Hippocampal dendritic spine dynamics as an estrous-modulated stochastic system
EEMB 247/IQB 247 Final Project
Dylan Martins

Introduction

Dendritic spines are the sites of synaptic communication between neurons and their presynaptic excitatory partners [1]. Spines are perhaps best studied in the hippocampus, an area of the brain implicated in memory consolidation and spatial navigation [2].

The rodent estrous cycle, a homolog of the human menstrual cycle, lasts 4 days and is divided into diestrus, proestrus, estrus, and metestrus stages. Pioneering work in the 1990s demonstrated that the number of spines on the dendritic arbor of hippocampal CA1 excitatory pyramidal neurons are modulated by the estrous cycle, increasing during the proestrus stage and decreasing in the estrus stage following ovulation [3]. However, methodological limits at the time required that the spine counts be measured by post-mortem staining. This mean that longitudinal measurements on single dendrites was not possible, and that the effect could only be observed across many rodents for which histology was performed at varied points along the estrous cycle. Recent work has expanded our understanding of the estrous cycle's modulation of spine density by utilizing two-photon microscopy to measure the density of spines along the dendrites of mouse hippocampal neurons longitudinally over the estrous cycle [4]. In addition to measuring the spine density over time, spines were classified by morphology as either filipodia, thin, stubby, or mushroom spines. Understanding these classes as separate populations of spines is important because they have unique biological functions, for example, fliopodia are mostly silent synapses which mediate circuit formation and refinement but require activity-dependent plasticity to mature into synapses that convey information from pre- to post-synaptic partners [5]. The authors [4] were able to measure new growth or pruning away of spines between imaging sessions over the estrous cycle. So too could the transition of one spine from one class to another be determined. They found that, as in [3], spine density peaked in proestrus and fell during estrus following ovulation [4]. The morphological observations coincided with increased coupling between the soma and dendrites, and an increased infiltration of back-propagating action potentials [4]. Additional experiments were performed measuring the somatic activity of hippocampal place cells and the functional plasticity of place cell remapping, but these fall outside the scope of my class project.

Here, I used publicly available collection of structural two-photon imaging spine data [4] to build a stochastic model of four spine class populations and observed the changes in population over the estrous cycle, modulated as a function of the endocrine factors estradiol, progesterone, luteinizing hormone (LH), and follicle-stimulating hormone (FSH).

Experimental methods / dataset

Here, I used a publicly available dataset of structural hippocampal CA1 spine imaging in female mice recorded longitudinally over the estrous cycle [4]. Methodological details are in the original paper, but a brief summary is given here for context.

To determine the estrous stage, animals were staged by vaginal lavage between one and two times per day, and images of stained slides were automatically classified following [6]. Female transgenic Thy1-GFP-M mice were implanted with a head plate and cranial window to which a custom-designed glass right angle prism microperiscope was attached, allowing for transverse hippocampal imaging of dendritic arbors in CA1. Spines were imaged every 12 hours for 8-11 days using two-photon microscopy. Fields across sessions were aligned based on the average projection from a reference session, rigidly registered, binarized, and individual spines on each dendritic segment were classified by finding the spine base as region closest to the dendritic shaft, calculating the length as Euclidean distance from midpoint of spine base to the most distant pixel, divided that vector evenly into three segments: head, neck, and base, and finally applying thresholds on these properties to partition the spines into four categories as: stubby

(neck length < 0.2 µm and aspect ratio < 1.3), thin (neck length < 0.2 µm, spine length < 0.7 µm, head circularity < 0.8 µm), mushroom (neck length < 0.2 µm, head circularity < 0.8 µm), and filopodium (neck length < 0.2 µm, spine length < 0.8 µm, aspect ratio < 1.3).

Spine class transition probabilities

I focused on four state variables, the population size of filopodia (F), thin (H, *not* represented by T to avoid confusion with variables for time), stubby, (S), and mushroom (M) dendritic spines of hippocampal CA1 pyramidal neurons. Let β_x be the spontaneous growth probability of new spines and δ_x be the per-individual death probability for $x \in \{F, H, S, M\}$. Then, let $\gamma_{x\to y}$ be the probability of transitioning from class x to y, for all distinct $x, y \in \{F, H, S, M\} | x \neq y$.

Transition frequencies were calculated using all spine locations that existed for at least one time point during longitudinal imaging. All dendrites, even if multiple dendrites were imagined in parallel from the same animal, were treated as independent. To calculate the transition frequency of all spine classes during the transition from stage s_1 to stage s_2, the number of observed transitions from class x to class y between stages s_1 and s_2 was counted and the transition fraction q was computed as

$$q_{x\to y} = \frac{\sum N_{x\to y}^{s_1\to s_2}}{\sum_x^{s_1}}$$

where $y \neq x$ (e.g., for $x = F$, $y \in H, S, M, n.s.$), where $n.s.$ means "no spine" to indicates a spine was either formed where none previously existed or an existing spine is pruned away. Expressed this way, $q_{n.s.\to x} = \beta_x$ and $q_{x\to n.s.} = \delta_x$. Estrous stages can occur only in sequence, so the possible pairs of $(s_1, s_2) \in \{(D, P), (P, E), (E, M), (M, D)\}$

Each entry in the transition matrix $Q \in \mathbb{R}^{5x5}$ gives the rate of transition from the current spine class x to the spine class y as $Q_{x\to y}$. Rows of Q are the current state, and columns of Q are the destination state. The zeroth row represents outgoing transitions from the $n.s.$ state (i.e., new growth), while the zeroth column represents incoming transitions to the $n.s.$ state (i.e., existing spines pruned away). The diagonal entries, α_x represent the total rate of leaving state x, where

$$\alpha_x = \delta_x + \sum_{y\neq x} \gamma_{x\to y}$$

The transition $\alpha_{n.s.}$ is left as a structural zero, as the $n.s.$ class is one-directional and can both receive and produce an unlimited number of spines – it supplies individual to and removes individuals from other classes, $\{F, H, S, M\}$, but does not have its own population size. The complete transition matrix takes the form

$$Q = \begin{bmatrix} \delta_M & \gamma_{M\to F} & \gamma_{M\to H} & \gamma_{M\to S} & -\alpha_M \\ \delta_S & \gamma_{S\to F} & \gamma_{S\to H} & -\alpha_S & \gamma_{S\to M} \\ \delta_H & \gamma_{H\to F} & -\alpha_H & \gamma_{H\to S} & \gamma_{H\to M} \\ \delta_F & -\alpha_F & \gamma_{F\to H} & \gamma_{F\to S} & \gamma_{F\to M} \\ 0 & \beta_F & \beta_H & \beta_S & \beta_M \end{bmatrix}$$

However, Q must be calculated for all four of the estrous cycle stage transitions, i.e., $Q_{s_1\to s_2} \forall (s_1, s_2) \in \{(D, P), (P, E), (E, M), (M, D)\}$. Values for each Q are shown in Figure 1.

