SimTracker Documentation

Release 0.1.1

Marianne Bezaire, Ivan Raikov

Aug 11, 2017

Contents

1 Introduction **1**

2 Installation **3**

3 SimTracker Suite **5**
- 3.1 SimTracker . . . 5
- 3.2 Network Clamp . . . 5
- 3.3 CellClamp . . . 5
- 3.4 CellData . . . 6
- 3.5 WebsiteMaker . . . 6

4 Settings **7**
- 4.1 General . . . 7
- 4.2 Machines . . . 7
- 4.3 Groups . . . 7
- 4.4 Errors . . . 7
- 4.5 Outputs . . . 8
- 4.6 Parameters . . . 8
- 4.7 Jobscripts . . . 8

5 Parameters **9**
- 5.1 NumData . . . 9
- 5.2 ConnData . . . 9
- 5.3 SynData . . . 9
- 5.4 Stimulation . . . 10
- 5.5 Connectivity . . . 10
- 5.6 Machine . . . 13
- 5.7 Code Version . . . 13

6 Models **15**
- 6.1 CA1 Model . . . 15
- 6.2 RingDemo Model . . . 15

7 Analyses **17**
- 7.1 Standard Analyses . . . 17
- 7.2 Custom Analyses . . . 18

8 Tutorials **19**
8.1 Install SimTracker and run a parallel ring network simulation 19
8.2 Network Clamp a cell using the NetworkClamp tool 26
8.3 Create a new model repository using SimTracker 27
8.4 Design, execute and upload a simulation 29
8.5 Analyzing a simulation 31
8.6 Organize simulations 31
8.7 Model Design Characterization 32
8.8 Automatically analyze experimental data using CellData 35

9 Workflow **39**

10 Support **41**

11 Indices and tables **43**

12 Bibliography **45**

Bibliography **47**

CHAPTER 1

Introduction

SimTracker works with a flexible network model template written for the NEURON simulator. SimTracker supports the modeler through every step of the modeling process, from designing the model and writing the code to executing the simulations and analyzing the results. SimTracker even helps modelers execute large scale, parallel neural network simulations on supercomputers. Additionally, SimTracker works with code versioning systems to track the code and parameters used to run every simulation, ensuring that each simulation is fully documented and that any model result can be reproduced.

SimTracker and its associated model code template allow modelers to quickly characterize their model components in experimental terms, including activation and inactivation curves for ion channels, paired recordings for synaptic connections, and current injection sweeps for intrinsic properties of single cells. In addition, SimTracker is suitable for those with limited modeling experience, providing a structured workflow that can serve as a didactic tool to introduce new modelers to good programming practices and to concepts and pitfalls associated with network models and parallel computing.

Tutorials

CHAPTER 2

Installation

SimTracker Installation on Windows, Mac, Linux.

Tutorials

CHAPTER 3

SimTracker Suite

SimTracker contains a suite of GUI-based tools to support the modeling process.

SimTracker

SimTracker

- *Tutorials*
- *Model Design Characterization*

Network Clamp

The Network Clamp technique ... from *[BRB+16]*

designnetinput designnetconn

- *Tutorials*
- *Model Design Characterization*

CellClamp

CellClamp info...

Tutorials Model Design Characterization

CellData

Experimental data

Tutorials

WebsiteMaker

WebsiteMaker

- *Tutorials*
- *Model Design Characterization*

SimTracker also includes interfaces for specifying particular parameter sets:

- *NumData*
- *ConnData*
- *SynData*

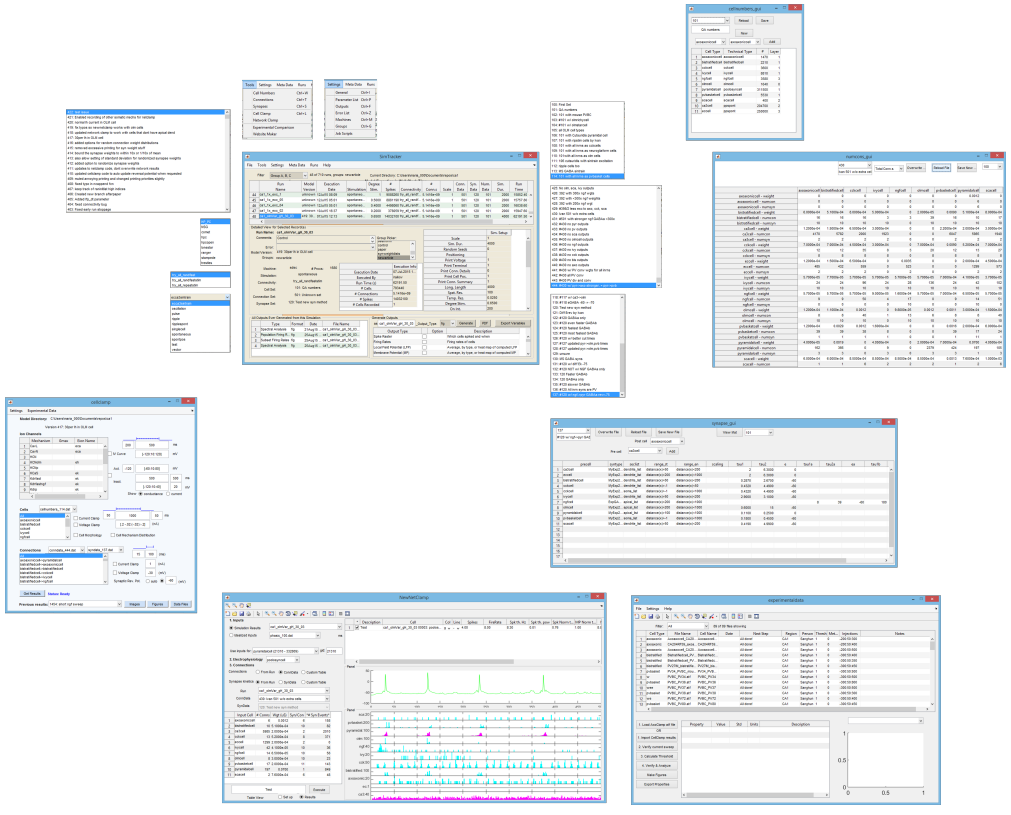


Fig. 3.1: Screen shots of the GUI-based tools and interfaces within SimTracker.

- *Tutorials*
- *Model Design Characterization*

CHAPTER 4

Settings

SimTracker includes interfaces for setting preferences and behavior of SimTracker:

General

Specification of general SimTracker behavior and interaction with other software.

Tutorials

Machines

Specification of the supercomputers to use

Tutorials

Groups

Specification of the groups by which simulations can be categorized

Tutorials

Errors

Specification of the errors by which failed simulations can be tracked

Tutorials

Outputs

Specification of the standard outputs and analyses available in SimTracker

Tutorials

Parameters

Specification of parameters used by SimTracker and in the associated NEURON model template.

Tutorials

Jobscripts

Specification of jobscripts used by SimTracker to submit simulations to batch queues on various supercomputers.

Tutorials

CHAPTER 5

Parameters

SimTracker includes interfaces for specifying particular parameter sets:

NumData

NumData specifiers the numbers of each cell type and the cell definition templates for each cell type.

- *Tutorials*
- *Model Design Characterization*

ConnData

ConnData specifiers the numbers and weights of each connection between each cell type in the model.

- *Tutorials*
- *Model Design Characterization*

SynData

SynData specifies the kinetics, reversal potential, location, and mechanisms of the synaptic connections in the model.

- *Tutorials*
- *Model Design Characterization*

Other list parameters include:

Stimulation

Stimulation Algorithms

Tutorials

Connectivity

Overview

There are two connectivity options available:

- fastconn
- repeatconn

They differ only in that *repeatconn* allows the same presynaptic cell to make multiple connections to a given postsynaptic cell while *fastconn* does not allow repeated connections. In scaled down networks with very small numbers of cells, in order to make the number of connections specified in the connection matrix (conndata_### file), it may be necessary to allow repeated connections by switching from the *fastconn* algorithm to the *repeatconn* algorithm.

The connectivity algorithm is implemented in a mod file (NMODL), compiled prior to simulation execution to allow the connection process to go much faster than if it were written completely in interpreted NEURON code.

Connectivity Algorithm

The algorithm in the NMODL file decides which cells will be connected. It is called once for each combination of presynaptic cell by postsynaptic cell type on each processor. For all cells of that type owned by that processor, first their positions are determined. Then, for each possible presynaptic type, the number of expected connections onto a given postsynaptic cell (as specified in the conndata_### file) is distributed into distance-bins according to the presynaptic cell type's axonal distribution profile. Next, the positions of all cells of the presynaptic type are determined and their distance from the postsynaptic cell calculated. For cells within a given distance-bin, the appropriate number of connections is randomly selected. If there are fewer potential presynaptic cells in a given distance-bin than are expected to fulfill both the axonal distribution and total the number of connections specified for the postsynaptic cell, the algorithm will attempt to make extra connections from other cells in nearby distance bins. The goal is to let the constraint for the total number of connections onto the postsynaptic cell allow slight distortion in the presynaptic cell's axonal distribution if that's what it takes to make the necessary number of connections. Sometimes there are not enough presynaptic cells available even considering the entire axonal extent of the presynaptic cell. In this case, when the *fastconn* version of the algorithm is used, the postsynaptic cell have fewer incoming connections from that presynaptic cell type than were specified in the connection matrix. Alternatively, when the *repeatconn* version of the algorithm is used, the postsynaptic cell can instead receive multiple connections from the presynaptic cells that are close enough, so that the total number of connections in the connection matrix is respected.

Note: when using SimTracker, it is easy to check how the actual connectivity of the network compares to the connectivity specified in the input connectivity matrix. From the list of possible analyses in the bottom right corner of SimTracker (the **Generate Outputs** panel), simply select the checkbox for **Connectivity Matrix** and type in the option 'grade' in the third column. Click the **Generate** button to see how the actual and expected numbers compare.

The NEURON model code works with the compiled connectivity algorithm in the following manner. In the NEURON code, each postsynaptic cell type is looped over in each processor. For each instance of the loop (each postsynaptic cell type), each presynaptic cell type is looped over. For each presynaptic X postsynaptic combination, a placeholder vector of connections-to-make is created and passed, along with connectivity parameters (presynaptic cell axonal distribution and positioning, postsynaptic cell positioning, number of connections expected, etc), to the compiled connectivity algorithm. The algorithm iterates through each postsynaptic cell of the given type owned by that processor, using the

parameters and positioning information of the cells to randomly select the presynaptic connections to make. It adds these presynaptic cells to the placeholder vector until all desired or possible connections have been determined for the postsynaptic cells.

