BioE 332 Assignment 2: A recurrent network model of decision making

Kwabena Boahen, TA: Tatiana Engel

Assignment due: April 19, 2013

1 Goals

In this assignment we will implement and analyze a neural network model that captures monkeys' behavior and neural activity in a perceptual decision task. The network has structured recurrent excitation. As a result two attractor states co-exist in the network during the decision period. Global inhibition generates competition between these attractors, which leads to the winner-take-all dynamics, where only one of two competing neural populations can reach high firing rate. We will perform phase-plane analysis of the mean-field reduced network model to understand the dynamical mechanism of the decision making process. We will explore how the network's ability to integrate inputs depends on the structure of recurrent connectivity, and then verify these results in a full spiking network model. By completion of this assignment we will learn about:

Scientifically:

1. Competing attractor states in a recurrent network model.
2. Dependence of the integration time on the coherence of stimulus input.
3. Mean-field reduction of spiking neural networks.
4. Phase-space dynamics and bifurcation analysis of the decision-making model.

Technically:

1. Dynamical systems analysis using PyDSTool.
2. Running large number of simulations on the cluster.

Readings:

1. Decision making model that will be implemented in this assignment, Ref. [1].
2. Mean-field reduction of the spiking network model and phase-plane analysis, Ref. [2].

2 Building neural circuit model of decision making

We will reproduce the decision making model from Ref. [1]. The model is a recurrent neural network that comprises $N_{\mathrm{E}} = 1600$ excitatory and $N_{\mathrm{I}} = 400$ inhibitory neurons. The recurrent excitation in the network is structured. There are two stimulus-selective groups of neurons; within each group there is a strong recurrent excitation that can maintain persistent activity. These two neural groups compete through the global feedback inhibition.

The decision making model has the same ingredients as the working memory model from the Assignment 1. In both models, the slow dynamics and non-linear saturation of the NMDA conductance are essential for generating multistability, whereby the network has a stable spontaneous state (uniform: low firing rate across all neurons) and stable high-activity states (non-uniform: high firing rate only in a sub-population of neurons). In the working memory model we used rotationally symmetric connectivity to implement a continuum of high-activity attractor states (a line attractor). In the decision making model, there are two neuronal clusters – defined by strong recurrent connectivity – and consequently there are only two high-activity attractor states. Competition between these attractors results in the decision-making dynamics. Note that for the default parameter set from Ref. [1], the decision making model also provides a working memory function: the high-activity attractor states are stable in the absence of stimulus, thus the memory of the decision is maintained autonomously in their persistent activity.

Since the dynamical equations for neurons and synaptic currents are identical in the working memory and decision making models, for the most part you should be able to implement the decision making model on your own. There are only a few details that you should pay attention to as outlined in the following subsections.

2.1 Specify integration method and time step

To perform a more accurate numerical integration of the model equations, we will use the second order Runge-Kutta method with small integration time step $dt = 0.02$ ms. By default Brian uses Euler method, but this can be changed by providing the argument `order=2` to the `NeuronGroup` constructor.

The integration time step can be specified using Brian's clock object. Many Brian objects store a clock that specifies at which times the object will be updated. If no clock is specified, the program uses the global default clock `defaultclock`. Each clock has the time step attribute `dt`, which you can set to the desired value. We will create our own clock `simulation_clock`, and use it to update the neuron groups as well as the external Poisson groups. We could have used the `defaultclock`, but we chose to do it this way:

```
simulation_clock=Clock(dt=0.02*ms)

Pi = NeuronGroup(NI, eqs_i, threshold=Vt_i, reset=Vr_i, \
refractory=tr_i, clock=simulation_clock, order=2)

PGi = PoissonGroup(NI, fext, clock=simulation_clock)
```

2.2 Subgroups of excitatory neurons

In the model, all excitatory neurons have the same cellular properties. Their segregation into two stimulus-selective populations and one non-selective population is determined solely by the structured recurrent connectivity. It is therefore convenient to create a group `Pe` with $N_E = 1600$ excitatory neurons, and then segregate three subpopulations `Pe1, Pe2, Pe0` within this group. For example:

```
Pe1 = Pe.subgroup(N1)
```

where N_1 is the number of neurons in the subgroup `Pe1`. This way we can specify the properties common to all excitatory neurons by referring to the group `Pe`, and setup the structured connectivity and stimulus selective inputs by referring to the subgroups `Pe1, Pe2`.

