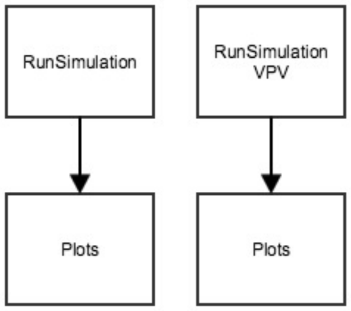
User Manual

The first few sections of this guide will give you a quick overview on how to get started on running the model. For a more detailed description of how the model works, please see the "Overview of HIV Model" section.

To start outputting raw data from the model as plots, the basic idea is as follows:



To run the model as it was built, you will only need to interact with RunSimulation.py and RunSimulationVPV.py. RunSimulation.py allows you to obtain data from a series of simulations, while keeping all physical parameter values constant. RunSimulationVPV.py allows you to perform sensitivity analysis.

First, modify either RunSimulation.py or RunSimulationVPV.py via a text editor with the settings of interest, looking into plottingkeys.csv to see what keys you want to collect data for in LIST_OF_KEY_NAMES. Each modifiable setting is accompanied with a description on how the value of that setting influences the model.

Suppose you want to collect data for the keys proteins_nuc and proteins_cyt, which you found in plottingkeys.csv. You would set LIST_OF_KEY_NAMES to have the following value:

```
# Determines what keys will be plotted
LIST_OF_KEY_NAMES = ['proteins_nuc', 'proteins_cyt']
```

To run RunSimulation.py,
visit the HIVModel directory via your terminal, and type

```
  python RunSimulation.py
```

To run RunSimulationVPV.py,
type

```
  python RunSimulationVPV.py
```

Prior to running RunSimulation/RunSimulationVPV, one also has the option to read or modify the physical parameter values inside of parameters.csv. These are the physical parameter values that are used when running each simulation. Note that most of the values stored in parameters.csv are supported by references listed in references.csv, which means that they are set by values found in the literature. When these values are unavailable, parameters are either fit heuristically using a different known parameter as a benchmark or set to a value where it does not affect the overall behavior of the system.

The following sections provide instructions on how to configure the script to output plots or raw csv files. Also, some sample use cases are found at the end of this guide.

Plot Output

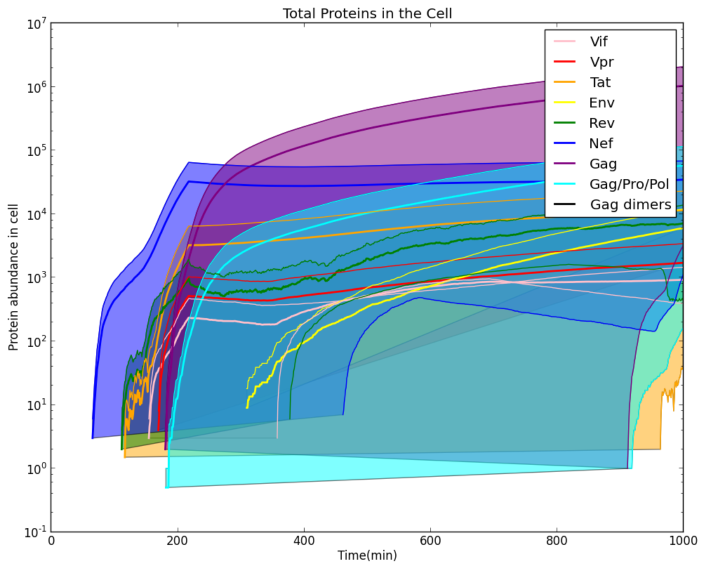
First, in RunSimulation.py or RunSimulationVPV.py, make sure that plotting is turned ON. This means that the setting to export raw data as CSV files should be turned off. That is,

```
# Export raw data as CSV from simulations instead of plotting the data
# Histograms are still plotted (if any)
EXPORT_RAW_DATA_NO_PLOT = False
```

Note: Users running this simulation on a Linux distribution with no GUI support should turn off plotting and export the data as raw data instead. See the **Raw Data CSV Output** section for more details.

