtain the samples of the parameters for
9 % performing the SA; 0 Log uniform distribution truncated at the parameter
10 % bounds 1 Log normal distribution with mu as the best value for a
11 % parameter and sigma as stg.sasamplesigma truncated at the parameter
12 % bounds 2 same as 1 without truncation 3 Log normal distribution centered
13 % at the mean of the parameter bounds and sigma as stg.sasamplesigma
14 % truncated at the parameter bounds 4 same as 3 without truncation.
15 % (Sensitivity analysis sampling mode)
16 stg.sasamplemode = 2;
17
18 % Sigma for creating the normal distribution of parameters to perform
19 % sensitivity analysis (Sensitivity analysis sampling sigma)
20 stg.sasamplesigma = 0.1;
21
22 stg.gsbootstrapsize = ceil(sqrt(stg.sansamples));

```

- **stg.sansamples** - (double) Number of samples to use in GSA (in total $(2+n_{\text{pars}})*\text{sansamples}$ simulations will be performed, where n_{pars} are the number of parameters).
- **stg.sasubmean** - (logical) Decide whether to subtract mean before calculating *SI* and *SIT*, see Halnes et al 2009.

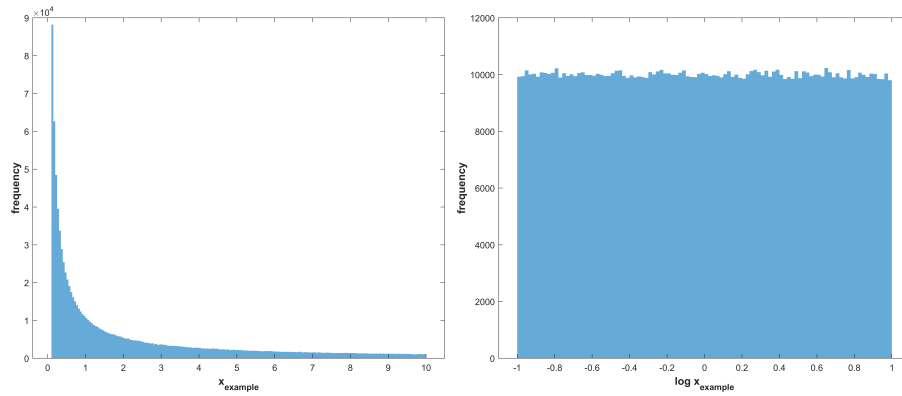
- **stg.sasamplemode** - (double) Choose the way you want to obtain the samples of the parameters for performing the GSA;

0. Reciprocal (log uniform) distribution

$$X_i \sim \text{Reciprocal}(a_i, b_i)$$

- i = Parameter index
- $a_i = \text{stg.lb}_i$
- $b_i = \text{stg.ub}_i$

Example distribution with $a = -1, b = 1$

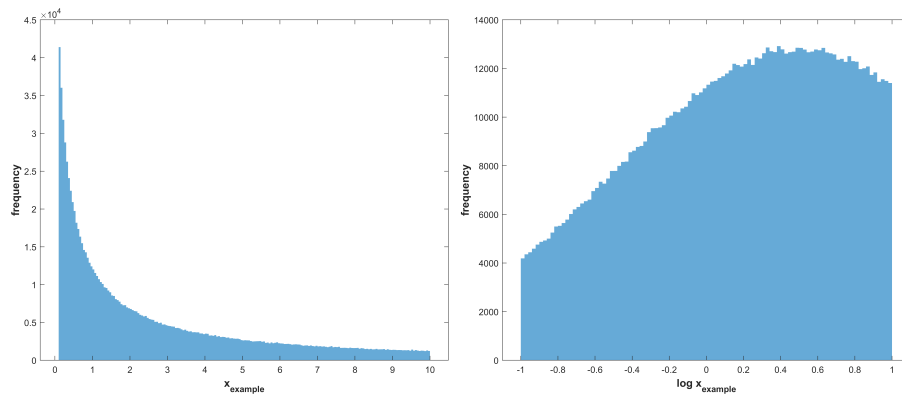


1. Log normal distribution with corresponding to the best value for a parameter, as recieved from the optimization, and as *stg.sasamplesigma* truncated at the parameter bounds

$$X_i \sim \text{TruncatedLogNormal}(i, , a_i, b_i)$$

- i = Parameter index
- $i = \text{best}x_i$
- $= \text{stg.sasamplesigma}$
- $a_i = \text{stg.lb}_i$
- $b_i = \text{stg.ub}_i$

Example distribution with $= 0.5, = 1, a = -1, b = 1$



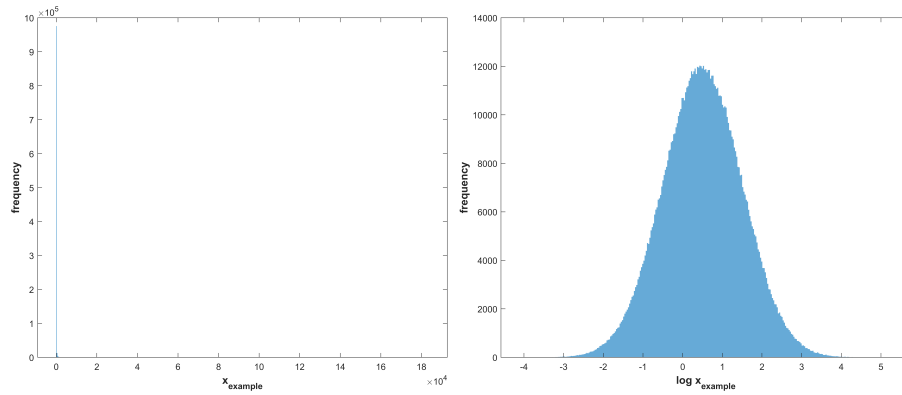
2. same as 1 without truncation

$$X_i \sim \text{LogNormal}(,)$$

- i = Parameter index

- $i = bestx_i$
- $= stg.sasamplesigma$

Example distribution with $= 0.5, = 1$

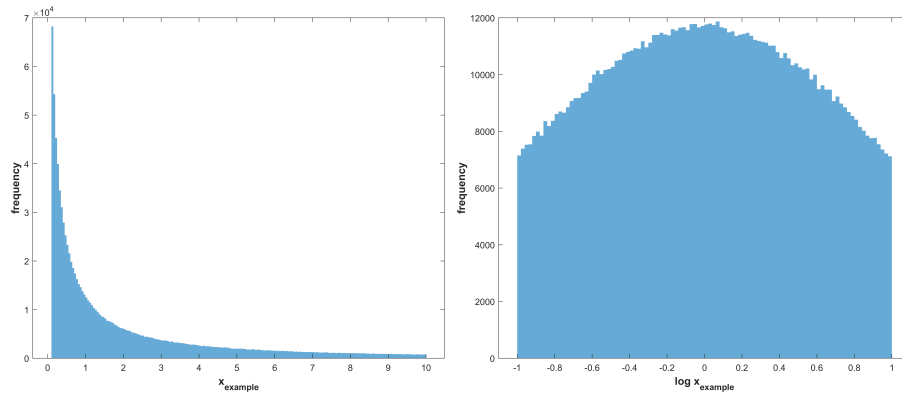


3. Log normal distribution with corresponding to the mean of the parameter bounds and as *stg.sasamplesigma* but truncated at the parameter bounds

$$X_i \sim TruncatedLogNormal(i, a_i, b_i)$$

- $i =$ Parameter index
- $i = \frac{stg.lb_i + (stg.ub_i - stg.lb_i)}{2}$
- $= stg.sasamplesigma$
- $a_i = stg.lb_i$
- $b_i = stg.ub_i$

Example distribution with $= \frac{a+(b-a)}{2}, = 1, a = -1, b = 1$

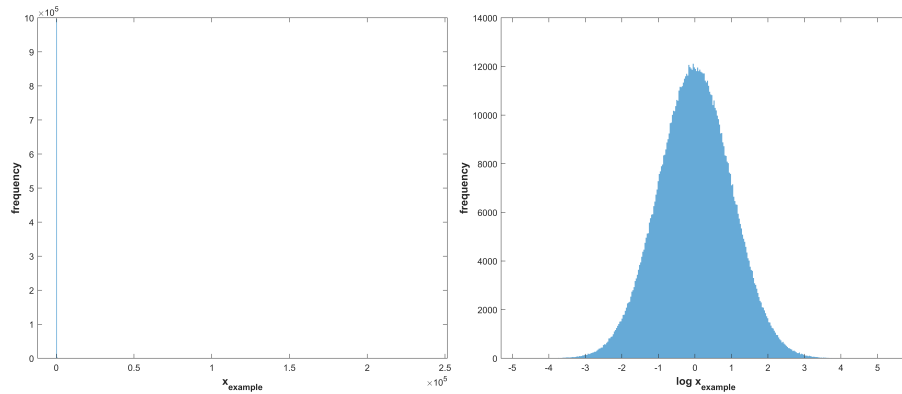


4. same as 3 without truncation.

