最終更新日： 2016/05/30

触角葉神経回路シミュレーションプログラム引き継ぎ資料

2016 年 3 月 18-19 日

朴希原

1. はじめに

本資料は，私が研究にて開発した触角葉神経回路シミュレーションプログラム(Antennal Lobe Simulator，以下 ALS)の基本的な使用方法をまとめたものです.

2. プログラムの動作環境構築

本使用の内容を実行するための最低限の環境を説明しておきます.

基本的に ALS はローカルマシンでも，スパコン上でも同じコードで動くように作っております．もちろん，設定によってネットワークの規模が大きい場合は計算時間やメモリの問題があるためローカルマシンで動かす事は推奨しません．ローカルマシンで動作させる時は基本的に小規模ネットワークで動かすときや，Debug 用に限ります．結果のデータを解析する際には基本的にスパコンのデータをローカルに落として Python の解析プログラムを動かす事になります.

必要項目

スーパーコンピュータ(京，FX100)：NEURON_KPLUS(git@github.com:sc4brain/neuron_kplus.git)（ALS を動かす場合 Version7.3，後藤さんのパラメータ推定プログラムを使ったパラメータ推定コードを動かす場合 Version7.2 が必要)，GitHub

ローカルマシン：NEURON_KPLUS，GitHub，Python，Numpy，matplotlib，stfio(pclamp データ読み込み用)

上記の項目の環境構築ができたら私のプログラム(ALS，git@github.com:heewonpark/als_v2.git)を clone します.

git@github.com:heewonpark/DataAnalysis.git

https://github.com/heewonpark/DataAnalysis.git

3. プログラムの構成

- ✧ fxjob：FX100 用の jobscript
- ✧ input
  - ➢ estimation_data：パラメータ推定用のデータ(データの仕様に関しては後藤さんの引き継ぎ資料を参照)
  - ➢ network_info：マルチコンパートメントシミュレーション実行時に必要な神経回路の情報
  - ➢ spiketiming：嗅覚受容細胞の発火タイミングを事前生成データおよびデータ生成ファイル
  - ➢ swc：マルチコンパートメントシミュレーション用の形態データフォルダ
  - ➢ synapse_info：マルチコンパートメントシミュレーション用のシナプス情報
  - ➢ synapse_list：各ネットワーク内結合に対するシナプスリスト
- ✧ job：K 用の jobscript
- ✧ mod：nmodl ファイル
- ✧ single-src：シングルコンパートメントシミュレーション用のソースコード
- ✧ src：マルチコンパートメントシミュレーション用のソースコード

プログラムの実行法によってはフォルダ内に result フォルダが生成される事もあります.