In RunSimulation.py, the # of plots generated can be calculated as follows:
 # of plots = # of keys in LIST_OF_KEY_NAMES

Take a look at the plot below, which is a plot of the abundances of different proteins (from the key total_proteins) in the cell over time across 2 simulations:



Each distinctly-colored region represents some indicator of the abundance of a particular protein over time. Taking into consideration that we are collecting data across 2 simulations, the thick line of each region represents the average abundance of the respective protein across those 2 simulations. The thin line above the thick line enclosing the region of a particular color represents the avg + std. dev. of the abundance, and the thin line below the thick line represents the avg – std. dev. The program colors in the areas between the lines of a particular color to help with visual clarity.

Note that each plot corresponds to 1 key in LIST_OF_KEY_NAMES
e.g.
Consider a plot representing the key proteins_cyt

protein_cyt's data for a particular simulation consists of a 9 x T matrix, where T = number of timesteps
Each row corresponds to a particular protein, and each column corresponds to a particular timestep

Two rows of this matrix are shown below:

	1	…	T
Vif (row 1)	0	…	368
…			
Gag (row 7)	0	…	139

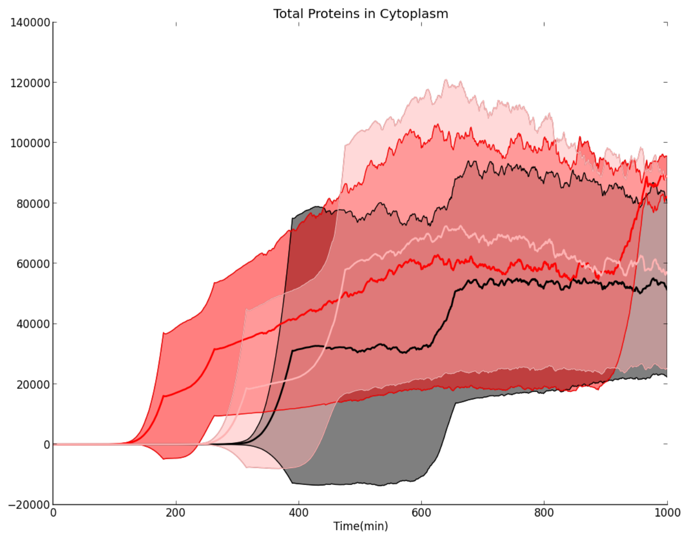
Thus, the element in the 7th row and Tth column of the proteins_cyt's matrix for a particular simulation would have a value of 368, and would correspond to the # of Gag protein in the cytoplasm during the Tth timestep of the simulation.

For RunSimulationVPV,
if plotting is enabled, which is equivalent to making sure exporting raw data as CSV is disabled:

```
# Export raw data as CSV from simulations instead of plotting the data
# Histograms are still plotted (if any)
EXPORT_RAW_DATA_NO_PLOT = False
```

then the # of figures generated can be calculated as follows:
  # of figures = # of keys in LIST_OF_KEY_NAMES
also, the number of subplots within a particular figure is:
  # of subplots = # of rows (items) in a particular key corresponding to that figure

Take a look at the sample figure below, which contains one subplot of the total abundance of proteins (from the key 'total proteins_cyt') in the cell over time across 2 simulations, for three different values of the physical parameter THRESH_TAT_FEEDBACK, 0.75, 0.8125, and 0.875.



Note that there is only one subplot in this figure, which means that there is only one row of data within the plotting key 'total proteins_cyt' If we were plotting a key like 'proteins_nuc', which gives us the abundance of different proteins across time, there would exist multiple rows within this key and each row of the key would correspond to a particular protein (e.g., Gag) and a particular subplot on the figure.

Each distinctly-colored region in a RunSimulationVPV-generated subplot corresponds to a batch, which is defined to be a set of simulations that are run under an environment where all the physical parameter values are fixed. The black region corresponds to the batch of simulations where THRESH_TAT_FEEDBACK has a value of 0.75, the red region corresponds to the batch where THRESH_TAT_FEEDBACK has a value of 0.8125, and the light red region corresponds to the batch where THRESH_TAT_FEEDBACK has a value of 0.875. In general, colors are assigned to batches from the spectrum black -> red -> light red, where an increasingly red color corresponds to a higher value of the modified parameter, and an increasingly lighter tint of red corresponds to an even higher value of the modified parameter. Each batch corresponds to a set of simulations, which can be interpreted using the explanation following the other sample plot displayed in this section.