Back in the NEURON code, the vector of connections to make (specific presynaptic and postsynaptic cells) is iterated through to make the actual connections. For each connection, the desired number of synapses are made by randomly selecting a location from a list of possible synaptic locations on the postsynaptic cell (for that particular presynaptic cell type) and connecting the presynaptic cell at those locations.

Printing Connectivity Information to File

Many users may wish to have a list of all connections in the model network. For small models, this is feasible and can be accomplished by setting the *PrintConnDetails* parameter prior to executing the simulation as follows:

- *PrintConnDetails = 1* // print out all synapses after the simulation is finished
- *PrintConnDetails = 2* // print out all synapses immediately after they are made
- *PrintConnDetails = 0* // do not print out all synapses ; only print out connections for small fraction of cells

If *PrintConnDetails > 0*, a *connections.dat* file will be printed along with the other results, containing a list of every synapse made in the network. For each synapse, the GID (global identifier, a unique number assigned to each cell in the network) of the presynaptic cell and the GID of the postsynaptic cell will be printed, along with a synapse ID unique to the postsynaptic cell type. Since many connections comprise multiple synapses, there are likely to be several times as many synapses (and therefore rows in this output file) as there are total connections in the model. The table below shows an example of 6 lines from a *connections.dat* file, where the postsynaptic cell with GID 45 is seen to receive 2 connections, from presynaptic cells 2 and 17, comprising 3 synapses each:

PreCellID	PostCellID	PostSynID
2	45	27
2	45	14
2	45	3
17	45	18
17	45	20
17	45	23

The *connections.dat* file can get quite large. For example, the full scale *ca1* network contains 5 billion synapses. Setting *PrintConnDetails > 0* for the ca1 network, which is not recommended, would print 15 billion datapoints to record those 5 billion synapses. Currently, there has been no attempt to allow the creation of an HDF5 file for large connectivity datasets in this model code.

Even for smaller networks for which it is feasible to print out all the synapses, printing to file may take a long time on a supercomputer. Therefore, the code by default will cause each processor to print to a separate file, and to print only the synapses that it owns. The resulting files may either be concatenated during simulation or later. The synapses owned by each processor are the synapses on the postsynaptic cells belonging to that processor. If *CatFlag==1*, the disparate files will be concatenated into a single file during the supercomputing job. To leave them as separate, set *CatFlag=0*. To print everything to a single file in serial, change the callsto *allCellConnsParallel()* (in *main.hoc* and *sim_execution_functions.hoc*) *to allCellConnsSerial()*.

To summarize:

- *(allCellConnsParallel() && CatFlag==1)* || *allCellConnsSerial()* produces a single connections.dat (allCellConnsSerial() does so very slowly)
- *(allCellConnsParallel() && CatFlag==0* produces multiple connections_[host#].dat files - one for each processor that lists only the synapses onto cells on that host/processor

Related Result Files

There are several other result files relevant to connectivity that are generated at the end of a simulation:

- allsyns.dat - gives the range of unique (possible) synapse IDs for each pre X post cell, where the synapse IDs are unique within each postsynaptic celltype
- cell_syns.dat - gives complete connectivity info for a small fraction of cells in the model
- celltype.dat - gives the mapping of cell type string to numerical typeIndex, as well as the range of gid values for that type
- numcons.dat - contains a summary of the number of connections (not synapses) in the network, broken down by preXpost combo per processor (connections are assigned to the processor that owns the postsynaptic cell):| host ID | precell type | postcell type | number of connections for that pre/post combo on that host |
- runreceipt.txt- contains parameter values used in the run. parameters of interest here would be:
  - Connectivity = 'try_all_randfaststim'; // this does not allow for repeated connections; use try_all_repeatstim to allow repeats and better innnervation in scaled down networks.
  - Scale = 1; // size of model in terms of how many real cells are represented by a single model cell (1000 is 1/1000 the size of the full model) and is used to scale down the model volume as well to keep cell density similar
  - PrintConnDetails = 0; // whether to print the full connections.dat file (see later in this email for explanation)
  - PrintConnSummary = 1; // whether to print numcons.dat summary file of connections per pre X post combo for each host/processor
  - TransverseLength = 1000; // y dimension of network volume in microns
  - LongitudinalLength = 4000; // x dimension of network volume in microns
  - ConnData = 501; // which connection matrix used to determine number of connections to make
  - SynData = 120; // synapses between each pre X post cell type must be defined here too for the connection to be made
  - CatFlag = 0; // relevant to how all connections are written out, see later in this email for explanation
  - RandomSeedsConn = 0; // to change the specific connections that are made (where along the random number generator stream to start picking numbers), change this number. This can be used to make unique networks that have the same numbers & types of connections
  - AxConVel = 0; // if set to 0, a constant delay of 3 ms is used. If set >0, this value is scaled by the distance between two cells to determine the conduction delay and added to a constantof 0.5 ms representing the cleft delay (and ensuring the delay > the timestep dt).
  - myConDelay = 1.2; //(ms) not used in this code, but could be used to allow varying the constant delay (or the constant part of the variable delay).
- sumnumout.txt - includes a line giving the total number of synapses in the model (misleadingly called NumConnections, should be NumSynapses): NumConnections = 5.14159e+09;
- synlist.dat - tells which section each possible synapse location is on the postsynaptic cell (but does not give x location within section):| postcell type | precell type | postsynaptic celltype-specific synapse ID | section on postsynaptic cell where synapse is located |

Users can also separately run launch_synapse_printer.hoc to obtain synlist.dat and allsyns.dat.

Tutorials

Machine

Computer on which to run the simulation

Tutorials

Code Version

Code versioning - how SimTracker uses git or mercurial code versioning.

Tutorials

Users may also set other parameters:

- Number of processors
- Name of simulation run
- Duration of simulation
- Scale of network

Users can also create their own parameters. More information is available in the section about customizing parameters.

CHAPTER 6

Models

Some sample models are available...

CA1 Model

The ca1 model repository...

RingDemo Model

The ringdemo model repository...

CHAPTER 7

Analyses

Simulations can be analyzed using SimTracker. A variety of analysis options are available, and users can add additional options if they wish (when using SimTracker via MATLAB scripts). Most analyses are performed on specific, defined simulations. However, users can choose to perform the same analysis on multiple simulations at once, and users can also design their own custom analyses that use any simulation data from all executed simulations in the repository.

Standard Analyses

When results from an executed simulation are uploaded to SimTracker, several standard analyses scripts become available. These scripts generate figures or tables to characterize the behavior of the network or individual cells. Multiple display or analyses options are available for several of the analyses:

- **Spike Raster: Which cells spiked and when**
  separate: the spikes of each cell type grouped separately
  pyremph: separate with pyramidal cell type emphasized
  interspersed: all celltypes plotted together
  spikedist: histogram by type of firing rate of each cell
  phasepre: phases of each spike
  individual: separate figure for each type
  rasthist: histogram of spikes by time, per type
  pyrpos: pyramidal spikes by time and position
- **Firing Rates: Firing rates of cells**
  all: include all cells in calculation
  subset: include only firing cells in calculation
- Local Field Potential (LFP): Calculated LFP analog
- **Membrane Potential (MP): Average, by type, or heat map of computed MP**
  [blank]:potential of each recorded cell (very long load time)
  [gid]:recording if cell with given gid
  average|avg:the average MP of all recorded cells
  pyr:average MP of all recorded pyramidal cells (3 character code for each cell type)
  type: average MP of all recorded cells, by cell type

- **Spectral Analysis: Spectral analysis of SDF, spike times, LFP, or MP by average, type, or heatmap, using Pwelch, periodo**

  analysis method: pwelch|gram|fft
  property to analyze: sdf|spikes|lfp|mp
  organization: type|all|{gid}|{type} ex: 3920 for {gid} or sca for {type}
  output: 2d|heatmap|table
  normalization desired: norm
- **Spectrogram: Spectral analysis as a function of time of spike times, LFP, or MP by average, type, or heatmap**

  property to analyze: spikes|lfp|mp
  organization: type|all|{gid}|{type} ex: 3920 for {gid} or sca for {type}
- **Phases & Modulation: Activity of each cell type as a function of oscillation phase**
  hist: histogram of spike times
  compass: phase and modulation of spike times
  wave: phase on an LFP wave
  table: table of phases and modulations
  mod: modulation level
  {Hz}: frequency of oscillation of interest, in Hz
  {activity}: which network property to analyze - spikes|lfp|mp|avglfp
- Cell Type Activity Table: Gid range and cell numbers for each cell type
- Cell Type Spectral Activity Graphs: Break down of cell type activity
- **Connectivity: Connection matrix, convergence, divergence, and axonal distributions for network**
  matrix: total number of connections
  convdiv: convergence & divergence
  weights: synaptic weights
  axondist: axonal distributions for recorded cells
  grade: desired v. actual connections
- Cell Positions: 3D position of each cell in network, using (hopefully) the same algorithm as the model code
- **Memory Usage: Memory used during simulation by host 0**
  Show results in: KB|MB|GB
- Run Time: Time required to run each part of simulation
- **Single Cell MP, LFP & Inputs: Single recorded cell's potentials and synaptic inputs with time**
  gid
- **Single Cell Spike Train: Spike times of a single cell**
  gid

Custom Analyses

Custom analyses for a specific model can be created as MATLAB *.m files and placed in the `customout` directory within the model code folder. SimTracker will list any files in that directory in a submenu within SimTracker when that repository is active.

CHAPTER 8

Tutorials

Here, we show how to use SimTracker to run a NEURON simulation of a ringdemo network (Tutorial *Install SimTracker and run a parallel ring network simulation*) and how to use the Network Clamp tool to run a network clamp simulation (Tutorial *Network Clamp*). Other tutorials are covered in the online documentation, including tutorials on particular functions within SimTracker, how to install SimTracker and its components individually rather than as part of a Docker package, and how to use SimTracker to run large scale parallel NEURON models on supercomputers.

All SimTracker functionality is open to customization. For users who want to customize SimTracker or its outputs, review the MATLAB code, or even run the NEURON code independently of SimTracker, we recommend consulting the online documentation at *readthedocs.org <readthedocs.org>*.

While working through the tutorials, keep in mind that steps that include the text `terminal-$ enter_this_command` mean that, at the user's local terminal, the text after the dollar sign should be entered (in this example, `enter_this_command`). Any step that advises entering "text" somewhere means that only the characters within the quotes, not the quotes themselves, should be entered.