$$X_i \sim LogNormal(mu_i,)$$

- $i =$ Parameter index
- $i = \frac{stg.lb_i + (stg.ub_i - stg.lb_i)}{2}$
- $= stg.sasamplesigma$

Example distribution with $= \frac{a+(b-a)}{2}, = 1, a = -1, b = 1$



- `stg.sasamplesigma` - (double) for creating the normal distribution of parameters to perform sensitivity analysis

Optimization

Default settings code

```

1  % Time for the optimization in seconds (fmincon does not respect this
2  % time!!) (Optimization time)
3  stg.optt = 60*5;
4
5  % Population size for the algorithms that use populations (Population size)
6  stg.popsz = 144;
7
8  % optimization start method, choose between: 1 Random starting point or
9  % group of starting points inside the bounds 2 Random starting point or
10 % group of starting points near the best point (Optimization start method)
11 stg.osm = 1;
12
13 % Distance from best parameter array to be used in stg.osm method 2
14 % (Distance from best parameter array)
15 stg.dbs = 0.1;
16
17 % True or false to decide whether to use Multistart (Multistart)
18 stg.mst = false;
19
20 % Multistart size
21 stg.msts = 1;
22
23 % True or false to decide whether to display Plots (Plots doesn't work if
24 % using multicore) (Optimization plots)
25 stg.optplots = true;
26
27 % True or false to decide whether to run fmincon (no gradient so this
28 % doesn't work very well, no max time!!)
29 stg.fmincon = false;
30
31 % Options for fmincon (fmincon options)
32 stg.fm_options = optimoptions('fmincon',...
33     'Algorithm','interior-point',...
34     'MaxIterations',2,'OptimalityTolerance',0,...

```

(continues on next page)

(continued from previous page)

```

35     'StepTolerance',1e-6,'FiniteDifferenceType','central',...
36     'MaxFunctionEvaluations',10000);
37
38 % True or false to decide whether to run simulated annealing (Simulated
39 % annealing)
40 stg.sa = false;
41
42 % Options for simulated annealing (Simulated annealing options)
43 stg.sa_options = optimoptions(@simulannealbnd, ...
44     'MaxTime',stg.optt,...
45     'ReannealInterval',40);
46
47 % True or false to decide whether to run Pattern search (Pattern search)
48 stg.psearch = false;
49
50 % Options for Pattern search (Pattern search options)
51 stg.psearch_options = optimoptions(@patternsearch,...
52     'MaxTime',stg.optt,'UseParallel',stg.optmc,...
53     'UseCompletePoll',true,'UseCompleteSearch',true,...
54     'MaxMeshSize',2,'MaxFunctionEvaluations',2000);
55
56 % True or false to decide whether to run Genetic algorithm (Genetic
57 % algorithm)
58 stg.ga = false;
59
60 % Options for Genetic algorithm (Genetic algorithm options)
61 stg.ga_options = optimoptions(@ga,'MaxGenerations',200,...
62     'MaxTime',stg.optt,'UseParallel',stg.optmc,...
63     'PopulationSize',stg.popsize,...
64     'MutationFcn','mutationadaptfeasible');
65
66 % True or false to decide whether to run Particle swarm (Particle swarm)
67 stg.pswarm = false;
68
69 % Options for Particle swarm (Particle swarm options)
70 stg.pswarm_options = optimoptions('particleswarm',...
71     'MaxTime',stg.optt,'UseParallel',stg.optmc,'MaxIterations',200,...
72     'SwarmSize',stg.popsize);
73
74 % True or false to decide whether to run Surrogate optimization (Surrogate
75 % optimization)
76 stg.sopt = false;
77
78 % Options for Surrogate optimization (Surrogate optimization options)
79 stg.sopt_options = optimoptions('surrogateopt',...
80     'MaxTime',stg.optt,'UseParallel',stg.optmc,...
81     'MaxFunctionEvaluations',5000,...
82     'MinSampleDistance',0.2,'MinSurrogatePoints',32*2+1);
83 end

```

Example settings code

```

1 % Time for the optimization in seconds (fmincon does not respect this
2 % time!!) (Optimization time)
3 stg.optt = 60*1;
4
5 % Population size for the algorithms that use populations (Population size)
6 stg.popsz = 10;
7
8 % optimization start method, choose between: 1 Random starting point or
9 % group of starting points inside the bounds 2 Random starting point or
10 % group of starting points near the best point (Optimization start method)
11 stg.osm = 1;
12
13 % Distance from best parameter array to be used in stg.osm method 2
14 % (Distance from best parameter array)
15 stg.dbs = 0.1;
16
17 % True or false to decide whether to use Multistart (Multistart)
18 stg.mst = false;
19
20 % Multistart size
21 stg.msts = 1;
22
23 % True or false to decide whether to display Plots (Plots doesn't work if
24 % using multicore) (Optimization plots)
25 stg.optplots = true;
26
27 % True or false to decide whether to run fmincon (no gradient so this
28 % doesn't work very well, no max time!!)
29 stg.fmincon = false;
30
31 % Options for fmincon (fmincon options)
32 stg.fm_options = optimoptions('fmincon',...
33     'UseParallel',stg.optmc,...
34     'Algorithm','interior-point',...
35     'MaxIterations',2,'OptimalityTolerance',0,...
36     'StepTolerance',1e-6,'FiniteDifferenceType','central',...
37     'MaxFunctionEvaluations',10000);
38
39 % True or false to decide whether to run simulated annealing (Simulated
40 % annealing)
41 stg.sa = false;
42
43 % Options for simulated annealing (Simulated annealing options)
44 stg.sa_options = optimoptions(@simulannealbnd, ...
45     'MaxTime',stg.optt,...
46     'InitialTemperature',...
47     ones(1,stg.parnum)*2,'ReannealInterval',40);
48
49 % True or false to decide whether to run Pattern search (Pattern search)
50 stg.psearch = false;
51
52 % Options for Pattern search (Pattern search options)
53 stg.psearch_options = optimoptions(@patternsearch,...

```

(continues on next page)

(continued from previous page)

```

54     'MaxTime',stg.optt,'UseParallel',stg.optmc,...
55     'UseCompletePoll',true,'UseCompleteSearch',true,...
56     'MaxMeshSize',2,'MaxFunctionEvaluations',2000);
57
58 % True or false to decide whether to run Genetic algorithm (Genetic
59 % algorithm)
60 stg.ga = true;
61
62 % Options for Genetic algorithm (Genetic algorithm options)
63 stg.ga_options = optimoptions(@ga,'MaxGenerations',200,...