2.3 Synaptic latency

In the decision making model from Ref. [1] all synapses have a latency of 0.5 ms, which you should specify when you implement the recurrent connections. For example:

```
Cie = Connection(Pi, Pe, 's_gaba', weight=1.0, delay=0.5*ms)
```

You should also add the synaptic latency to the implementation of the presynaptic NMDA gating. In the working memory model there was no synaptic latency, and the auxiliary x-variable in the equations for the pre-synaptic NMDA gating was incremented instantaneously at the time of each spike. This instantaneous update was achieved by modifying the reset rule for the excitatory population. In the decision making model, the increment of x should be delayed by 0.5 ms after a spike. This can be implemented in Brian with the `IdentityConnection` class, which you have already used to add the external Poisson inputs:

```
selfnmda=IdentityConnection(Pe,Pe,'x',weight=1.0,delay=0.5*ms)
```

2.4 Recurrent AMPA current

In the decision making model the recurrent excitation is mediated by both AMPA and NMDA conductances (in the working memory model there was no recurrent AMPA conductance). We will therefore add the recurrent AMPA connections to the network. Because we use a linear model for the AMPA-synapse, we can connect the external and recurrent AMPA inputs to the same AMPA gating variable; we just need to scale the connection weights appropriately. Particularly, determine the AMPA current in the voltage equation using the conductance $g_{\text{rec, AMPA}}$:

$$I_{\text{AMPA}} = -g_{\text{rec, AMPA}} \; s_{\text{AMPA}} \; (V - E_{\text{AMPA}}). \tag{1}$$

Then connect the external Poisson inputs to the same `s_ampa` gating variable with the weight equal to `wext`$= g_{\text{ext, AMPA}}/g_{\text{rec, AMPA}}$:

```
Cpi = IdentityConnection(PGi, Pi, 's_ampa', weight=wext_i)
```

To implement the structured recurrent AMPA connections, use the parameters w_+ (wp) and w_- (wm) to set the weight of the recurrent AMPA connections that are different from 1:

```
C11 = Connection(Pe1, Pe1, 's_ampa', weight=wp, delay=0.5*ms)
```

2.5 Structured NMDA connections

Due to the non-linearity of NMDA gating equations, we will again track the NMDA gating variable presynaptically for each excitatory neuron. The update operation for the postsynaptic NMDA conductance should also be modified to include the structured NMDA connectivity:

```
@network_operation(simulation_clock, when='start')
def update_nmda(simulation_clock):
    s_NMDA1 = Pe1.s_nmda.sum()
    s_NMDA2 = Pe2.s_nmda.sum()
    s_NMDA0 = Pe0.s_nmda.sum()
    Pe1.s_tot = (wp*s_NMDA1+wm*s_NMDA2+wm*s_NMDA0)
    Pe2.s_tot = (wm*s_NMDA1+wp*s_NMDA2+wm*s_NMDA0)
    Pe0.s_tot = (s_NMDA1+s_NMDA2+s_NMDA0)
    Pi.s_tot = (s_NMDA1+s_NMDA2+s_NMDA0)
```

Here we passed the `simulation_clock` object that we created earlier as an argument to the network operation function. Therefore this network operation will be executed at each `dt` of the `simulation_clock` (i.e. each 0.02 ms).