Install SimTracker and run a parallel ring network simulation

There are three ways to install SimTracker: 1) using Docker, a virtual software container which has a preconfigured software environment that includes all software necessary to run SimTracker and NEURON in Linux or Mac, 2) by installing SimTracker on a system that has MATLAB, NEURON, and Mercurial or Git, and 3) by installing a stand-alone version of SimTracker that doesn't require a separate MATLAB license on a system with NEURON and Mercurial/Git. The latter two options are available for Linux, Mac OS X, and Windows computers. In this tutorial, the user will use the first approach on a Linux computer to create a fresh instance of a Docker container with SimTracker installed. For other approaches, see our documentation at *readthedocs.org <readthedocs.org>*

1. To create a fresh container instance with SimTracker installed, install Docker either by downloading it from https://www.docker.com or by using the package manager corresponding to the Linux distribution on the computer. Docker installation instruction for different Linux distributions are available at https://docs.docker.com/engine/installation/linux/.
2. After installing Docker, download and install the SimTracker ringdemo image with the following command:

   ```
   terminal-$ docker pull solteszlab/simtracker
   ```

This will download the SimTracker Docker image and save it on the computer.

3. Next, create a 'repos' directory which will be used for storing model code and results data generated when running simulations:

   ```
   terminal-$ mkdir repos
   ```

4. Then, start the SimTracker container by entering the following command:

   ```
   terminal-$ docker run --rm -it -e DISPLAY=$DISPLAY -v
   /tmp/.X11-unix:/tmp/.X11-unix -v $PWD/repos:/home/docker/repos
   solteszlab/simtracker
   ```

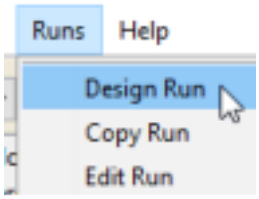
   Note: The options `-e` and `-v` ensure that GUI applications based on the X11 standard can run in the Docker container and their graphics can appear on the host computer. The second `-v` option mounts the newly created 'repos' directory as a volume inside the Docker container. It is important to mount this directory from the host computer, making it available within Docker. Unless special steps are taken, files and data saved to other directories within the Docker package, which are not accessible outside of Docker, will also not be available even within Docker after closing and reopening the Docker package.

5. Finally, run SimTracker with the following command:

   ```
   terminal-$ ./simtracker/run_SimTracker.sh
   ```

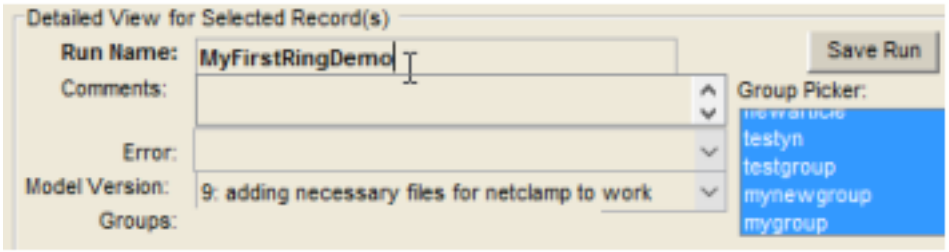
   SimTracker will start and ask for a repository directory. Select the 'ringdemo' repository included in the sample repo directory within Docker.

6. To design a new simulation, from the Runs menu choose "Design Run". The property fields in the SimTracker will become editable.



   Design new run menu item.

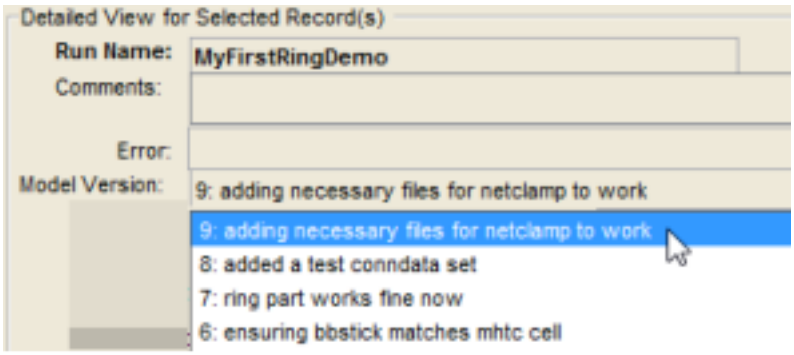
7. Enter a unique name for the simulation (allowed characters include letters, numbers, and the underscore character), such as MyFirstRingDemo.



   Edit run name.

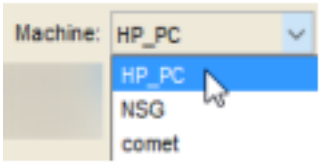
8. Optionally, enter comments about the simulation. It is often helpful to users if they note the specific question or idea that triggered them to run this particular simulation.

9. Specify the model code version to use for this simulation. Usually, this will be the most recent version of the code. The list of available code versions will correspond to each time users "committed" a version of the code in their model code repository and will also list the comments they entered when committing that code version. Select the most recent version, #9.

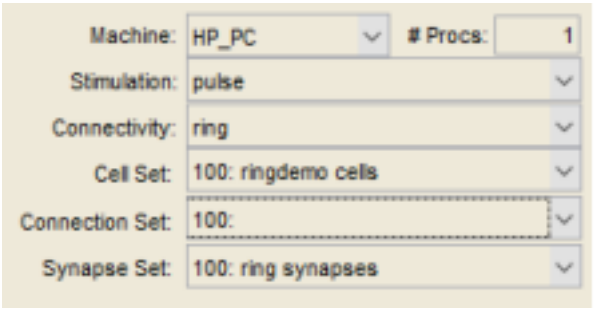


Specify model code version.

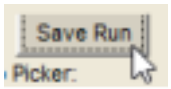
10. Specify the machine to use for running the simulation. For users to run this simulation on their own computers, they must find the unique name of their computer from the list and select it. Then, in the field to the right of the machine name, specify the number of processors to be used to execute the run. On a personal computer, users can either set this to 1 or, to run the network in parallel, set it to the total number of cores (aka processors) on their machines.

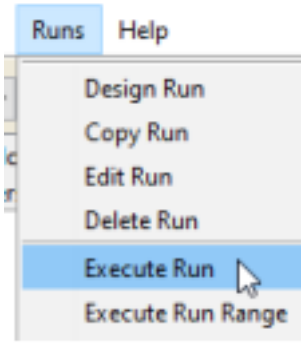


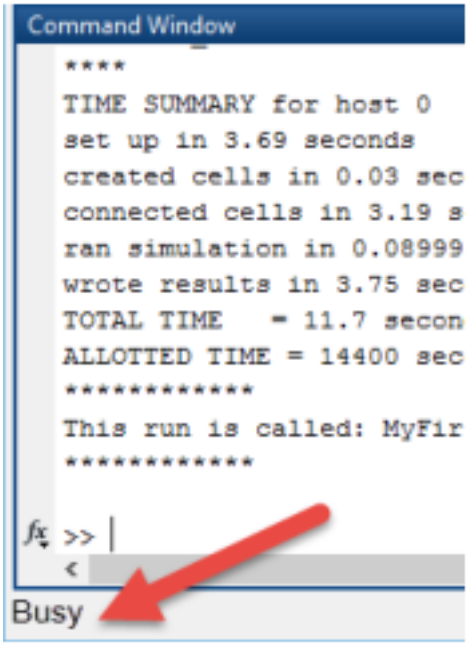
11. The default values in the various menus for simulation control (stimulation, connectivity algorithm, cell numbers, connection numbers, and synapse kinetics) can be used to run a ringdemo network of size XXXX (specify network configuration details that will be used).

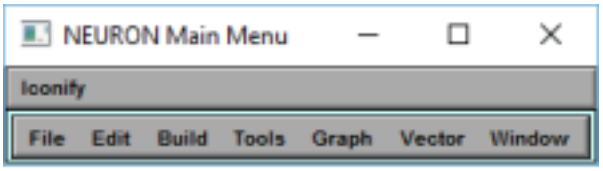


12. Upon finishing the specification of simulation properties, click the "Save Run" button that appeared when upon starting to design the simulation. The simulation will now appear in the list of simulations at the top of the SimTracker. The property fields will become uneditable (to change the run prior to execution, choose "Edit Run" from the "Runs" menu). The process for executing a simulation differs somewhat depending on whether users are executing it on their personal computers, using the NSG portal, or accessing a supercomputer independently of the NSG portal. The following list details the process for using a personal computer. For instructions on using the NSG portal or a supercomputer directly, see Tutorial *Design, execute and upload a simulation*.
13. To execute the simulation, select its row from the table at the top of the SimTracker, and then from the "Runs" menu choose "Execute Run".
14. The computer"s CPU may become very active and users who are using MATLAB scripts to run the SimTracker, will see the MATLAB status bar display "Busy" until the simulation completes Additionally, a NEURON window may pop up.

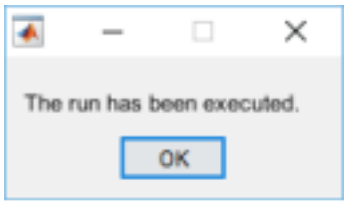

Save Run
Picker:


Runs
Help
Design Run
Copy Run
Edit Run
Delete Run
Execute Run
Execute Run Range

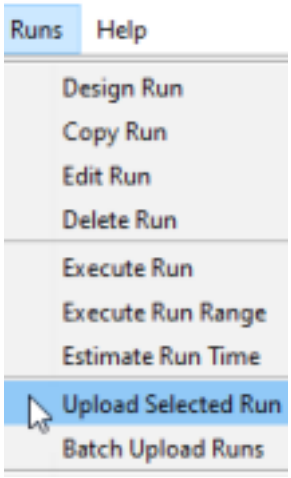

Command Window
****
TIME SUMMARY for host 0
set up in 3.69 seconds
created cells in 0.03 sec
connected cells in 3.19 s
ran simulation in 0.08999
wrote results in 3.75 sec
TOTAL TIME = 11.7 secon
ALLOTTED TIME = 14400 sec
****
This run is called: MyFir
****
>>
Busy


NEURON Main Menu
Iconify
File
Edit
Build
Tools
Graph
Vector
Window

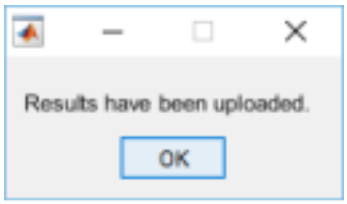
15. When the simulation finishes, a dialog box will appear that states "The run has been executed". Once this message appears, continue to the next step.