64     'MaxTime',stg.optt,'UseParallel',stg.optmc,...
65     'PopulationSize',stg.popsize,...
66     'MutationFcn','mutationadaptfeasible','Display','diagnose');
67
68 % True or false to decide whether to run Particle swarm (Particle swarm)
69 stg.pswarm = false;
70
71 % Options for Particle swarm (Particle swarm options)
72 stg.pswarm_options = optimoptions('particleswarm',...
73     'MaxTime',stg.optt,'UseParallel',stg.optmc,...
74     'SwarmSize',stg.popsize);
75
76 % True or false to decide whether to run Surrogate optimization (Surrogate
77 % optimization)
78 stg.sopt = false;
79
80 % Options for Surrogate optimization (Surrogate optimization options)
81 stg.sopt_options = optimoptions('surrogateopt',...
82     'MaxTime',stg.optt,'UseParallel',stg.optmc,...
83     'MaxFunctionEvaluations',5000,...
84     'MinSampleDistance',0.2,'MinSurrogatePoints',32*2+1);
85 end

```

- **stg.optt** - (double) Time for the optimization in seconds (fmincon does not respect this time!!)
- **stg.popsize** - (double) Population size (for the algorithms that use populations)
- **stg.osm** - (double) optimization start method, choose between
 1. Get a random starting parameter set or group of starting parameter sets inside the bounds
 2. Get a random starting parameter set or group of starting parameter sets near the best parameter set
- **stg.dbpa** - (double) Distance from best parameter set to be used in *stg.osm* method 2
- **stg.mst** - (logical) Decide whether to use one or multiple starting parameter sets for the optimization
- **stg.msts** - (double) Number of starting parameter sets for the optimizations
- **stg.optplots** - (logical) Decide whether to display optimization plots (They aren't plotted if running the code in multicore)
- **stg.fmincon** - (logical) Decide whether to run `fmincon` (no gradient in our models so this doesn't work very well, does not respect *time set for the optimization*!!)
- **stg.fm_options** - (optim.options.Fmincon) Options for `fmincon`
- **stg.sa** - (logical) Decide whether to run `simulated annealing`

- **stg.sa_options** - (optim.options.SimulannealbndOptions) Options for simulated annealing
- **stg.psearch** - (logical) Decide whether to run [Pattern search](#)
- **stg.psearch_options** - (optim.options.PatternsearchOptions) Options for [Pattern search](#)
- **stg.ga** - (logical) Decide whether to run [Genetic algorithm](#)
- **stg.ga_options** - (optim.options.GaOptions) Options for [Genetic algorithm](#)
- **stg.pswarm** - (logical) Decide whether to run [Particle swarm](#)
- **stg.pswarm_options** - (optim.options.Particleswarm) Options for [Particle swarm](#)
- **stg.sopt** - (logical) Decide whether to run [Surrogate optimization](#)
- **stg.sopt_options** - (optim.options.Surrogateopt) Options for [Surrogate optimization](#)

Automatically generated at Import

- **stg.expn** - (double) Total number of experiments stored in the SBtab
- **stg.outn** - (double) Total number of experimental outputs specified in the SBtab

References

Halnes, G., Ulfhielm, E., Ljunggren, E.E., Kotaleski, J.H. and Rospars, J.P., 2009. Modelling and sensitivity analysis of the reactions involving receptor, G-protein and effector in vertebrate olfactory receptor neurons. *Journal of Computational Neuroscience*, 27(3), p.471.

3.2.7 Results

Scoring and saved simulation output

Every time a simulation is run the simulated results are compared to the results provided and a score is calculated. Additionally the end point of the experimental output of all simulations is also stored. When performing the diagnostics function an MATLAB® representation of the entire run is also saved.

- **simd** - Simulation results (MATLAB® representation)
- **st** - Total score

To simplify representations the following correspondence has been used

$$score_{i,j,k} = \frac{1}{n} \sum_{i=1}^n \left(\frac{Y_{i,j,k} - y_{i,j,k}}{i,j,k} \right)^2$$

if *stg.useLog* = 0

$$st(; Y,) = \sum_{k=1}^l \sum_{j=1}^m score_{i,j,k}$$

if *stg.useLog* = 1

$$st(; Y,) = \sum_{k=1}^l \sum_{j=1}^m \log_{10}(score_{i,j,k})$$

if $stg.useLog = 2$

$$st(; Y,) = \sum_{k=1}^l \log_{10}(\sum_{j=1}^m score_{i,j,k})$$

if $stg.useLog = 3$

$$st(; Y,) = \log_{10}(\sum_{k=1}^l \sum_{j=1}^m score_{i,j,k})$$

- **se** - Score of each experiment

if $stg.useLog = 0$ or 3

$$se(; Y,) = \begin{bmatrix} \sum_{j=1}^m score_{i,j,1} \\ \sum_{j=1}^m score_{i,j,2} \\ \dots \\ \sum_{j=1}^m score_{i,j,k} \end{bmatrix}$$

if $stg.useLog = 1$

$$se(; Y,) = \begin{bmatrix} \sum_{j=1}^m \log_{10}(score_{i,j,1}) \\ \sum_{j=1}^m \log_{10}(score_{i,j,2}) \\ \dots \\ \sum_{j=1}^m \log_{10}(score_{i,j,k}) \end{bmatrix}$$

if $stg.useLog = 2$

$$se(; Y,) = \begin{bmatrix} \log_{10}(\sum_{j=1}^m score_{i,j,1}) \\ \log_{10}(\sum_{j=1}^m score_{i,j,2}) \\ \dots \\ \log_{10}(\sum_{j=1}^m score_{i,j,k}) \end{bmatrix}$$

- **sd** - Score of each experimental outputs in all experiments

if $stg.useLog = 0, 2, \text{ or } 3$

$$sd(; Y,) = \begin{bmatrix} score_{i,1,1} & score_{i,2,1} & \dots & score_{i,j,1} \\ score_{i,1,2} & score_{i,2,2} & \dots & score_{i,j,2} \\ \dots & \dots & \dots & \dots \\ score_{i,1,k} & score_{i,2,k} & \dots & score_{i,j,k} \end{bmatrix}$$

if $stg.useLog = 1$

$$sd(; Y,) = \begin{bmatrix} \log_{10}(score_{i,1,1}) & \log_{10}(score_{i,2,1}) & \dots & \log_{10}(score_{i,j,1}) \\ \log_{10}(score_{i,1,2}) & \log_{10}(score_{i,2,2}) & \dots & \log_{10}(score_{i,j,2}) \\ \dots & \dots & \dots & \dots \\ \log_{10}(score_{i,1,k}) & \log_{10}(score_{i,2,k}) & \dots & \log_{10}(score_{i,j,k}) \end{bmatrix}$$

- **xfinal** - Value of each experimental outputs at the end of the simulation

$$x_{final} (; Y,) = \begin{bmatrix} y_{n,1,1} & y_{n,2,1} & \dots & y_{n,j,1} \\ y_{n,1,2} & y_{n,2,2} & \dots & y_{n,j,2} \\ \dots & \dots & \dots & \dots \\ y_{n,1,k} & y_{n,2,k} & \dots & y_{n,j,k} \end{bmatrix}$$

- F = Objective function for Particle Swarm optimization
- Y = Data provided for fitting
- y = Simulation results of the updated model under parameterization

- = New parameterization for y
- = Allowed mismatch between the two simulation results, analogous to the standard deviation of a Gaussian noise model in data fitting
- n/i = Number/index of points in a given experimental output
- m/j = Number/index of experimental outputs
- l/k = Number/index of experiments

Diagnostics

When running the diagnostics a struct gets created that stores all the *outputs* of the *f_sim_score* function.