Spine transition probability as a function of endocrine factors

Here, we focus on four key endocrine factors that change in concentration over the estrous cycle: estradiol, progesterone, luteinizing hormone (LH), and follicle-stimulating hormone (FSH). Samples of endogenous endocrine factors over the estrous cycle were turned into a more continuous set of measurements using a cyclical cubic spline interpolation to 0.25 day (6 hour) samples over the 4 day cycle, totaling 16 samples.

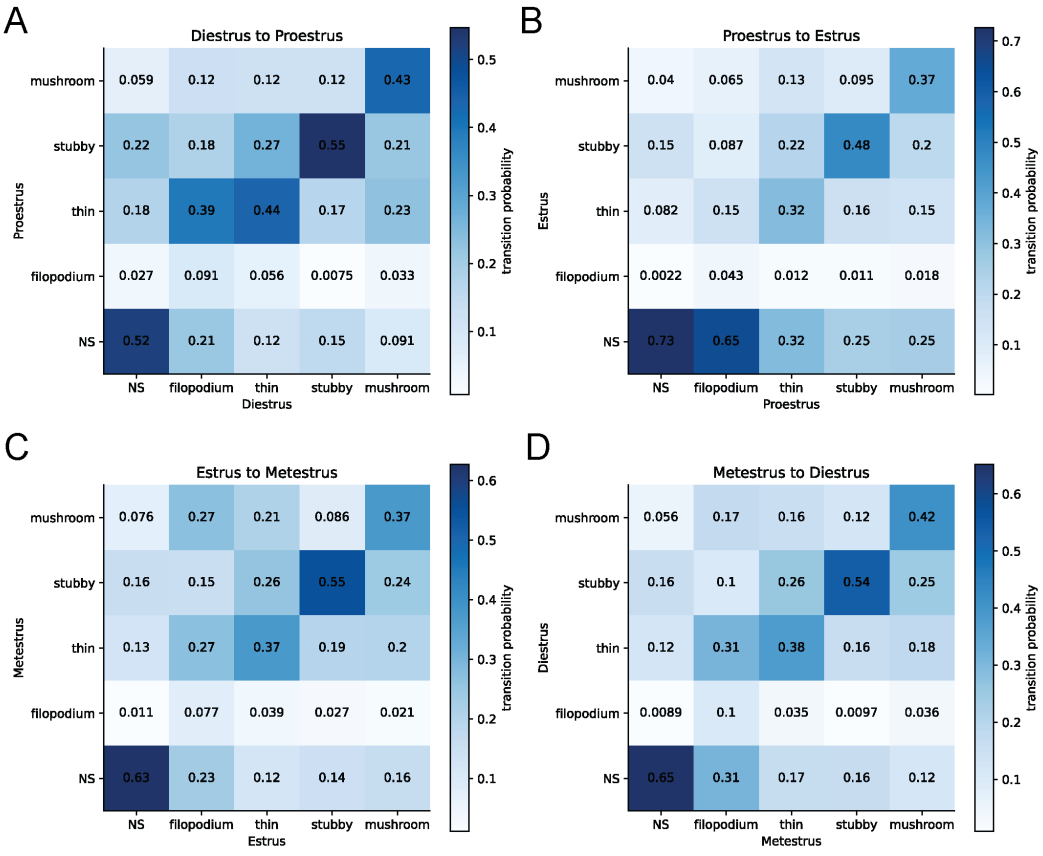

Figure 1: Transition matrices for Diestrus to Proestrus (A), Proestrus to Estrus (B), Estrus to Metestrus (C), and Metestrus to Diestrus (D) estrous stage changes.

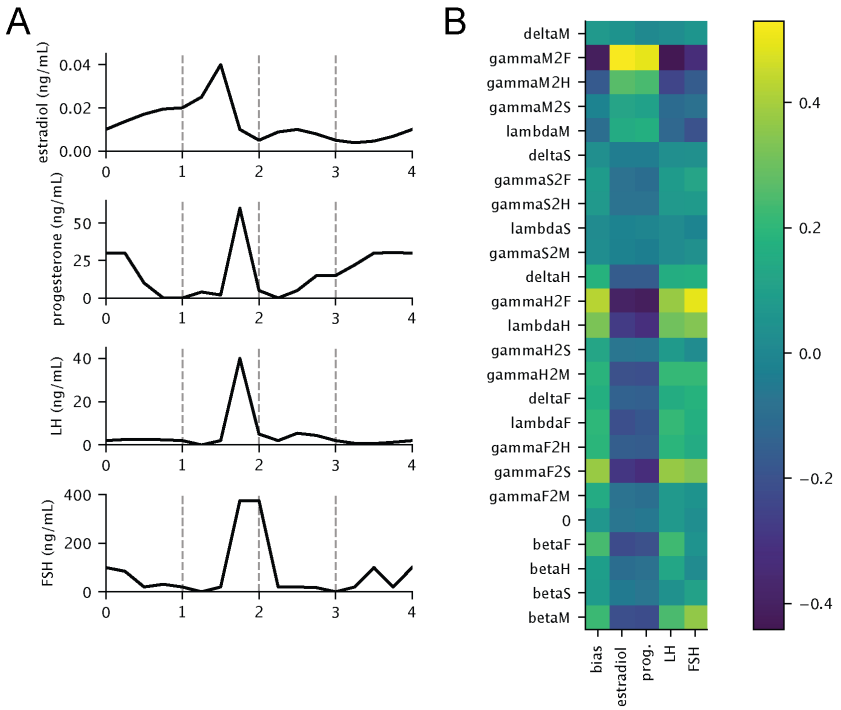

Figure 2: (A) Interpolated endogenous endocrine factor concentrations over the 4-day estrous cycle. (B) Column-normalized values of GLM weights, i.e., values of $\hat{y}$ from Equation 1.

Endogenous endocrine factor levels in female mice follow the four functions shown in Figure 2 in units of ng/mL (values from [7]) over a typically 4 day cycle through diestrus ($0 \leq t < 24$), proestrus ($24 \leq t < 48$), estrus ($48 \leq t < 72$), and metestrus ($72 \leq t < 96$) stages. Ovulation occurs at $t = 2.1 days$ and during the estrus stage of the estrous cycle. In this case, $t = 0$ and $t = 4$ can be thought of as simultaneous timepoints of the cyclical function. The transition probabilities of Q can be thought of as a function of endogenous endocrine factors, as in Figure 3.