16. Select the simulation in the SimTracker (make sure to view the "Not Ran" view) and, from the "Runs" menu, choose "Upload Run" to display the results of the run in the SimTracker.



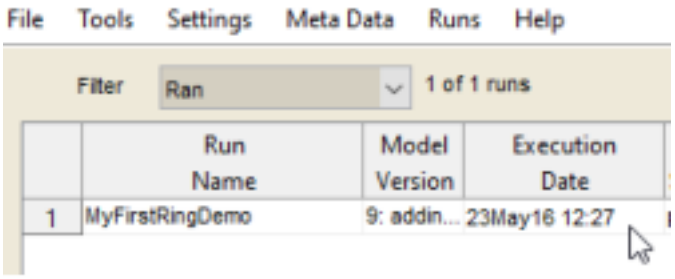
17. Once the simulation is successfully uploaded to the SimTracker, a dialog box will appear stating that "Results have been uploaded." The SimTracker table will switch to the "Ran" view. In both the list of runs and the form view below, SimTracker will show the executed simulation"s run time, number of spikes, and execution date.

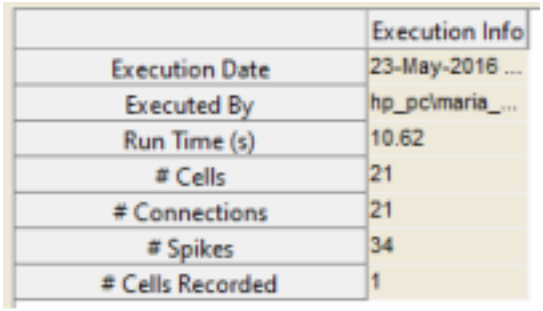


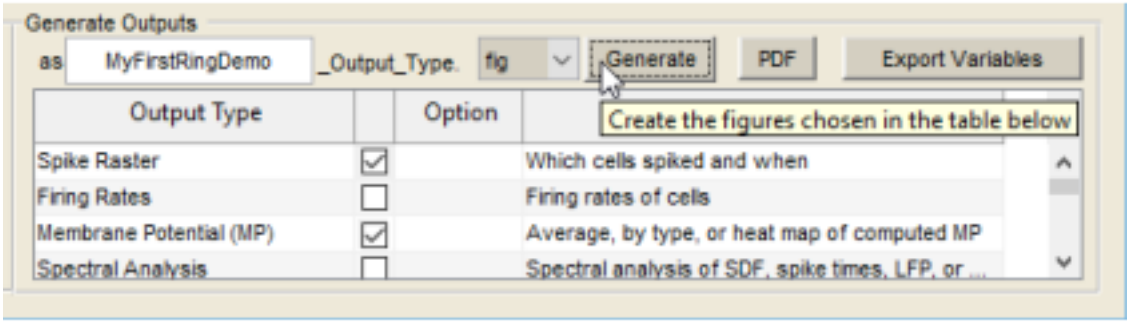
After the simulation has been executed and uploaded, it is time to analyze the results. The ringdemo simulation produces a spikeraster file that contains the times found in Table [tab:outputs:raster], which compare well to the results from *[HC08]*, found in section *3.3.5 Reporting simulation results*.

18. Select the completed simulation of interest from the table in the SimTracker.
19. Look at the table on the bottom right side of the SimTracker for a list of available outputs for that particular simulation (the list will vary depending on which output files users choose to print for their simulations). For now, click to select the checkboxes next to the 'Spike Raster' and 'Membrane Potential' outputs.

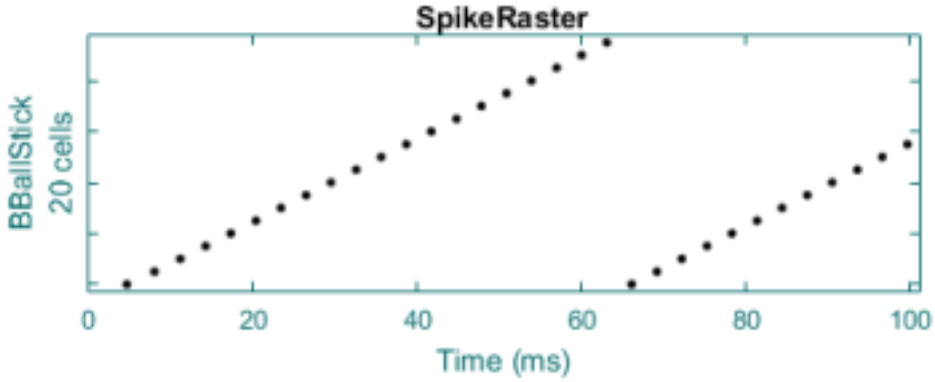
    Note: For more information on any particular output, click on that row to highlight that output type and then hover the mouse anywhere over the output table to see a tooltip with guidance about the selected output type.

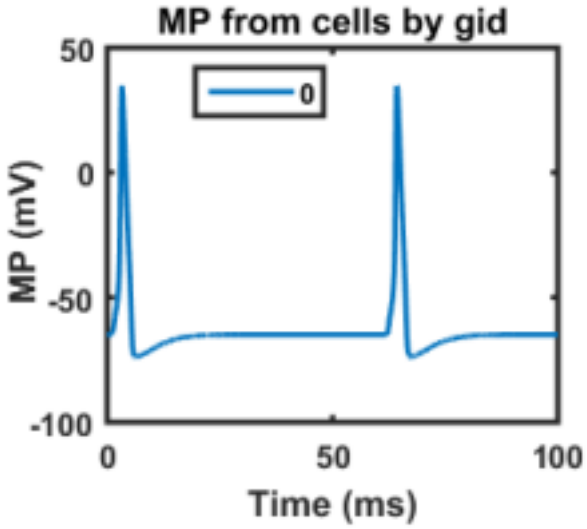

File
Tools
Settings
Meta Data
Runs
Help
Filter
Ran
1 of 1 runs
Run Name
Model Version
Execution Date
1
MyFirstRingDemo
9: addin...
23May16 12:27


Execution Info
Execution Date
23-May-2016 ...
Executed By
hp_pc\maria_...
Run Time (s)
10.62
Cells
21
Connections
21
Spikes
34
Cells Recorded
1

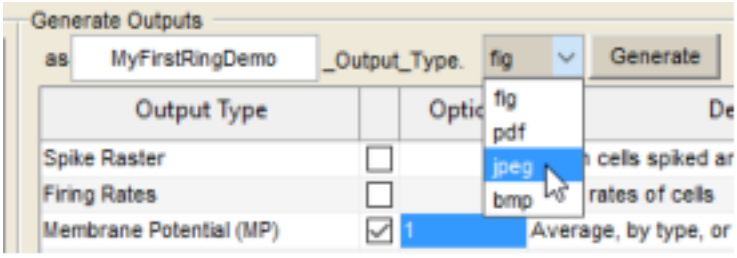

Generate Outputs
as
MyFirstRingDemo
_Output_Type
fig
Generate
PDF
Export Variables
Output Type
Option
Create the figures chosen in the table below
Spike Raster
Which cells spiked and when
Firing Rates
Firing rates of cells
Membrane Potential (MP)
Average, by type, or heat map of computed MP
Spectral Analysis
Spectral analysis of SDF, spike times, LFP, or ...

20. After selecting the outputs of interest, click the "Generate" button to produce them as MATLAB figures. If prompted to enter a GID (global ID number for a cell type), enter '0' to display the membrane potential for cell #0. Users can zoom in, pan, and otherwise edit or save the figures once the MATLAB figures are generated.

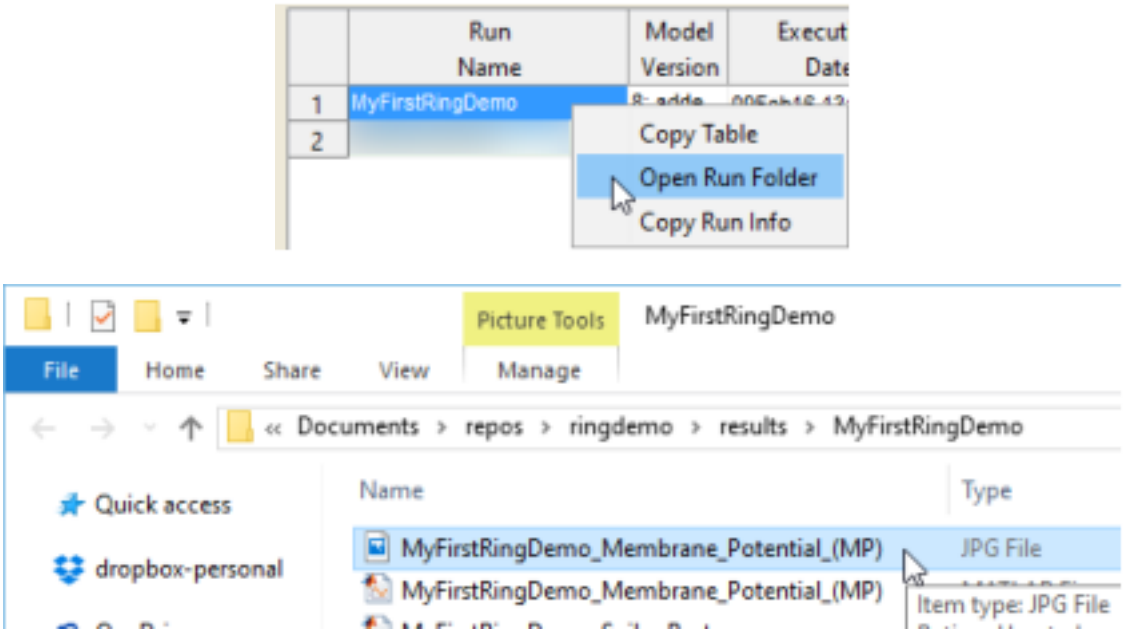




21. Alternatively, users can directly export the figures to an image format of their choice by selecting the "fig" menu option in the file extension pop-up menu and choosing a different file format (such as jpeg, bmp, pdf, etc.).



22. Exported images will be saved in the results folder for that particular simulation, which will be logged in the figure-generation log on the lower left side of the SimTracker. To view the file location of the saved image, users can right click in the figure-generation log or in the main simulation list window and choose to open the simulation results directory.