Note that each figure corresponds to 1 key in LIST_OF_KEY_NAMES
e.g.
Consider a figure representing the key proteins_cyt

protein_cyt's data for a particular simulation consists of a 9 x T, where T = number of timesteps
Each row corresponds to a particular protein, and each column corresponds to a particular timestep

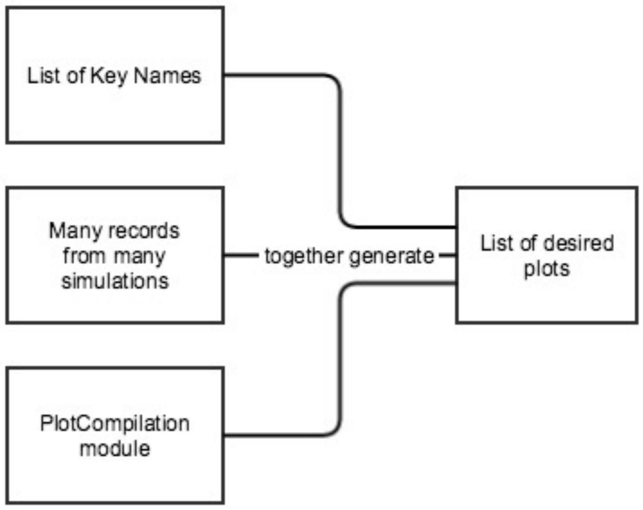
Two rows of this matrix are shown below:

	1	…	T
Vif (row 1)	0	…	540
		…	
Gag (row 7)	0	…	447

Thus, the element in the 7th row and Tth column of the proteins_cyt's matrix for a particular simulation would have a value of 447, and would correspond to the # of Gag protein in the cytoplasm during the Tth timestep of the simulation.

Here, each row (item) corresponds to a particular subplot in the figure, and note that each simulation is run under a fixed set of physical parameter values (e.g. AVE_GAG_PER_VIRON = 2500)

Plotting as a Whole
Note: 1 record corresponds to 1 simulation



<u>Definitions</u>
PlotCompilation: A module located in mainaux/ folder. This is where you set plotting options corresponding to each key. If a key is defined in terms of other keys (e.g. 'all_Golgi_forms'), you state its definition in this module. You don't really have to touch this section unless you want to modify how a particular plot is displayed (e.g., different legend colors) or want to add a new type of plot that's not already supported.

Plotting options are a list of Python parameters of this form:
[legend_colors, legend_names, plot_title, y_axis, log='off', show_legend=True, show_plot=True]
e.g.
[['Blue'], ('virion_num'), 'Number of Virion with >=40 percent expected num of Gag', 'Virion Quantity', 'off']

Note: the last two elements in the plotting options list are optional (and default to True ea.)

Raw Data CSV Output

Data obtained from RunSimulation.py/RunSimulationVPV.py can also be outputted in csv format

First, turn the .csv output setting ON in RunSimulation.py or RunSimulationVPV.py:

```
# Export raw data as CSV from simulations instead of plotting the data
# Histograms are still plotted (if any)
EXPORT_RAW_DATA_NO_PLOT = True
```