グラフを生成するための python コードの中には

シングルコンパートメントプログラム
マルチコンパートメントプログラム
入力データの仕様
シミュレーションの実行法
結果解析法

データ解析用プログラム

ファイル構造

4. Single-compartment Antennal Lobe Simulation 仕様書

入力ファイル

[K] : 京コンピュータ, [FX100] : 名古屋の FX100, [L] : ローカルマシン

Single-ALS 用の並列実行用 Job script 解説

赤色は[K]で実行する時必要な部分

青色は[FX100]で実行する時必要な部分

```
#!/bin/bash -x

### ノード数と時間はシミュレーション時間とシミュレーションで Record する変数の数によって変わる. Record する変数の数がメモリをかなり大きく消費するため計算が必要（適当に大きくすれば良い）
### [K]
#PJM --rsc-list "node=12"
#PJM --rsc-list "rscgrp=small"
#PJM --mpi "proc=96"
### [FX100]
#PJM --rsc-list "node=3"
#PJM --rsc-list "rscgrp=fx-small"
#PJM --mpi "proc=96"

#PJM --rsc-list "elapse=1:00:00"
#PJM -s

# staging
#PJM --stg-transfiles all
#PJM --mpi "use-rankdir"

### [K]
```

```
#--------------------------------------------------------------------------------
# CHANGE TO YOUR OWN DIR
#PJM --stgin-basedir /home/hp120263/k01793/code/al_V2/  ###  自分の環境に合わせて変更

#--------------------------------------------------------------------------------
# STAGE IN ANTENNAL LOBE SIMULATION PROGRAM
#PJM --stgin "rank=* ./input/* %r:../input/"
#--#PJM --stgin "rank=* ./input/spiketiming/40stim/* %r:../input/spiketiming/40stim/"
#PJM                                                                    --stgin
"rank=* ./input/spiketiming/1000dose_30stims_filtering/* %r:../input/spiketiming/1000dose_30stims_filter
ing/"
#PJM                                                                    --stgin
"rank=* ./input/spiketiming/1000dose_30stims_filtering/* %r:../input/spiketiming/1000dose_30stims_filter
ing_adaptation/"
#PJM                                                                    --stgin
"rank=* ./input/spiketiming/100dose_30stims_filtering/* %r:../input/spiketiming/100dose_30stims_filterin
g/"
#PJM                                                                    --stgin
"rank=* ./input/spiketiming/10dose_30stims_filtering/* %r:../input/spiketiming/10dose_30stims_filtering/"
#PJM --stgin "rank=* ./src/* %r:./"
#PJM --stgin "rank=* ./single-src/* %r:./"

#--------------------------------------------------------------------------------
# STAGE IN NEURON_KPLUS SIMULATOR
#--#PJM --stgin "rank=* ../../github/neuron_kplus/stgin/* %r:./"
#--#PJM --stgin "rank=* ../../github/neuron_kplus/specials/sparc64/special %r:./"
#--#PJM --stgin "rank=* ../../github/neuron_kplus_tune/stgin/* %r:./"
#--#PJM --stgin "rank=* ../../github/neuron_kplus_tune/specials/sparc64/special %r:./"

#PJM --stgin "rank=* ../../github/neuron_kplus73/stgin/* %r:./"
#PJM --stgin "rank=* ../../github/neuron_kplus73/specials/sparc64/special %r:./"

#--------------------------------------------------------------------------------