When done using SimTracker, the Docker container can be exited with the 'exit' command. Please consult the online Docker documentation at https://docs.docker.com/engine/userguide/containers/dockerimages/ for further details on how to use Docker images and software containers, including use of the 'docker commit' command to save data and settings within the package.

- *Tutorials*
- *Create a new model repository using SimTracker*
- *Design, execute and upload a simulation*
- *Organize simulations*
- *Analyzing a simulation*
- *Network Clamp*

Network Clamp a cell using the NetworkClamp tool

Next, use the Network Clamp tool to employ the Network Clamp technique *[BRB+16]* on the model. The Network Clamp is most useful for a large network, especially one with many inputs to each cell. The Network Clamp tool can be used for investigating how various inputs affect individual cells and for studying network dynamics without requiring a supercomputer. For this example, we will study how a particular cell's behavior changes as a function of the strength of the incoming synapse. This will gives us an idea of how to alter our connectivity strength in the full model without having to run a lot of full-network simulations to find the proper synapse strength. We will run this tutorial from the ca1 repository, so please ensure that SimTracker currently has the ca1 repository open before proceeding. The first few steps of the tutorial describe how to download the results and configuration [BezaireSciData] from the control, full scale CA1 network run used in *[BRB+16]*, so that they can be used as the basis for a network clamp run in the rest of the tutorial.

1. First, download the CA1 control simulation results from Open Science Framework: https://osf.io/v4ceh/
2. Then, from within SimTracker, choose File > Import and select the newly downloaded ca1_centerlfp_long_exc_065_01 file to import it. The simulation run should then appear in the SimTracker runs table.

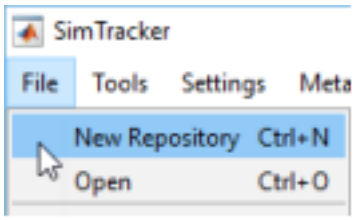
3. From the SimTracker Tools menu, choose "Network Clamp" and the Network Clamp tool will open.
4. Within the Network Clamp tool, at the top left area in part "1. Inputs", select the "Simulation Results" radio button.
5. Then, from the popup menu next to it, choose the network simulation to use as the baseline: "ca1_centerlfp_long_exc_065_01" and change the duration from 4000 to 100 (ms).
6. Next, find the popup menu for "Use inputs for" and choose "pyramidalcell (21310 - 332809)", and for gid number, enter 21310.
7. For part "2. Electrophysiology", choose the electrophysiology of "poolosyncell", which is the technical cell template used in the model CA1 network, derived from the pyramidal cell model published by [Poolos02].
8. Finally, for part "3. Connections", choose to use the connections and synapses from the corresponding network simulation by clicking the "From Run" radio button for the Connections line as well as for the Synapse kinetics line. Then verify that the "Run" menu below says "ca1_centerlfp_long_exc_065_01", as the simulation run from which to take the connections and synapse kinetics.
9. At the bottom left area, enter a description for this Network Clamp run, such as "Running baseline netclamp for pyramidal cell", click the "Table View" radio button for "Results" and then click the "Execute" button.
10. After the run completes, the intracellular membrane potential recording for the cell will appear on the graph on the right side, as well as histograms of each input type received by the cell throughout the simulation. In the table at the top right, a summary of the run will appear. Users can toggle whether the design or the results of the network clamp run are shown using the radio buttons at the bottom left of the NetClamp tool. Next, try altering the inputs to this cell and observing the effect. This tutorial will arbitrarily focus on investigating the effect of distal feedback inhibition, which is mediated by neurogliaform cell inputs on the pyramidal cell.
11. In part "3. Connections" on the left side of the NetworkClamp, switch the connections radio button to "Custom Table". Then in the table below, find the row for input cell "ngfcell" (for neurogliaform cell) and the column for "Wgt (uS)". Change the value from 1.4500e-04 to 1.4500e-05, decreasing by 90% the incoming weight of inhibition from neurogliaform cells to the network-clamped pyramidal cell.
12. In the description field at the bottom left of the tool, enter a new description of "90% reduction of neurogliaform cell input" and then click "Execute" button. The simulation will run again with all the same times of spike inputs to the pyramidal cell, and the incoming synaptic weights and kinetics will remain the same for all inputs except for neurogliaform cell inputs. The neurogliaform cell inputs will occur at the same time, with the same kinetics, but have a synaptic amplitude that is only 10% of the baseline condition.
13. After a few minutes, a new record will appear in the upper right table and a new trace will appear on top of the old one in the graph just below it. The input histograms will update to reflect the inputs from this new run.
14. Various run results can be hidden or displayed by clicking the checkbox next to the corresponding description in the results table. The color and dash pattern of each results line can also be customized by adjusting the settings in columns "Col" and "Line" of the results table. Currently, the input histograms always display the input spike times from the most recent network clamp simulation run.
15. Now, try running a Network Clamp simulation that uses patterned stimulation input designed by the user, rather than stimulation based on the activity of a large scale network simulation result.

- *Tutorials*
- *Model Design Characterization*

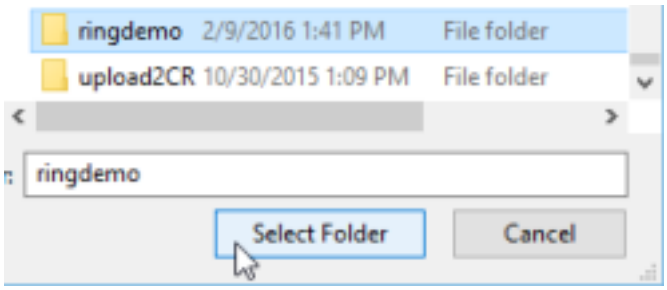
Create a new model repository using SimTracker

This, tutorial will show how to set up a new model, which will be an occasional task. To create or add a model repository to SimTracker, do the following:

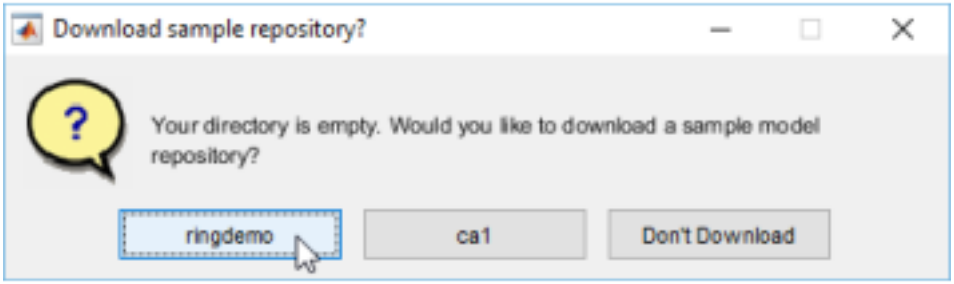
1. From the File menu, choose 'New Repository'.



2. In the folder picker dialog that appears, create a new folder or select an existing folder that will contain or already contains some model repository files. For the purposes of this tutorial, create a new, empty folder called 'ringdemo', select it, and then click the 'Open' button.



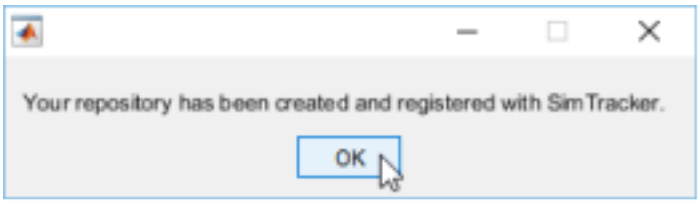
3. Because the repository directory is empty, SimTracker will offer a choice of downloading a sample repository from BitBucket, such as the ringdemo network model code at https://bitbucket.org/mbezaire/ringdemo. Click the 'ringdemo' button to download the code as the remaining tutorials in this section will use the ringdemo network. Note: if users choose not to download code because, for example, they are adding a model repository that already contains its own code, they will instead be given the option for SimTracker to perform a check of the files in the repository and add any missing ones to complete the NEURON template used with SimTracker.

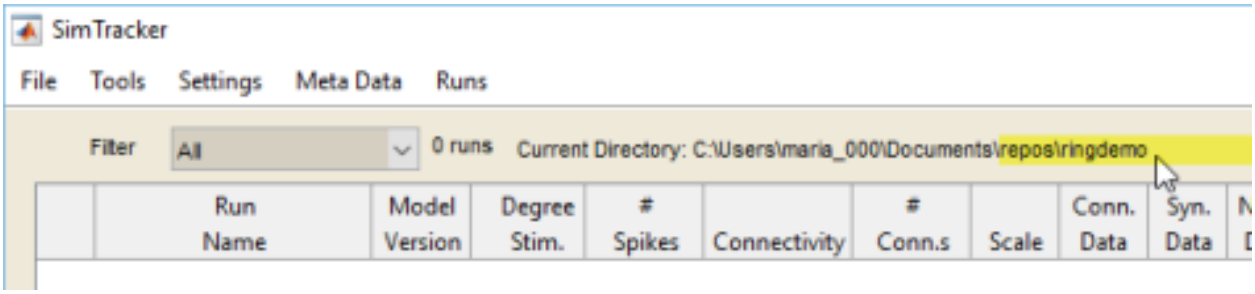


4. SimTracker will download the code and register the repository, then display a message when finished and will now list the path to the repository at the top of the SimTracker window.

   Note: If users choose not to download the sample code, at this point a text file would open containing a to-do list of any missing files and folders the repository requires. Make sure to address each item on the to-do list before continuing to work with the new model repository. The model repository is now ready. The next tutorial will walk through the processes of creating, executing, and analyzing a simulation, tasks that users will perform many times.

- *Tutorials*
- *Design, execute and upload a simulation*





Design, execute and upload a simulation

This tutorial covers designing and executing simulations, which will be a frequent task. Before designing specific simulations for a model, users will likely want to create some datasets to specify the cells and size of the model, the numbers and weights of its connections, and the kinetics of its synapses. However, those steps are not necessary for this example ringdemo network because those data sets have already been created and are included in the download; therefore the steps are detailed in the appendix. This tutorial will simply show how to view the datasets that have already been created, for the purpose of understanding all the components of the model before designing and executing a simulation (Figures [fig:tutNewRun]-[fig:tutNewRun3]).