- **rst.diag.simd** - Simulation results (MATLAB® representation)
- **rst.diag.st** - Total score
- **rst.diag.se** - Score per experiment
- **rst.diag.sd** - Score per experimental outputs in all experiments
- **rst.diag.xfinal** - x value of all the species being tested at the end of the simulation

Optimization

- **rst.opt.name** - Name of optimizer that was used
- **rst.opt.x** - Best parameter set found by the optimization
- **rst.opt.fval** - Score for that best parameter set
- **rst.opt.exitflag** - Diagnostics to see how the optimization went
- **rst.opt.output** - Diagnostics to see how the optimization went

Sensitivity Analysis

The calculations performed to obtain these sensitivities were performed according to the equations described in Haines et al 2009.

- **rst.SA.M1** - Matrix with $(r * k)$ random numbers within the lower and upper bound ranges set for each parameter

$$M_1 = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \dots & x_k^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \dots & x_k^{(2)} \\ \dots & \dots & \dots & \dots \\ x_1^{(r)} & x_2^{(r)} & \dots & x_k^{(r)} \end{bmatrix}$$

- x = Parameters
- k = Total number of parameters (*stg.parnum*)
- r = Total number of Samples (*stg.sansamples*)

- **rst.SA.M2** - Same as *rst.SA.M1* but different random initialization

$$M_2 = \begin{bmatrix} x_1^{(1')} & x_2^{(1')} & \dots & x_k^{(1')} \\ x_1^{(2')} & x_2^{(2')} & \dots & x_k^{(2')} \\ \dots & \dots & \dots & \dots \\ x_1^{(r')} & x_2^{(r')} & \dots & x_k^{(r')} \end{bmatrix}$$

- x = Parameters
- k = Total number of parameters (*stg.parnum*)
- r = Total number of Samples (*stg.sansamples*)

- **rst.SA.N** - Matrix of size $(r * k * k)$ with columns exchanged between M1 and M2 as follows:

$$N_i = \begin{bmatrix} x_1^{(1')} & x_2^{(1')} & \dots & x_i^{(1)} & \dots & x_k^{(1')} \\ x_1^{(2')} & x_2^{(2')} & \dots & x_i^{(2)} & \dots & x_k^{(2')} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ x_1^{(r')} & x_2^{(r')} & \dots & x_i^{(r)} & \dots & x_k^{(r')} \end{bmatrix}$$

- x = Parameters
- k = Total number of parameters (*stg.parnum*)
- r = Total number of Samples (*stg.sansamples*)
- i = Index of each parameter

- **rst.SA.fM1** -

$$fM_1 = \begin{bmatrix} f(M_1^{(1)}) \\ f(M_1^{(2)}) \\ \dots \\ f(M_1^{(r)}) \end{bmatrix} = \begin{bmatrix} f(x_1^{(1)} & x_2^{(1)} & \dots & x_k^{(1)}) \\ f(x_1^{(2)} & x_2^{(2)} & \dots & x_k^{(2)}) \\ \dots & \dots & \dots & \dots \\ f(x_1^{(r)} & x_2^{(r)} & \dots & x_k^{(r)}) \end{bmatrix}$$

- k = Total number of parameters (*stg.parnum*)
- r = Total number of Samples (*stg.sansamples*)

- **rst.SA.fM2** -

$$fM_2 = \begin{bmatrix} f(M_2^{(1')}) \\ f(M_2^{(2')}) \\ \dots \\ f(M_2^{(r')}) \end{bmatrix} = \begin{bmatrix} f(x_1^{(1')}) & x_2^{(1')} & \dots & x_k^{(1')} \\ f(x_1^{(2')}) & x_2^{(2')} & \dots & x_k^{(2')} \\ \dots & \dots & \dots & \dots \\ f(x_1^{(r')}) & x_2^{(r')} & \dots & x_k^{(r')} \end{bmatrix}$$

- k = Total number of parameters (*stg.parnum*)
- r = Total number of Samples (*stg.sansamples*)

- **rst.SA.fN** -

$$fN_i = \begin{bmatrix} f(N_i^{(1)}) \\ f(N_i^{(2)}) \\ \dots \\ f(N_i^{(r)}) \end{bmatrix} = \begin{bmatrix} f(x_1^{(1')}) & x_2^{(1')} & \dots & x_i^{(1)} & \dots & x_k^{(1')} \\ f(x_1^{(2')}) & x_2^{(2')} & \dots & x_i^{(2)} & \dots & x_k^{(2')} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ f(x_1^{(r')}) & x_2^{(r')} & \dots & x_i^{(r)} & \dots & x_k^{(r')} \end{bmatrix}$$

- k = Total number of parameters (*stg.parnum*)
- r = Total number of Samples (*stg.sansamples*)
- i = Index of each parameter

- **rst.SA.SI** - First order effects

$$S_i = \frac{V_i(E_{-i}(Y|i))}{V(Y)} = \frac{U_i - E^2(Y)}{V(Y)}$$

$$U_i = \frac{1}{n-1} \sum_{r=1}^n f(M_1^r) f(N_i^r)$$

$$E^2(Y) = \frac{1}{n} \sum_{r=1}^n f(M_1^r) f(M_2^r)$$

$$V(Y) = \frac{1}{n-1} f^2(M_1^r) - E^2(Y)$$

- V = Variance
- $E(\dots|\dots)$ = Conditional expected value
- = Parameters of the model
- Y = Scalar output from the model
- n = Total number of Samples (*stg.sansamples*)
- r = Index of the Samples
- i = Index of each parameter

- **rst.SA.STI** - Total order effects

$$S_{Ti} = \frac{V(Y) - V_i(E_i(Y|i))}{V(Y)} = 1 - \frac{U_{-i} - E^2(Y)}{V_T(Y)}$$

$$U_{-i} = \frac{1}{n-1} \sum_{r=1}^n f(M_2^r) f(N_i^r)$$

$$E^2(Y) = \frac{1}{n} \sum_{r=1}^n f(M_1^r) f(M_2^r)$$

$$V_T(Y) = \frac{1}{n-1} f^2(M_2^r) - E^2(Y)$$

- V = Variance
- $E(\dots|\dots)$ = Conditional expected value
- = Parameters of the model
- Y = Scalar output from the model

- n = Total number of Samples (*stg.sansamples*)
- r = Index of the Samples
- i = Index of each parameter

References

Halnes, G., Ulfhielm, E., Ljunggren, E.E., Kotaleski, J.H. and Rospars, J.P., 2009. Modelling and sensitivity analysis of the reactions involving receptor, G-protein and effector in vertebrate olfactory receptor neurons. *Journal of Computational Neuroscience*, 27(3), p.471.

3.2.8 Model files and folders

Here we have the file hierarchy of the models used in MATLAB® portion of our workflow.

Some of these folders and files need to be user generated before running the code and some are automatic generated at runtime, in the list below the former are identified as (u-gen) and the later as (a-gen)

When importing one of our provided models the user should place the model folder inside the “Subcellular_workflow/Matlab/Model/” all this folders and files are relative to this folder.

- **“Model Folder name”/ (u-gen)**

Folder containing all the model files the model, we recomend using the same name as the repository name for the models we provide but there is no restrictions.

- “Source_sbt_name”.xlsx (u-gen)

Contains the SBtab in .xlsx format.

- **Matlab/** (u-gen)

Folder containing all model files related to MATLAB®, the model is used in other softwares so here reside all the files that MATLAB® uses or generates

- * **Data/** (a-gen)

Folder containing the model in severall diferent formats relevant for the analysis and files contaning model metadata such as experimental inputs and outputs

- * model_”model name”.sbproj (a-gen)

Contains the model derived from the SBtab in .sbproj (MATLAB® SimBiology®) format.