2.6 Fluctuating Poisson inputs during the stimulus presentation

The decision making network models the accumulation of sensory evidence in experiments, where sensory stimuli might be noisy and their strengths may fluctuate. Therefore we have to generate fluctuating inputs to the network. In Ref. [1], noisy sensory inputs are modeled as Poisson spike trains with the firing rate sampled from a Gaussian distribution each 50*ms. We implement this by defining a new clock for the rate update, and changing the rate of the Poisson groups at the desired times:

```
from numpy.random import *

rate_clock = Clock(dt=50*ms)

# Stimulus firing rates sampled from Gaussian distribution
@network_operation(rate_clock, when='start')
def update_rates(rate_clock):
    if (rate_clock.t>=tstart and rate_clock.t<tstop):
        PG1.rate = fext + mu*(1.0+0.01*c) + randn()*sigmaMu
        PG2.rate = fext + mu*(1.0-0.01*c) + randn()*sigmaMu
    else:
        PG1.rate = fext
```

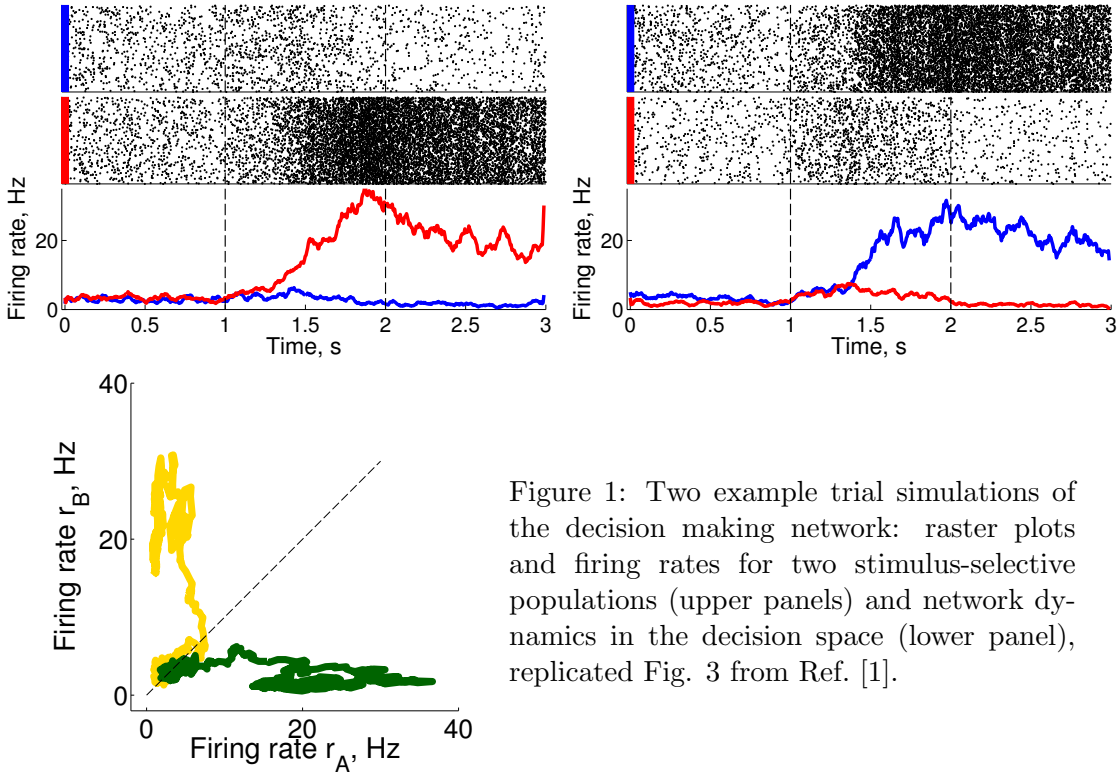

Figure 1: Two example trial simulations of the decision making network: raster plots and firing rates for two stimulus-selective populations (upper panels) and network dynamics in the decision space (lower panel), replicated Fig. 3 from Ref. [1].

```
PG2.rate = fext
```

Note that the mean of the fluctuating stimulus rate is different for two populations and depends on the parameter c representing the coherence of random dot motion.