1. Within the 'Tools' menu, choose the 'Cell Numbers' option (Figure [fig:tutNewRun:01]).
2. A dialog box will open, allowing users to view the Cell Numbers datasets associated with this model repository.
3. From the popup menu at top, choose number #100 to display the dataset that will be used to set the types and numbers of cells in the ringdemo network (Figure [fig:tutNewRun:02]). The table will refresh, showing two rows with information about the two cell types in our model (Figure [fig:tutNewRun:03]). The left-most column lists the name users have given the cell in their model. The next column lists the name of the cell template file to use for this cell, from the list of cell template code files in our model repository. The third column gives the number of that cell type to include in the model. The last column is only relevant for 3D models with layers, listing the layer in which that cell type is found (starting from layer 0). After viewing, close the window.
4. Next, from the 'Tools' menu, choose 'Connections' (Figure [fig:tutNewRun:04]). From the popup menu at top, choose number #100 to display the connectivity dataset to use (Figure [fig:tutNewRun:05]). A matrix will appear, including all the cell types in the network. Down the left side of the matrix (the row labels) are given the presynaptic cells, while the top row (the column labels) gives the postsynaptic cells. At the intersection of each presynaptic and postsynaptic cell, there are entries for the average number of connections from all cells of that presynaptic type onto one postsynaptic cell of that type (aka, the convergence), the number of synapses comprising one connection, and the weight of each synapse. After viewing the connection information, close the connection tool.
5. Finally, from the 'Tools' menu choose 'Synapses' (Figure [fig:tutNewRun:06]) and a table will appear. From the popup menu at the top left, choose dataset #100 to view the synapse data for the model (Figure [fig:tutNewRun:07]). Only one postsynaptic cell at a time will be displayed; choose from the pop-up menu at the top middle of the table (Figure [fig:tutNewRun:08]) to specify which post-synaptic cell to view. Then the table will display the possible presynaptic cell types for that postsynaptic cell, along with entries for the rise and decay time constants of the synapse and the reversal potential of the synapse (Figure [fig:tutNewRun:09]).

When finished, close the window. The next steps show how to design a simulation that uses these datasets.

6. To design a new simulation, from the Runs menu choose 'Design Run' (Figure [fig:tutNewRun:10]). The property fields in the SimTracker will become editable (Figure [fig:tutNewRun:11]).
7. Enter a unique name for the simulation (allowed characters include letters, numbers, and the underscore character), such as MyFirstRingDemo (Figure [fig:tutNewRun:11]).
8. Optionally, enter comments about the simulation. It is often helpful to users if they note the specific question or idea that triggered them to run this particular simulation.
9. Specify the code version to use for this simulation. Usually, this will be the most recent version of the code. The list of available code versions will correspond to each time users 'committed' a version of the code in their model code repository and will also list the comments they entered when committing that code version. Select the most recent version, #9 (Figure [fig:tutNewRun:12]).
10. Specify the machine to use for running the simulation. For users to run this simulation on their own computers, they must find the unique name of their computer from the list and select it (Figure [fig:tutNewRun:13]). Then, in the field to the right of the machine name, specify the number of processors to be used to execute the run. On a personal computer, users can either set this to 1 or, to run the network in parallel, set it to the total number of cores (aka processors) on their machines.
11. Select from the various menus for simulation control to specify the stimulation, connectivity algorithm, cell numbers, connection numbers, and synapse kinetics for the model. For this ringdemo network, choose the following settings (Figure [fig:tutNewRun:14]):
    - Stimulation: pulse
    - Connectivity: ring
    - Cell set: 100
    - Connection set: 100
    - Synapse set: 100
12. In the table on the right side of the SimTracker, specify additional properties for simulation control, including the scale (size) of the model network, the length of time the simulation should last, the temporal resolution, whether to print specific results files, and many other options (the options that appear in this table can be customized by choosing 'Parameters' from the 'Settings' menu). Because this ringdemo network is so small, it can run it at full scale. Set the following parameters and leave the remainder set to their defaults (Figure [fig:tutNewRun:15]):
    - Scale: 1
    - Sim. Duration: 101
    - Temporal Resolution: 0.025
13. Upon finishing the specification of simulation properties, click the 'Save Run' button that appeared when upon starting to design the simulation (Figure [fig:tutNewRun:16]). The simulation will now appear in the list of simulations at the top of the SimTracker. The property fields will become uneditable (to change the run prior to execution, choose 'Edit Run' from the 'Runs' menu). The process for executing a simulation differs somewhat depending on whether users are executing it on their personal computers, using the NSG portal, or accessing a supercomputer independently of the NSG portal. The following list details the process for using a personal computer. For instructions on using the NSG portal or a supercomputer directly, see the Appendix.
14. To execute the simulation, select its row from the table at the top of the SimTracker, and then from the 'Runs' menu choose 'Execute Run' (Figure [fig:tutNewRun:17]).
15. The computer's CPU may become very active and users who are using MATLAB scripts to run the SimTracker, will see the MATLAB status bar display 'Busy' until the simulation completes (Figure [fig:tutNewRun:18]). Additionally, a NEURON window may pop up (Figure [fig:tutNewRun:19]).

16. When the simulation finishes, a dialog box will appear that states 'The run has been executed' (Figure [fig:tutNewRun:20]). Once this message appears, continue to the next step.
17. Select the simulation in the SimTracker (make sure to view the 'Not Ran' view) and, from the 'Runs' menu, choose 'Upload Run' to display the results of the run in the SimTracker (Figure [fig:tutNewRun:21]).
18. Once the simulation is successfully uploaded to the SimTracker, a dialog box will appear stating that 'Results have been uploaded.' (Figure [fig:tutNewRun:22]) The SimTracker table will switch to the 'Ran' view (Figure [fig:tutNewRun:23]). In both the list of runs and the form view below, SimTracker will show the executed simulation's run time, number of spikes, and execution date (Figure [fig:tutNewRun:24]). After the simulation has been executed and uploaded, it is time to analyze the results (Figure [fig:tutAnalyzeStep]).

Tutorials

Analyzing a simulation

Any simulation that has been executed and uploaded to the SimTracker is available for analysis. This tutorial walks through how to use some standard analyses available within SimTracker. However, users may also access the results from each simulation directly, by opening the directory with the same name as the simulation, found within the 'results' directory of the model repository. The results directory can also be opened by right-clicking (Ctrl-click on Mac) the entry for the simulation in the main SimTracker runs list. The spikeraster.dat file within the results folder contains a list of all spike times and which cell caused each spike during the simulation. The ringdemo simulation produces a spikeraster file that contains the times found in Table [tab:outputs:raster], which compare well to the results from *[HC08]*, found in their section *3.3.5 Reporting simulation results*.

1. Select the completed simulation of interest from the table in the SimTracker.
2. Look at the table on the bottom right side of the SimTracker for a list of available outputs for that particular simulation (the list will vary depending on which output files users choose to print for their simulations). For now, click to select the checkboxes next to the 'Spike Raster' and 'Membrane Potential' (Figure [fig:tutAnalyzeStep:01]) outputs. Note: For more information on any particular output, click on that row to highlight that output type and then hover the mouse anywhere over the output table to see a tooltip with guidance about the selected output type.
3. After selecting the outputs of interest, click the 'Generate' button to produce them as MATLAB figures. If prompted to enter a GID (global ID number for a cell type), enter '0' to display the membrane potential for cell #0. Users can zoom in, pan, and otherwise edit or save the figures once the MATLAB figures are generated (Figure [fig:tutAnalyzeStep:02a] and [fig:tutAnalyzeStep:02]).
4. Alternatively, users can directly export the figures to an image format of their choice by selecting the 'fig' menu option in the file extension pop-up menu and choosing a different file format (such as jpeg, bmp, pdf, etc.) (Figure [fig:tutAnalyzeStep:03]).
5. Exported images will be saved in the results folder for that particular simulation, which will be logged in the figure-generation log on the lower left side of the SimTracker (Figure [fig:tutAnalyzeStep:04]). To view the file location of the saved image, users can right click in the figure-generation log or in the main simulation list window and choose to open the simulation results directory (Figures [fig:tutAnalyzeStep:04] and [fig:tutAnalyzeStep:05]).

Tutorials

Organize simulations

After completing many runs, users may wish to organize them in various ways. One possibility is to add labels to the runs as well as adding searchable comments. Users can set the view filter to only show runs with (or without) particular

labels, as well as runs with similar names, or runs meeting a custom search requirement of their choice. Users can and should back up their runs (both configurations and results sets) using SimTracker. Archive runs that are no longer needed (deleting executed runs from the SimTracker is not recommended). Users can also, export and import runs to share with others. This tutorial will illustrate labeling, filtering, and archiving a completed simulation, then importing it back into SimTracker. To prepare for this tutorial, please try running an additional simulation following the steps in Tutorial *Design, execute and upload a simulation*, giving it a unique name. After uploading the results, proceed with this tutorial (Figures [fig:tutOrganize] - [fig:tutOrganize2]).

1. First, add a grouping category to SimTracker by going to the Settings menu and choosing Groups (Figure [fig:tutOrganize:01]).
2. When the group list appears, click the 'Add line' button and in the left cell of the new line that appears, enter a group name 'testgroup' (Figure [fig:tutOrganize:02]). Then click the 'Save' button and close the window. The entry 'testgroup' will now appear in the list of groups on the SimTracker.
3. Next, ensure that the view is set to 'Successfully ran' so that the completed runs show. Then, click the MyFirstRingDemo entry in the table to select it.
4. Next, find the 'testgroup' entry in the group list and click it. The label 'testgroup' should appear in the 'labels' section of the simulation entry (Figure [fig:tutOrganize:03], highlighted section).
5. Now, from the view filter menu, choose 'Group A,B,C filter' (Figure [fig:tutOrganize:04]).
6. In the dialog box that appears, find the line for 'testgroup' and from the menu next to it, pick the equals sign entry (=), then click OK (Figure [fig:tutOrganize:05]).
7. Now, the only run showing in the SimTracker should be MyFirstRingDemo (Figure [fig:tutOrganize:06]).
8. Again select the MyFirstRingDemo run and now from the File menu, choose 'archive' (Figure [fig:tutOrganize:07]). Users can archive many runs simultaneously by selecting all desired runs before choosing the 'archive' command, but this tutorial will only archive one.
9. An input dialog will appear, allowing the user to enter a name for the archive. Enter a name such as 'MyTestArchive', and click 'Save' (Figure [fig:tutOrganize:08]). SimTracker will create an archive in the backup folder under the name specified by the user. The backup folder is automatically created in the same directory where the other repositories are stored.
10. After MATLAB has created the archive, it will ask if the user is ready to delete the archived files from SimTracker. Click 'Yes' and the archival process is complete (Figure [fig:tutOrganize:09]). Note: users who want to create a backup file of a run without removing it from SimTracker, can do so by choosing File>Backup rather than the Archive command.
11. Now, the run should have disappeared from the SimTracker. Change the view setting to 'All' and verify that the run no longer appears (Figure [fig:tutOrganize:10]).
12. Next, import the run back into SimTracker by going to the File menu and choosing 'Open' (Figure [fig:tutOrganize:11]).
13. In the dialog box that appears, click the second option and then select the saved archive from the list (Figure [fig:tutOrganize:12]).
14. Click OK. The run should now appear in the SimTracker table again.

