Easybiotics User Guide

Jonathan Naylor
ICOS, School of Computing Science, Newcastle University
j.r.d.naylor@ncl.ac.uk

Contents

1 Introduction

Overview

Easybiotics is a graphical user interface (GUI) for the Simbiotics platform. Easybiotics allows for the design, simulation and analysis of Simbiotics models via an easy to use graphical interface which does not require programming experience to operate. It is a light-weight program developed in Python which has minimal dependencies.
This document describes how to get and install Easybiotics, as well as some tutorials on how to use it. **You can also try Easybiotics in a Virtual Machine for easy out-of-the-box use, it can be found on the website along with video tutorials on how to use the software.**

`https://bitbucket.org/simbiotics/simbiotics/wiki/Home`

License

Technical overview

Easybiotics is written in Python 2.7, and depends on the Kivy, Matplotlib and Pandas library.

Terminology

To clarify some of the terminology used in this document, we list some keywords and their meaning.

Term	Meaning
Library module	Java classes within the Simbiotics library which describe model-specific behaviour.
$SIMBIOTICS	The main Simbiotics folder, which contains the *src* folder

Table 1: User manual terminology

2 Getting Easybiotics

Downloading and installing

Simbiotics and Easybiotics are distributed together and can be downloaded at `https://bitbucket.org/simbiotics/simbiotics/wiki/Home`.

Simbiotics is developed in Java 1.7, and Easybiotics in Python 2.7.

You must have the following packages installed:

- Open JDK >= 6 (GNU General Public Licence + classpath exception) or Oracle Java SE >= 6 (Oracle Binary Code Licence)
- Python >= 2.7 - (https://www.python.org/)
- Kivy >= 1.8 - (https://kivy.org/)
- Matplotlib >= 1.4.2 - (https://matplotlib.org/index.html)
- Pandas >= 0.17 - (https://pandas.pydata.org/)

And optionally, if you wish to use SBML integration, you must have:

- libSBML - http://sbml.org/Software/libSBML (GNU LGPL)
- libSBMLSim - http://fun.bio.keio.ac.jp/software/libsbmlsim/ (GNU LGPL)

If you already have these dependencies installed, skip to **2.3. Running easybiotics**. Note that you must have Simbiotics installed to use Easybiotics. If you do not have Simbiotics installed please refer to the *simbiotics_guide.pdf* document.

Getting Dependencies

Easybiotics depends on modules from the Kivy, Pandas and Matplotlib libraries, which can be found here:

[Kivy]
[Pandas]
[Matplotlib]

If you are using a linux system, these can be installed on command line via aptitude with the following commands:

Listing 1: Getting Easybiotics dependencies via aptitude

```
sudo apt-get install python-kivy
sudo apt-get install python-pandas
sudo apt-get install python-matplotlib
```

Running Easybiotics

Simbiotics can be run in multiple ways allowing the user to choose which is most appropriate for them. Simbiotics can either by run via Easybiotics, by command-line, or opened in an IDE.

To run Easybiotics in Linux/MAC open a terminal/command prompt, navigate to the *$SIMBOTICS* directory and run the *"start_easybiotics"* script. For example:

Listing 2: Running Easybiotics

```
cd $SIMBIOTICS
./start_easybiotics
```

If you are using Windows you must use start the python application with *gui/qSimbiotics.py*

3 Developing models in Easybiotics

Here we elaborate on the features of Easybiotics and how to use them. First describing what each part of the GUI is for. Examples of how to use Easybiotics can be found the in tutorials section of this document, and in the Easybiotics video tutorials which can be found at

`https://bitbucket.org/simbiotics/simbiotics/wiki/Tutorials`

Easybiotics provides features to accomplish the following:

1. Develop models
2. Run models
3. Analyse models
4. Render visualisations

Develop models
The model development environment allows for Simbiotics models to be composed easily. The model specification is represented as an interactive tree structure to which library modules can be attached.

Run models
Models can be run from inside the model development environment. Models can be run with optional real time rendering of the simulation domain.

Analyse models
Simulation data can be exported to file by attaching exporters to the model specification. This data can optionally be visualised in live plots during the simulation run. The Simbiotics library also contains some analysis modules for characterisation of the model during run time.

Render visualisations
In addtion to real-time simulation rendering, simulations can be visualised after run time as 3D scenes. You can create static 'image' 3D scenes, or animated 3D scenes composed of a sequence of static scenes.

Overview

Creating a model

Models can be developed in Simbiotics by selecting the "Develop and run models" on the home screen. You may either create a new model, or load an existing model. There are a collection of example models which can be loaded directly from the load screen. Additionally if you select browse and navigate to *examples/models/* you will find some more example models.

We will first go over the basics on model development in Easybiotics. A tutorial can be found below this section which walks through the development of a basic model.

Once you have created a new model or loaded an existing one, you will be presented with the model development screen. The model development screen has 5 major components: the Filebar, the Config Editor, the Model Editor, the Display Panel and the Button Panel.

Filebar
The filebar (highlighted in yellow on Figure 3) gives access to functions such as the Easybiotics settings, handling configuration and model files, running simulations with optional live visualisation, graphs and parameter sweeps, along with information about Easybiotics.

Config Editor
The configuration editor (highlighted in blue on Figure 3) gives access to the Simbiotics platform settings, such as whether to run the simulation with real time rendering, and how many CPU threads should be created for the simulation.

Model Editor
The model editor (highlighted in green on Figure 3) allows for the manipulation of a model specification. This includes adding/removing modules, setting the modules parameters, and connecting modules together to represent the target system.

Display Panel
The display panel (highlighted in red on Figure 3) has three tabs. The description tab displays details of the selected module. The properties tab allows for the modification of any parameters of that module. The edit tab allows for the direct manipulation of the model specification file values.

Button Panel
The button panel (highlighted in orange on Figure 3) has three buttons - to run the current model, to quicksave the current model, and to go back to the Easybiotics homepage.

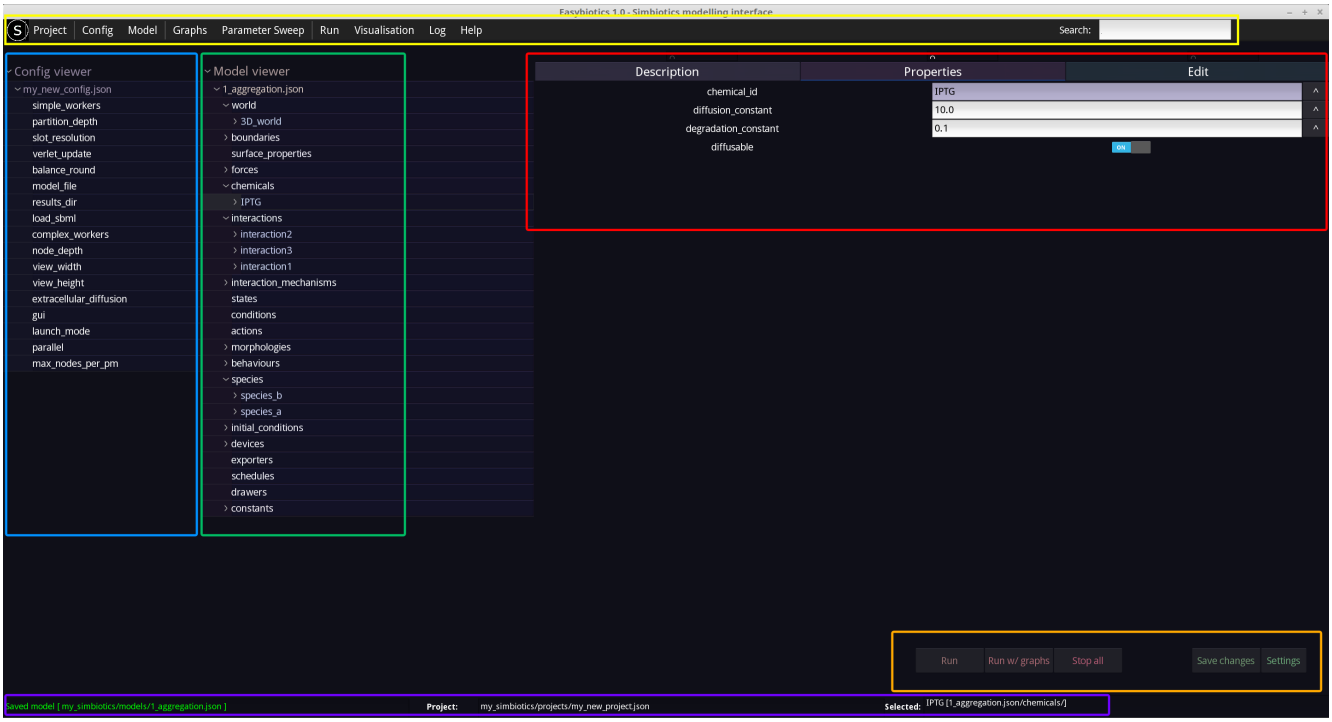


Figure 1: Overview of the Easybiotics modelling interfaces. The file bar is highlighted in yellow, the Simbiotics configuration editor in blue, the model specification editor in green, the display panel in red, and the button panel in orange.

Running the model

Models can be run by either clicking the 'Run' button in the button panel, or by selecting one of the run options on the filebar.

You may select the amount of RAM that the JVM (Java Virtual Machine) uses for the simulation by selecting 'Settings - Run Settings' from the file bar. Here you can set the initial amount of RAM allocated as well as the maximum amount of RAM the simulation may use.

The simulation can be run with an optional live rendering. This is set in the configuration editor, by setting the *gui* variable to *true* or *false*. If using the live renderer, the Simbiotics GUI is opened, more information can be found in the Live Visualisation of Simulations section.

Analysing the model

Models can be analysed by attaching exporters to the model specification. Numerous exporters can be defined, where each collects specific data about the simulation and writes it to file. Graphs may then be defined, which are set to plot data from the exporters. Graphs can be saved and loaded to/from file for easy reuse. To run real time graphs during the simulation run, select the 'Run - Run with live graph plotting' option from the file bar, and specifying the graphs which are to be rendered.

Easybiotics also provides a feature to perform parameter sweeps. Similar to graphs, parameter sweep objects can be defined, which iterate over properties in the model. All simulations are run for the defined parameter sweep ranges and their results saved to independent directories. Live graph plotters may also be attached to parameter sweeps, plotting the data from all simulations on one graph for easy comparison.

In addition, the library contains numerous analytical tools such as those able to calculate the mean squared displacement of agents and their velocity autocorrelation function. There are also some simulated lab tools, such as microsensors and a spectrophotometer. This allows for probing of the simulated experiment as would typically be done for the real experiment.

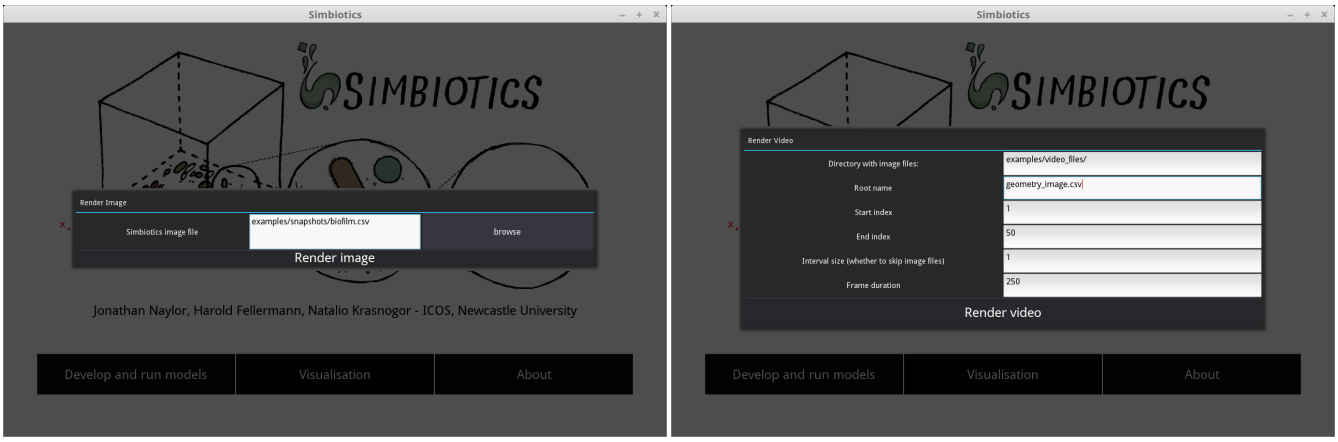
Examples of analysis with the tool are shown in the Easybiotics tutorials below.