2.7 Record population firing rates

The network implementation is now complete. To track the firing rates of two stimulus-selective populations, we will compute their population firing rates in a 50 ms time window sliding with a time step of 5ms. To this end, you can set up a `PopulationRateMonitor` in Brian

```
rate_Pe1 = PopulationRateMonitor(Pe1,5*ms)
```

and then smooth the the recorded rate using 50 ms windows.

Simulate 3 s of the network dynamics with the motion stimulus of $c = 0\%$ coherence presented during the time interval from 1 to 2 s. Repeat the simulation 10 times with different random seeds. Plot spike rasters for two stimulus-selective populations and their population firing rates. Observe the divergence of firing rates of two populations (winner-take-all dynamics). Observe that the winning population is randomly changing from trial to trial (neuronal "coin tossing"). Plot spike rasters and population firing rates for two example trials, when different stimulus-selective populations win the competition **[Deliverable]**. Plot the firing rates of two populations on these example trials against each other in the decision space **[Deliverable]**; an example is shown in Fig. 1. Observe

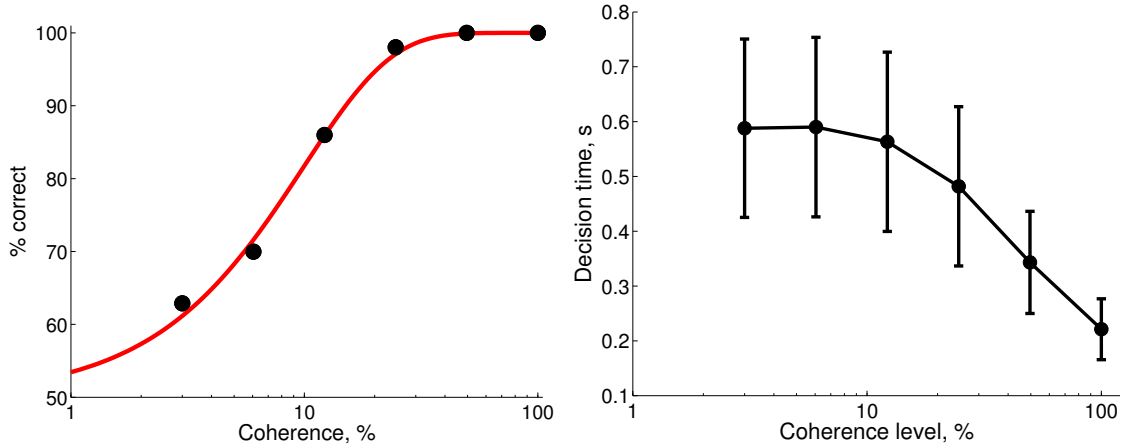

Figure 2: Left: Neurometric function obtained from simulation results (circles) and the fit with the Weibull function $1 - 0.5\exp(-(x/\alpha)^{\beta})$ (red line). Right: average decision time for several input coherence levels, replicated Fig. 5 from Ref. [1].

that the trajectories cluster around three locations in the decision space corresponding to a symmetric low-activity attractor state and two asymmetric attractor states with high firing rate in one population and low in the other.

3 Behavioral performance and reaction time simulations

We saw that for zero coherence stimuli network decisions are random. We will now explore how the network behavior changes when we inject biased inputs, i.e. the mean input firing rate is higher for one population than for the other (non-zero coherence). Perform simulations of the network model using the same protocol as before, presenting stimuli for 1 s, but now with non-zero coherence c. Run at least 200 simulation trials at 6 different coherence levels: 3%, 6.05%, 12.2%, 24.6%, 49.59%, and 100%. Set the firing rate threshold at 15 Hz, and determine the network decision on each trial by detecting which of the two populations crosses the firing rate threshold first. Determine the proportion of correct choices at each stimulus coherence (psychometric function). Determine the mean and the standard deviation of the decision time (time from the stimulus onset until the threshold crossing) at each coherence. Plot these functions [**Deliverable**]; an example is shown in Fig. 2.