Tutorials

Model Design Characterization

Here, we demonstrate how to add cell types and alter the network constituents or connectivity. The following tutorials also highlight additional tools and features of SimTracker. These tutorials work best with a larger network model rather than the ringdemo network. Feel free to start a new repository using the ca1 model code by following Tutorial

3 and selecting 'ca1' rather than 'ringdemo' when adding the new repository to SimTracker. Then proceed with the following tutorials.

Add model cell types and specify model cell numbers and connections

This tutorial will illustrate the necessary steps to add another interneuron type to the ca1 model network, such as perforant path-associated cells (PPA cells). First, we would need to create a new model cell template for this cell type, and then we would need to create some datasets containing information about how to incorporate that cell type into the model network. We will walk through both of these goals in the following tutorial.

The tutorial will begin by copying an existing, working model cell template. It is a good idea to create the model cell template by adapting it from an existing file. Users can make incremental changes to a copy of the working template to ensure that they maintain the standardization and functionality of the template and also to ensure the ability to check that the template still works after each change.

1. First make a copy of the Schaffer Collateral-Associated (S.C.-A) cell template, as the S.C.-A. cell type is quite similar to the PPA cell type. Within the 'cells' subdirectory of the model repository, open the 'class_scacell.hoc' file and choose 'Save As' and then type 'class_ppacell.hoc'. Now the new cell template is ready to be edited and tuned to become a model PPA cell.
2. Next, rename the template within the file. The first and last code lines of the file refer to the template name, which should be 'ppacell' instead of 'scacell'. To work most efficiently, use find and replace to change all references to 'scacell' to 'ppacell' in the document.
3. Next, alter the morphology and electrophysiological properties defined in the file as we see fit. Later on, there will be an opportunity to test and further tune the cell using the CellClamp tool to ensure that it behaves as a physiological PPA cell.
4. Finally, save the file and then return to the SimTracker tool to add the cell to the network model. Now that the model PPA cell template has been created and added to the 'cells' directory, it is time to create the new datasets that will include this cell type in the network.
5. From the SimTracker 'Tools' menu, choose the 'Cell Numbers' option.
6. A dialog box will open, allowing the user to select an existing cell numbers set or to create a new set that specifies which cell types to include in the model and how many of each should be included. To create a new set based on the existing set #101, choose 101 from the top menu to load that cell numbers dataset.
7. Before making any changes, click the 'Save New...' button and add a comment to the box that appears, stating '#101 with ppacell added.' This will save a new dataset that can be edited.
8. Now, from the celltype menu, select 'ppacell' on both sides and then click 'Add'.
9. A new line will be added with an entry for 'ppacell', Enter the remaining information on that line, including the number of ppacells in the full size network and the layer in which they are found (stratum lacunosum-moleculare is layer 3, since stratum oriens is layer 0).
10. Next, click the 'Save' button. A dialog box will appear stating that a new axonal distribution file has been created for the ppacell. To specify the axonal extent and distribution of boutons of the ppacell, modify its axondist file from within the axondists directory of the cells directory in the repository. Further information about this file and its parameters is included in the User Manual.
11. Now, close the Cell Numbers dialog box.
12. Next, from the SimTracker 'Tools' menu, choose 'Connections'. A matrix will appear. From the top left menu, select '430' to load the basic dataset from which we will derive our new one. Then, from the top right menu, select the number corresponding to the new cell numbers dataset just saved. The matrix will now expand to include ppacells as well. Down the left side of the matrix (the row labels) are given the presynaptic cells, while the top row (the column labels) gives the postsynaptic cells. At the intersection of each presynaptic

and postsynaptic cell, add the average number of connections from all cells of that presynaptic type onto one postsynaptic cell of that type (aka, the convergence).

13. After filling out all the connection information for connections to or from ppacells (convergence numbers, synapses per connection, and synaptic weights), save the connection tool and close it.
14. Finally, from the 'Tools' menu choose 'Synapses.' A table will appear. From the top left menu, select the number '120' to load a baseline synapse dataset.
15. Then, from the top right menu, select the number corresponding to the recently created cell numbers dataset that contains the PPA cells.
16. Now, notice the pop-up menu at the top middle to specify a post-synaptic cell. Select the ppacell from this menu and then add potential presynaptic cells to the list to specify their specific kinetics. Save the dataset by clicking the 'Save' button.
17. Now, switch the top middle menu to each cell that can receive input from the PPA cell in turn. Add the synaptic entry for the ppacell connection. Then, before switching to a different post-synaptic cell, again save the current table. After completing the definitions for all postsynaptic celltypes, then save the table again and push the button to close. One of each type of the parameter dataset has been created to specifying the network and its connections: cell numbers, connections, and synapse kinetics. These datasets resemble those used in the control CA1 network simulation except that they also include PPA cells. Now, users can launch the CellClamp tool, set the dataset menus in it to the ones just created, and test both the intrinsic cell properties of the PPA cell and the connections to and from PPA cells (using the paired recording feature). The next tutorial covers the use of the CellClamp tool.

Tutorials Model Design Characterization

Characterize model network components with CellClamp

Here, we walk through how to characterize basic components of the model using the Cell Clamp tool. This tool can characterize ion channel dynamics, single cell intrinsic properties, and synaptic connections in common experimental terms. This tutorial shows each of these characterizations using the CA1 network.

1. From the SimTracker Tools menu, choose 'Cell Clamp'. The Cell Clamp tool will open.
2. Within Cell Clamp, first, scroll through the table of ion channels to find the entry for Nav, or the voltage-gated sodium channel. Fill out the entries for this row in the table, entering a Gmax (maximum conductance density) value of 0.001 (micro Siemens per square centimeter) and an ENa (reversal potential) value of +55 (mV). The variable used to store the reversal potential for this channel, ena, has already been populated in the table.
3. Next, click the checkboxes to the right of the ion channel table, for 'IV Curve' and 'Act./Inact.'. Then, click the 'Conductance' radio button below them so that the results will display in terms of ion channel conductance rather than current.
4. Then, click the 'Get Results' button below. A dialog box will appear, asking the user to add comments for this particular characterization run. Enter 'Characterize Nav channel in conductance terms' and then click 'OK'. It will take a little while for the characterization simulations to complete before the result figures appear. Because the Nav row in the table was filled out, the CellClamp will run an activation/inactivation curve and a current/voltage relation for the Nav channel and display the resulting graphs in terms of channel conductance.
5. Next, prepare CellClamp to run a simulation at the single cell level rather than the macroscopic ion channel level. Uncheck the 'IV Curve' and 'Act./Inact.' checkboxes so that these protocols will not run again, and instead check the 'Current Clamp' checkbox to the right of the list of cells (in the middle of the tool).
6. Check the popup menu above the list of cells and set it to the 'cellnumbers_100' dataset. All the cell types included in that dataset will populate the list of cells below.
7. From the list of cells, click to select the 'pyramidalcell' cell type.

8. Set the current clamp protocol to be applied to the selected cells. The top row of numbers gives the time before the pulse is applied, then the length of the pulse, followed by the time after the pulse has finished that the cell behavior continues to be recorded. The row below gives all the current injection (or voltage clamp) levels that will be used. Note that MATLAB's vector syntax can be used to specify which currents to apply. For example, rather than listing -0.300, -0.250, -0.200, -0.150 ... all the way up to +0.500, users can specify that every current injection level between -.300 nA and +.500 nA should be tested, increasing in steps of 0.050 nA, with the following syntax: [-0.300:0.050:0.500]. Users can also add additional values to use before or after that syntax within the brackets: [-0.500 -0.300:0.050:0.500 0.510 0.540 0.580 0.600]. For this tutorial, leave the row of times alone but change the current list to say: [-0.3:0.05:0.5].
9. Now click 'Get Results' and enter into the comments dialog that appears 'Single cell characterization'. After a few moments, several figures will appear, characterizing the cell's behavior. Note: For even more in-depth characterization, the results of this CellClamp protocol can be loaded into the CellData tool to compare the model cell's properties directly with those of experimental cells.
10. Next, prepare the CellClamp tool to run a paired recording rather than a single cell recording by unchecking the 'Current Clamp' checkbox in the single cell section.
11. In the lower left area of CellClamp, set the popup menu to 'syndata_120' to populate the synapse list below with all synapses listed in that dataset.
12. From the synapse list, select the pvbasketcell -> pyramidalcell entry to characterize connections from PV+ basket cells to pyramidal cells.
13. Click the current clamp checkbox to apply a current clamp to the postsynaptic cell in the pair (the pyramidal cell), and to the right of the checkbox, enter '0' (nA) into the current injection box so that the cell's response to the incoming synaptic activity will be recorded relative to its baseline resting potential.
14. Above the current injection amount, the row of boxes indicated relevant times for the paired recording protocol: 15 ms after the recording starts, the presynaptic cell will spike, triggering the synaptic activity in the postsynaptic cell after a short delay due to axonal conduction. The next entry, 100 ms, shows the length of time to record the postsynaptic cell's response after the presynaptic spike. For most current clamp of synapse types, 100 ms is adequate; only for slow synapses such as those with GABAB would the recording window need to be longer in a current clamp paired recording.
15. Finally, click the 'Get Results' button and for the comment dialog, enter 'Synaptic recording' and then click 'OK'. The CellClamp will now perform 10 paired recordings of the synaptic connection using the protocol specified, and then will average the results. It will then display a graph of the individual recordings and the averaged recording, along with the time constants and amplitude of the synaptic connection computed from the averaged trace.

Tutorials Model Design Characterization

Tutorials

Automatically analyze experimental data using CellData

Both experimentalists and modelers may wish to automate analysis of AxoClamp recordings from biological cell current sweeps. The CellData tool reads in AxoClamp files in binary (ABF) or text (ATF) format. If the file contains recordings from a current injection sweep, such as the protocols commonly used to characterize firing curves and intrinsic electrophysiological properties of single cells, the CellData tool will process the file to determine spike threshold points, action potential peaks, sag and steady state peaks. Then, it will allow the user to review and edit the points for accuracy (or choose a different strategy for determining the action potential threshold), before computing over 30 common electrophysiological properties of the cell.