Rendering visualisations

Visualisations of simulations can be rendered after they have finished executing, as an alternative to a live visualisation. This can be achieved by attaching a certain type of exporter, called a *geometry_image*, to the model specification. This exporter writes all geometry properties to a file periodically, writing a new file for each time point. It produces a series of indexed files which can be found in the results folder you set for the exporter.

Each geometry image file can be rendered independently into an static 3D scene, which is loaded in the Simbiotics GUI allowing the user to move the camera around the scene and modify what properties are visualised.

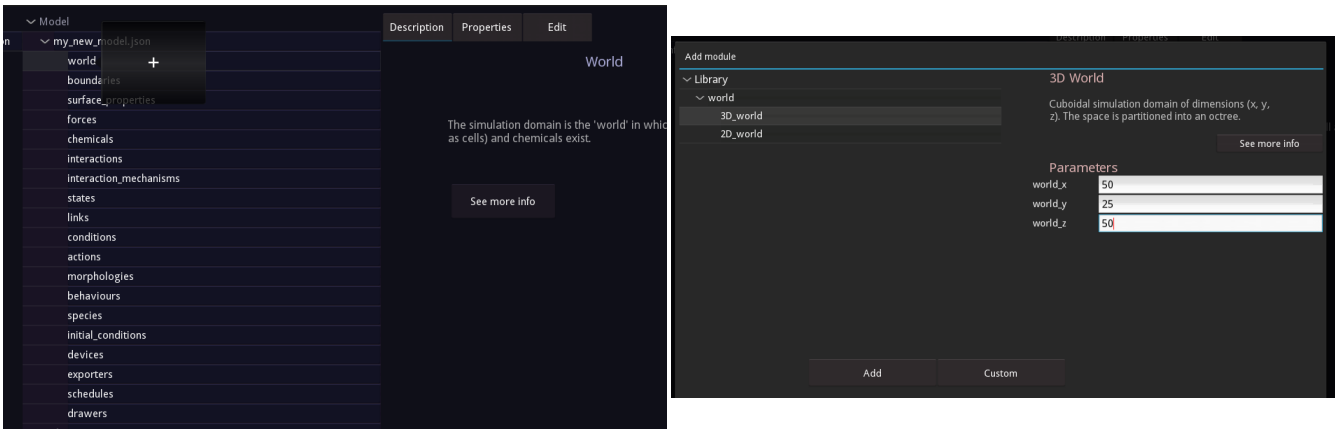
Alternatively, a sequence of geometry images can be loaded into an animated 3D scene. The user may set the delay between the animation frames, and whether the renderer should skip indexes. The animation renderer also runs a camera to record the animated 3D scene and convert it into an *.avi* which can be found in the *$SIMBIOTICS/results* folder.



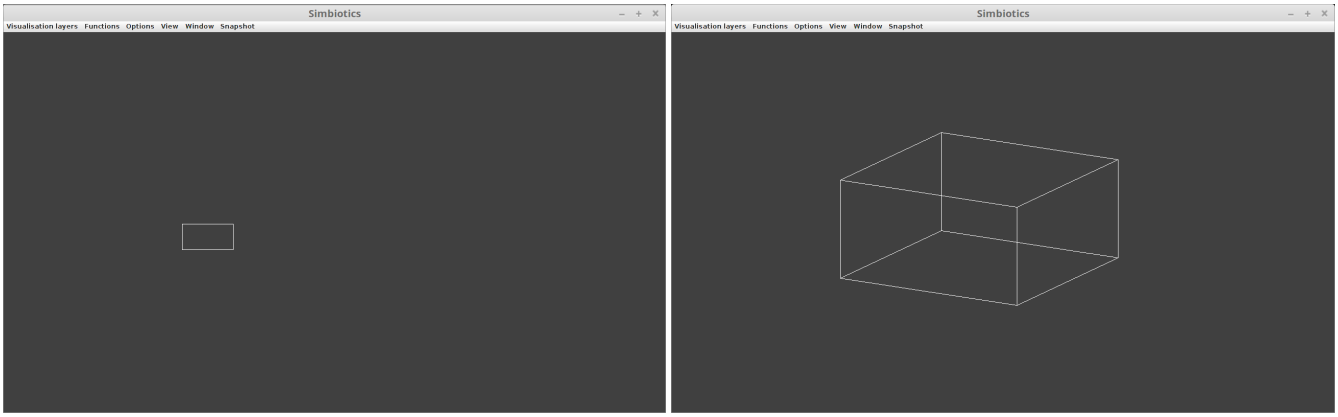
Tutorial 1 - Creating your first model

To create a new model in Easybiotics selected *Develop and run models - New model* from the Easybiotics home page, and enter the file name for your model, then press *Create*. The modelling interface should then display, showing the configuration editor on the left, and the model specification editor in the middle.

Firstly, we need to create a simulation domain, which we call a *world*. To do this, right-click on the *world* node on the model specification tree. This will display all world modules in the Simbiotics library. For this example, select the *3D_world* module, and set the world dimensions to be 50*25*50, then press the *Add* button. An example of this can be seen below.



To see that baby in action, hit run (either from the filebar or from the button bar bottom right). Make sure you have the *gui* property in the configuration set to *true*! The Simbiotics GUI should open, showing the simulation domain and nothing else. You can rotate the camera by right-clicking and dragging, and scrolling to zoom in and out. For more information on the Simbiotics GUI, see the Live visualisation of simulations section nearer the start of this document.

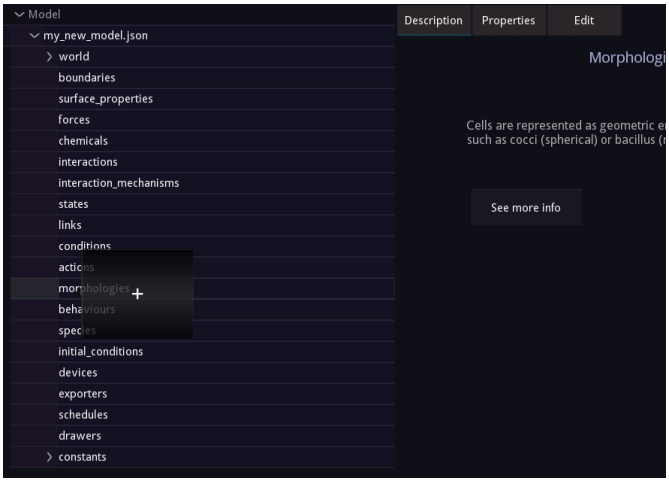


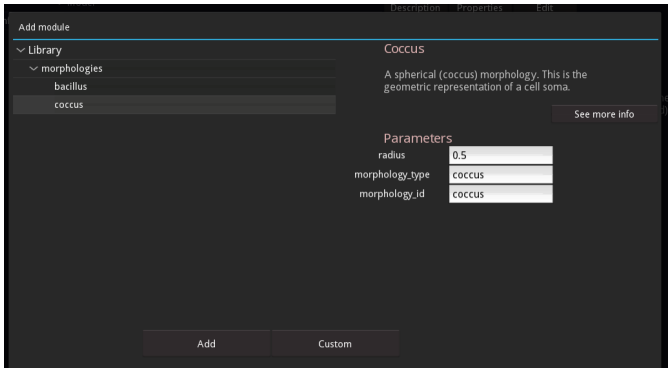
Ok, lets add some cells!

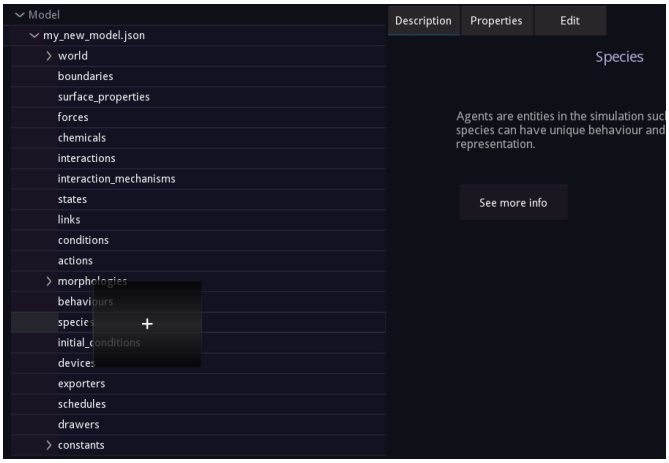
In brief, a cell is considered as an agent (an individual) in the model, and each agent is represented by a physical geometry in the simulation. In the model specificaiton, a physical geometry can be created by add a *morphology*. To do this, right click on the *morphologies* node on the model specificaiton, and select the *coccus* (spherical) module. The default parameters are ok, so just press *Add*. This process can be seen below.

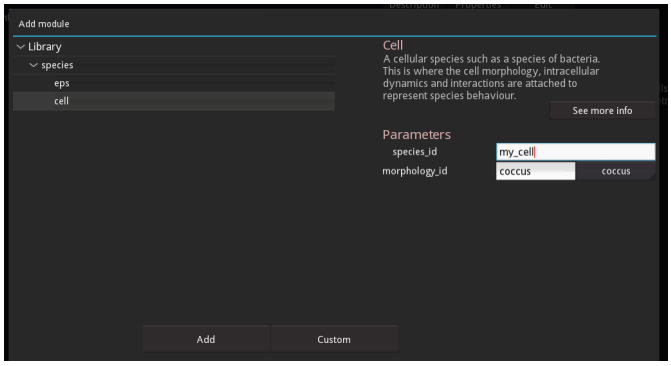
Next, we need to create a cellular species (a type of agent) which has the morphology as its geometric shape. To do this, right-click on the *species* node. Select the *cell* module, and in the parameters set the *morphology_id* to be the *coccus* morphology we have just defined. When linking modules in this way, you either have to type in the id, or can select one of the valid modules you've already defined by the ... drop down menu. Let's also change the name of this cellular species, set the *species_id* to be *"my_cell"*.

Finally, let's create a population of cells. This can be done by right clicking on the *initial_conditions* node and adding an *initial_population* module. Set the *species_id* to be *"my_cell"*, add set the population size to be 50, and

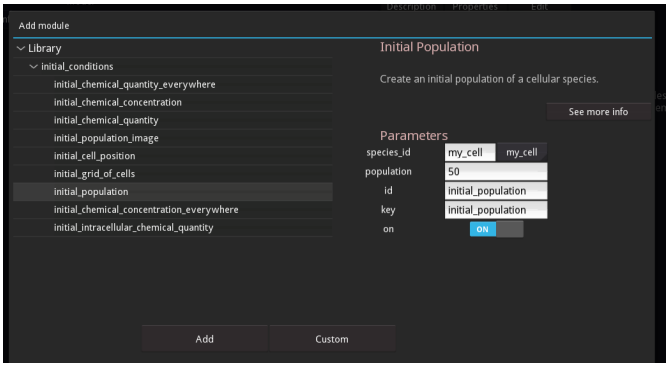


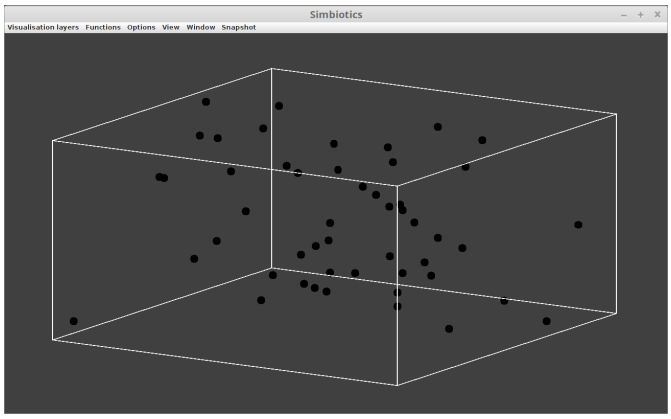






press add.



