

# Virtual lab #5 (Building a planarian)

## Overview

In this lab, we will experiment with how a planarian (a type of flatworm) might build a  $V_{\text{mem}}$  pattern that eventually determines its body shape. The  $V_{\text{mem}}$  pattern will in principle be quite simple: relatively negative at the tail, more positive at the head, and smoothly changing in between. However, we will also see patterns with multiple heads, and will see some worms establishing a much bigger head-to-tail  $V_{\text{mem}}$  gradient than others.

Our job will be to explain all these results!

## How to run the code

The worm code is in the file *main\_worm.py*. It uses the function *setup\_lab\_worm()*. Unlike the previous Python labs, you run this one by simply typing “**python main\_worm.py.**” Everything else just happens automatically.

Most of *main\_worm.py* is infrastructure that you don’t really have to understand. More details are at the end of this document, if you’re interested. The top-level code for this simulation is in *setup\_lab\_worm()*, which is just a short function that uses the main infrastructure.

## What the code does

- Our worm consists of 5 cells, connected as a straight line of cells. I.e., each cell is connected by a gap junction to its two neighbors (one ahead of it and one behind). The head and tail cells thus have only one neighbor each.
- The code for this simulation is in *setup\_lab\_worm()*. It
  - creates a worm and does some initial simulation.
  - keeps increasing the density of GJs (i.e., increasing their conductance) until the worms are no longer able to build a head-to-tail gradient of  $V_{\text{mem}}$ .
  - at each stage (i.e., for each value of GJ density), it prints out worm results; namely, the worm’s  $V_{\text{mem}}$  gradient and the worm’s shape.
- Some biological details:
  - In addition to the usual Na, K, Cl and P, each cell has a “mystery” morphagen  $M$  with a valence of -2. There is initially more  $M$  in the head(s) than the tail(s), since the higher (i.e., more positive)  $V_{\text{mem}}$  in the head attracts  $M$ .
  - The worm cells K channels provide positive feedback. A higher  $[M]$  in any cell tends to shut down that cell’s K ion channels, which tends to make the cell’s  $V_{\text{mem}}$  more positive (which in turn attracts still more  $M$ , and so on). We call these *ligand-gated* channels, since their conductivity is controlled by the concentration of a *ligand* (i.e., a local chemical that attaches to them).

## Understanding the output

The output will start with initial setup and then have a few lines for each worm simulation. The few lines may look like

```
240 func evals, status=0 (The solver reached the end of the integration
interval.), success=True
vm=[-29.775 -51.477 -58.855 -51.477 -29.775]mV
[M]=[1.621 0.654 0.449 0.654 1.621] => HTH
```

Simulation: gradient=29.1mV => no gradient  
At GJ density=0.0161,  $\Delta V_{\text{mem}}$ =29.08mV and shape=HTH

Here is how to interpret this:

- The first line is irrelevant except if you have to debug a failed simulation.
- The “Vm=” line gives  $V_{\text{mem}}$  in cells 0-4; the “[M]=” line likewise gives [M].

The worm’s shape (HTH in this case) is determined by the [M] across the five cells. Peaks of [M] indicate a head; valleys indicate a tail. In this case, the peaks of 1.621 in cells #0 and #4 are the heads, and the .449 in cell #2 is a tail – hence the shape of HTH, or a two-headed worm.

The  $V_{\text{mem}}$  gradient (29.08mV in this case) is defined as the difference of the highest  $V_{\text{mem}}$  in any cell (in this case, -29.775mV in cells #0 and #4) minus the lowest  $V_{\text{mem}}$  in any cell (in this case, -58.855mV in cell #2). So,  $-29.775\text{mV} - 58.855\text{mV} = 29.08\text{mV}$ .

#### **To turn in:**

- Since you’re not really changing main\_worm.py, you do *not* need to turn it in.
- Turn in a graph of  $\Delta V_{\text{mem}}$  vs. GJ density (using your favorite graphing tool). Mark each segment of the graph with the associated body shape.
- Turn in your report, including the graph and the questions below.

#### **Questions**

1. As GJ density increases, the worm shape changes from HTHTH to HTH and then to HT. Can you explain why this is?
2. As GJ density increases, the gradient generally keeps shrinking. However, each time there is a shape change (from HTHTH to HTH or from HTH to HT) the gradient rises. Again, can you explain why this is?

For each of the two questions above, first answer it intuitively. You should use the worm model from slide set 4c slides #10 and 28-30, but your description should be intuitive and qualitative rather than numerical. Make reference to the appropriate circuit-analysis tools (e.g., Ohm’s Law, KCL, etc.), but don’t actually solve equations numerically.

After answering both question intuitively, do the math to illustrate your point for just one simple case – a HT worm. Show that reducing the GJ resistance reduces the  $V_{\text{mem}}$  gradient – everything else should follow from there, given your intuitive arguments. To make the circuit analysis easier, you need only put ion channels in at cells that are a head or a tail. I.e., you can ignore the ion channels at cells #1, #2 and #3, only using them at cells #0 and #4. And just pick some reasonable values for the various batteries (that don’t have to be completely exact), and useful values for the various resistors (that don’t have to be at all accurate but should make your math relatively easy and demonstrate the concepts).

#### **Summary of what we learned from this lab**

This is a fairly simple model, and it certainly does not fully explain morphogenesis. It’s unlikely to even fully explain how a planarian decides which end will be its head and which will be its tail. Then what’s the point of the lab?

This lab hopefully shows that a simple model can often give valuable insights that explain data trends. These insights can be easily backed up by simulations that hopefully do match the intuitive predictions.

You can contrast this to the other end of the spectrum; using molecular-level assays to get very detailed information on a system's low-level functionality. The advantage of that technique is that its results can be much more authoritative than simulations. However, it also has a tendency to get "lost in the weeds" of large amounts of detail and to not generalize well.

Certainly both approaches can be valuable 😊. Anyway, one of the main messages of this course is that circuit-analysis techniques can give valuable insights into system-level biology.

### ***Worm-class details***

You don't really need to understand much of the programming, but here it is in case you're interested.

The code in *main\_worm.py* is object-oriented code that sets up a class called Worm. Each Worm object represents one planarian. The object remembers all of the various parameters that one might reasonably change in different worm species or individuals. A Worm object has several methods that are useful to us:

- `__init__()` is the standard Python function to initialize a new Worm object. It mostly just stores all of the parameters of this worm in a parameter dictionary *PD*.
- `__repr__()` is the standard Python function to print a Worm object.
- `setup_Bitsey_network()` is a function that ties a Worm to BITSEY. It looks at the parameters of this worm and builds a corresponding BITSEY network for the worm.
- `sim()` is our simulation function. In an object-oriented world, a Worm should know how to simulate itself. That's what `sim()` does, using the standard BITSEY infrastructure to actually run the simulation.
- `Gradient()` and `shape()` are functions that you can call after calling `sim()`. They look at the simulation results and return the planarian's  $V_{\text{mem}}$  gradient and/or body shape, according to the definitions above.