CellData can be used independently of SimTracker. Both a compiled version and the MATLAB source code of CellData is available for download at http://mariannebezaire.com/simtracker/.

1. From the Tools menu of SimTracker, choose the 'Experimental Comparison (CellData)' option to open the CellData tool (Figure [fig:tutCellData:01])
2. In the CellData tool, load data from an AxoClamp file by clicking the button 'Load AxoClamp file' (Figure [fig:tutCellData:02]) and selecting the desired AxoClamp file (in ABF or ATF format) from the file picker dialog box that appears (Figure [fig:tutCellData:03]).
3. As the recording data is imported, a dialog box will appear asking for additional metadata (Figure [fig:tutCellData:04]). Enter the cell type and location for the cell, using whichever nomenclature is appropriate to the user's field of study. Also enter the name of the person who recorded the cell, for future reference. Then click OK.
4. Once the recording data is read in from the cell, CellData will display some of the traces from the file in a graph at the lower right corner of the tool. The traces displayed will include recorded membrane potential from the most hyperpolarized trace, the most depolarized trace, and the trace from the current injection at the zero level, if performed (Figure [fig:tutCellData:05]). Overlaid on these traces will be the corresponding current injection plots.
5. Users should look these over and ensure that the current injection timing looks correct and corresponds to the displayed potential traces, and then click the 'Verify Current Sweep' button (Figure [fig:tutCellData:06]). A dialog box will then appear, confirming 'Current Sweep Verified' (Figure [fig:tutCellData:07]).
6. The lower right corner of the CellData tool will now display the some membrane potential traces from the current sweep, including the potential recordings corresponding to all hyperpolarized current injections and the most depolarized current injection (Figure [fig:tutCellData:08]). However, the traces must be further analyzed to calculate the electrophysiological properties of the cell. Click the 'Calculate Threshold' button to start the analysis process (Figure [fig:tutCellData:09]).
7. The threshold calculation process can be time intensive, so the CellData tool will display a status bar to alert the user to the progress of the calculation (Figure [fig:tutCellData:10]).
8. A second window will appear, displaying one at a time the membrane potential recorded at each current injection. In this interface, users can choose from one of three possible threshold calculation strategies, visually observing how the point chosen depends on the threshold calculation used. Select '1' from the dropdown menu picker to set the calculation strategy for this sample cell (Figure [fig:tutCellData:11]).
9. A line will be drawn through each threshold point calculated in the displayed membrane potential trace (Figure [fig:tutCellData:12], see red arrow). Users can browse through the membrane potential recordings from each current injection level by selecting the left or right arrow buttons (Figure [fig:tutCellData:12], see orange outline), while keeping track of which level they are currently viewing by looking at the description in the upper left corner of the window (Figure [fig:tutCellData:12], see red circle). Whichever threshold calculation strategy is chosen will be used to find the threshold points in the recordings from all current injection levels.
10. When satisfied with the threshold calculation strategy chosen, push the 'Set Threshold' button (Figure [fig:tutCellData:13]) and wait for CellData to calculate all spike threshold points from all current injection levels in the file. When the calculation has finished, the second window will close.
11. Back in the CellData window, click the button to 'Verify & Analyze' (Figure [fig:tutCellData:14]). Another status bar will appear in CellData to signal the progress of the review step, which can take awhile for large files.
12. After the tool has identified all points of interest in the membrane potential recordings from each current step, the second window will again appear, allowing users to review and correct any point (Figure [fig:tutCellData:15]). The points found by the program include (for hyperpolarized traces) the trough point of the sag potential, (for hyperpolarized and subthreshold depolarized traces) the steady state potential, (for subthreshold traces only) the transient peak depolarization potential, (for suprathreshold traces) the action potential threshold, action potential peak, and afterhyperpolarization potential.
13. Figure [fig:tutCellData:16] shows various buttons available for modifying the points automatically chosen by the CellData tool. To move or delete an incorrect point, either click the corresponding action button (Figure

[fig:tutCellData:16]) and then select the point by clicking it with the mouse, or scroll through the table of points to the right side and select the checkbox next to the point, then click the appropriate action button.

14. As when selecting the threshold, users may scroll through the membrane potential recordings associated with each current injection level, and may correct points at each level. When finished, click the 'Done with cell' button to close the second window and return to the main CellData interface.

15. Now CellData will display additional buttons for displaying and exporting the analyses of the cell, as well as a table of calculated electrophysiological properties (Figure [fig:tutCellData:17]) and, in the lower right corner, a graph that can be set to display several different analyses by making a selection from the drop-down menu above the graph (Figure [fig:tutCellData:18]).

16. Click the 'Make Figures' button (Figure [fig:tutCellData:19]) to display an interface where users can pick which graph types to display and the format in which to display them (Figure [fig:tutCellData:20]). Figures such as those pictured in Figure [fig:tutCellDataOut] will be displayed.

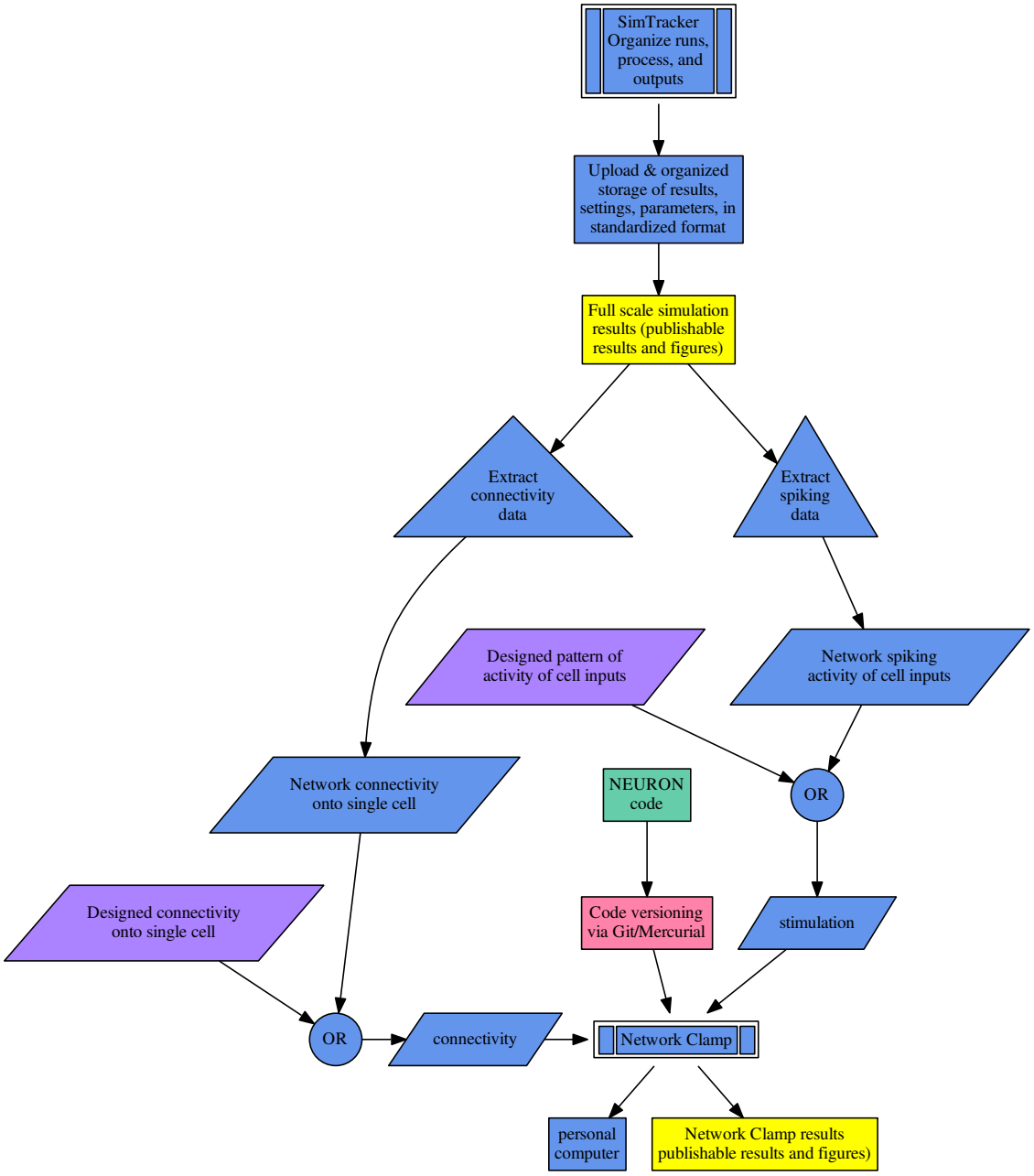
17. Alternatively, click the 'Export Properties' button to export the table of electrophysiological properties as a tab-delimited text file, comma-separated values text file, or Microsoft Excel file.

Tutorials

CHAPTER 9

Workflow

The workflow


SimTracker
Organize runs, process, and outputs
Upload & organized storage of results, settings, parameters, in standardized format
Full scale simulation results (publishable results and figures)
Extract connectivity data
Extract spiking data
Designed pattern of activity of cell inputs
Network spiking activity of cell inputs
Network connectivity onto single cell
NEURON code
OR
Designed connectivity onto single cell
Code versioning via Git/Mercurial
stimulation
OR
connectivity
Network Clamp
personal computer
Network Clamp results publishable results and figures)

CHAPTER 10

Support

Looking for answers to questions:

- SimTracker support forum: https://www.neuron.yale.edu/phpBB/viewforum.php?f=42
- Documentation

Bugs, errors, unable to find answers:

- Email
- Post in Forum: https://www.neuron.yale.edu/phpBB/viewforum.php?f=42
- Add issue to repository: https://bitbucket.org/mbezaire/simtrackercode

Tutorials

CHAPTER 11

Indices and tables

- genindex
- search

CHAPTER 12

Bibliography

Bibliography

[BRB+16] Marianne J. Bezaire, Ivan Raikov, Kelly Burk, Dhrumil Vyas, and Ivan Soltesz. Interneuronal mechanisms of hippocampal theta oscillations in full-scale models of the CA1 circuit. *eLife*, 2016. doi:10.7554/eLife.18566.

[HC08] M.L. Hines and N.T Carnevale. Translating network models to parallel hardware in NEURON. *J. Neurosci. Methods*, 169:425–455, 2008. doi:10.1016/j.jneumeth.2007.09.010.