- * model_”model name”.mat (a-gen)

Contains the model derived from the SBtab in .mat format.

- * data_”model name”.mat (a-gen)

Contains data derived from the SBtab in a .mat format. This data is used to run the model taking into account all the inputs and outputs of the model.

- * model_”model name”.xml (a-gen)

Contains the model derived from the SBtab in .xml (SBML) format.

- * **Input_**"model name".mat (a-gen)
Contains input data derived from the SBtab in a .mat format for all the experimental inputs.
- * **SBtab_**"model name".mat (a-gen)
Contains the SBtab in .mat format.
- * **Exp/** (a-gen)
Contains a version of the model for each experiment contained in the SBtab. They include all the necessary inputs and outputs to simulate the supplied experimental conditions.
 - **Model_**"model name"_i.mat (a-gen)
Tailor made for the main run of the simulation.
 - **Model_eq_**"model name"_i.mat (a-gen)
Tailor made for the equilibration step of the simulation.
 - **Model_detail_**"model name"_i.mat (a-gen)
Tailor made for the main run of the simulation. The step size is reduced to generate better graphs
- * **Input_functions/** (a-gen)
Folder containing the functions that are used at run time to give the correct input to all experiments
 - "model name"_input_Ligand.mat (a-gen)
These functions interpolate the input that is supposed to be given to the model at run time.
 - "model name"_input_creator.mat (a-gen)
Creates the functions above for all experimental inputs.
- * **Results/** (a-gen)
Folder containing the results of the various possible Matlab analysis provided by our workflow
 - **"Analysis name"/** (a-gen)
Each analysis output is stored in its own folder, depending on the analysis run the results can be saved in either a "Diagnostics", "Optimization" or "Sensitivity Analysis". An "Examples" folder is also provided with analysis that were pre-run by us.
 - **"date"/** (a-gen)
The date and time of when the analysis was run, this is auto generated when an user choses to run any alysis.
 - **All_figures.fig** (a-gen)
All the plots generated by the analysis stored in a Matlab figure assembly
 - **Analysis.mat** (a-gen)
All the data used as input to the analysis, saved as the SBtab and the setting fileconverted to a matlab structs called "sb" and "stg" respectively. And all the outputs generated by running the analysis saved also in a matlab struct called "*rst*"
 - **"Figure name".png** (a-gen)
All the plots generated by the analysis stored individually as images

- * **Settings/** (u-gen)

Folder containing the *settings file*

- “Settings file name” (u-gen)

Settings file of the model. A place for the user to define all the relevant properties of model simulation that are not stored in SBtab. These are usually things that need to change during optimizations or model development.

- **tsv/** (a-gen)

Folder containing the SBtab converted to .tsv files for each SBtab that has been run through one of our analysis.

- * **“model name”** (a-gen)

Contains the SBtab in .tsv format.

Description of the general terms:

“Model Folder name” - Name of the folder containing the model files and folders. We recommend using the same name as the repository name for the models we provide but there are no restrictions.

“Source_sbtab_name” - Name of the SBtab provided by the user

“model name” - Name given to the model in all the automatically generated model files, chosen in the *settings file*

“Settings file name” - Name chosen by the user for the *settings file* of the model. By default we chose the same as the “model name”.

“Analysis name” - Depending on the analysis run the results can be saved in either a *“Diagnostics”*, *“Optimization”* or *“Sensitivity Analysis”* folder. An “Examples” folder is also provided with analysis that were pre-run by us.

“date” - The date and time of when the analysis was run, this is auto generated when a user chooses to run any analysis.

“Figure name” - Catch all for all description for the names of all the plots that are generated by our analysis.

3.3 NEURON

Here we describe the usage of the MOD files for two of the provided examples (the third one does not use MOD files).

3.3.1 Nair et al. 2016 (D1 MSN subcellular cascade)

The model requires input in the form of dopamine and calcium. These need to be specified by the user:

1. Dopamine is set to be 20 nM in `assign_calculated_values()`. This line should be replaced to whatever the user needs for input. For example, it could be replaced with an expression for dopamine such as the one provided in this mod file, which makes a dopamine pulse with a double exponential shape. The line

DA = 20

should instead read

DA = DA_expression.

1. The same goes for the calcium input. Alternatively, calcium could be provided via the intracellular concentration of a calcium ion. For example, in this MOD file we use the intracellular calcium concentration due to influx from NMDA receptors, `ca_nmdai`, which is calculated through a mechanism for calcium accumulation provided in a

separate MOD file. Access to the ionic concentrations is provided by NEURON's "USEION" statement. In this case the variable `ca_nmdai` needs to be added to the ASSIGNED block.

—————*IMPORTANT NOTE*—————

When specifying the calcium input like this care needs to be taken to make sure the units used by NEURON and the units in the model exported to the MOD file match. As mentioned in the article, the models exported as MOD files could have different units for the parameter values from the default units used by NEURON. For instance, NEURON's default units for internal calculations of ionic concentrations are in millimolars (mM), but the model's parameters are expressed in nanomolars (nM). It is absolutely *paramount* to match units, i.e. use the correct scaling for, in this case, the variable `ca_nmdai`, to provide the model with the right quantity of calcium so that it runs properly:

`Ca = ca_nmdai * (1e6)`

3.3.2 Viswan et al. 2016 (EGF-stimulated MAPK cascade)

In this model we only use EGF as an input (only this input is used in Figure 7 in Viswan et al. 2018 which we reproduce; otherwise the model may have various other inputs, see the original paper for details [2]).

1. The input is a step in the concentration of EGF, given by an expression for a sharp sigmoidal function for EGF in `assign_calculated_values()`:

`EGF = EGF_level/(1+exp(-(t-EGF_start) * EGF_steepness))`

1. The SBtab format of this model expresses the parameter values in seconds, whereas NEURON's default unit for time is milliseconds. The conversion script `SBtab_to_vfgen` provides automatic scaling of time units in SBtab to milliseconds. The concentration units are given in micromolar, and if some species needs to be coupled to a NEURON variable which is expressed in other units (such as NEURON's default millimolar units), the species or the NEURON variable need to be rescaled as in the example above.

3.3.3 References

- [1] Nair, A.G., Bhalla, U.S. and Hellgren Kotaleski, J., 2016. Role of DARPP-32 and ARPP-21 in the emergence of temporal constraints on striatal calcium and dopamine integration. *PLoS computational biology*, 12(9), p.e1005080.
- [2] Viswan, N.A., HarshaRani, G.V., Stefan, M.I. and Bhalla, U.S., 2018. FindSim: a framework for integrating neuronal data and signaling models. *Frontiers in neuroinformatics*, 12, p.38.

3.4 Subcellular application

Subcellular application (<https://subcellular.humanbrainproject.eu/model/meta>) provides a web interface for simulation of biomolecular networks expressed on bionetgen language (<https://bionetgen.org/>) using network free solver NFsim and reaction-diffusion stochastic systems solver STEPS (<http://steps.sourceforge.net/STEPS/documentation.php>) Models can be imported from an sbml file. In this repository we used two model examples to exemplify the usage of this tool.