Observe that the psychometric function plateaus at 100% for high coherence levels and that the mean and variance of the decision time greatly increase at lower coherence. Interestingly, the decision time at low coherence can be as long as ~ 1 s, which means that the network can integrate inputs on a 10-fold slower time scale than the longest biophysical time constant in the network ($\tau_{\text{NMDA, decay}} = 100$ ms). These observations are consistent with the behavior of monkeys in the random-dot motion discrimination task. In the next section, we will perform phase-plane analysis of the mean-field reduced network model to elucidate the dynamical mechanisms underlying this network behavior.

4 Phase-plane analysis of the reduced two-variable model

The mean-field reduction of the full spiking network model to the simplified two-variable model is derived in Ref. [2]. In our analysis we will use the following model equations:

$$\frac{ds_1}{dt} = -\frac{s_1}{\tau_s} + (1 - s_1)\gamma f(I_{\text{syn},1}); \quad (2)$$

$$\frac{ds_2}{dt} = -\frac{s_2}{\tau_s} + (1 - s_2)\gamma f(I_{\text{syn},2}). \quad (3)$$

Here s_1 and s_2 are NMDA gating variables of the two stimulus-selective populations, and the function $f(I)$ is the populations' f-I curve, which specifies how neurons' firing rate f depends on their input current I:

$$f(I) = \frac{aI - b}{1 - \exp[-d(aI - b)]}. \quad (4)$$

The parameters are $\tau_s = 0.06$ s, $\gamma = 0.641$, $a = 270$ Hz/nA, $b = 108$ Hz, $d = 0.154$ s.

$I_{\text{syn},i}$ is the synaptic input current to each population:

$$I_{\text{syn},1} = J_{11}s_1 - J_{12}s_2 + I_0 + I_1; \quad (5)$$

$$I_{\text{syn},2} = -J_{21}s_1 + J_{22}s_2 + I_0 + I_2. \quad (6)$$

Here parameters J_{ij} represent the effective network connectivity. We will use $J_{11} = J_{22} = 0.3725$ nA, and $J_{12} = J_{21} = 0.1137$ nA. The background current $I_0 = 0.3297$ nA, and parameters I_1 and I_2 represent input currents to two populations from the motion stimulus.

The advantage of the reduced model is that we can understand what dynamical behaviors the model can generate for a particular parameter set using phase-plane analysis, and then explore how this behavior changes when the model parameters are varied (bifurcation analysis). To this end, we will use the Python package PyDSTool.

PyDSTool provides several tutorial examples. To accomplish the goals of this lab, it is sufficient for you to go through the calcium channel model example here:
http://www.ni.gsu.edu/~rclewley/PyDSTool/Tutorial/Tutorial_Calcium.html.

We will construct the phase portraits of the reduced model Eqs. 2-3 for different stimulus inputs; see example in Fig. 3. First, we need to create a data structure that will contain all the model parameters:

```
import PyDSTool as dst

# initialize model parameters structure
DSargs = dst.args(name='Decision_making_model')

# model parameters
DSargs.pars = { 'tauS' : 0.06, 'gam' : 0.641,
                'a' : 270.0, 'b' : 108.0, 'd' : 0.154,
                'J11' : 0.3725, 'J12' : 0.1137,
                'I0' : 0.3297, 'I1' : 0, 'I2' : 0}

# auxiliary functions: fI-curve and recurrent current
```