To allow the population model to predict spine population changes based on a time-varying environmental variable (and not directly using transition matrices) endogenous concentrations of these four endocrine factors were used to predict the probability of each type of formation, pruning, and transition event using a generalized linear model (GLM) of the form

$$\hat{y} = w_0 + w_{es}[es] + w_{pr}[pr] + w_{lh}[lh] + w_{fs}[fs] \tag{1}$$

where $\hat{y}$ is the transition probability, w_0 is the bias term, and $[es]$, $[pr]$, $[lh]$, and $[fs]$ are the endogenous concentrations in units of ng/mL of estradiol, progesterone, LH, and FSH, respectively. Only the four timepoints at the edges of each estrous stage (i.e., transition points) were used to determine the GLM weights (i.e., the GLM was trained with a y of shape (4) timepoints and an X of shape $(4, 4)$ for the 4 timepoints and 4 endocrine factors). An independent set of the weights $w = \begin{bmatrix} w_0 & w_{es} & w_{pr} & w_{lh} & w_{fs} \end{bmatrix}$ were fit for each possible spine class change $x \rightarrow y \quad \forall \quad x, y \in \{F, H, S, M, n.s.\} | x \neq y$. The column-normalized weights of the GLM for each transition are show in Figure 2.

Spine dynamics as a system of ordinary differential equations

This can be described by a continuous time system of ordinary differential equations for the

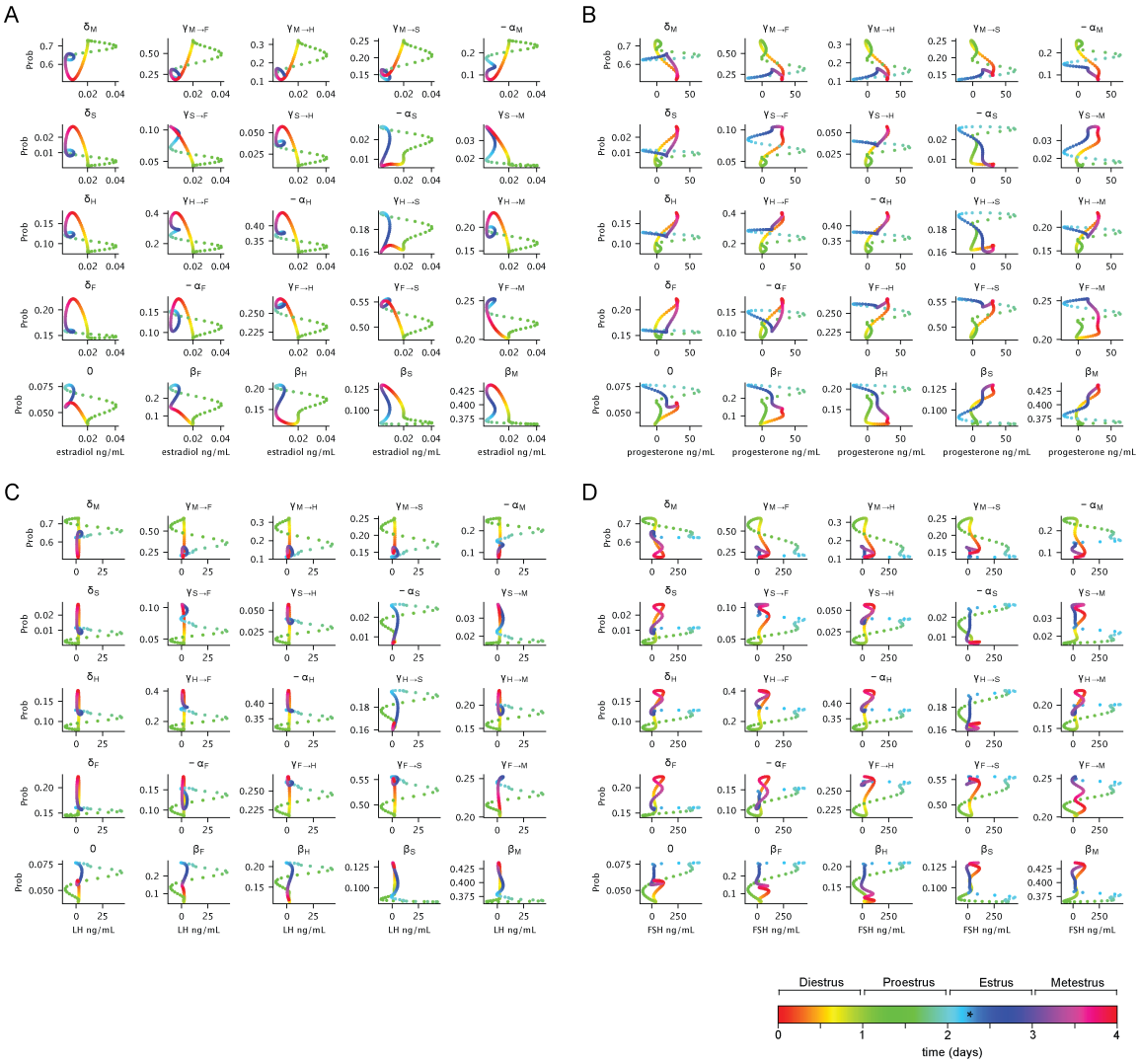

Figure 3: Transition probabilities (i.e., values of $Q_{s_1 \to s_2}$) as an interpolated function of estradiol (A), progesterone (B), LH (C), and FSH (D) concentrations in units of ng/mL. Each point is a timepoint along the estrous cycle. the asterisk (*) in the legend's colorbar indicates the timepoint of ovulation, also indicated by a dark-teal color.

populations of the four spine classes:

$$\begin{aligned}
\frac{dF}{dt} &= \underbrace{\beta_F}_{\text{formation}} - \underbrace{\delta_F F}_{\text{pruning}} + \underbrace{\gamma_{HF}H + \gamma_{SF}S + \gamma_{MF}M}_{\text{transition in}} - \underbrace{F(\gamma_{FH} + \gamma_{FS} + \gamma_{FM})}_{\text{transition out}} \\
\frac{dH}{dt} &= \beta_H - \delta_H H + \gamma_{FH}F + \gamma_{SH}S + \gamma_{MH}M - H(\gamma_{HF} + \gamma_{HS} + \gamma_{HM}) \\
\frac{dS}{dt} &= \beta_S - \delta_S S + \gamma_{FS}F + \gamma_{HS}H + \gamma_{MS}M - S(\gamma_{SF} + \gamma_{SH} + \gamma_{SM}) \\
\frac{dM}{dt} &= \beta_M - \delta_M M + \gamma_{FM}F + \gamma_{HM}H + \gamma_{SM}S - M(\gamma_{MF} + \gamma_{MH} + \gamma_{MS})
\end{aligned} \tag{2}$$