# STAGE OUT TO DATA DIRECTORY  ### 自分の環境に合わせて変更
#PJM --stgout "rank=* %r:./*.txt /data/hp120263/park/al_V2/%j/record/"
#PJM --stgout "rank=* %r:./*.dat /data/hp120263/park/al_V2/%j/spike/"
#PJM --stgout "rank=* %r:./pd/* /data/hp120263/park/al_V2/%j/pd/"
```

```
# SET UP ENVIRONMENT OF LANGUAGE
. /work/system/Env_base

#--#export OMP_NUM_THREADS=8

### [FX100]
### fx100 は Stage-out が無いため，シミュレーション中の記録ファイルを保存するディレクトリを生成する必要がある
RESULT_DIR="../fx-result/"
RECORD_DIR="${RESULT_DIR}${PJM_JOBID}/record" ###スパイクタイミング以外のものを保存するフォルダ
SPIKE_DIR="${RESULT_DIR}${PJM_JOBID}/spike"  ###スパイクタイミングを記録したものを保存するフォルダ
OUT="${RESULT_DIR}${PJM_JOBID}/out"  ###print で表示するものはこれに保存される
mkdir -p ${RECORD_DIR}  ###フォルダ生成
mkdir -p ${SPIKE_DIR}  ###フォルダ生成

### [K]
NRNIV="./special -mpi"
### [FX100] NEURON の special ファイル，  ### 各自自分の環境に合わせて設定
NRNIV="/home/usr7/z48927t/github/neuron_kplus/specials/sparc64/special -mpi"
HOC_NAME="./main.hoc"

NRNOPT=¥
" -c JOBID=${PJM_JOBID}"¥  ###NEURON に JOBID を教えるために使用
" -c STOPTIME=12000"¥  ###シミュレーション時間
### [K]
" -c IS_SUPERCOMPUTER=1"¥  ### K では 1，FX100 では 2，  ローカルマシンでは 0
### [FX100]
" -c IS_SUPERCOMPUTER=2"¥  ### K では 1，FX100 では 2，  ローカルマシンでは 0
" -c START_TIME=0"¥  ###  刺激開始時間
" -c GABAB_ON=1"¥  ### GABA_B シナプスの作動可否(1 の時のみ作動)
" -c GABAA_ON=1"¥ ### GABA_A シナプスの作動可否(1 の時のみ作動)
" -c PTOL_ON=1"¥ ### PN→LN のシナプス作動可否(1 の時のみ作動)
" -c NSYNAPSE=100"¥  ### RN→LN,PN の数(1 個の LN,PN は 100 個の RN からシナプス入力を受ける)
" -c NPN=50"¥  ### PN の数
" -c NLN=350"¥  ### LN の数
" -c NRN=2000"¥  ### RN の数
" -c WEIGHT_RNtoPN=0.04"¥  ### RN→PN の結合強度
" -c WEIGHT_RNtoLN=0.018"¥ ### RN→LN の結合強度
```

```
" -c GABAA_LTOP=0.50"¥  ### LN→PN の GABA_A の最大コンダクタンス
" -c GABAA_LTOL=0.50"¥ ### LN→LN の GABA_A の最大コンダクタンス
" -c GABAB_LTOP=12.5"¥  ### LN→PN の GABA_B の最大コンダクタンス
" -c GABAB_LTOL=0.00"¥ ### LN→LN の GABA_B の最大コンダクタンス
" -c DOSE=1000"¥ ### 刺激濃度
" -c NSTIM=30"¥  ###  刺激回数(1 より大きい場合，大きい意味は無い... )
" -c PROB_LTOP=0.5"¥ ### LN→PN の結合確率
" -c PROB_LTOL=1.0"¥ ### LN→LN の結合確率
" -c PROB_PTOL=0.5"¥  ### PN→LN の結合確率
" -c WEIGHT_PTOL=0.036"¥  ###  PN→LN の結合強度
" -c RND_SEED=0"  ###  RANDOM SEED, DEFAULT は 0, 変えるともちろん結果も変わる

LPG="lpgparm -t 4MB -s 4MB -d 4MB -h 4MB -p 4MB"
MPIEXEC="mpiexec -mca mpi_print_stats 1"
#MPIEXEC="mpiexec -mca mpi_print_stats 2 -mca mpi_print_stats_ranks 0"

#PROF="fapp -C -d ./pd -L1 -Hevent=Statistics"
#PROF="fipp -C -Ihwm,call -d ./prof"
#PROF="fipp -C -Ihwm,call -d pd"
#PROF="fipp -C -Ihwm,call -Puserfunc -i 20 -d ./pd"
PROF=""

echo "${PROF} ${MPIEXEC} ${LPG} ${NRNIV} ${NRNOPT} ${HOC_NAME}"
time ${PROF} ${MPIEXEC} ${LPG} ${NRNIV} ${NRNOPT} ${HOC_NAME}

sync
```