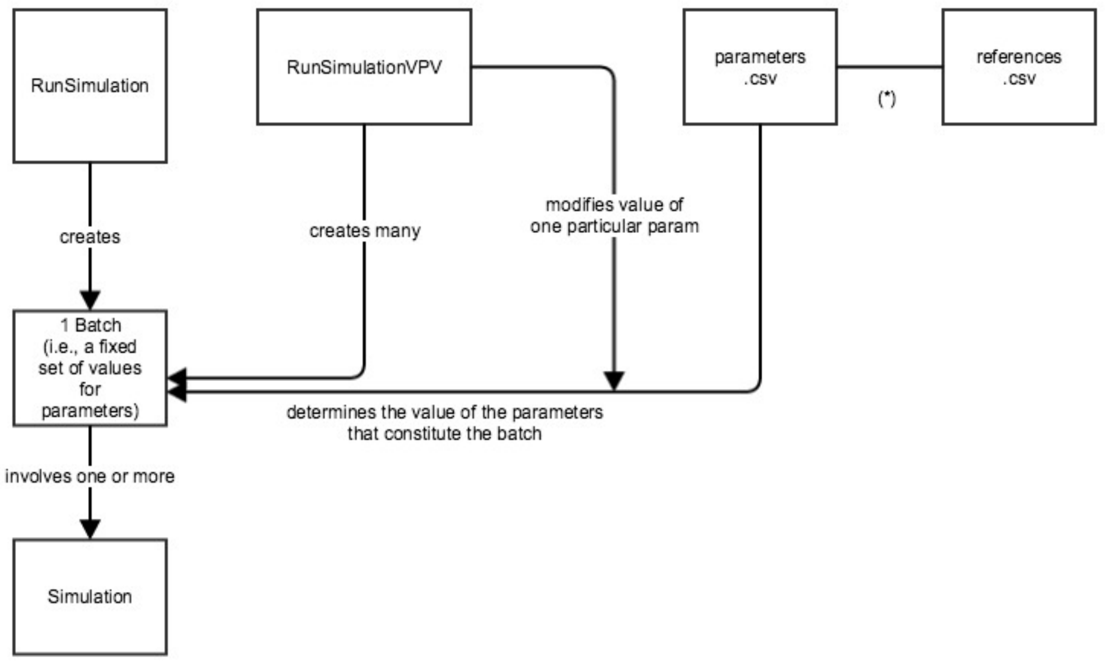
For RunSimulation, each key's data is outputted under
  outputdata/Batch<timestamp>/<key_name>.csv

For RunSimulationVPV, each key's data is outputted under
  outputdata/Batch<timestamp>/<key_name>_<param_of_interest>_<param_val>.csv
where
  <key_name>: key whose data we're collecting
  <param_of_interest>: name of physical parameter whose value we're varying prior to running each simulation
  <param_val>: value of physical parameter

When GROUP_BY_ROW_NUM is set to False,
rows that are not separated by a blank line belong to a particular set and have values that represent data from the same simulation.
Each set comprised of rows corresponds to a particular simulation number.

When GROUP_BY_ROW_NUM is set to True,
rows that are not separated by a blank line belong to a particular category within the key of interest (e.g., 'Gag' category inside the 'proteins_nuc' key)
Each row belonging to the same category has values that correspond to data collected from a different simulation.

Overview of How HIV Model Works


RunSimulation
RunSimulationVPV
parameters .csv
references .csv
(*)
creates
creates many
modifies value of one particular param
1 Batch (i.e., a fixed set of values for parameters)
determines the value of the parameters that constitute the batch
involves one or more
Simulation

*Note: Physical parameter values in parameters.csv are set by values found in the literature listed in references.csv. When these values are unavailable, parameters are either fit heuristically using a different known parameter as a benchmark or set to a value where it does not affect the overall behavior of the system.

Definitions

Batch: environment where all the physical parameter values are fixed (cannot change between simulations; e.g. AVE_GAG_PER_VIRON = 2500)

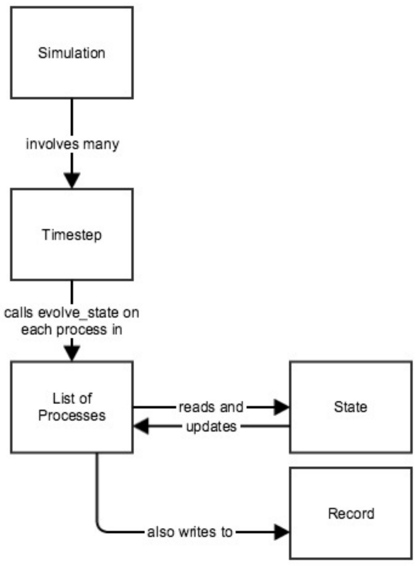
Simulation: one instance of an HIV cell interacting with a T-cell under the physical conditions specified by the batch

From the diagram, the user interacts with the user through 3 files:
RunSimulation.py, RunSimulationVPV.py, and parameters.csv

Running the script inside RunSimulation.py will lead to creation of an environment (batch) with conditions specified by the physical parameter values specified in parameters.csv. Many simulations are run under these physical conditions and data is collected