The equilibria of $\frac{dF}{dt}$ can be calculated as

$$\beta_F - \delta_F F^* + \gamma_{HF}H^* + \gamma_{SF}S^* + \gamma_{MF}M^* - F^*(\gamma_{FH} + \gamma_{FS} + \gamma_{FM}) = 0$$

$$F^*(\delta_F + \gamma_{FH} + \gamma_{FS} + \gamma_{FM}) = \beta_F + \gamma_{HF}H^* + \gamma_{SF}S^* + \gamma_{MF}M^*$$

$$F^* = \frac{\beta_F + \gamma_{HF}H^* + \gamma_{SF}S^* + \gamma_{MF}M^*}{\delta_F + \gamma_{FH} + \gamma_{FS} + \gamma_{FM}}$$

and the other equations can be solved similarly as

$$H^* = \frac{\beta_H + \gamma_{FH}F^* + \gamma_{SH}S^* + \gamma_{MH}M^*}{\delta_H + \gamma_{HF} + \gamma_{HS} + \gamma_{HM}}$$

$$S^* = \frac{\beta_S + \gamma_{FS}F^* + \gamma_{HS}H^* + \gamma_{MS}M^*}{\delta_S + \gamma_{SF} + \gamma_{SH} + \gamma_{SM}}$$

$$M^* = \frac{\beta_M + \gamma_{FM}F^* + \gamma_{HM}H^* + \gamma_{SM}S^*}{\delta_M + \gamma_{MF} + \gamma_{MH} + \gamma_{MS}}$$

This can be expressed as the four equations

$$\begin{aligned}
f_1(F^*, H^*, S^*, M^*) &= \frac{\beta_F + \gamma_{HF}H^* + \gamma_{SF}S^* + \gamma_{MF}M^*}{\delta_F + \gamma_{FH} + \gamma_{FS} + \gamma_{FM}} - F^* \\
f_2(F^*, H^*, S^*, M^*) &= \frac{\beta_H + \gamma_{FH}F^* + \gamma_{SH}S^* + \gamma_{MH}M^*}{\delta_H + \gamma_{HF} + \gamma_{HS} + \gamma_{HM}} - H^* \\
f_3(F^*, H^*, S^*, M^*) &= \frac{\beta_S + \gamma_{FS}F^* + \gamma_{HS}H^* + \gamma_{MS}M^*}{\delta_S + \gamma_{SF} + \gamma_{SH} + \gamma_{SM}} - S^* \\
f_4(F^*, H^*, S^*, M^*) &= \frac{\beta_M + \gamma_{FM}F^* + \gamma_{HM}H^* + \gamma_{SM}S^*}{\delta_M + \gamma_{MF} + \gamma_{MH} + \gamma_{MS}} - M^*
\end{aligned} \tag{3}$$

To find the steady states, I use the system in Equation 3 and plug in the experimentally measured values for the β, δ, and γ parameters. First, we use only the parameters for the environmental transition $D \to P$ and compute the coefficient matrix $C_{D\to P}$ for the state transition from Diestrus to Proestrus states:

$$C_{D\to P} = \begin{bmatrix} 0 & \frac{\gamma_{HF}}{D_F} & \frac{\gamma_{SF}}{D_F} & \frac{\gamma_{MF}}{D_F} \\ \frac{\gamma_{FH}}{D_H} & 0 & \frac{\gamma_{SH}}{D_H} & \frac{\gamma_{MH}}{D_H} \\ \frac{\gamma_{FS}}{D_S} & \frac{\gamma_{HS}}{D_S} & 0 & \frac{\gamma_{MS}}{D_S} \\ \frac{\gamma_{FM}}{D_M} & \frac{\gamma_{HM}}{D_M} & \frac{\gamma_{SM}}{D_M} & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0.1348 & 0.1756 & 0.1027 \\ 0.4091 & 0 & 0.0336 & 0.1482 \\ 0.2617 & 0.2926 & 0 & 0.1531 \\ 0.1180 & 0.1725 & 0.3548 & 0 \end{bmatrix} \tag{4}$$

The vector b which contains all terms that do not depend on the variables F^*, H^*, S^*, or M^* and pushes the system away from zero by only including the formation of new spines, can be written as

$$b = \begin{bmatrix} \frac{\beta_F}{D_F} \\ \frac{\beta_H}{D_H} \\ \frac{\beta_S}{D_S} \\ \frac{\beta_M}{D_M} \end{bmatrix} = \begin{bmatrix} 0.4321 \\ 0.1250 \\ 0.1189 \\ 0.1212 \end{bmatrix} \tag{5}$$

Then, the expression $(I - C)x = b$ can be solved for x as $x = inv(I - C)b$ which gives the steady-state/equalibria at the environmental transition boundaries.

$$\begin{bmatrix} F^* \\ H^* \\ S^* \\ M^* \end{bmatrix}_{D\to P} = \begin{bmatrix} 0.6262 \\ 0.4639 \\ 0.4870 \\ 0.4479 \end{bmatrix}$$

Likewise, for the other stage changes,

$$\begin{bmatrix} F^* \\ H^* \\ S^* \\ M^* \end{bmatrix}_{P\to E} = \begin{bmatrix} 0.5798 \\ 0.4032 \\ 0.4164 \\ 0.3327 \end{bmatrix}, \quad \begin{bmatrix} F^* \\ H^* \\ S^* \\ M^* \end{bmatrix}_{E\to M} = \begin{bmatrix} 0.6019 \\ 0.4573 \\ 0.5840 \\ 0.6210 \end{bmatrix}, \quad \begin{bmatrix} F^* \\ H^* \\ S^* \\ M^* \end{bmatrix}_{M\to P} = \begin{bmatrix} 0.6189 \\ 0.4673 \\ 0.5267 \\ 0.4977 \end{bmatrix}$$

The Jacobian matrix, $J_{ij} = \frac{\partial f_i}{\partial x_j}$ can be computed, where $x_j = \{F^*, H^*, S^*, M^*\}$. For f_1 from Equation 3, the partial derivatives are

$$\frac{\partial f_1}{\partial F^*} = -1 \quad , \quad \frac{\partial f_1}{\partial H^*} = \frac{\gamma_{HF}}{D_F} \quad , \quad \frac{\partial f_1}{\partial S^*} = \frac{\gamma_{SF}}{D_F} \quad , \quad \frac{\partial f_1}{\partial M^*} = \frac{\gamma_{MF}}{D_F}$$

where $D_F = \delta_F + \gamma_{FH} + \gamma_{FS} + \gamma_{FM}$ $D_H = \delta_H + \gamma_{HF} + \gamma_{HS} + \gamma_{HM}$ $D_S = \delta_S + \gamma_{SF} + \gamma_{SH} + \gamma_{SM}$, and $D_M = \delta_M + \gamma_{MF} + \gamma_{MH} + \gamma_{MS}$. The same can be done for all four equations to create the Jacobian matrix

$$J = \begin{bmatrix} -1 & \frac{\gamma_{HF}}{D_F} & \frac{\gamma_{SF}}{D_F} & \frac{\gamma_{MF}}{D_F} \\ \frac{\gamma_{FH}}{D_H} & -1 & \frac{\gamma_{SH}}{D_H} & \frac{\gamma_{MH}}{D_H} \\ \frac{\gamma_{FS}}{D_S} & \frac{\gamma_{HS}}{D_S} & -1 & \frac{\gamma_{MS}}{D_S} \\ \frac{\gamma_{FM}}{D_M} & \frac{\gamma_{HM}}{D_M} & \frac{\gamma_{SM}}{D_M} & -1 \end{bmatrix}$$