スクリプトを実行する前に

```
./als_v2/ $ cp ./mod/*.mod NEURON_KPLUS/mod/
```

als_v2/mod/の中にある mod ファイルを NEURON_KPLUS/mod/に入れて special を生成する．もしくは，als_v2 の中に special を生成しても問題無いです．各自便利な方法でやりましょう．

スクリプトを各自の環境に合わせて変更

[K]

```
……途中省略……
#PJM --stgin-basedir /home/hp120263/k01793/code/als_v2/
……途中省略……
# STAGE OUT TO DATA DIRECTORY
```

```
#PJM --stgout "rank=* %r:./*.txt /data/hp120263/park/al_V2/%j/record/"
#PJM --stgout "rank=* %r:./*.dat /data/hp120263/park/al_V2/%j/spike/"
#PJM --stgout "rank=* %r:./pd/* /data/hp120263/park/al_V2/%j/pd/"
```

[FX100]

```
NRNIV="/home/usr7/z48927t/github/neuron_kplus/specials/sparc64/special -mpi"
```

スクリプトの実行法

[K]

```
./als_v2/job $ pjsub single12.sh
```

[FX100]

```
./als_v2/single-src $ pjsub ../fxjob/single12.sh
```

結果の解析法[K],[FX100]

ファイルの整理

```
DATA_DIR/JOBID $ sh file_arrange.sh JOBID
```

グラフの生成[L]

JOBID(ファイル名)で指定したファイルを Download して解凍するプログラム

[K]データダウンロード

```
DATA_DIR/JOBID $ sh download.sh JOBID
```

[FX100]データダウンロード

```
DATA_DIR/JOBID $ sh fx100_download.sh JOBID
```

グラフの生成

```
DATA_DIR/JOBID $ python drawGraph.py JOBID/record
DATA_DIR/JOBID $ sh spike.sh JOBID/spike
```

Single-ALS のローカルマシン用 Job Script 解説

```
#! /bin/bash

### 実行時刻でフォルダを設定してデータを保存
# Make directory to save data file
Time=`date '+%m%d%H%M%S'`
echo "TIME : ${Time}"
RESULT_DIR="../single-result/"
RECORD_DIR="${RESULT_DIR}${Time}/record"
SPIKE_DIR="${RESULT_DIR}${Time}/spike"
OUT="${RESULT_DIR}${Time}/out"
#SPIKERECORD_DIR="${BASE_DIR}${Time}/spike"
echo "DATA DIRECTORY : ${RECORD_DIR}"
mkdir -p ${RECORD_DIR}
mkdir -p ${SPIKE_DIR}

NRNIV="../specials/x86_64/special -mpi"  ### 実行 special ファイル，環境に応じて変更
HOC_NAME="./main-local.hoc"

NRNOPT=¥
" -c STOPTIME=6000"¥
" -c IS_SUPERCOMPUTER=0"¥
" -c START_TIME=${Time}"¥
" -c GABAB_ON=1"¥
" -c GABAA_ON=1"¥
" -c PTOL_ON=1"¥
" -c NSYNAPSE=350"¥
" -c NPN=10"¥
" -c NLN=70"¥
" -c NRN=0"¥
" -c GABAA_LTOP=18.0"¥  ### シナプスの最大コンダクタンス
" -c GABAA_LTOL=0.0"¥
" -c GABAB_LTOP=22.0"¥
" -c GABAB_LTOL=0.7"¥
" -c PN_NACH_GMAX=0.38"¥
" -c LN_NACH_GMAX=0.10"¥
" -c PtoL_NACH_GMAX=0.1"¥
" -c DOSE=1000"¥
```

```
" -c NSTIM=30"¥
" -c PROB_LTOP=0.5"¥
" -c PROB_LTOL=1.0"¥
" -c PROB_PTOL=0.5"¥
" -c RND_SEED=0"¥
" -c JOBID=0"¥
" -c RNtoLN_Latency=150"¥
" -c VOLTAGERECORD=1"¥
" -c CURRENTRECORD=0"

MPIEXEC="mpiexec -n 8"
EXEC="${MPIEXEC} ${NRNIV} ${NRNOPT} ${HOC_NAME}"

echo $EXEC
time $EXEC |tee $OUT

python ../src/drawGraph.py $RECORD_DIR  ### グラフ生成
#python ../src/drawISF.py $SPIKE_DIR
python ../src/spike_analyze.py $SPIKE_DIR  ### スパイクタイミングのグラフ生成
python ../src/whole_in_one_spike.py $SPIKE_DIR ###  細胞別のスパイクデータをまとめる
```