3.4.1 BioNetGen translation of SBtab Nair_2016 model

The model was translated from SBtab model format to rule-based BioNetGen language for the simulation with STEPS and NFsim solvers embedded in the subcellular web app and with the RuleBender

Conversion steps

- Run *convert_Nair_2016_from_SBTAB_to_SBML.R* in RStudio to translate SBtab model to SBML. This step depends on SBtab to SBML converter
- Run *convert_Nair_2016_from_SBML_to_BNGL.ipynb* jupyter notebook to translate from SBML to BioNetGen language
- Import the resulted BioNetGen model *Nair_2016_optimized_alternative.bngl* to the subcellular web app. Add spine geometry *.json*, *.node*, *.ele*, *.face* files and stimulation pattern *stim_DA_complex.tsv*. See the subcellular web app help for details
- Simulate the final model *Nair_2016_optimized_alternative.ebngl* in the subcellular web app using STEPS or NFsim solvers

Files and folders

Nair_2016_optimized_alternative.ebngl - extended BNGL model corresponding to the *optimized Nair 2016* SBtab model with added geometry and stimulation patterns. Can be imported and simulated in the subcellular web app

SBTAB_Nair_2016 - the folder with the *optimized Nair 2016* SBtab model tsv tables.

Nair_2016_optimized.xml - SBML model translated from the *optimized Nair 2016* model by *convert_Nair_2016_from_SBTAB_to_SBML.R* script based on SBtab to SBML converter

Nair_2016_optimized_alternative.bngl - BioNetGen model obtained from *Nair_2016_optimized.xml* by *convert_Nair_2016_from_SBML_to_BNGL.ipynb* jupyter notebook which is based on *sbml_to_bngl.py* conversion tool

spine.ele, *spine.face*, *spine.node*, *spine.json* - these files specify TetGen meshes and model geometry needed for the subcellular web app *Geometry* section and STEPS solver (see the subcellular web app help for details)

stim_DA_complex.tsv, *stim_noDA_complex.tsv* - these files specify the stimulation pattern in *Simulations* section of the subcellular web app (corresponds to the experiments E0 - E9 of the SBtab model).

SBtabVFGEN-master - the folder containing a copy of SBtab to SBML converter

sbml_to_bngl.py - the python tool for conversion of SBML models to BioNetGen language.

3.4.2 BioNetGen translation of SBtab Viswan_2018 model

The model was translated from SBtab model format to rule-based BioNetGen language for the simulation with STEPS and NFsim solvers embedded in the subcellular web app and with the RuleBender

Conversion steps

- Run *convert_Viswan_2018_for_STEPS_optimised_from_SBTAB_to_SBML.R* in RStudio to translate SBtab model to SBML. This step depends on [SBtab to SBML converter](#) and [LibSBML](#)
- Run *convert_Viswan_2018_for_STEPS_optimised_from_SBML_to_BNGL.ipynb* jupyter notebook to translate from SBML to BioNetGen language. This step depends on *sbml_to_bngl.py* and on [LibSBML](#) or [pySB](#)
- Import the BioNetGen model (*SBTAB_Viswan_2018_alternative.bngl*) to the [subcellular web app](#). Add spine geometry (*.json*, *.node*, *.ele*, *.face* files) and stimulation pattern (*stim_E0.tsv*). See the [subcellular web app help](#) for details
- Simulate final model (*SBTAB_Viswan_2018_alternative.ebngl*) in the [subcellular web app](#) using **STEPS** or **NFsim** solvers
- Simulate the BioNetGen model with the [RuleBender](#)

Files and folders

SBTAB_Viswan_2018_alternative.ebngl - extended BNGL model corresponding to the *Viswan_2018_for_STEPS_optimised.xlsx* SBTAB model with added geometry and stimulation patterns. Can be imported and simulated in the [subcellular web app](#)

Viswan_2018_for_STEPS.xlsx - SBtab model equivalent to the original *Viswan_2018* model. It was obtained from *Viswan_2018.xlsx* by the modification of the model features incompatible with BNGL.

Viswan_2018_for_STEPS_optimised.xlsx - SBtab model equivalent to the optimized *Viswan_2018* model. It was obtained from *Viswan_2018_optimized.xlsx* by the modification of the model features incompatible with BNGL.

SBTAB_Viswan_2018.xml - SBML model translated from *Viswan_2018_for_STEPS_optimised.xlsx* model by *convert_Viswan_2018_for_STEPS_optimised_from_SBTAB_to_SBML.R* script based on [SBtab to SBML converter](#)

SBTAB_Viswan_2018_alternative.bngl - BioNetGen model obtained from *SBTAB_Viswan_2018.xml* by *convert_Viswan_2018_for_STEPS_optimised_from_SBML_to_BNGL.ipynb* jupyter notebook which is based on *sbml_to_bngl.py* conversion tool

cell.ele, *cell.face*, *cell.node*, *cell.json* - these files specify TetGen meshes and model geometry needed for the [subcellular web app](#) *Geometry* section and STEPS solver (see the [subcellular web app help](#) for details)

stim_E0.tsv, *stim_E1.tsv* - these files specify the stimulation pattern in *Simulations* section of the [subcellular web app](#) (corresponds to the experiments *E0* and *E1* of the SBtab model).

Viswan_2018_alternative_RuleBender.bngl - BNGL model corresponding to the *Viswan_2018_for_STEPS_optimised.xlsx* SBtab model with additional section specifying stimulation and BioNetGen solver. Can be imported and simulated in the [RuleBender](#)

SBtabVFGEN-master - the folder containing copy of [SBtab to SBML converter](#)

SBTAB_Viswan_2018_for_STEPS_optimised - the folder containing *tsv* tables of *Viswan_2018_for_STEPS_optimised.xlsx*

sbml_to_bngl.py - the python tool for conversion of SBML models to BioNetGen language.

3.4.3 Conversion of SBML to BioNetGen language

The conversion is implemented in *sbml_to_bngl.py* python module. Two approaches are supported by *sbml_to_bngl.transform()* function:

- if *converter='pysb'* - the converter based on the *Atomizer* implemented in *pysb.importers.sbml.sbml_translator()* function within *pySB* framework will be used. The *Atomizer* will try to modify the set of model molecules and reactions to convert them from reaction network to rule-based BioNetGen format.
- if *converter='plain'* - a libsbml based converter for sbml level 2, version 4 will be used. This converter produces a bngl approximation to reaction network format of a model. It is assumed that sbml models were obtained by exporting a MATLAB simbiology model to sbml, or by translation of SBTAB model by *SBtab to SBML converter*.

Models expressed by SBML and SBTAB often are not fully compatible with BNGL. Additional model adaptation steps are required in this case to obtain a working BNGL model. These steps will be partially automatized by *sbml_to_bngl.transform()* function if *adapt_steps* argument dictionary '*list_of_steps*' is nonempty.

The adaptation steps include:

1. The STEPS and NFsim solvers require different units for species quantities and kinetic rates. An adapted BNGL model provides modified expressions for all species concentrations and kinetic rates and provides an easy way for units changing by specification of auxiliary bngl model parameters: *Na* and *V_compartment_name*. These parameters should be selected to: *Na=6.022e23* and *V_compartment_name* = volume of corresponding compartment in liters for NFsim and to: *Na=1* and *V_compartment_name=1* for STEPS.
2. Species with fixed concentrations are not supported by NFsim solver. The BNGL model adaptation will modify model reactions such that a fixed species concentration became a model parameter. This parameter can be used for the clamping of species concentration or for the stimulation pattern application
3. If SBML to BNGL converter implemented in Atomizer is selected then additional transformation steps include renaming of duplicated molecule sites and repairing incorrect molecule names and kinetic rate transformations
4. In case when MATLAB simbiology is used for SBML model creation, *adapt_steps* will repair incorrect molecule and parameter names
5. Compartmental model of BNGL assumes tree structure of the set of model compartments. This assumption is often incompatible with mesh geometries supported by STEPS. The model adaptation steps can produce bngl models with flexible compartmental structure
6. There is a number of incompatibilities between SBTAB and BNGL which still require manual correction. These include concentrations fixed to an expression, functional expressions for reaction rates in case of STEPS solver, some nonstandard types of reaction kinetic functions etc. The *adapt_steps* detects the cases of known incompatibilities and produces corresponding warning messages

3.5 Conversion tools

For this workflow we have developed some conversion tools to facilitate model development.