Using the equations for the $D \to P$ transition, I find

$$J_{D\to P} = \begin{bmatrix} -1 & 0.3157 & 0.0729 & 0.1700 \\ 0.2751 & -1 & 0.0579 & 0.1231 \\ 2.645 & 0.8034 & -1 & 0.7493 \\ 0.2133 & 0.2297 & 0.0328 & -1 \end{bmatrix}$$

The eigenvalues λ (such that $det(J - \lambda I) = 0$) and eigenvectors v (such that $(J - \lambda I)v = 0$) for the Jacobian $J_{D\to P}$ are

$$\lambda_{D\to P} = \begin{bmatrix} -0.2313 \\ -1.415 \\ -1.226 \\ -1.128 \end{bmatrix}, \quad v_{D\to P} = \left\{ \begin{bmatrix} -0.1889 \\ -0.1625 \\ -0.958 \\ -0.1419 \end{bmatrix}, \begin{bmatrix} -0.1472 \\ -0.0473 \\ 0.9877 \\ 0.0238 \end{bmatrix}, \begin{bmatrix} -0.0501 \\ -0.2914 \\ 0.9323 \\ 0.2082 \end{bmatrix}, \begin{bmatrix} -0.1212 \\ -0.2483 \\ -0.5499 \\ 0.7882 \end{bmatrix} \right\}$$

For the three other transitions,

$$\lambda_{P\to E} = \begin{bmatrix} -0.1612 \\ -1.4972 \\ -1.1089 \\ -1.2327 \end{bmatrix}, \quad v_{P\to E} = \left\{ \begin{bmatrix} -0.0855 \\ -0.0866 \\ -0.9921 \\ -0.0293 \end{bmatrix}, \begin{bmatrix} -0.086 \\ 0.0311 \\ 0.9958 \\ -0.0047 \end{bmatrix}, \begin{bmatrix} 0.048 \\ -0.1165 \\ 0.9909 \\ -0.0473 \end{bmatrix}, \begin{bmatrix} 0.1001 \\ -0.3554 \\ -0.9232 \\ 0.1069 \end{bmatrix} \right\}$$

$$\lambda_{E\to M} = \begin{bmatrix} -0.2059 \\ -1.4901 \\ -1.108 \\ -1.196 \end{bmatrix}, \quad v_{E\to M} = \left\{ \begin{bmatrix} 0.1373 \\ 0.1340 \\ 0.9773 \\ 0.0900 \end{bmatrix}, \begin{bmatrix} 0.1298 \\ 0.0206 \\ -0.9909 \\ -0.0275 \end{bmatrix}, \begin{bmatrix} 0.0479 \\ 0.1478 \\ 0.8599 \\ -0.4862 \end{bmatrix}, \begin{bmatrix} 0.0748 \\ -0.6114 \\ 0.6711 \\ 0.4127 \end{bmatrix} \right\}$$

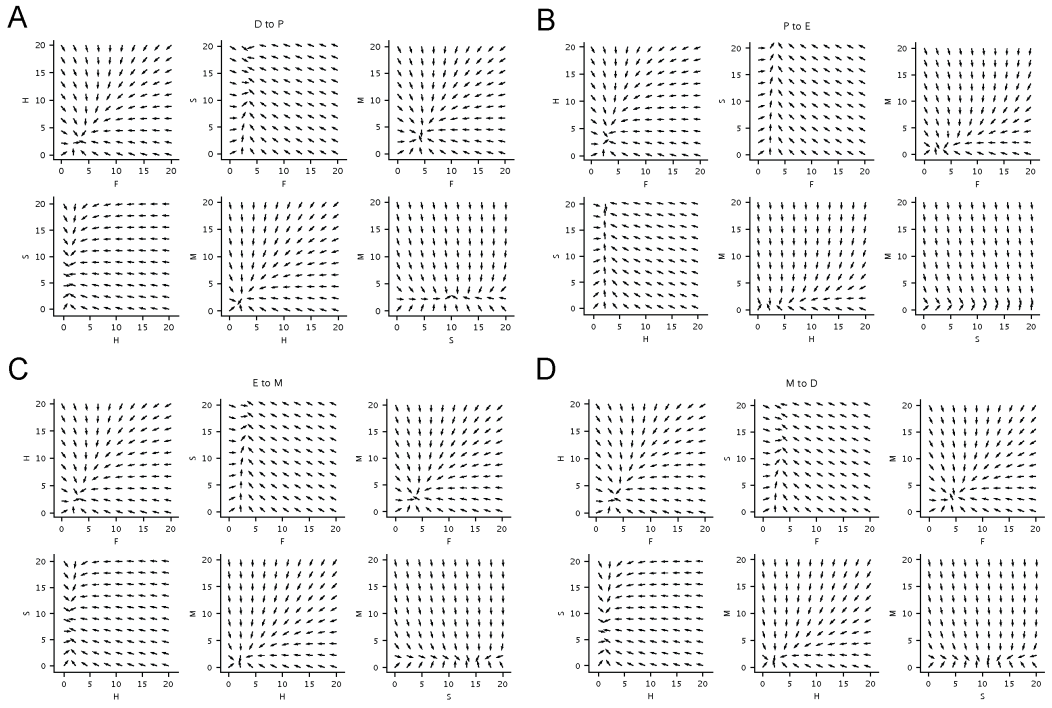

Figure 4: Phase portrait of system described by Equation 2 for the Diestrus to Proestrus (A), Proestrus to Estrus (B), Estrus to Metestrus (C), and Metestrus to Diestrus (D) estrous stage changes.

$$\lambda_{M\to D} = \begin{bmatrix} -0.1910 \\ -1.5116 \\ -1.1082 \\ -1.1891 \end{bmatrix}, \quad v_{M\to D} = \left\{ \begin{bmatrix} 0.1855 \\ 0.1590 \\ 0.9640 \\ 0.1047 \end{bmatrix}, \begin{bmatrix} 0.1815 \\ -0.0272 \\ -0.983 \\ -0.0073 \end{bmatrix}, \begin{bmatrix} 0.0708 \\ -0.0815 \\ 0.9575 \\ -0.2675 \end{bmatrix}, \begin{bmatrix} 0.0368 \\ -0.5339 \\ 0.8110 \\ 0.2364 \end{bmatrix} \right\}$$

For all four transitions, the real part of the eigenvalues are negative. Therefore the equilibria are stable such that small perturbations decay over time and the system will return to its equilibrium. The vector fields of the system are shown in Figure 4.