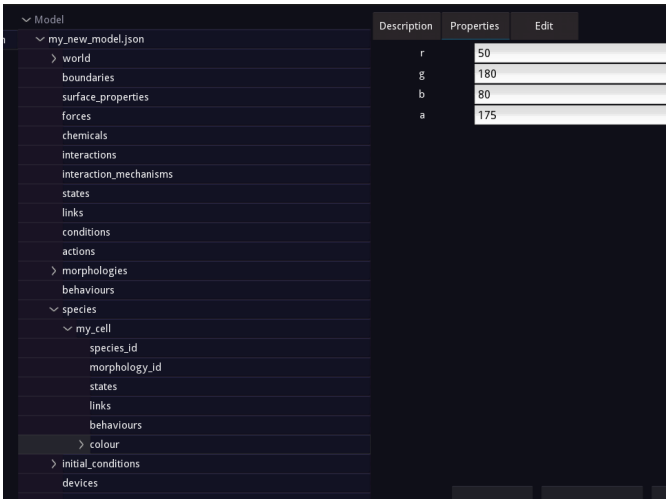


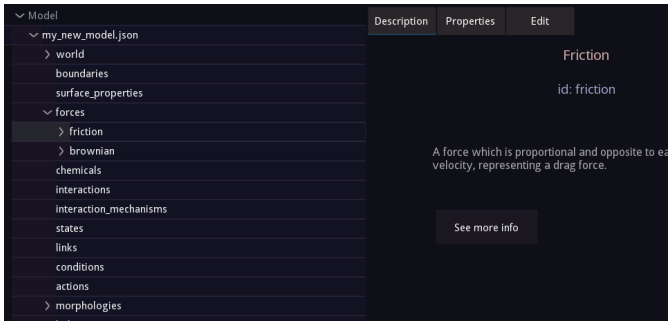
Ok... so the cells aren't doing much, lets make this more interesting. To set the colour of the cells, navigate to the *colour* property of the *cell* species you defined (by clicking on the dropdown icons on the model specification.) Select the *Properties* tab in the display panel on the right, it should show the options to set the RGBA values. Set the colour to whatever you like, we'll do a nice green (50, 180, 80, 175).

Also, let's add some movement to the system. We'll do this by adding a *friction* and *brownian* module to the *forces* node with their default parameters.

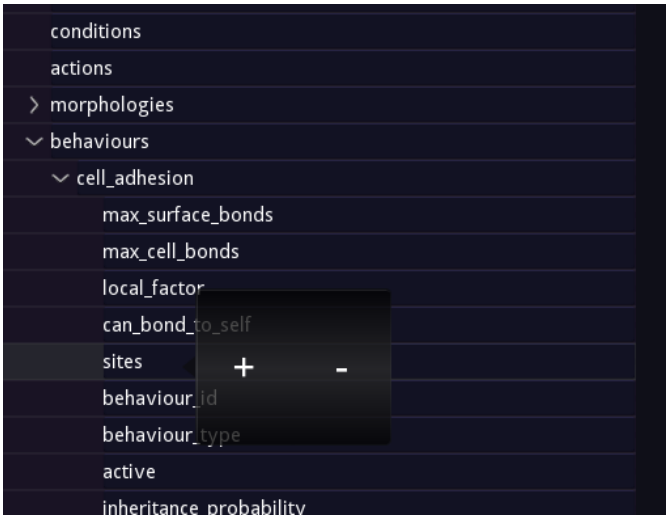
Now, when you press *Run*, you should see a population of moving cells which are the colour you set!

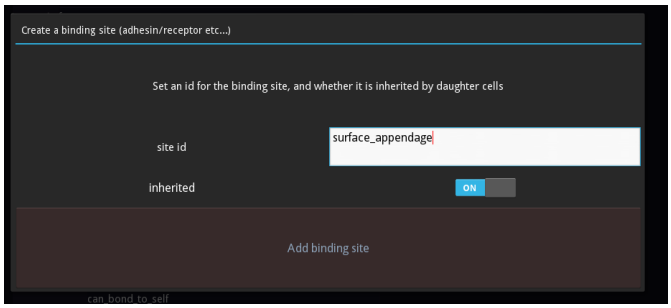
Let's now add some basic behaviour to the cells to finish off tutorial 1. We'll model that the cells have surface appendages that cause them to aggregate. To do this, we must create a *behaviour* module, called *cell_adhesion*.





Add this module to the model specification with its default parameters. We must then add a property to this *cell_adhesion* module representing a surface appendage. To do this, open up the property list for the *cell_adhesion* module, and right on *sites* then add a site called *"surface_appendage"* as seen below. The *inherited* boolean sets whether any child cell would directly inherit this surface structure, though for this model it is irrelevant we will not include cell growth.





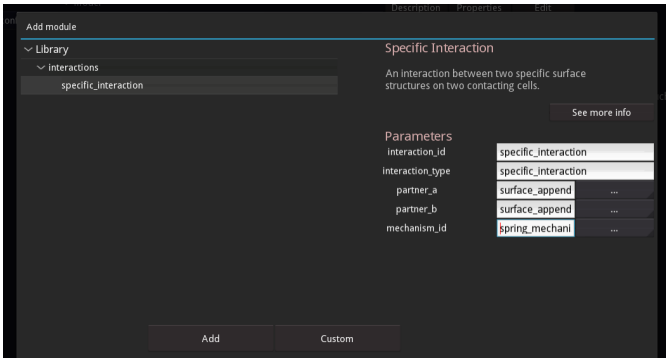
Next we need to define an *interaction_mechanism*, which is the sub-model describing how two surface appendages interact. Add a *spring_mechanism*, which describes the interaction as a Hookian spring. This forms a spring connecting the two interacting geometries according to the springs parameters. For this model, we can leave the parameters as default.

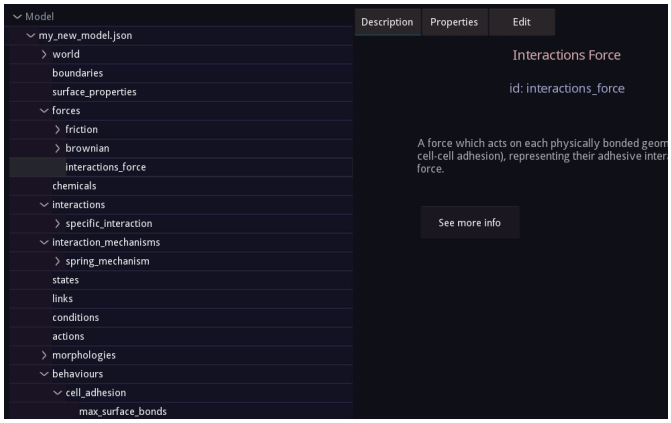
We must then create an *interaction*, which associates the surfaces appendages we defined with the mechanism. Add a *specific_interaction* module, and set the partners both to be the *"surface_appendage"*, and the *mechanism_id* to be the *"spring_mechanism"* we defined above.

Finally, we must add the *interaction_force* module to our *forces*. This is a module which includes the physical force generated by interactions to the total force that a cell experiences. Without this force module, interactions will have no physical effect.

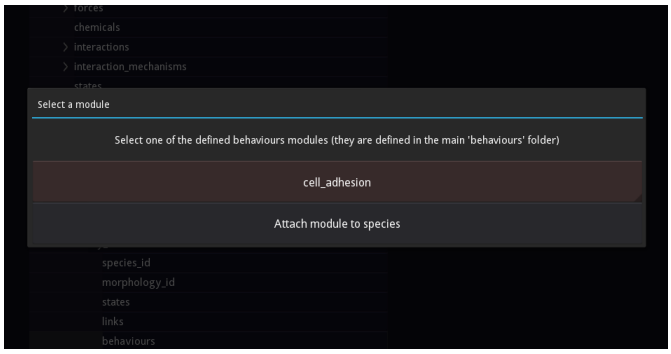
Ok, almost there! Now we have defined a *behaviour* module which has a surface appendage, and we have defined an *interaction* that states if two of those surface appendages (from two different cells) comes into contact, then the *interaction mechanism* we also defined is used to model that interaction. The only thing left is to say that our cell species, *"my_cell"*, has this type of behaviour.

To do this do, open up the species definition, and right click on the *behaviours* property which is inside the *"my_cell"* definition, press the + button, and select the the *"cell_adhesion"* module from the drop down box, and

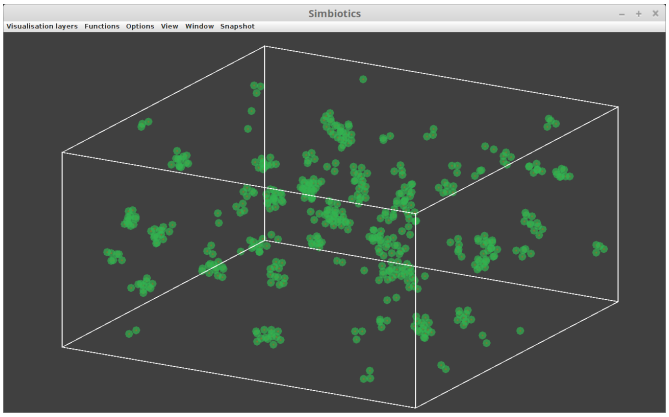


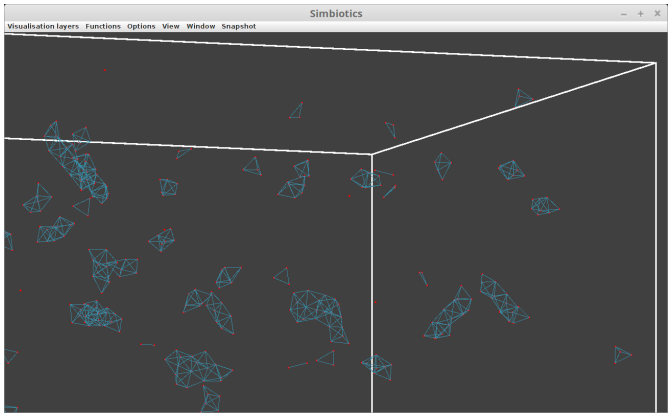


select *Attach*.



Now when you run the model, the cells should begin to aggregate, such as you see below. You may want to increase the population size so that the aggregation is more apparent. To render the wire-frame of the interactions, turn off the rendering of *"my_cell"* in the *Visualisation layers* menu of the Simbiotics GUI, and turn on the *"interactions"* layer.





This concludes tutorial 1. You can play around with the parameters in the model, and more information on these parameters can be found in the publications surrounding Simbiotics, which can be found in the *Related publications* section.

Tutorial 2 - Collecting some data from a model

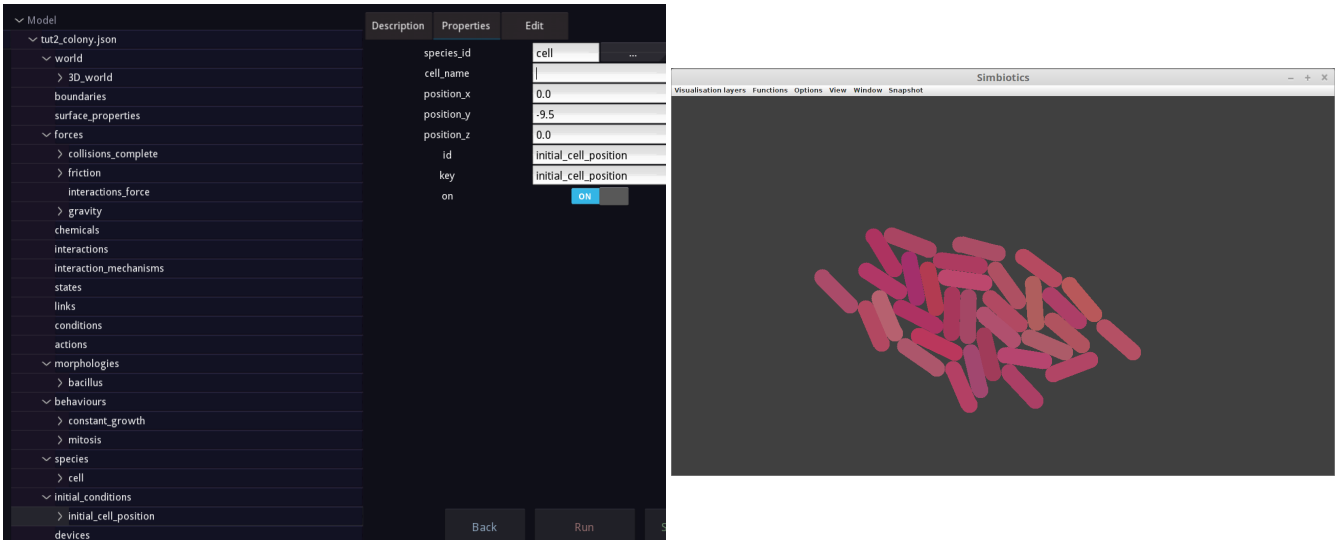
Here we run through an example of collecting data from a model, including how to visualise it live during the simulation.