スクリプトの実行法

シミュレーション実行修了後，グラフ生成コードも実行される

```
als_v2/single-src $ sh run.sh
```

5. Multi-compartment Antennal lobe Simulation 仕様書

マルチコンパートメント ALS 解説

multi-ALS 制作で考えた事

- 1コアでもマルチコアでも同じコードで動かせられるようにする(OK)
- シナプスの設定を簡単に変えられるようにする→シナプスのいち情報，細胞間の結合情報を分かりやすくする，シナプスの種類(nACh，GABA_A，GABA_B，電気シナプス)の管理　(現在シナプス種類を簡単に変更する事はできない．しかし，シナプスリストファイルに項目を追加する事で実現可能)
- 細胞数を自由に変えられるようにする(細胞情報ファイルを書き換える事で可能，OK)
- PN を 1 つ、LN を 7 つを基本単位として増やせられるようにする(LN の SWC ファイルがもっと欲しい…)
- PN と LN は基本的に 1 細胞 1 コアで動かすことを前提にする。また、RN は余った領域で動くようにする(OK)

マルチコンパートメント ALS のローカルマシン用の job script

```
#! /bin/bash
### データを保存するフォルダを生成
# Make directory to save data file
Time=`date '+%m%d%H%M%S'`
echo "TIME : ${Time}"
RESULT_DIR="../result/"
RECORD_DIR="${RESULT_DIR}${Time}/record"
SPIKE_DIR="${RESULT_DIR}${Time}/spike"
OUT="${RESULT_DIR}${Time}/out"
#SPIKERECORD_DIR="${BASE_DIR}${Time}/spike"
echo "DATA DIRECTORY : ${RECORD_DIR}"
mkdir -p ${RECORD_DIR}
mkdir -p ${SPIKE_DIR}

NRNIV="../specials/x86_64/special -mpi"
### 実行するファイルを設定
#HOC_NAME="./main_antenna.hoc"
HOC_NAME="./ln_test.hoc"
#HOC_NAME="./main_test.hoc"
#HOC_NAME="./loadbalance_test.hoc"

NRNOPT=¥
" -c STOPTIME=40"¥
" -c IS_SUPERCOMPUTER=0"¥
" -c START_TIME=${Time}"¥
```

```
" -c WEIGHT_200=0.05"¥
" -c WEIGHT_300=0.004"¥
" -c WEIGHT_301=0.008"¥
" -c GABAA_GMAX_LTOL=5.0"¥
" -c GABAB_GMAX_LTOL=5.0"¥
" -c GABAA_GMAX_LTOP=0.2"¥
" -c GABAB_GMAX_LTOP=0.0065"¥
" -c GABAB_ON=1"¥
" -c GABAA_ON=1"

### 実行するコア数を設定
#MPIEXEC="mpiexec -n 4"
#MPIEXEC="mpiexec -n 5"
MPIEXEC="mpiexec -n 1"
#MPIEXEC=""

EXEC="${MPIEXEC} ${NRNIV} ${NRNOPT} ${HOC_NAME}"

#mpiexec -np 4 $NRNMPI/nrniv -mpi parallel_simulation1201.hoc
#mpiexec -np 8 ./mod/x86_64/special -mpi main.hoc
echo $EXEC
time $EXEC |tee $OUT

python drawGraph.py $RECORD_DIR
python drawISF.py $SPIKE_DIR
```

スクリプトの実行法

シミュレーション実行修了後，グラフ生成コードも実行される

```
als_v2/src $ sh run.sh
```

ネットワーク情報ファイル

```
# gid, cellid, swcid, cloneid, SWC file path, position&rotation file, Synapse information file
$ CELLS 総細胞数
$ PN 細胞数
細胞情報
$ LN 細胞数
細胞情報
```

- 細胞情報は gid, cellid, swcid, cloneid, SWC file path, position&rotation file, Synapse information file で記入される，また，各項目間は tab を入れる．
- cellid は細胞種別に番号を付ける

	cellid
RN	1
PN	2
LN	3

- swcid は同種の細胞の中で SWC ファイル別に細胞を管理するために付ける番号，現在の swcid は以下の通り

../input/swc/050622_4_sn_bestrigid0106_mkRegion.swc ： PN, cellid = 2, swcid = 0

../input/swc/040823_5_sn_bestrigid0106_mkRegion.swc ： LN, cellid = 3, swcid = 0

../input/swc/050205_7_sn_bestrigid0106_mkRegion.swc ： LN, cellid = 3, swcid = 1

- cloneid は同じ swc ファイルを複数回使った場合の識別番号，0 から始まる
- 最終的に細胞に付ける gid は

$$\mathrm{gid} = \mathrm{cellid} \times 100000 + \mathrm{swcid} \times 1000 + \mathrm{cloneid}$$