Spine dynamics as a system of stochastic equations

Considering that the structural imaging from which the formation, pruning, and transition probability of spines was measured from just a small patch of the dendritic arbor of a small subset of CA1 neurons with small population sizes, it is best to implement the model as a stochastic system. Equation 2 can be reformulated for the 20 possible events in Table 1 as a stochastic birth-death-transition process

$$\begin{aligned}
\frac{dp_n^F}{dt} &= k_1 p_{n-1}^F + k_9 p_{n-1}^F + k_{10} p_{n-1}^F + k_{11} p_{n-1}^F - (k_5 + k_{12} + k_{13} + k_{14}) p_n^F \\
\frac{dp_n^H}{dt} &= k_2 p_{n-1}^H + k_{12} p_{n-1}^H + k_{15} p_{n-1}^H + k_{16} p_{n-1}^H - (k_6 + k_9 + k_{17} + k_{19}) p_n^H \\
\frac{dp_n^S}{dt} &= k_3 p_{n-1}^S + k_{13} p_{n-1}^S + k_{17} p_{n-1}^S + k_{18} p_{n-1}^S - (k_7 + k_{10} + k_{15} + k_{20}) p_n^S \\
\frac{dp_n^M}{dt} &= k_4 p_{n-1}^M + k_{14} p_{n-1}^M + k_{19} p_{n-1}^M + k_{20} p_{n-1}^M - (k_8 + k_{11} + k_{16} + k_{18}) p_n^M
\end{aligned} \tag{6}$$

where, for example, the probability of having n spines of type F at time t is given by $p_n^F(t)$.

The system in Equation 6 affects the state vector

$$X(t) = \begin{bmatrix} F(t) \\ H(t) \\ S(t) \\ M(t) \end{bmatrix}$$

The master equation, $P(n_F, n_H, n_s, n_m, t)$, takes the form

$$\frac{dP(\vec{n}, t)}{dt} = \sum_{\vec{n}'} W(\vec{n}|\vec{n}')P(\vec{n}', t) - W(\vec{n}'|\vec{n})P(\vec{n}', t) \tag{7}$$

where $\vec{n} = \begin{bmatrix} n_F & n_H & n_s & n_m \end{bmatrix}$, $W(\vec{n}|\vec{n}')$ represents transition from class $\vec{n}'$ to class $\vec{n}$, and $W(\vec{n}'|\vec{n})$ represents transition from class $\vec{n}$ to class $\vec{n}'$.

Table 1: Stochastic parameters for Equation 6.

#	Event	Definition	Vector	#	Event	Definition	Vector
1	New F grown	$k_1 = \beta_F$	$v_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$	11	M transitions to F	$k_{11} = \gamma_{MF} M$	$v_{11} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ -1 \end{bmatrix}$
2	New H grown	$k_2 = \beta_H$	$v_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$	12	F transitions to H	$k_{12} = \gamma_{FH} F$	$v_{12} = \begin{bmatrix} -1 \\ 1 \\ 0 \\ 0 \end{bmatrix}$
3	New S grown	$k_3 = \beta_S$	$v_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$	13	F transitions to S	$k_{13} = \gamma_{FS} F$	$v_{13} = \begin{bmatrix} -1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$
4	New M grown	$k_4 = \beta_M$	$v_4 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$	14	F transitions to M	$k_{14} = \gamma_{FM} F$	$v_{14} = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$
5	Existing F pruned	$k_5 = \delta_F F$	$v_5 = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$	15	S transitions to H	$k_{15} = \gamma_{SH} S$	$v_{15} = \begin{bmatrix} 0 \\ 1 \\ -1 \\ 0 \end{bmatrix}$
6	Existing H pruned	$k_6 = \delta_H H$	$v_6 = \begin{bmatrix} 0 \\ -1 \\ 0 \\ 0 \end{bmatrix}$	16	M transitions to H	$k_{16} = \gamma_{MH} M$	$v_{16} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ -1 \end{bmatrix}$
7	Existing S pruned	$k_7 = \delta_S S$	$v_7 = \begin{bmatrix} 0 \\ 0 \\ -1 \\ 0 \end{bmatrix}$	17	H transitions to S	$k_{17} = \gamma_{HS} H$	$v_{17} = \begin{bmatrix} 0 \\ -1 \\ 1 \\ 0 \end{bmatrix}$
8	Existing M pruned	$k_8 = \delta_M M$	$v_8 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -1 \end{bmatrix}$	18	M transitions to S	$k_{18} = \gamma_{MS} M$	$v_{18} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ -1 \end{bmatrix}$
9	H transitions to F	$k_9 = \gamma_{HF} H$	$v_9 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0 \end{bmatrix}$	19	H transitions to M	$k_{19} = \gamma_{HM} H$	$v_{19} = \begin{bmatrix} 0 \\ -1 \\ 0 \\ 1 \end{bmatrix}$
10	S transitions to F	$k_{10} = \gamma_{SF} S$	$v_{10} = \begin{bmatrix} 1 \\ 0 \\ -1 \\ 0 \end{bmatrix}$	20	S transitions to M	$k_{20} = \gamma_{SM} S$	$v_20 = \begin{bmatrix} 0 \\ 0 \\ -1 \\ 1 \end{bmatrix}$

Implementation using nonhomogenous Gillespie's stochastic simulation algorithm

The model was implemented as a modified form of Gillespie's stochastic simulation algorithm [8] which allows the model to account for the effects of both demographic stochasticity and environmental variability through time [9]. In this case, the environmental variable is the endogenous endocrine factor levels of the animal at each timepoint, where the combination of endocrine factor concentrations, $[es]$, $[pr]$, $[lh]$, and $[fs]$, are used to determine a probability, $\hat{y}$, for each spine class formation, pruning, and transition term using the GLM model in Equation 1. Then, Equation 1's output transition terms are used in the approach of [9], termed "stochastic simulation algorithm plus," (SSA+):

1. Set intensities of all demographic processes (i.e., the probability the processes occur, per unit time), the effects of those processes (e.g., birth=+1 individual), the starting population sizes, and the end time of the simulation.
2. Generate a random number u from a uniform distribution over the interval $[0, 1]$.
3. Find the value of x which solves $f(x) = u$, where f is the cumulative distribution $f(t) = 1 - exp\big(-\int_0^t \lambda(T+\tau)d\tau\big)$ for the nonhomogenous Poisson process with the rate function $\lambda(t) = \alpha t^{-\beta}$ (i.e., a power law function).
4. Set inter-event time τ to x.
5. Determine which demographic event occurs at $t + \tau$ by sampling from the list of possible processes.
6. Update the time based on Steps #2-#4 and update population sizes based on Step #5. Then return to #Step 2 until all demographic rates are zero or the end time has been reached.