We will illustrate this example through a model of a colony of bacillus (rod-shaped) cells growing on a surface. We will assume a constant nutrient supply and thus uninhibited growth. The bacillus cells secrete a chemical which diffuse out of the cell membrane into extracellular space.

To start, please create a new model, and define the following specification:

1. To world, add a 3D_world which is 50*20*50.
2. To forces, add a collisions_complete module, with a range of 10 and a k value of 50.
3. To forces, add a friction module, with a constant value of 1.0.
4. To forces, add a interaction_force module.
5. To forces, add a gravity module, with a constant value of 0.2.
6. To morphologies, add a bacillus module with a length of 1.0 and radius of 0.5.
7. To behaviours, add a constant_growth module with a growth rate of 0.025.
8. To behaviours, add a mitosis module.
9. To species, add a cell module with a the bacillus morphology.
10. In the species - modules property, attach the constant_growth and mitosis modules.
11. To initial_conditions, add an initial_cell_position, setting the species_id to the cell species you defined, and the position to be x=0.0, y=9.5, z=0.0.

Press run, and you should see a single bacillus cell at the bottom of the simulation domain which is growing and dividing.



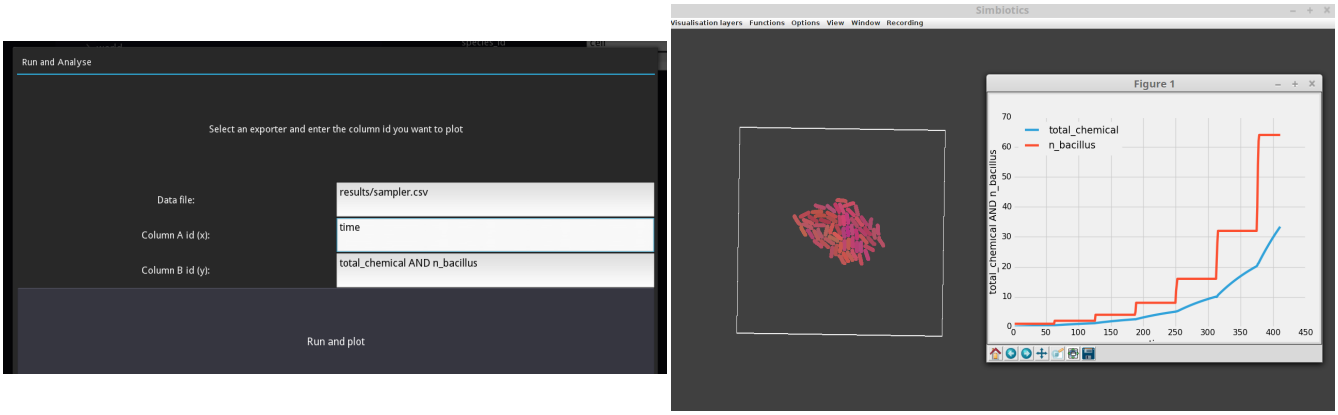
Next, we will add the module that describes their cell behaviour, specifically that they synthesize a chemical, and secrete it out of their membrane.

This is done by defining a chemical species. Add a *chemical* module to the *chemicals* definitions with a diffusion_coefficient of 1.0, a degradation_coefficient of 0.1 and diffusable set to true (ON).

Next, add a *synthesizing_grn* to the *behaviours* definitions. Set the chemical_id to be equal to the chemical you just defined, ensure the velocity_constant is 1.0 and add the module. Add this new behaviour module to the species, in the same way that you attached the mitosis and constant_growth modules to the species.

To check that this works correctly, we can plot the amount of the chemical in simulation. To do this, we must first collect the data - add a *sampler* module to the *exporters* definitions, with the the file_path set to be "results/". Now we have a sampler defined, open it up in the model specification and right click on *samples*, and add a *TotalChemicalQuantity* sample, with the *chemical_id* set to the chemical you defined, and sample title to be "total_chemical". Add a *WorldTime* sample, and title it *time*, and a *CellNumber* sample with the species_id set to the bacillus species you defined.

To periodically write the data to file, add an *export_periodically* module to the *schedules* definitions. We can now run the simulation with live graph plotting. Click *Run* on the top file bar, then select *Run with live graph plotting*. Set the data file to be the sampler exporter you defined, which is the [file path+exporter id+file extension]. In our instance, this is *results/sampler.csv* (the default results folder is in the $SIMBIOTICS main folder). The graph axes must then be defined, and they must be one of the column headers (sample titles) in the csv data file. We set the X axis to be the time data, and on the Y axis we plot both the total_chemical quantity (scale = molecule number) and the number of the bacillus cell species (scale = cell number).

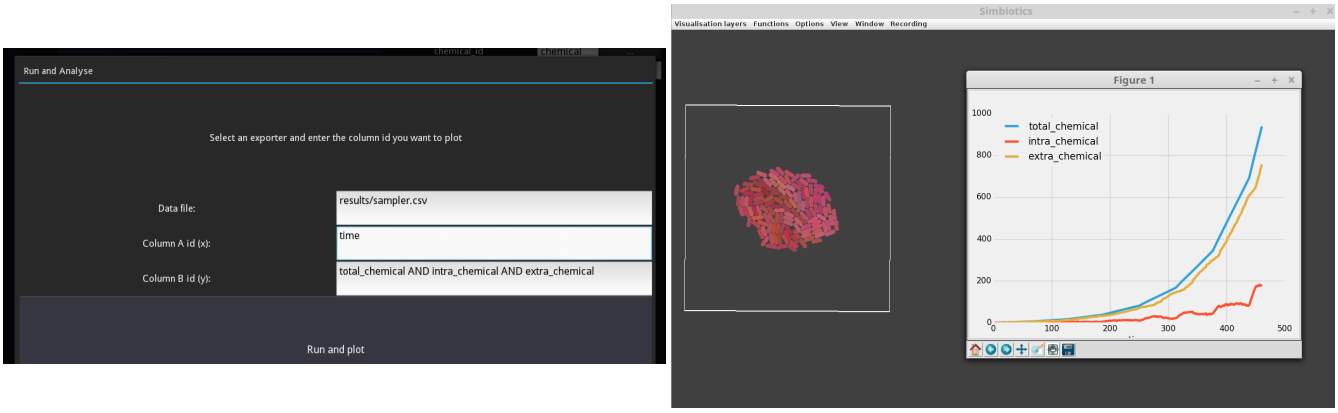


We see that as the population of cells is growing at a constant rate, doubling every 60 units of time (Note: The Simbiotics platform is unit-agnostic, meaning that whatever units you put in are the units you get out. For more information on this see the related publications section.) We also see that as their are more cells, the total rate of production of the chemical increases.

Next, we add the behaviour to the cell describing that the chemical can diffuse out of the cell membrane into the extracellular space. To do this, add a *membrane* module to the *behaviours* definition, leave its parameters as default. Open up the *membrane* definition in the model specification, and right click on the *membrane_fluxes* property. Add a flux with the chemical_id the chemical you defined, and the rate to be 0.2. Set poisson to be true (ON), which sets the solver to take a poisson distributed around the average permeation rate. Set osmotic to be true, which means that the flux direction is always from high to low concentrations. Interpolated can be left on false.

Now, we must attach that membrane module to our cell species definition, in the same way we added cell_adhesion, mitosis and the sythnesizing_grn. To ensure this works, let's plot the intracellular and extracellular quantity of the chemical over time. To do this, navigate back to the *samples* defined our *sampler* in the *exporters* definitions. Add a *TotalIntracellularChemicalQuantity* and a *TotalExtracellularChemicalQuantity*, with the titles *"intra_chemical"* and *"extra_chemical"* respectively, which record the defined chemical_id. Also, change the velocity_constant of the synthesizing_grn to 10.0, and set the *chemical* to have an degradation_constant of 0.0.

We then plot the total_chemical, intra_chemical and extra_chemical over time.


Run and Analyse
Select an exporter and enter the column id you want to plot
Data file:
results/sampler.csv
Column A id (x):
time
Column B id (y):
total_chemical AND intra_chemical AND extra_chemical
Run and plot
Simbiotics
Visualisation layers Functions Options View Window Recording
Figure 1
total_chemical
intra_chemical
extra_chemical
1000
800
600
400
200
0
0
100
200
300
400
500

Tutorial 3 - Intracellular dynamics with Gillespie submodels + multiple bacterial and chemical species

In this tutorial we will create a 2D model with a static population of bacteria (species_1). We will model a chemical influx (chemical_1) from the left hand side (X axis minimum face) and an influx of a seocnd species bacteria (species_2) from the right hand side (X axis maximum face.) The bacterial species_2 will have an active motility, called chemotaxis, such that they ascend the chemical gradient and try find the highest concentration of the chemical. The bacterial species has an active transport mechanism, taking the chemical in the extracellular space inside the cell. It also has metabolic behaviour transforming the chemical into a second chemical (chemical_2), this second chemical can diffuse out of the membrane in the extracellular space. Chemical_2 is toxic to bacterial species_1 in high concentrations, resulting in cell death.

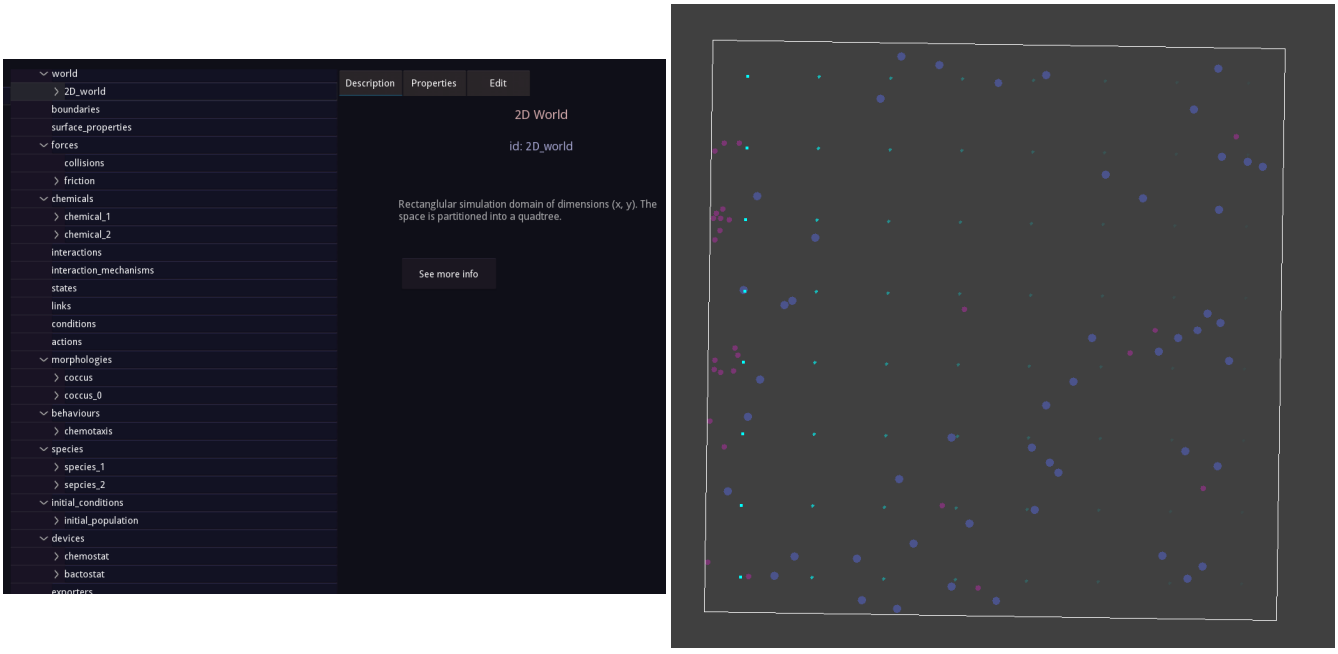
To start, please create a new model, and define the following specification:

1. To world, add a 2D_world which is 100*100
2. To forces, add a collisions module,
3. To forces, add a friction module, with a constant value of 1.0.
4. To chemicals, define chemical_1 with a diffusion constant of 2.0 and degradation 0.1.
5. To chemicals, define chemical_2 with a diffusion constant of 1.0 and degradation 0.0.
6. To forces, add a friction module, with a constant value of 1.0.
7. To morphologies, add a coccus module with a radius of 0.75.
8. To morphologies, add a coccus module with a radius of 0.5.
9. To behaviours, add a chemotaxis module with the chemical_id set to chemical_1, and interpolated set to true (ON).
10. To species, create species_1 with the first coccus morphology.
11. To species, create species_2 with the second coccus morphology, and attach the chemotaxis module to its behaviour list.
12. To initial_conditions, add an initial_population, setting the species_id to the species_1, and the population size to 50.
13. To devices, add a chemostat with default parameters.
14. Open the chemostat properties in the model specification editor, right click on fluxes and add a flux of chemical_1 with a rate of 1.0.
15. Right click on the environment_interfaces property of the chemostat, add set it to be the axis to be X, and the axis_face to be MIN.
16. To devices, add a bactostat with default parameters.
17. Open the bactostat properties in the model specification editor, right click on fluxes and add a flux of species_2 with a rate of 0.1.
18. Right click on the environment_interfaces property of the bactostat, add set it to be the axis to be X, and the axis_face to be MAX.

Press run, and you should see a static population of species_1, and species_2 cells coming into the simulation domain from the right hand size (X Max interface), which move around and hunt down the high concentration of chemical_1, causing them to migrate left to the X Min interface.

To add the membrane and metabolic pathway to species_2, do the following:

1. To behaviours, add a membrane module with default parameters.



2. Open the membrane module properties in the model editor, and add a membrane_flux of chemical_1, with a flux=0.1, osmotic=false(OFF), poisson=true(ON), interpolated=false(OFF).

3. To behaviours, add a gillespie module with default parameters.

4. Open the gillespie module in the model editor, and add a reaction (right click on reactions and click +). The id is the reaction name, call it something like "my_reaction". Set the reactants to be "chemical_1" and the products to be "chemical_2", with a rate of 0.1.

5. Add the gillespie and membrane modules to the cell species_2 behaviour definitions.

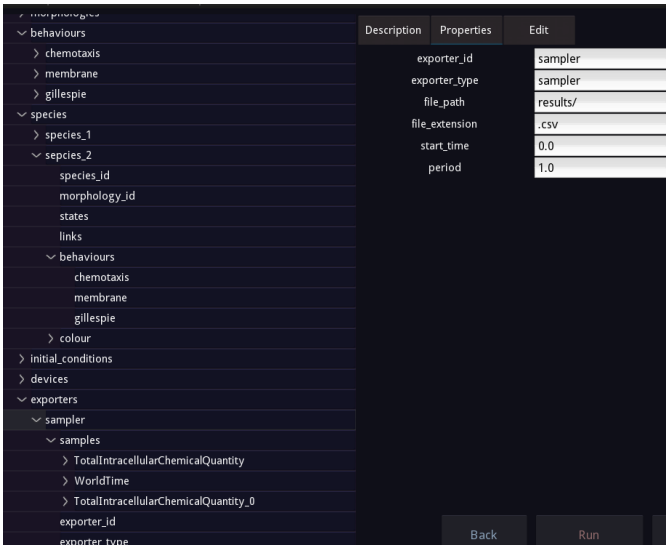
6. To exporters, add a sampler with the file path set to "results/"

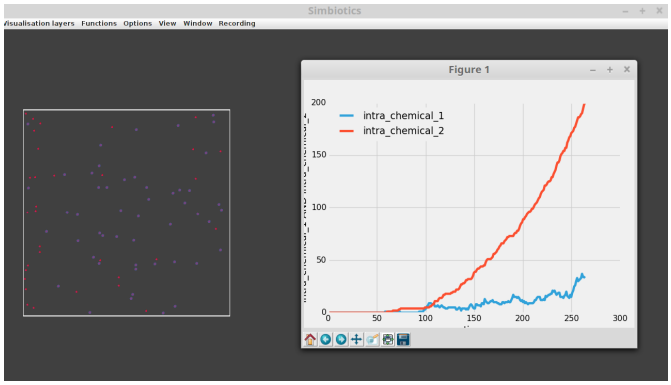
7. Open the sampler in the model editor, and add 3 samples: WorldTime with title "time", and 2 TotalIntracellularChemicalQuantity modules, one for chemical_1 called "intra_chemical_1" and one for chemical_2 called "intra_chemical_2".

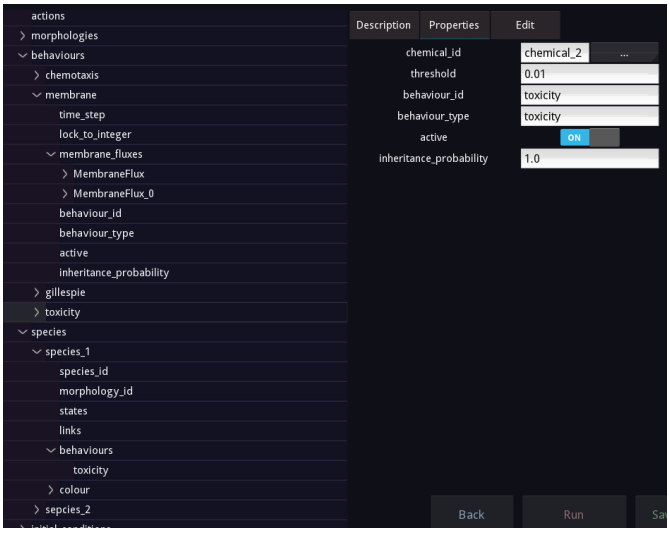
If you run the model with live plotting, and run with a custom graph, plot data file="results/sampler.csv", x="time" and y="intra_chemical_1 AND intra_chemical_2". You should see a similar output to below.

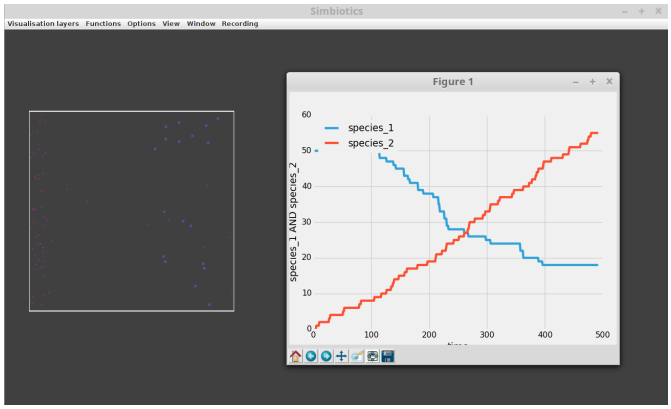
Next, we must add the transportation of chemical_2 across bacterial species_2's membrane, with a rate of 0.5 and set as osmotic process (osmotic on, poisson on, interpolated off). This causes chemical_2 to diffuse into the extracellular space. We then define a toxicity module in in our behaviour definitions, setting the chemical_id to be chemical_2, and the threshold to be 0.01. We must then add the toxicity module to bacterial species_1's behaviour list.

Additionally, we'll add two new samples to our sampler (in exporters). Add two CellNumber modules, one for species_1 and one for species_2, with their titles the same as their species_ids. The result should be that after some time species_1 cells begin to die caused by species_2 cells secreting the toxic chemical_2 product. Below we plot the cell numbers over time.


behaviours
chemotaxis
membrane
gillespie
species
species_1
sepcies_2
species_id
morphology_id
states
links
behaviours
chemotaxis
membrane
gillespie
colour
initial_conditions
devices
exporters
sampler
samples
TotalIntracellularChemicalQuantity
WorldTime
TotalIntracellularChemicalQuantity_0
exporter_id
Description
Properties
Edit
exporter_id
sampler
exporter_type
sampler
file_path
results/
file_extension
.csv
start_time
0.0
period
1.0
Back
Run


Simbiotics
Visualisation layers
Functions
Options
View
Window
Recording
Figure 1
intra_chemical_1
intra_chemical_2
200
150
100
50
0
0
50
100
150
200
250
300


actions
morphologies
behaviours
chemotaxis
membrane
time_step
lock_to_integer
membrane_fluxes
MembraneFlux
MembraneFlux_0
behaviour_id
behaviour_type
active
inheritance_probability
gillespie
toxicity
species
species_1
species_id
morphology_id
states
links
behaviours
toxicity
colour
sepcies_2
Description
Properties
Edit
chemical_id
chemical_2
...
threshold
0.01
behaviour_id
toxicity
behaviour_type
toxicity
active
ON
inheritance_probability
1.0
Back
Run


Simbiotics
Visualisation layers
Functions
Options
View
Window
Recording
Figure 1
species_1
species_2
species_1 AND species_2
60
50
40
30
20
10
0
0
100
200
300
400
500

Tutorial 4 - Cellular logic and decision making with Triggers

One way to represent cellular logic is with triggers. Triggers are composed of conditions and actions. If all conditions are true, then all actions are executed.

As a brief example, we will create a population of cells which may adhere to a surface with two types of interaction. We will tell the cell to 'differentiate' by changing colour depending on the dominant interaction it is having with the surface.

1. To world, add a $3D\_world$ with default parameters
2. To boundaries, add a *solid_boundary* with $axis = Y, axis\_face = MIN, property\_id = adhesive$
3. To surface_properties, add an *adhesive* module with $surface\_structures = structure1\ AND\ structure2$
4. To forces, add a $collisions, interaction\_force, brownian$ and $friction$ module with default parameters, then add a *gravity* module with $gravity\_constant = 0.25$
5. To behaviours, add a *cell_adhesion* module with default parameters - then add a *site* with $site\_id = adhesin$ (via the model editor drop down menu of the *cell_adhesion* behaviour module)
6. To interaction_mechanisms, add two *spring_mechanism* modules, and set $rate = 50$ for both
7. To interactions, add a two $specific\_interaction$ modules, the first should have $partner\_a = adhesin$ and $partner\_b = structure1$, with $mechanism\_id = spring\_mechanism$. The second should have $partner\_a = adhesin$ and $partner\_b = structure2$, with $mechanism\_id = spring_m echanism_0$
8. To conditions, add two *has_interactions* modules, the first with $interaction\_id = specific\_interaction$, $relation =>$ and $value = 1$. The second with $interaction\_id = specific\_interaction_0$, $relation =>$ and $value = 1$
9. To actions, add two *change_colour* modules, set the first one to have $action\_id = change\_red$, and an RGBA value of 100, 0, 0, 0. The second should have $action\_id = change\_blue$, and an RGBA value of 0, 0, 200, 100.
10. To behaviours, add two *trigger* moudles, for the first set $conditions = has\_interactions1$, $actions = change\_red$. For the second set $conditions = has\_interactions2$, $actions = change\_blue$
11. To morphologies, add a *coccus* module with default parameters
12. To species, add a *cell*. Set $morphology\_id = coccus$, and in the model editor attach the three behaviour modules (cell_adhesion, trigger and trigger_0) to the species. Also, set the *colour* property of the cell to be 80, 100, 60, 100
13. To initial_conditions, add an $initial\_population$ module with $species\_id = cell$ and $population = 100$