The MATLAB® code takes the SBtab file in excel format and generates tab separated file (.tsv) of this SBtab, an SBML file (.xml) of the model, and two MATLAB® versions of the model (.m and .sbproj). This conversion happens as a setup step of running any of the Matlab analysis, it might be added as a standalone option in the future.

SBtab (.xls,.xlsx) -> Matlab® model (.m .sbproj), SBtab (.tsv), Matlab® SBML (.xml)

We also have R code to perform other conversions in two external repositories:

- Code for fixing the SBML produced by MATLAB®

<https://github.com/a-kramer/simbiology-sbml-fix>

Matlab® SBML (.xml) -> SBML (.xml)

- A standalone SBtab to VFgen SBML and NEURON tool

<https://github.com/a-kramer/SBtabVFGEN>

SBtab (.tsv or .ods) -> VFGEN (.vf) + SBML (.xml) + Neuron (.mod)

3.6 Models

We have used multiple models to validate our software tools. Each model has its own repository. In these repositories you can find;

- The model in different formats relevant for the various tools that we provide
- “Matlab” folder with:
 - Settings file relevant to use the model with our MATLAB® tools
 - Some examples of the output provided by our MATLAB® tools after running an Analysis

This is also the folder where all the run time outputs of the matlab analysis are stored.

- “Bionetgen and Steps” folder with relevant files for running the model in the subcellular application
- “Neuron” folder with relevant files for running the model in NEURON.

Note: Model_Fujita_2010 does not contain “Bionetgen and Steps” and “Neuron” folders as this model was not implemented in these softwares.

Links to the model repositories:

- https://github.com/jpgsantos/Model_Nair_2016
- https://github.com/jpgsantos/Model_Fujita_2010
- https://github.com/jpgsantos/Model_Viswan_2018

3.6.1 Fujita et al. 2010

Model by Fujita et al 2010 as one of the proposed benchmark models provided by Hass et al. 2019. The model represents epidermal growth factor (EGF)-dependent Akt pathway. Data from [Fig. 1b] with a corresponding EGF step input from [Fig. 1a] in Fujita et al. 2010 was used for estimation and Global Sensitivity Analysis of 12 parameters. We performed parameter estimation starting from the model parameters provided in the publication, using a search space 5 order of magnitude above and below for each parameter.

- $\text{EGFR} + \text{EGF} \rightleftharpoons \text{EGF_EGFR}$
- $\text{pEGFR} + \text{Akt} \rightleftharpoons \text{pEGFR_Akt}$
- $\text{pEGFR_Akt} \rightarrow \text{pEGFR} + \text{pAkt}$
- $\text{pEGFR} \rightarrow \text{null}$
- $\text{pAkt} + \text{S6} \rightleftharpoons \text{pAkt_S6}$
- $\text{pAkt_S6} \rightarrow \text{pAkt} + \text{pS6}$
- $\text{pAkt} \rightarrow \text{Akt}$
- $\text{pS6} \rightarrow \text{S6}$

- EGF_EGFR -> pEGFR
- EGFR -> null

https://github.com/jpgsantos/Model_Fujita_2010

Fujita, K.A., Toyoshima, Y., Uda, S., Ozaki, Y.I., Kubota, H. and Kuroda, S., 2010. Decoupling of receptor and downstream signals in the Akt pathway by its low-pass filter characteristics. *Science Signaling*, 3(132), pp.ra56-ra56.

Hass, H., Loos, C., Raimúndez-Álvarez, E., Timmer, J., Hasenauer, J. and Kreutz, C., 2019. Benchmark problems for dynamic modeling of intracellular processes. *Bioinformatics*, 35(17), pp.3073-3082.

3.6.2 Nair et al. 2016

We illustrate the Subcellular Workflow with a model depicting the emergence of the eligibility trace observed in reinforcement learning in striatal D1 medium spiny neurons (D1 MSN) (Nair et al. 2016). Here, an intracellular increase in calcium representing excitatory synaptic input leads to synaptic potentiation only when it is followed by a reinforcing dopamine input. These two signaling cascades, starting with a calcium train and a dopamine transient, are illustrated in Fig. 1a. The first pathway (depicted in blue) represents calcium-dependent activation of Ca²⁺/calmodulin-dependent protein kinase II (CaMKII), its autophosphorylation, and the phosphorylation of a generic CaMKII substrate that represents long term potentiation (LTP). In the second pathway (species in red), dopamine initiates a G-protein dependent cascade which results in the phosphorylation of dopamine- and cAMP-regulated phosphoprotein, 32 kDa (DARPP-32) turning into an inhibitor of protein phosphatase 1 (PP1) that otherwise dephosphorylates CaMKII and its substrate. Substrate phosphorylation is maximal when the time window between the calcium and dopamine inputs is sufficiently short (input-interval constraint mediated by DARPP-32 via PP1 inhibition), and when intracellular calcium elevation is followed by dopamine (input-order constraint mediated by another phosphoprotein, the cyclic AMP-regulated phosphoprotein, 21 kDa (ARPP-21) that sequesters calcium/calmodulin if dopamine arrives before calcium).

CaMKII is autophosphorylated both in the cytosol and the post synaptic density (PSD) with a custom MATLAB rate function as described in Li et al. (2012). To run the model in different software, we substitute the autophosphorylation rate function with the same set of bimolecular reactions (simplified version of reactions from Pepke et al. 2010) for both compartments. The reactions represent the formation of a complex with two fully activated CaMKII monomers and a catalytic step in which one monomer phosphorylates the other. Schematics can be found in Fig. 1b along with the six new parameters. We estimated the parameters using simulated data from the published model with simulation setups in Fig. 1c depicting different timings of the dopamine input relative to the calcium input.

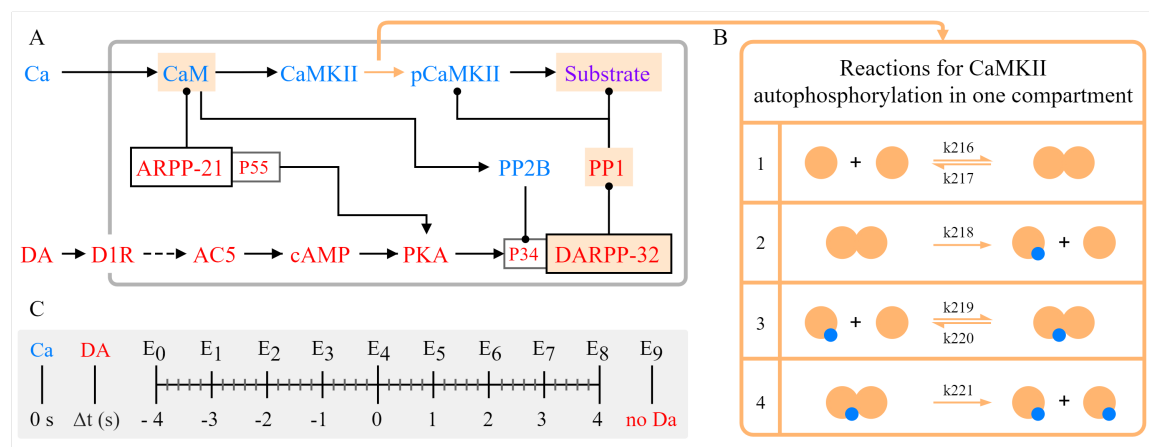


Figure 1. (A) Simplified schematics of the model. Species in the calcium cascade are depicted in blue, and species in the dopamine cascade are depicted red. Simulated time course data of the species with a beige background were used in parameter estimation. Figure adapted from Nair et al. (2016). (B) Illustration of the bimolecular reactions in CaMKII autophosphorylation with yellow circles representing activated CaMKII, and blue circles representing phosphate groups. Newly introduced parameters with their IDs in the updated model are shown above/below the arrows.