Steps #2-#4 are the major difference between SSA (homogenous) and SSA+ (nonhomogeneous), and can be accomplished by searching an interval (generally 0 to the maximum time of a simulation) for the root (zero) of the equation $f(x) - u$ with respect to x. I implemented this approach in Python based on publicly available R code from [9].

I used initial conditions which occurred in one of the experimental recordings at the start of Diestrus (i.e., the population immediately following an $M \rightarrow D$ transition), where $init = \begin{bmatrix}1 & 6 & 23 & 2\end{bmatrix}$. The output of 100 simulations, all beginning with the same initial conditions, is shown in Figure 5. Increases in spine growth are expected immediately before ovulation (in proestrus stage), and increases in pruning are expected immediately following ovulation (in estrus stage).

Discussion

The first 2 days of the simulation see very few changes in spine populations, and I have no idea why this happened. At the start of the SSA+ simulation's changes (after those 2 days), there is a substantial increase in mushroom spines and a decrease in stubby spines, this appears to sudden and transient surge in $\gamma_{S \rightarrow M}$, after which the S population appears to stabilize. The simulation begins at the metestrus-to-diestrus stage change, and the initial values used are from an animal at this stage, so it is unclear to me why an immediate major change is needed to reach a more steady state. The simulation is very slow to run (running N=100 takes 3 hours), so I haven't tried a range of initial conditions. That's something I'd like to try next.

The SSA+ algorithm, while improved from the version I presented in class last week, is not capturing the dynamics of the real recordings very convincingly. I've tried some other next steps, including calculating the percent spine turnover per 10 µm dendrite length so that I can compare the turnover in the simulation population to that of the real recordings. I've also considered interpolating/resampling the data so that every estrous cycle stage lasts exactly 1 day (i.e., two recording samples) even if the estrous stage cytology determined that it lasted shorter or longer. Doing so on the data before measuring the transition probabilities may improve the simulation's fidelity to the real spine populations over time.

I think this model could address many outstanding questions in modulation of in vivo spine

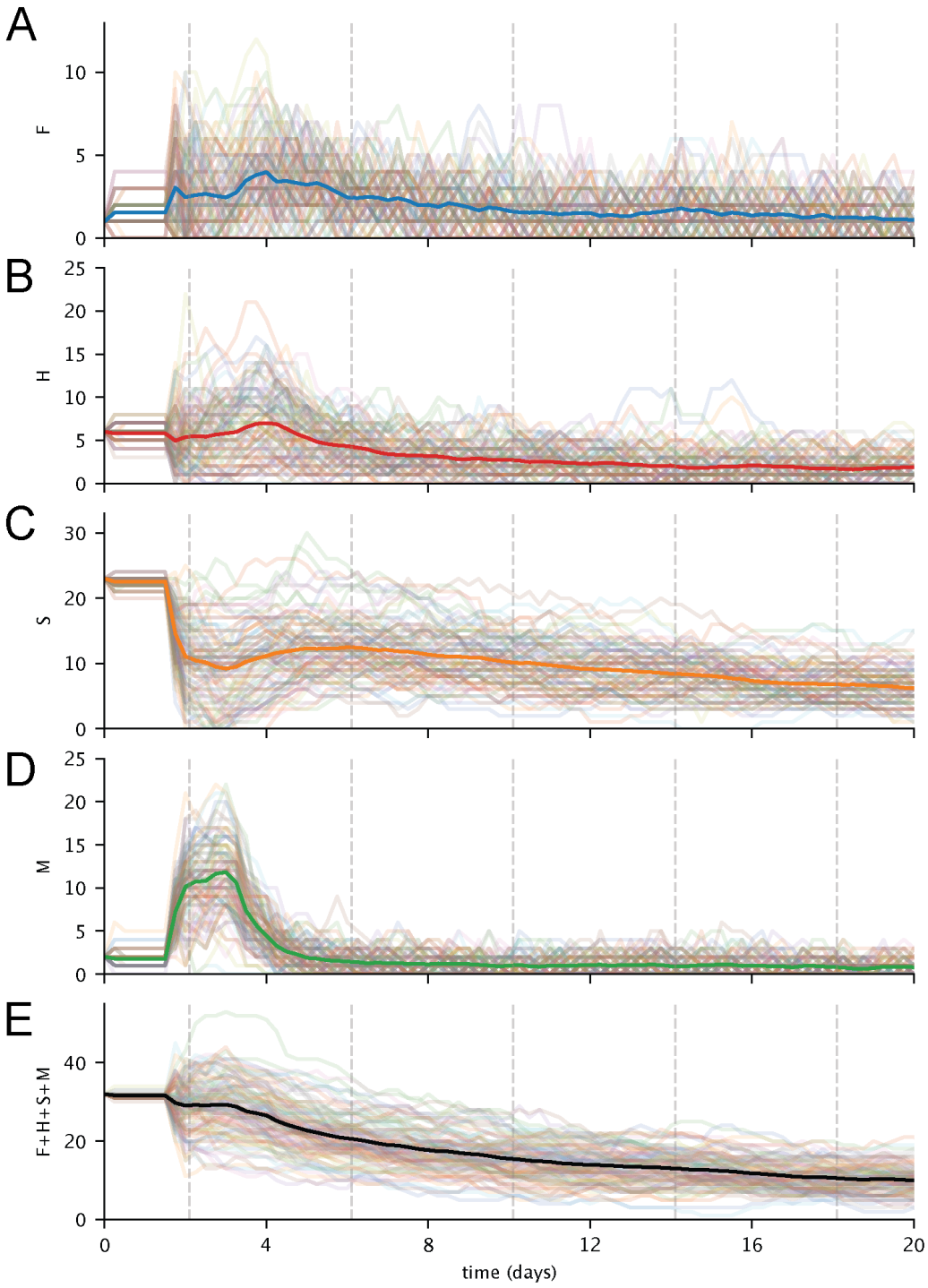

Figure 5: SSA+ populations at each timepoint ($dt = 0.25$ days) for filopodium (A), thin (B), stubby (C), and mushroom (D) spines. (E) Sum of the population, i.e., $F + H + S + M$ at each time point. Each trace within a panel shows the individual simulations (N=100) with identical initial conditions. The thick colored lines indicate the mean population size across simulations. Vertical dashed lines indicate the time of ovulation.

turnover by endocrine factors. Which endocrine factors are most important in modulating the hippocampal spine turnover? If I fix the concentration of controlled combinations of the four endocrine factors used by the model, I can determine which endocrine factors are necessary or sufficient to modulate spine turnover. The model could be used to make testable predictions, for example, if I set the alter the endocrine factors functions of this model to mimic particular animal life stages, for example, endocrine factor levels during pregnancy, at the onset of menopause, etc. While pregnancy or cessation of the estrous cycle are difficult experiments to perform with animal models, they are achievable, and this modeling approach could be a useful tool for making testable predictions before engaging in real experiments. Done alongside the experiments, it could be a useful way to interrogate mechanisms that underly the experimental results – if the spine fluctuation in pregnant mice do not match the model's prediction, what changes to the model make it congruent with those measures of pregnant spine turnover?