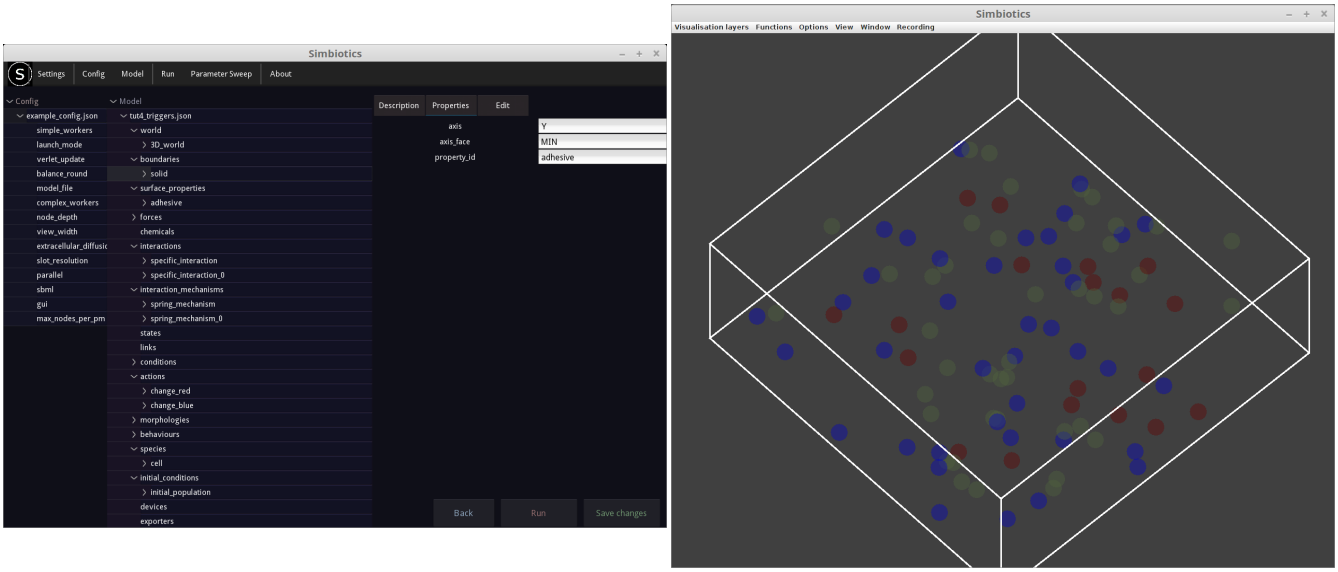
The resulting model will be a population of cells which are motile, and when they stick to the surface they will either turn blue or red, signifying which of the two interactions is dominant.

The model works as follows: cells may adhere to the Y MIN surface (bottom domain boundary), this occurs by their *adhesin* interacting with either *structure*1 or *structure*2 on the boundary. The cells have *triggers* which check which interaction is dominant, and the cell changes colour to indicate this. A trigger consists of *conditions* and *actions*, if all the conditions are true, then all the actions are executed.

The parameters to note are *rate* property of the *spring_mechanism* interaction mechanism module. This is the rate at which that interaction occurs, we originally set it to 50% for both interactions, therefore we got roughly an equal distribution of red and blue cells. The *cell_adhesions* property $max\_surface\_bonds$ is the maximum number of interactions the cell may have with the surface (domain boundary). The default value for this is 3, which is why in the conditions we check if the number of a given interacton is > 1 (as a cell could have two type 1 interactions and one type 2 interaction, visa versa, or have all 3 interactions 'occupied' by a single type of interaction.)

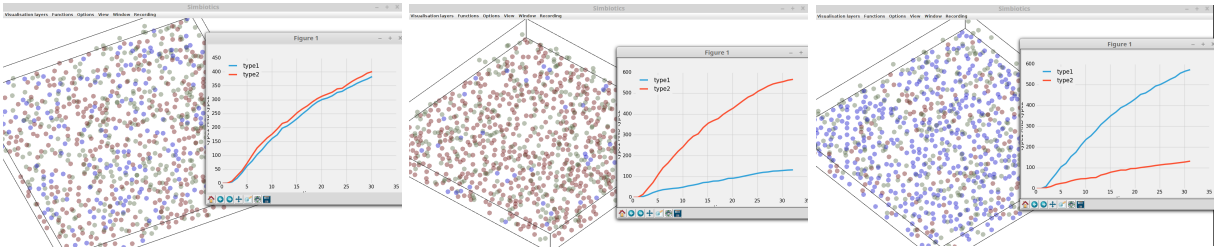
You can test out changing these parameters and seeing how the model behaviours - you could even say that a given type of interaction has a weak *spring_constant* meaning the interaction could be reversed (the spring can break easier due to the random motion on the cells).

To plot a graph showing the number of red and blue cells can be achieved by adding a *sampler* to the exporters definitions, and adding two *CellCondition* modules. The first should have $conditions = has\_interactions1$, and set the title to something like *type*1, the second should have $conditions = has\_interactions$ and $title = type2$.



Again, you may wish to add a *export_periodically* module to the schedules (alternatively press 'a' on the keyboard to write all buffered results to file).

Try playing around with the parameters to see what you get, bellow we show results for a some different rates of the type1 and type2 interactions.



(a) (b) (c)

Figure 20: **(a)** rate of type 1 = 50, rate of type 2 = 50 **(b)** rate of type 1 = 25, rate of type 2 = 75 **(c)** rate of type 1 = 75, rate of type 2 = 25

Tutorial 5 - Intracellular dynamics using SBML files

Another way to represet cellular logic/dynamics is through an SBML model. An SBML model can be loaded as a behaviour - as with all modules, each cell has its own 'version' of this model.

Before we start the SBML tutorial, note the following steps when using SBML files:

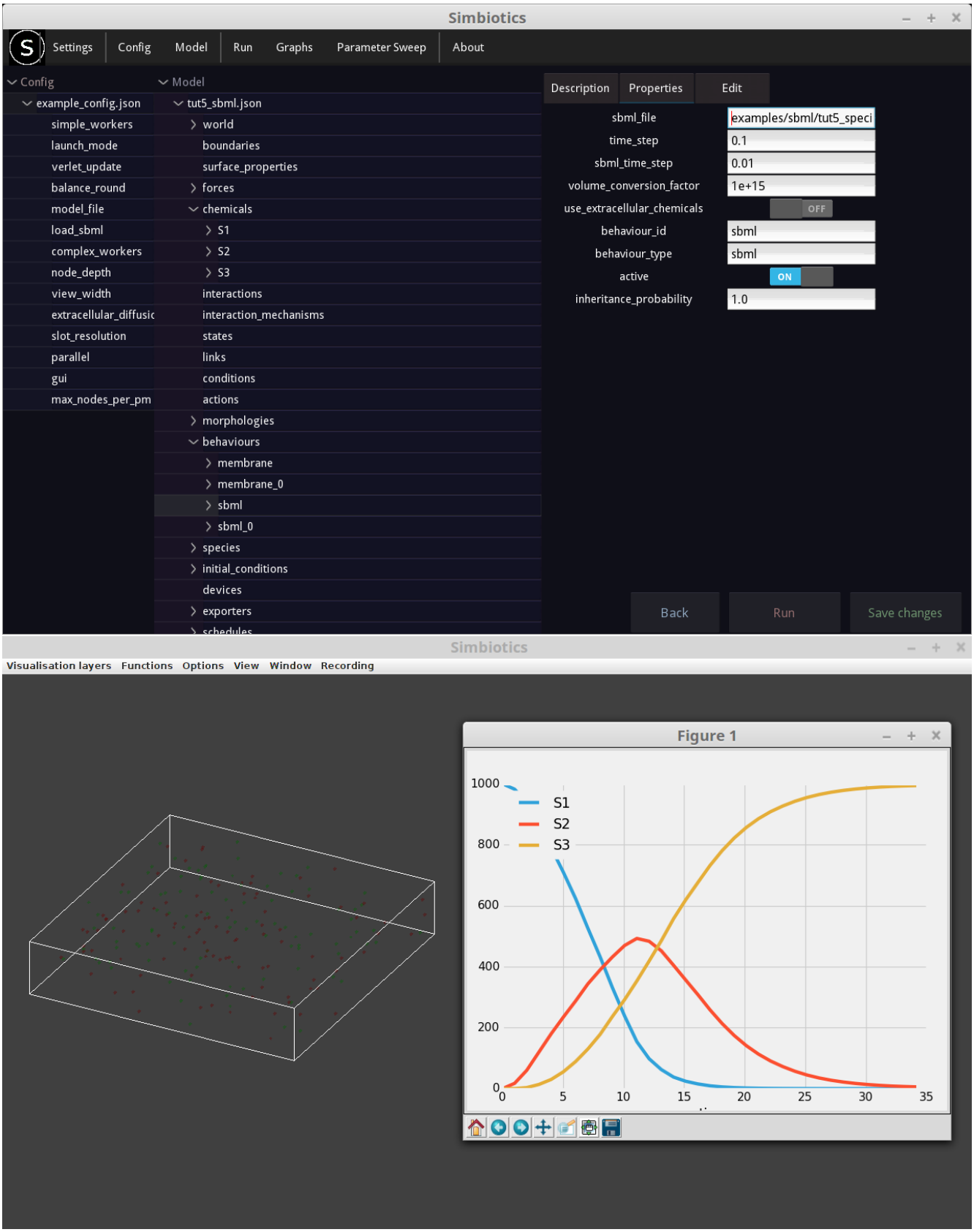
1. Each file should only have a single compartment (which represents a cell's volume)
2. Any species you want to diffuse in/out of the cell need to be defined in the Simbiotics model (with the exact same ID!)
3. Those diffusable species will be handled by the Simbiotics membrane transport system that you'll define, therefore you do not need membrane related reactions in your SBML model.
4. Remember, you can just use 1 SBML file, and create many unique instances of it, or you may use many SBML files (to define multiple species)
5. PLEASE CHECK THE SBML FILE VALIDITY. You can use this link .
6. The SBML simulator Simbiotics uses, libSBMLsim, does not handle SBML events

SBML is integrated into Simbiotics by calling the SBML simulator (libSBMLsim) to solve each cell's intracellular dynamics. The SBML simulator is called every *time_step* of simulation time (we'll discuss the parameters below), and the solver has its own internal *sbml_time step*, which is solved for the whole *time_step*. Simbiotics then integrates the SBML simulator result.

For this exercise, we will create 2 species of cells using 2 SBML model files. These files can be found in $\$SIMBIOTICS/examples/$, called "tut5_species1.xml" and "tut5_species2.xml".

The species 1 SBML model has a reaction which turns chemical A into chemical B, and the species 2 SBML model has a reaction which turns chemical B into chemical C. The A to B reaction is set to be at a rate double that of the B to C reaction.

1. To world, add a $3D\_world$ with default parameters
2. To forces, add a *brownian* and $friction$ module with default parameters
3. To chemicals, add 3 *chemical* modules, called S1, S2 and S3. All of them shouldd have $diffusable = ON$, $diffusion\_rate = 10$ and $degradation\_rate = 0$
4. To behaviours, create a *membrane* module. Add two $membrane\_fluxes$, the first should transports S1 inside the cell at a rate of 1.0, and set only *poisson* to be ON. The second should transport S2 outside the cell at a rate of -1.0, and again set only *poisson* to be true.
5. To behaviours add another *membrane* module, add a flux that that transports S2 at a rate of 1.0 and one that transports S3 at a ratio of -1.0. Again, both should only have *poisson* set to be ON.
6. To behaviours, add two *sbml* modules. For the first, set the $sbml\_file$ to $examples/sbml/tut5\_species1.xml$, and the $time\_step$ to be 0.1 and $sbml\_time\_step$ to 0.01. The second should be the same except its $sbml\_file$ should be set to $examples/sbml/tut5\_species2.xml$. Note: the $time\_step$ variable is the step between the SBML model being solved, and the $sbml\_time\_step$ is the internal time step for the SBML solver
7. To morphologies, add a *coccus* module
8. To species, add two *cell* species modules. Try and guess what's next - we're going to attach the first *membrane* and the first *sbml* module (the ones that deal with S1 and S2, to the first *cell*, and to the second *cell* we add the second *membrane* and *sbml* modules, which deal with S2 and S3.
9. To initial_conditions, add two $initial\_population$ modules, creating 100 of each of the *cell* species.
10. To initial_conditions, add an $initial\_chemical_1{}_quantity$, and set it to be 1000 molecules of S1. (These molecules will be placed at the *position* defined, it's 0, 0, 0 by default, which is the very center of the simulation domain).
11. To exporters, add a *sampler*, and add $TotalChemicalQuantity$ samples for all 3 chemicals S1, S2 and S3. If you want, you could also add a $TotalIntracellularChemicalQuantity$ and a $TotalExtracellularChemicalQuantity$ for each of them too.
12. To schedules, add an $export\_periodically$ module.