である．すべての細胞で重ならないように設定

- SWC file path は swc ファイルの保存場所
- position&rotation は swc ファイルの移動，回転情報(現在は何も設定してない)
- Synapse information file は細胞毎にシナプスの情報を含んだファイル

例) als_V2/info/network_info/network_info_3_1PN2LNs.dat

```
# gid, cellid, swcid, cloneid, SWC file path, position&rotation file, Synapse information file
$ CELLS 3
$ PN 1
200000          2          0          0          ../input/swc/050622_4_sn_bestrigid0106_mkRegion.swc
none ../input/synapse_info/3_1PN2LNs/200000syn.dat
$ LN 2
300000          3          0          0          ../input/swc/040823_5_sn_bestrigid0106_mkRegion.swc
none ../input/synapse_info/3_1PN2LNs/300000syn.dat
301000          3          1          0          ../input/swc/050205_7_sn_bestrigid0106_mkRegion.swc
none ../input/synapse_info/3_1PN2LNs/301000syn.dat
```

シナプス情報ファイル

例)../input/synapse_info/3_1PN2LNs/300000syn.dat

```
# This file shows the file path of synapse
$ fromRN  ←RN からシナプス入力を受ける場所
../input/synapse_list/fromRN/040823_5_sn_SynapseList.dat
$ CtoC  ←細胞間のシナプス
```

```
2  ←シナプス設定ファイルの数
presynaptic cell の gid, postsynaptic cell の gid, シナプス情報ファイル
300000 200000 ../input/synapse_list/3_1PN2LNs/300000_200000_manual.txt
300000 301000 ../input/synapse_list/3_1PN2LNs/300000_301000_randomize.txt
```

シナプスリスト

例）../input/synapse_list/3_1PN2LNs/300000_200000_manual.txt

```
$ PRE_CELL 300000  ←PRESYNAPTIC CELL の gid
$ POST_CELL 200000 ←POSTSYNAPTIC CELL の gid
$ NCONNECTIONS 10 ←シナプス結合の数
Presynaptic cell のコンパートメント番号, Postsynaptic cell のコンパートメント番号, シナプスの gid
2739 110 2000000
2730 107 2000001
2366 104 2000002
2363 101 2000003
2055 98 2000004
2050 97 2000005
1809 94 2000006
1801 114 2000007
1590 109 2000008
1579 102 2000009
```

Synapse の gid は細胞の gid より一桁多い.

Synapse gid の仕様

X,YYY,YYY

X ：

結合のタイプ	X
RN→PN, LN	1
LN→PN	2
LN→LN	3

YYY,YYY ：

生成した順に 0 から付ける

現在 multi-als をローカルで動かす時に single-core で動かすと spikeout でエラー発生

mpiexec -n 4 にすれば問題無い. 理由は把握中

マルチコンパートメント ALS のコード解説

Goto Estimator(GE):後藤さんのパラメータ推定プログラム
Antennal Lobe Simulator(ALS):触角葉神経回路シミュレーションプログラム

*** main.c ***
load_file:
nrngui.hoc
stdlib.hoc
CellSwc3.hoc
loadinfo.hoc
areaList.hoc
exSectionList.hoc

Goto Estimator のパラメータ
SPIKEGEN_POS, SPIKEGEN_SIZE, GKDRBAR, GNABAR, GKCBAR, KCA, KNA, KK, CM, GL, EL, CAGAIN