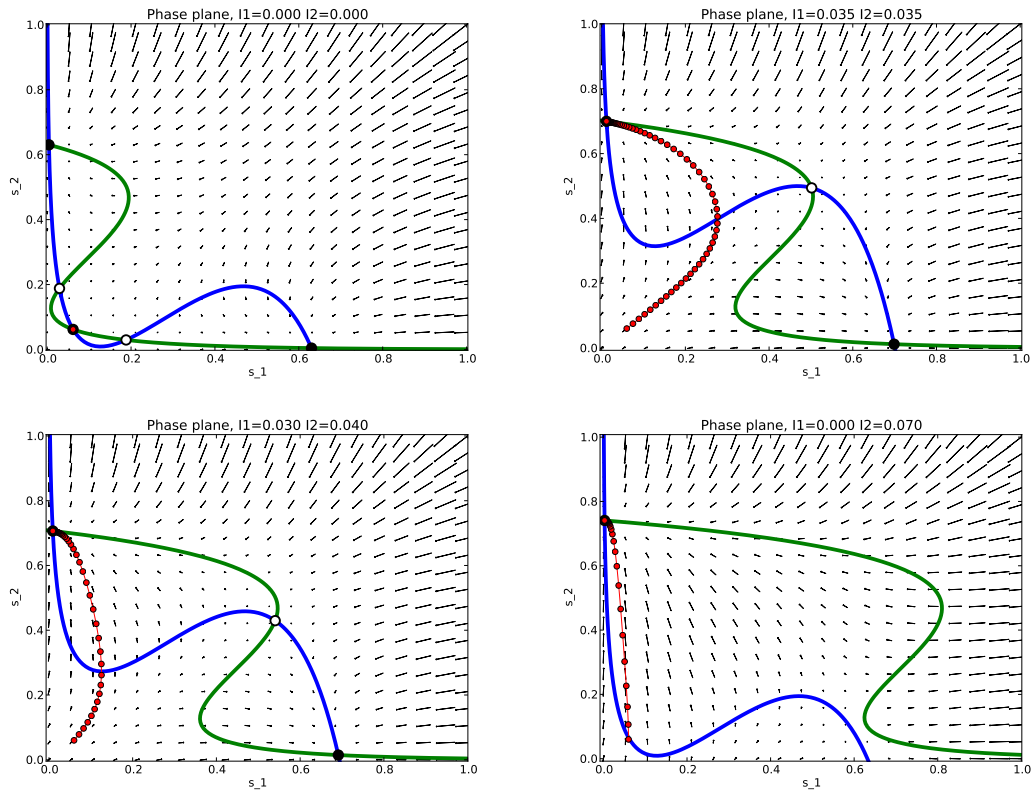

Figure 3: Phase-plane diagrams of the two-dimensional mean-field network model for different stimulation conditions: no stimulus (upper left), symmetric stimulus (0% coherence, upper right), biased stimulus (14% coherence, lower left) and stimulus to one population only (100% coherence, lower right). Replicated Fig. 5 from Ref. [2].

```
DSargs.fnspecs  = {
'fRate': (['I'], '(a*I-b)/(1.0-exp(-d*(a*I-b)))'),
'recCurr': (['x', 'y'], 'J11*x-J12*y+I0') }

# rhs of the differential equations
DSargs.varspecs = {
's1': '-s1/tauS+(1-s1)*gam*fRate(recCurr(s1,s2)+I1)',
's2': '-s2/tauS+(1-s2)*gam*fRate(recCurr(s2,s1)+I2)' }

# initial conditions
DSargs.ics      = {'s1': 0.06, 's2': 0.06 }
# set the range of integration.
DSargs.tdomain = [0,30]
# variable domain for the phase plane analysis
DSargs.xdomain = {'s1': [0, 1], 's2': [0, 1]}
```

Now we can create the model object using these arguments:

```
dmModel = dst.Vode_ODEsystem(DSargs)
```

Now open a figure and plot the vector field determined by the model equations:

```
from PyDSTool.Toolbox import phaseplane as pp

pp.plot_PP_vf(dmModel, 's1', 's2', scale_exp=-1.5)
```

Find the model fixed points:

```
fp_coord = pp.find_fixedpoints(dmModel, n=4, eps=1e-8)
```

and then find and plot the null-clines:

```
nulls_x, nulls_y = pp.find_nullclines(dmModel, 's1', 's2',
n=3, eps=1e-8, max_step=0.01, fps=fp_coord )

plot(nulls_x[:,0], nulls_x[:,1], 'b')
plot(nulls_y[:,0], nulls_y[:,1], 'g')
```