Simbiotics
Settings
Config
Model
Run
Graphs
Parameter Sweep
About
Config
example_config.json
simple_workers
launch_mode
verlet_update
balance_round
model_file
load_sbml
complex_workers
node_depth
view_width
extracellular_diffusic
slot_resolution
parallel
gui
max_nodes_per_pm
Model
tut5_sbml.json
world
boundaries
surface_properties
forces
chemicals
S1
S2
S3
interactions
interaction_mechanisms
states
links
conditions
actions
morphologies
behaviours
membrane
membrane_0
sbml
sbml_0
species
initial_conditions
devices
exporters
Description
Properties
Edit
sbml_file
examples/sbml/tut5_speci
time_step
0.1
sbml_time_step
0.01
volume_conversion_factor
1e+15
use_extracellular_chemicals
OFF
behaviour_id
sbml
behaviour_type
sbml
active
ON
inheritance_probability
1.0
Back
Run
Save changes
Simbiotics
Visualisation layers
Functions
Options
View
Window
Recording
Figure 1
1000
800
600
400
200
0
S1
S2
S3
0
5
10
15
20
25
30
35

Tutorial 6 - Intracellular dynamics using differential equations

Intracellular dynamics can also be specified using differential equations. The *grn* behaviour module is used to do this. Note: despite the name 'grn' it can be used to represent things other than a gene regulatory network, for example it may be used to modify an agents mass (it can be used for biomass growth).

To exemplify use of a differential equation module, we will create a population of a single bacterial species, in an domain (world) filled with chemical S1. The cell species can uptake the chemical though its membrane, and consumes it to both grow in mass and synthesize chemical S2, which it secretes out into the extracellular space. We'll set chemical S1 not to degrade in the extracellular space, and chemical S2 to degrade quite fast in the extracellular space.

You can find the model at "$SIMBIOTICS/examples/models/tut6_equations.json". We will not go through all the steps of the model building as they can be found above, rather we'll focus on the differential equation module.

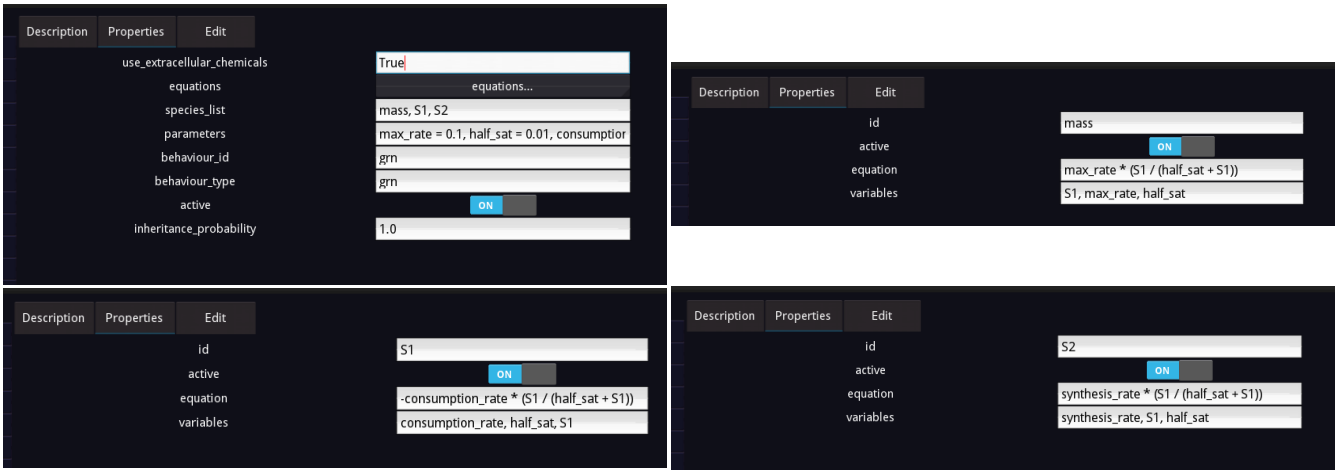
Create a world set up with a single cellular species in it, and defined the two chemical species, set S1's degradation rate to 0.0, and S2's degradation rate to 0.1. Add a membrane flux so that S1 is transported into the cell at a rate of 1.0, and S2 out of the cell at a rate of -1.0, both with the poisson sampler set to be on.

Now, create the *grn* behaviour module. You must first set the *species_list* - as we'll be working with S1, S2 and modifying the cell's *mass* (which is an accessible property via its ID), we must define those three, separated by a comma:

$species\_list = mass, S1, S2$

Now that we have defined the species, we must add some equations. Right click on the *equations* property in the *grn* module (in the model editor view). The equation *id* should be the species you are modifying (either mass, S1 or S2). The *equation* field sets the calculation to work out $\frac{dS_i}{dT}$. It may be an expression with variables/constants too, for example you may refer to any of the species in the *species_list* you just defined. You may also refer to a custom named variable, for example k, and you must define it in the *parameters* field, in the form of $variable = value$.

An example can be seen in the figure below of us setting the equations for this system.



Here are all the values for the *grn* behaviour module:

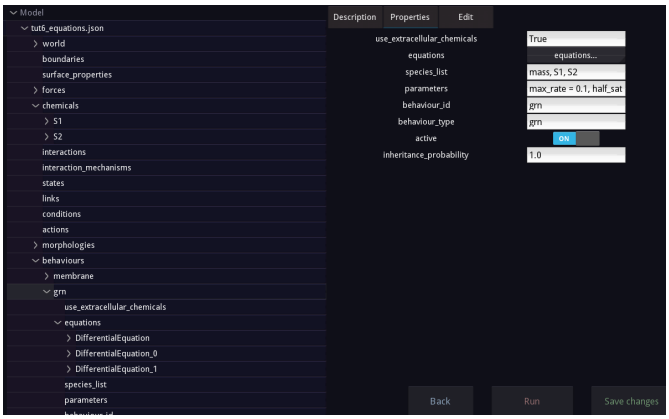
1. species_list = mass, S1, S2
2. parameters = max_rate = 0.1, half_sat = 0.01, consumption_rate = 0.1, synthesis_rate = 0.1
3. Equation 1, id: = mass
4. Equation 1, equation: = max_rate * (S1 / (half_sat + S1))
5. Equation 1, variables: = S1, max_rate, half_sat
6. Equation 2, id: = S1
7. Equation 2, equation: = -consumption_rate * (S1 / (half_sat + S1))
8. Equation 2, variables: = consumption_rate, half_sat, S1
9. Equation 3, id: = S2

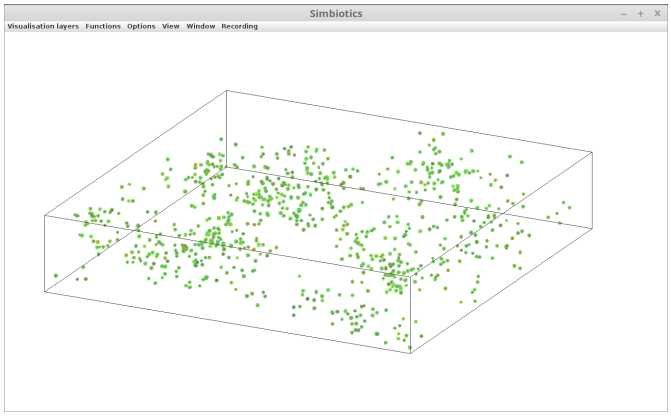
10. Equation 3, equation: = synthesis_rate * (S1 / (half_sat + S1))

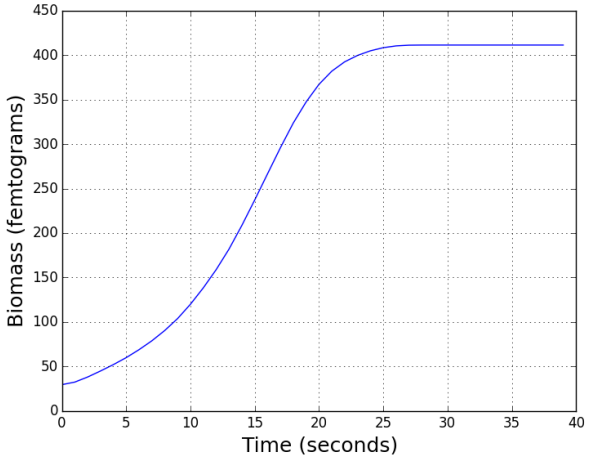
11. Equation 3, variables: = synthesis_rate, S1, half_sat

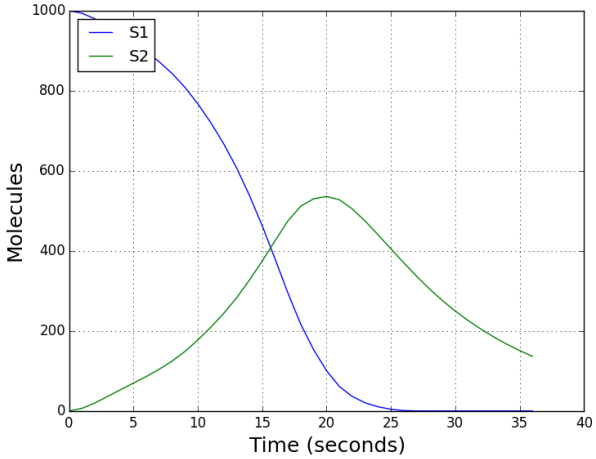
Now create an initial population of the cell species, and add an initial amount of S1 into the extracellular space. Also, don't forget to attach the behaviours you defined to the cell species! (Plus attach some data exporters if you want to generate some graphs)

You can run the model get a result similar to what we see below. The cells should stop growing and producing S2 once all of S1 is consumed - the remaining S2 then degrades in the extracellular space. Try playing around with the parameters, or add another species which consumes S2.









Tutorial 7 - Live graph plotting

Live graph plotting can be achieved by defining graph objects and attaching them to a simulation run. The defined graphs can be saved to file and loaded again for easy reuse.