(C) Timing of the dopamine input ($t = \{-4, -3, -2, -1, 0, 1, 2, 3, 4\}$ corresponding to experiments E0-E8) relative to calcium increase (time 0), and a single experiment without dopamine (E9).

Four reactions representing autophosphorylation in one compartment.

Reactions in cytosol:

- $\text{CaMKII_CaM_Ca4} + \text{CaMKII_CaM_Ca4} \rightleftharpoons \text{CaMKII_CaM_Ca4_CaMKII_CaM_Ca4}$
- $\text{CaMKII_CaM_Ca4_CaMKII_CaM_Ca4} \rightarrow \text{pCaMKII_CaM_Ca4} + \text{CaMKII_CaM_Ca4}$
- $\text{pCaMKII_CaM_Ca4} + \text{CaMKII_CaM_Ca4} \rightleftharpoons \text{pCaMKII_CaM_Ca4_CaMKII_CaM_Ca4}$
- $\text{pCaMKII_CaM_Ca4_CaMKII_CaM_Ca4} \rightarrow \text{pCaMKII_CaM_Ca4} + \text{pCaMKII_CaM_Ca4}$

Reactions in PSD:

- $\text{CaMKII_CaM_Ca4_psd} + \text{CaMKII_CaM_Ca4_psd} \rightleftharpoons \text{CaMKII_CaM_Ca4_psd_CaMKII_CaM_Ca4_psd}$
- $\text{CaMKII_CaM_Ca4_psd_CaMKII_CaM_Ca4_psd} \rightarrow \text{pCaMKII_CaM_Ca4_psd} + \text{CaMKII_CaM_Ca4_psd}$
- $\text{pCaMKII_CaM_Ca4_psd} + \text{CaMKII_CaM_Ca4_psd} \rightleftharpoons \text{pCaMKII_CaM_Ca4_psd_CaMKII_CaM_Ca4_psd}$
- $\text{pCaMKII_CaM_Ca4_psd_CaMKII_CaM_Ca4_psd} \rightarrow \text{pCaMKII_CaM_Ca4_psd} + \text{pCaMKII_CaM_Ca4_psd}$

https://github.com/jpgsantos/Model_Nair_2016

Li, L., Stefan, M.I. and Le Novère, N., 2012. Calcium input frequency, duration and amplitude differentially modulate the relative activation of calcineurin and CaMKII. *PloS one*, 7(9), p.e43810.

Nair, A.G., Bhalla, U.S. and Hellgren Kotaleski, J., 2016. Role of DARPP-32 and ARPP-21 in the emergence of temporal constraints on striatal calcium and dopamine integration. *PLoS computational biology*, 12(9), p.e1005080.

Pepke, S., Kinzer-Ursem, T., Mihalas, S. and Kennedy, M.B., 2010. A dynamic model of interactions of Ca^{2+} , calmodulin, and catalytic subunits of Ca^{2+} /calmodulin-dependent protein kinase II. *PLoS Comput Biol*, 6(2), p.e1000675.

3.6.3 Viswan et al. 2018

Model from the FindSim framework by Viswan et al. 2018. The model represents an epidermal growth factor (EGF)-dependent mitogen-activated protein kinase (MAPK) signaling pathway (the green block from [Fig. 7a]) which measures MAPK phosphorylation. Here, two simulation experiments with EGF step inputs of different sizes are used to perform parameter estimation on 29 model parameters corresponding to reactions involved in MAPK phosphorylation (see below). For this we used activated MAPK curves in [Fig. 7b and 7c].

- $\text{GTP_Ras} + \text{craf_1_p} \rightleftharpoons \text{Raf_p_GTP_Ras}$
- $\text{GEF_p} \rightarrow \text{inact_GEF}$
- $\text{GTP_Ras} \rightarrow \text{GDP_Ras}$
- $\text{GAP_p} \rightarrow \text{GAP}$
- $\text{MAPK_p_p} + \text{craf_1_p} \rightleftharpoons \text{MAPK_p_p_feedback_cplx}$
- $\text{MAPK_p_p_feedback_cplx} \rightarrow \text{MAPK_p_p} + \text{craf_1_p_p}$
- $\text{MAPKK_p_p} + \text{MAPK} \rightleftharpoons \text{MAPKKtyr_cplx}$
- $\text{MAPKKtyr_cplx} \rightarrow \text{MAPKK_p_p} + \text{MAPK_p}$
- $\text{MAPKK_p_p} + \text{MAPK_p} \rightleftharpoons \text{MAPKKthr_cplx}$
- $\text{MAPKKthr_cplx} \rightarrow \text{MAPKK_p_p} + \text{MAPK_p_p}$
- $\text{MAPKK} + \text{Raf_p_GTP_Ras} \rightleftharpoons \text{Raf_p_GTP_Ras_1_cplx}$
- $\text{Raf_p_GTP_Ras_1_cplx} \rightarrow \text{MAPKK_p} + \text{Raf_p_GTP_Ras}$

- $\text{MAPKK_p} + \text{Raf_p_GTP_Ras} \rightleftharpoons \text{Raf_p_GTP_Ras_2_cplx}$
- $\text{Raf_p_GTP_Ras_2_cplx} \rightarrow \text{MAPKK_p_p} + \text{Raf_p_GTP_Ras}$
- $\text{inact_GEF} + \text{GDP_Ras} \rightleftharpoons \text{basal_GEF_activity_cplx}$
- $\text{basal_GEF_activity_cplx} \rightarrow \text{inact_GEF} + \text{GTP_Ras}$
- $\text{GEF_p} + \text{GDP_Ras} \rightleftharpoons \text{GEF_p_act_Ras_cplx}$
- $\text{GEF_p_act_Ras_cplx} \rightarrow \text{GEF_p} + \text{GTP_Ras}$
- $\text{GAP} + \text{GTP_Ras} \rightleftharpoons \text{GAP_inact_Ras_cplx}$
- $\text{GAP_inact_Ras_cplx} \rightarrow \text{GAP} + \text{GDP_Ras}$

https://github.com/jpgsantos/Model_Viswan_2018

Viswan, N.A., HarshaRani, G.V., Stefan, M.I. and Bhalla, U.S., 2018. FindSim: a framework for integrating neuronal data and signaling models. *Frontiers in neuroinformatics*, 12, p.38.

REFERENCES

- Santos, J.P., Pajo, K., Trpevski, D., Stepaniuk, A., Eriksson, O., Nair, A.G., Keller, D., Koteleski, J.H. and Kramer, A., 2020. A Modular Workflow for Model Building, Analysis, and Parameter Estimation in Systems Biology and Neuroscience. *bioRxiv*.
- Lubitz, T., Hahn, J., Bergmann, F.T., Noor, E., Klipp, E. and Liebermeister, W., 2016. SBtab: a flexible table format for data exchange in systems biology. *Bioinformatics*, 32(16), pp.2559-2561.
- Halnes, G., Ulfhielm, E., Ljunggren, E.E., Koteleski, J.H. and Rospars, J.P., 2009. Modelling and sensitivity analysis of the reactions involving receptor, G-protein and effector in vertebrate olfactory receptor neurons. *Journal of Computational Neuroscience*, 27(3), p.471.
- Nair, A.G., Bhalla, U.S. and Hellgren Koteleski, J., 2016. Role of DARPP-32 and ARPP-21 in the emergence of temporal constraints on striatal calcium and dopamine integration. *PLoS computational biology*, 12(9), p.e1005080.