To determine the stability of fixed points we need to evaluate the model's Jacobian at each fix point. PyDSTool can compute the Jacobian function symbolically for you:

```
jac, new_fnspecs = \
dst.prepJacobian(dmModel.funcspec._initargs['varspecs'],
['s1', 's2'], dmModel.funcspec._initargs['fnspecs'])
scope = dst.copy(dmModel.pars)
scope.update(new_fnspecs)
jac_fn = dst.expr2fun(jac, ensure_args=['t'], **scope)
```

Now we can add fixed points to the phase portrait. Stable fixed points are marked by filled circles, and unstable by open circles:

```
for i in range(0,len(fp_coord)):
        fp = pp.fixedpoint_2D(dmModel, dst.Point(fp_coord[i]),
                    jac=jac_fn, eps=1e-8)
        pp.plot_PP_fps(fp)
```

Finally, we will compute and plot an example trajectory, starting at the symmetric low activity state $s_1 = s_2 = 0.06$:

```
traj = dmModel.compute('trajectory1')
pts = traj.sample()
plot(pts['s1'], pts['s2'], 'r-o')
```

As you can see, in the absence of the stimulus current $I_1 = I_2 = 0$, there is a stable fixed point near the initial conditions that we have chosen for our trajectory. The trajectory converges to this fixed point.

Now generate phase portraits of the model with different stimulus inputs corresponding to 3 different coherence levels: (i) $I_1 = I_2 = 0.035$ nA, (ii) $I_1 = 0.03$ nA, $I_2 = 0.04$ nA, (iii)

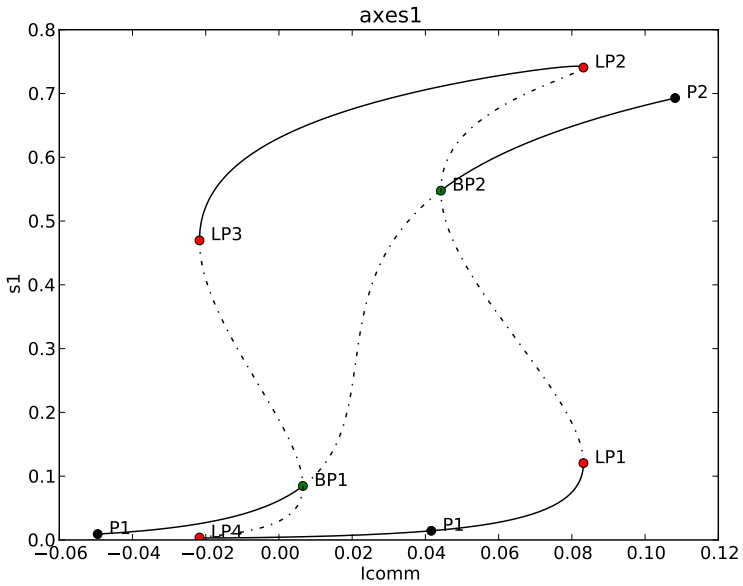

Figure 4: Bifurcation diagram of the reduced model with stimulus strength I_{comm} as a parameter for zero coherence level, reproduced Fig. 10 from Ref. [2].

$I_1 = 0$ nA, $I_2 = 0.07$ nA **[Deliverable]**; an example is shown in Fig. 3. Explain how the changes you see in the phase portrait of the system and in the time course of the example trajectory correspond to the observations you made about the behavior of the full spiking model in Fig. 2 **[Deliverable]**.

5 Bifurcation diagram of the reduced two-variable model

To see in a single plot how the phase portrait of the system changes when we change the stimulus current, we will generate a bifurcation diagram for the reduced model. On the bifurcation diagram the fixed points of the model are shown as a function of a changing parameter – in our case this is the stimulus current. Stable fixed points are plotted with solid lines, and unstable ones with dashed lines. Bifurcation is a qualitative change in the system's phase space that happens at some parameter value. For example, fixed points can change their stability at the bifurcation, new fixed points can appear, or existing ones can disappear.