ALS のパラメータ
STOPTIME : シミュレーション時間
IS_SUPERCOMPUTER : ローカルマシンでは 0,スパコンでは 1 に設定
START_TIME : 刺激を開始する時間
WEIGHT_200 : RN→PN の結合強度
WEIGHT_300 : RN→LN1 の結合強度
WEIGHT_301 : RN→LN2 の結合強度
GABAA_GMAX_LTOL : LN↔LN の GABA_A の最大コンダクタンス
GABAB_GMAX_LTOL : LN↔LN の GABA_B の最大コンダクタンス
GABAA_GMAX_LTOP : LN→PN の GABA_A の最大コンダクタンス
GABAB_GMAX_LTOP : LN→PN の GABA_B の最大コンダクタンス
GABAB_ON : GABAB_ON==1 のみ GABA_B シナプスを設定
GABAA_ON : GABAA_ON==1 のみ GABA_A シナプスを設定
DOSE : 刺激濃度
NSTIM : 刺激回数

RNperSYN : RN 一つ当たりのシナプス結合数,
info_filename : PN と LN の情報ファイル名
info_filename_rn : RN の情報ファイル名

spt_file_dir : 刺激用入力ファイルのフォルダ名

ShowInitialStatus() : プログラムの最初に設定したパラメータを表示
mkNetwork() : make network, mkCells と mkConnections を実行

load_file:
　　divider.hoc　: 並列実行時に core 別にシミュレーションする細胞するを計算

NPN : 総 PN 数
NLN : 総 LN 数
NRN : 総 RN 数
NCELLS : PN, LN, RN を含めた全体の細胞数

mkCells :
　　読み込んだデータを基に細胞を宣言
　　全体のコア数によって RN の配置が変わる. 全体のコア数が NPN+NLN より大きい場合は RN は PN, LN とは異なるコアに配置
　　全体のコア数が NPN+NLN 以下の場合は全コアに均等に配分する

load_file:
　　connect.hoc : 各種の神経回路関連設定

****** connect.hoc ******

NUMAXONS_PER_RN : RN の軸索の数
NUMSYNAPSE_PER_CELL : 1 細胞当たりの RN から入力されるシナプスの数

mkPreSynapseOnRN() : RN に Pre-synapse を設定
calcSyngid() : Calculate Synapse Gid, Presynapse の gid を計算(gid に重複が無いように設定)
calcSrcgid() : Calculate Source Gid, postsynapse と繋げる presynapse の gid を計算
mkPostSynapseOfRN() :　PN と LN に RN からシナプス入力を受ける postsynapse を設定する
connectRNtoOthers() : RN と LN,PN を繋げる

set_netcon_gid():
get_netcon_gid():
print_netcon_gid():
netcon_gid_exists():
netcon_cmpt_exists():
netcon_nid_exists():

readSynapseList2():
makePreSynapse():
makePostSynapse():

makePostSynapse_NET_GABA():

setPostSynapseVoltageRecord():
setPostSynapseVoltageRecord_t():
fprintPostSynapseVoltageRecord():

makeNET_GABAsynapse():
makeNET_GABAsynapse2():

makePRL_GABAsynapse():

connectNtoN():
connectNtoN_NET_GABA():
connectNtoN_NET_GABA2():

connectNtoN_GABA():
connectNtoN_DUMMY(): NEURON の仕様上の問題で何も継ががって無いとエラーになった気がする

****** connect.hoc ******

initModel_PN() ： GE からコピー， PN の初期設定
mkConnections() : ネットワークの設定

load_file :
　　stim.hoc : 刺激関数

****** stim.hoc ******
mkIntermitStim() : RN へ電流刺激（論文の 4.2 章参照）
mkIclamp(), mkIclamp_pn(), mkIclamp_ln(), mkIclamp_ln2() テスト用刺激関数

mkSerialStim() : 刺激タイミングファイルを読み込んで，それに合わせて RN を発火させる

****** stim.hoc ******

```
mkStimulations() : 刺激の設定

load_file :
        recorder.hoc : record 用の関数設定

pc.setup_transfer(): シナプス設定の為，シミュレーションが始まる前に必ず実行する必要がある

setPrinter() : record 結果の出力

*** main.hoc ***
```