Running the script inside RunSimulationVPV.py will lead to the creation of many environments (batches) with conditions specified by the physical parameter values specified in parameters.csv and the VPV script. RunSimulationVPV will allow the user to pick a parameter via name and modify its value incrementally between batches. For a particular batch (where the physical parameter values are fixed), many simulations are run. Data is collected across the different batches so that sensitivity analysis can be performed

Looking inside a Particular Simulation



Definitions:

State: an object that contains all the relevant information regarding our T-cell during any given timestep (e.g. num of Gag proteins in the nucleus)

Process: module that takes in a State object and reads and writes values to that State object; each Process represents an underlying aspect of the HIV-T cell interaction

Record: contains information about the state across many different timesteps, in a representation that is optimal for plotting or csv file output. A matrix is associated with each key, where the row corresponds to an item (e.g. Gag in the key proteins_cyt), and the column corresponds to the timestep

From the diagram, we can see that a simulation runs many timesteps. During each timestep, a particular function evolve_state is called for each Process in a list of processes. This function takes in a State object characteristic to the simulation and reads/updates that State object depending on what aspect of the HIV-T cell interaction it represents. The Process module also writes to a Record object which collects data cross many different timesteps in a format that is optimal for plotting or csv file output.

For reference,
the list of processes run during each timestep are:
Tat feedback, Transcription, Alternative splicing, Rev binding, mRNA export, Translation, Protein localization, Degradation, Packaging, Env processing

Testing

To run all tests, cd to the HIV Model directory and type:

```
  python -m unittest discover
```

To run a test corresponding to a particular process module, type:

```
  python -m unittest tests.test_PROCESS_NAME
```

where PROCESS_NAME corresponds to the name of a supported process within the simulation.

Currently supported process names for testing include the following: AlternativeSplicing, Degradation, EnvProcessing, MRNAExport, Packaging, ProteinLocalization, RevBinding, TatFeedback, Transcription, Translation

Sample Use Cases

Example 1. Want a plot of the avg. different proteins in the cytoplasm vs. time, across multiple simulations

Steps

1. Go into plottingkeys.csv and find the key corresponding to the desired plot above. In this case, the key name is 'proteins_cyt' with the description 'Protein abundance in the cytoplasm'

2. Open RunSimulation.py with a text editor.
a) Change NUM_OF_SIMULATIONS to 5
b) Set NUM_OF_TIMESTEPS to desired number
Recommended: NUM_OF_TIMESTEPS = 48 * 60
c) Change LIST_OF_KEY_NAMES to only contain the text 'proteins_cyt'
i.e., LIST_OF_KEY_NAMES = ['proteins_cyt']
d) Make sure EXPORT_RAW_DATA_NO_PLOT is set to False
e) Set TYPE_OF_PLOT to None
i.e., TYPE_OF_PLOT = None
Note: for more than one simulation, the average value of a particular quantity across multiple simulations is always plotted as a thick line

3. cd into the HIV Model folder and execute the following command:

```
  python RunSimulation.py
```

Example 2. Want a figure containing a plot of the number of proteins in the nucleus vs. time, for three batches of one simulation each, varying the parameter 'THRESH_TAT_FEEDBACK' by a value of 0.0625 each batch.

Steps

1. Go into plottingkeys.csv and find the key corresponding to the desired plot above. In this case, the key name is 'total proteins_nuc' with the description 'Total protein abundance in nucleus'

2. Open RunSimulationVPV.py with a text editor.
a) Change NUM_OF_SIMULATIONS to 1
b) Set NUM_OF_TIMESTEPS to desired number
Recommended: NUM_OF_TIMESTEPS = 48 * 60
c) Set INIT_VAL to 0.75, which is the value reported in parameters.csv
d) Set INCREMENT_TYPE to 'linear'
e) Set INCREMENT_VAL to 0.0625
f) Set NUM_OF_INCREMENTS = (# of desired batches) - 1 = 3 - 1 = 2
g) Make sure EXPORT_RAW_DATA_NO_PLOT is set to False
h) Change LIST_OF_KEY_NAMES to only contain the text 'proteins_cyt'
i.e., LIST_OF_KEY_NAMES = ['total proteins_nuc']

3. cd into the HIV Model folder and execute the following command:

```
  python RunSimulationVPV.py
```