Bifurcation diagrams can be generated by numerical continuation. The algorithm starts in the vicinity of a fixed point, slightly changes the control parameter, and sees how the fixed point changes. PyDSTool has a continuation class `ContClass` that implements these algorithms.

We will generate a bifurcation diagram for the reduced decision-making model for the zero coherence level. Change the model specification so that both populations get equal stimulus current I_{comm}. Then prepare the system to start in the vicinity of the low-activity fixed pint:

```
# Set lower bound of the control parameter
dmModel.set(pars = {'Icomm': -0.05} )
# Initial conditions
dmModel.set(ics =  {'s1': 0.01, 's2': 0.01} )
```

Now we can set up the continuation class, specify parameters for numerical continuation, find the first equilibrium curve and plot it:

```
# Set up continuation class
PC = dst.ContClass(dmModel)

# Equilibrium Point Curve (EP-C). The branch is labeled EQ1:
PCargs = dst.args(name='EQ1', type='EP-C')
PCargs.freepars     = ['Icomm'] # control parameter
PCargs.MaxNumPoints = 1000
PCargs.MaxStepSize  = 1e-4
PCargs.MinStepSize  = 1e-5
PCargs.StepSize     = 1e-3
PCargs.LocBifPoints = 'all' # detect all bifurcation types
PCargs.SaveEigen    = True # to determine stability of branches

PC.newCurve(PCargs)
PC['EQ1'].forward()

PC['EQ1'].display(['Icomm','s1'], stability=True, figure=1)
```

We found one branch of the bifurcation diagram, which shows how stability of the symmetric fixed point changes when the stimulus current is varied. But what happens with the the other asymmetric fixed point when the stimulus strength changes? We can find the second branch of the bifurcation diagram by choosing initial conditions for continuation near the asymmetric fixed point:

```
PC.model.icdict={'s1': 0.01,'s2': 0.8 }
PC.model.setPars = {'Icomm': -0.05}
PCargs.MaxNumPoints = 3000
PCargs.LocBifPoints = ['LP']
PCargs.name = 'EQ2'
PC.newCurve(PCargs)
PC['EQ2'].forward()

PC['EQ2'].display(['Icomm','s1'], stability=True, figure=1)
```

You should get a diagram similar to one shown in Fig. 4 **[Deliverable]**. Explain, which part of this diagram corresponds to the spontaneous activity, decision making dynamics and working memory of the decision **[Deliverable]**.

6 Extension question

Explore how the network's ability to perform decision making computation depends on the strength of the recurrent excitation (determined by the parameter w_+).

First, explore this question in the reduced mean-field model. The mapping of the parameters of the full spiking model on the parameters of the reduced mean-field model is complicated, and here we will not use the exact mapping. Instead, we will mimic changes of the parameter w_+ by changing the effective connectivity parameters J_{11} and J_{12} in such a way that their difference remains constant (i.e. $J_{11} - J_{12} = \text{const}$). Keeping this difference constant ensures that the stable symmetric fix point does not change upon parameter variation, which replicates the fact that in the full spiking model the variation of the parameter w_+ does not affect the spontaneous activity state. Increasing w_+ corresponds to increasing the sum of J_{11} and J_{12} (while keeping their difference constant). See what effect increasing and decreasing w_+ has on the bifurcation diagram, and describe in which parameter regime decision making computation is robust. Use Fig. 12 from Ref. [2] as a guide.

Based on what you learn from the reduced model, go back to the full spiking network model and perform simulations for different values of the parameter w_+. Explain the results using what you learned from the reduced model. You can refer to Fig. 7 from Ref. [1] for guidance.

References

[1] X-J Wang. Probabilistic decision making by slow reverberation in cortical circuits. *Neuron*, 36:955–968, 2002.

[2] K-F Wong and X-J Wang. A recurrent network mechanism of time integration in perceptual decisions. *J Neurosci*, 26:1314–28, 2006.