パラメータ推定プログラム
K を使ってパラメータ推定を行った結果
1024 遺伝子, 300 世代
/data/estimation/dose_MsPN/4978194

パラメータ推定で得られたパラメータを使って Random Seed を変えながら 10 回程度動かした結果
single-result/0131151149~0131151959/

パラメータ推定のプログラムは後藤さんの推定プログラムを少し変えて,Dose-Response の Fitting ができるようにして, 動かして見た程度で終わっている. 推定プログラムの設定や推定パラメータを変えながら試すところまでは到達して無い. また, Restart 戦略も使ってないので試してみる価値はあると思われる.

付録

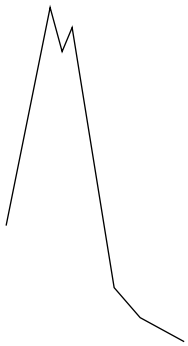
藤原さんのシミュレーションコードの使い方

画像データの解析リスト

渡すデータリスト

未完成リスト

- 実験データとの定量的な比較(試問時の竹内先生の質問)
- 4 章の実験データ B(200ms 刺激データ)の解析方法では PSTH を基準にフィッティングする時に Bar の中心点の高さを基準にフィッティングを行った．しかし，正確なフィティングを行うためには，PSTH の面積が一致するようなフィッティングを行わなければならない．(時間が足りなかったので実装してない)
- Peter’s Rule を使ってシナプス位置を推定するときにはコンパートメントの体積や長さなどで規格化をした後にランダムで候補を選ぶようにしなければならない．(現在はコンパートメントの番号をランダムで選ぶようになっている)
- 4 章の実験データ A(1 秒刺激データ)では，スパイクの検出を微分値の符号が反転する (t_(n)-t_(n-1))*(t_(n+1)-t_(n))<0 の時をスパイクであると考えた．



図

しかし，上の図のようにスパイクによっては頂点が 2 つあるようなスパイクも存在する．したがって，実験データにおいてスパイク間隔がすごく短いものは上記のようなケースを考えなければならない

<u>各データの保存場所</u>

修士論文の保存場所：/media/data/MASTER_THESIS_TEX_FILES_20160331.tar

修士論文と Abstract の TEX ファイルが乗っています．必要であれば参考に使って下さい．

als_v2：/media/data/PROGRAM_BACKUP/als_v2_20160331_TAKINGOVER_FINAL

(リモートで git に変更内容を更新しようとしたら何故かエラーになったので git に上がっているデータと多少差があるかも知れないです．コードの実行結果が一部追加されています．それ以外のソースコードは git と同じものです．)

DataAnalysis: /media/data/PROGRAM_BACKUP/DataAnalysis_20160331_TAKINGOVER_FINAL.tar

藤原さんの matlab 実行コード：/media/data/DROPBOX_BACKUP/fujiwara_terufumi_matlab.zip

INCF Short Course の講義資料：/media/data/DROPBOX_BACKUP/INCF_ShortCourse2015.zip

プログラムの実行結果データなどのデータファイル：/media/data/DROPBOX_BACKUP/研究データ.zip

Antennal Lobe シミュレーション実行結果：/media/data/al_V2(スパコンの実行結果, Neuroinformatics2015 以降), al_V2_antenna_2015_11to2015_12(ローカルの実行結果), al_V2_antenna_data_Before_Neuroinformatics2015(ローカルの実行結果), al_V2_Before_Neuroinformatics2015(スパコンの実行結果, Neuroinformatics2015 以前)
パラメータ推定プログラムの実行結果：/media/data/estimation/

追加予定

Python Script ファイルの実行方法
drawISFofSPT.py
python drawISFofSPT.py ファイル名
python drawISFofSPT.py フォルダ名
Instantaneous Frequency グラフを生成