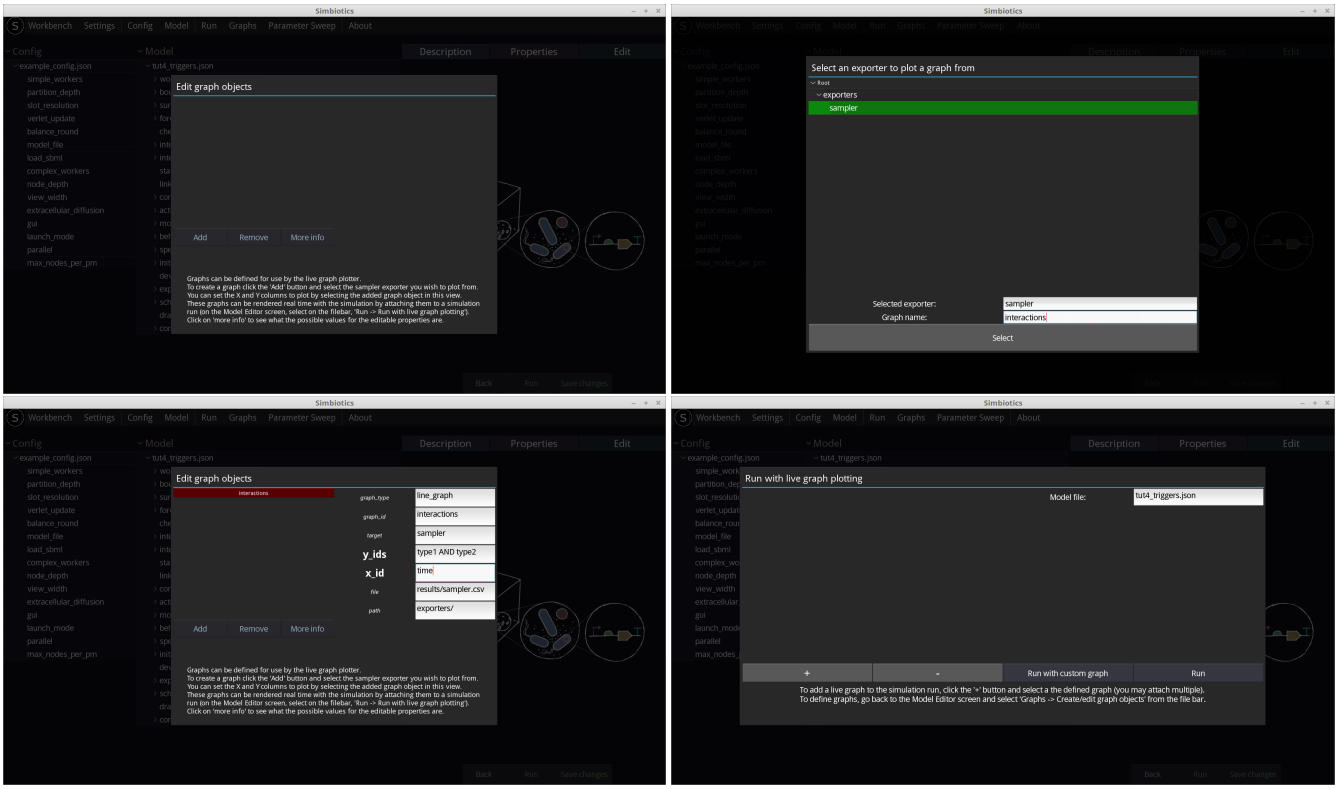
In this tutorial we'll reuse the model from Tutorial 4, and build the graph objects to render our graphs, rather than using the custom graph mode as we did in Tutorial 4. So, load up that model (a copy of it can be found at *examples/models/tut4_triggers.json*.

To create a graph, click on *Graphs* on the file bar, and select *Create/edit graph objects*. Click on *Add*, and select the *sampler* exporter. You can name the graph, in our case we call it *interactions* (see figures below), then press *Select*.

Now you have added a graph for that exporter, you can set what X and Y values you want to plot. Click on the graph in the list view, and it will bring up the properties on the right hand side. You may set the **bold** variables. *y_ids* are the columns in the data file to be plotted on the Y axis, and *x_id* is, as you can guess, is the column to be plotted for the X value. The values for *y_ids* and *x_id* are the sample titles that you wish to plot (the titles become the column headers in the *.csv* file which is exporter by the sampler). To see your sample titles, go back to the model editor view, open the nodes for *exporters - sampler - samples*, then you can click on each individual sample, and see/set its title in the property view.

Note: the *x_id* should only be one value (one sample title), but the *y_ids* may be set to be many values. This is achieved by chaining together sample titles separated by *AND*. See the figures below for clarification on this.

For our graph we'll plot *time* on the X axis, and for the Y axis we'll plot *type1 AND type2*. Once you've set the X and Y columns to plot, go back to the model editor. We can now run a simulation with this graph attached to the run. To do this, select *Run - Run with live graph plotting* from the file bar. Now you can click the + button, and select your graph, then press *Attach*. You can now run the model, and you should have a lovely live graph alongside your simulation!



Tutorial 8 - Parameter sweeps

Parameter sweeps can be conducted. The value of a selected parameter is iterated across a defined range, and a simulation is run for each value in that range. The data for each individual simulation is stored in its own subfolder for further processing. You can also run live graphs/post rendererd graphs with the parameter sweep, where each individual simulation data is plotted on the same graph for easy comparison.

Many parameter sweeps can be defined, and you can set the to run in a combinatorial manner to explore the entirity of the parameter space (all combinations of the attached parameter sweeps are simulated), or alternatively you can run them indepedently to observe the effect of each indiviudal parameter on the system.

In this tutorial we'll use the model as we left it after the previous tutorial (Tutorial 7), so load that up. To create a parameter sweep, click on *Parameter sweeps - Creat/edit parameter sweep objects* on the file bar of the model editor. Same as for the graphs, press *Add* and select the associated model object. For parameter sweeps you can only select properties with a numerical value. In this tutorial we'll select the *rate* variable for the *spring_mechanism*.

Once you have your parameter object defined, click on it in the list view, and edit the properties which appear on the right hand side. The *range* property can take values which match the following forms:

range = [A,B,C,...,Z]
range = [A-Z]

Where A-Z are placeholders for numerical values. And the *interval* is a numeric value describing the interval between each value when iterating through the range. If you use the first form then the *interval* setting is ignored, as each value (separated by a comma) is iterated through, so you can leave the parameter value empty. For this tutorial, we want to set the interaction rate (of *type1* interactions occuring) to be 0, then 50, then 100. There are two ways we can do this, choose either:

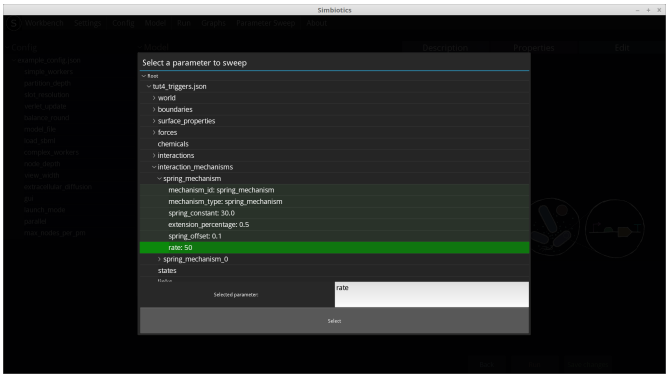
range = [0, 50, 100]
range = [0-100], interval = 50

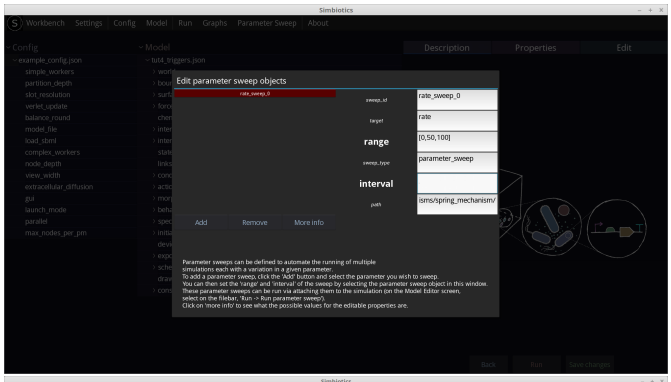
Now that you have your parameter object and its properties set, you can go back to the model editor view, and press *Run - Run parameter sweep* from the file bar. To attach the defined parameter sweep object to this simulation run, press the + button and attach your sweep object. You're now good to go! But first, lets look at the options we have:

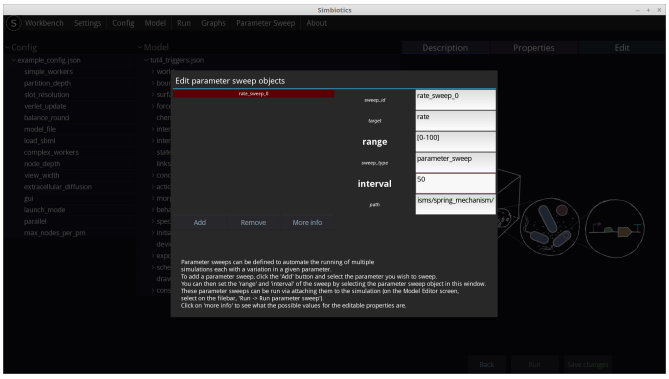
Run each sweep separately? If there are multiple parameter sweep objects attached to a run, you can either sweep each parameter individually, or you can run all combinations of all sweeps. (ON runs them separately, OFF runs all combinations).

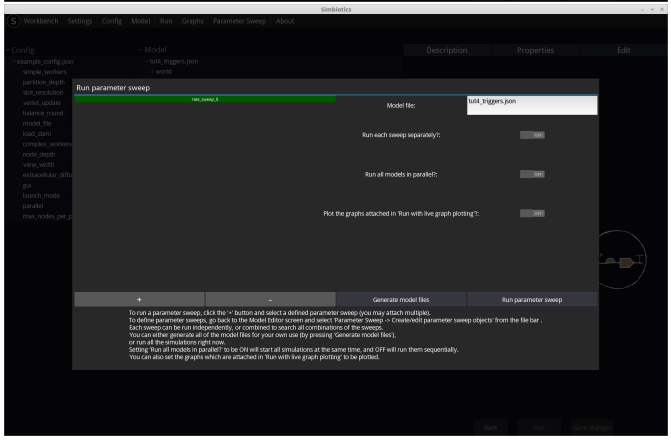
Run all models in parallel? Sets whether to run all simulations at the same time, or whether to run them sequentially one after another. Be careful if you run all simulations at the same time (in parallel), as this could consume a lot of RAM and potentially freeze your machine, so please determine how much RAM a single simulation consumes before thinking about running many at the same time. (OFF runs them one after another, ON runs them at the same time)

Run the graphs attached in 'Run with live graph plotting' If you have graphs attached then these can also be run with the parameter sweeps. (ON renders the graphs, OFF doe... forget it, you get the picture)


Simbiotics
Select a parameter to sweep
tut4_triggers.json
world
boundaries
surface_properties
forces
chemicals
interactions
interaction_mechanisms
spring_mechanism
mechanism_id: spring_mechanism
mechanism_type: spring_mechanism
spring_constant: 30.0
extension_percentage: 0.5
spring_offset: 0.1
rate: 50
spring_mechanism_0
states
Selected parameter:
rate
Select


Simbiotics
Workbench
Settings
Config
Model
Run
Graphs
Parameter Sweep
About
Edit parameter sweep objects
rate_sweep_0
sweep_id
rate_sweep_0
target
rate
range
[0,50,100]
sweep_type
parameter_sweep
interval
path
sims/spring_mechanism/
Add
Remove
More info
Parameter sweeps can be defined to automate the running of multiple simulations each with a variation in a given parameter.
To add a parameter sweep, click the 'Add' button and select the parameter you wish to sweep.
You can then set the 'range' and 'interval' of the sweep by selecting the parameter sweep object in this window.
These parameter sweeps can be run via attaching them to the simulation (on the Model Editor screen, select on the fileBar, 'Run -> Run parameter sweep').
Click on 'more info' to see what the possible values for the editable properties are.


Simbiotics
Edit parameter sweep objects
rate_sweep_0
sweep_id
rate_sweep_0
target
rate
range
[0-100]
sweep_type
parameter_sweep
interval
50
path
sims/spring_mechanism/
Add
Remove
More info
Parameter sweeps can be defined to automate the running of multiple simulations each with a variation in a given parameter.
To add a parameter sweep, click the 'Add' button and select the parameter you wish to sweep.
You can then set the 'range' and 'interval' of the sweep by selecting the parameter sweep object in this window.
These parameter sweeps can be run via attaching them to the simulation (on the Model Editor screen, select on the fileBar, 'Run -> Run parameter sweep').
Click on 'more info' to see what the possible values for the editable properties are.


Simbiotics
Run parameter sweep
Model file:
tut4_triggers.json
Run each sweep separately?:
OFF
Run all models in parallel?:
OFF
Plot the graphs attached in 'Run with live graph plotting'?:
OFF
+
-
Generate model files
Run parameter sweep
To run a parameter sweep, click the '+' button and select a defined parameter sweep (you may attach multiple).
To define parameter sweeps, go back to the Model Editor screen and select 'Parameter Sweep -> Create/edit parameter sweep objects' from the file bar .
Each sweep can be run independently, or combined to search all combinations of the sweeps.
You can either generate all of the model files for your own use (by pressing 'Generate model files'),
or run all the simulations right now.
Setting 'Run all models in parallel?' to be ON will start all simulations at the same time, and OFF will run them sequentially.
You can also set the graphs which are attached in 'Run with live graph plotting' to be plotted.