The consensus is that while turnover is modulated by female endocrine factors in a way that is not observed in male animals, overall spine turnover is not higher or lower in female compared to male animals. Rather, the overall rate of turnover is about equal, but the female rate of turnover is time-locked to endocrine factor changes in this very striking pattern. There may be questions related to male hormonal fluctuations that are made accessible using this model, although I'm not sure what they would be since the dynamics of that system are less interesting.

Complete Python code All of my custom Python code is included below, but it's also in a github repository here: https://github.com/dylanmmartins/spine-population-modeling.

The analysis uses a portion of the dataset from [4], which is publicly available here: https://data.mendeley.com/datasets/fwd9b5jv9h/1. These data were converted from the available ".mat" files to ".h5" files as in:

```
import os
import numpy as np
from scipy.io import loadmat
import pandas as pd

def convert_data(datadir, savedir):

    list_of_dendrites = sorted([f for f in os.listdir(datadir) if 'mat' in f])

    for f in list_of_dendrites:

        print('Converting {}'.format(f))

        d = loadmat(os.path.join(datadir, f))
        mouse_id = str(d['spine_data'][0][0][0][0])
        dendrite_id = int(d['spine_data'][0][0][1])
        stages = np.array([str(i[0]) for i in d['spine_data'][0][0][2][0]])

        spine_data = np.zeros([
            np.size(d['spine_data'][0][0][5], 0),
            np.size(d['spine_data'][0][0][5], 1)
        ])

        spine_data = spine_data.astype(str)
        for day in range(np.size(d['spine_data'][0][0][5], 0)):
            for spine_id in range(np.size(d['spine_data'][0][0][5], 1)):
                try:
                    spinetype = d['spine_data'][0][0][5][day,spine_id][0][0][0][0]
                    while type(spinetype) == np.ndarray:
                        spinetype = spinetype[0]
                except:
                    spinetype = 'NS'
```

```
                spine_data[day, spine_id] = str(spinetype)

        df = pd.DataFrame(spine_data)
        df.columns = ['spine_{:02}'.format(n) for n in range(len(df.columns.values))]

        # Drop last day for the only recording that a day was removed from spine
        # data but not from cycle data.
        if (len(stages)>len(df.index.values)) and (mouse_id=='WTR042') and (dendrite_id==4):
            stages = stages[:-1]

        df['hrs'] = np.arange(0, 12*len(df.index.values), 12)
        df['stage'] = stages
        df.attrs = {
            'mouse': mouse_id,
            'dendrite': dendrite_id-1
        }

        df.to_hdf(
            os.path.join(savedir, '{}.h5'.format(os.path.splitext(f)[0])),
            key='key'
        )

if __name__ == '__main__':

    datadir = '/Users/dmartins/Dropbox/spine_modeling/matlab_data'
    savedir = '/Users/dmartins/Dropbox/spine_modeling/clean_data'
    convert_data(datadir, savedir)
```

Next, the transition matrix at each estrous cycle edge was calculated from the ".h5" files.

```
import pandas as pd
import os
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
mpl.rcParams['axes.spines.top'] = False
mpl.rcParams['axes.spines.right'] = False
mpl.rcParams['pdf.fonttype'] = 42
mpl.rcParams['ps.fonttype'] = 42
mpl.rcParams['svg.fonttype'] = 'none'

def calculate_transition_matrix(file_paths, f, t):
    # file_paths is the list of hdf files
    # f is the 'from' state
    # t is the 'to' state

    if isinstance(file_paths, str):
        file_paths = [file_paths]

    # Initialize an empty list to accumulate all the transitions across files
    all_transitions = []

    # Iterate through each file
    for file_path in file_paths:
        # Read the DataFrame from the file
        df = pd.read_hdf(file_path)
        df = df.replace('filopodia', 'filopodium')

        # Ensure the 'stage' column exists
        if 'stage' not in df.columns:
            raise ValueError(f"'stage' column is missing in {file_path}.")

        # Iterate through the rows of the dataframe and check for f -> t transitions
```

```
        for i in range(0, len(df)-1):
            # Look for transitions from 'D' to 'P'
            if df.iloc[i]['stage'] == f and df.iloc[i+1]['stage'] == t:
                # Collect the string values for transition
                for col in [x for x in df.columns.values if x!='hrs' or x!='stage']:
                    transition_pair = (df.iloc[i][col], df.iloc[i+1][col])
                    all_transitions.append(transition_pair)

    state_options = ['NS','filopodium','thin','stubby','mushroom']
    state_to_idx = {s: i for i, s in enumerate(state_options)}

    # Initialize count matrix
    count_matrix = np.zeros((5, 5), dtype=np.float64)

    # Fill the count matrix
    for from_state, to_state in all_transitions:
        # Skip any transitions that do not match expected naming
        if (from_state not in state_options) or (to_state not in state_options):
            continue
        i = state_to_idx[from_state]
        j = state_to_idx[to_state]
        count_matrix[i, j] += 1

    # Normalize rows to get transition probabilities
    row_sums = count_matrix.sum(axis=1, keepdims=True)
    transition_matrix = np.divide(count_matrix, row_sums, where=row_sums != 0)

    return transition_matrix

def main():

    base_path = '/Users/dmartins/Documents/GitHub/spine-population-modeling/clean_data'
    file_paths = sorted([os.path.join(base_path, x) for x in os.listdir('./clean_data')])
    file_paths = [x for x in file_paths if 'DS_Store' not in x]

    transition_matrix_DtoP = calculate_transition_matrix(file_paths, f='D', t='P')
    transition_matrix_PtoE = calculate_transition_matrix(file_paths, f='P', t='E')
    transition_matrix_EtoM = calculate_transition_matrix(file_paths, f='E', t='M')
    transition_matrix_MtoD = calculate_transition_matrix(file_paths, f='M', t='D')

    transition_matrix_list = [
        transition_matrix_DtoP,
        transition_matrix_PtoE,
        transition_matrix_EtoM,
        transition_matrix_MtoD
    ]

    state_options = ['NS','filopodium','thin','stubby','mushroom']

    title_list = [
        ['Diestrus','Proestrus'],
        ['Proestrus','Estrus'],
        ['Estrus','Metestrus'],
        ['Metestrus','Diestrus']
    ]
    for tind in range(4):

        print('visualizing for {}/4'.format(tind+1))

        name_pair = title_list[tind]

        fig_save_name = '{}to{}_transition.svg'.format(name_pair[0][0], name_pair[1][0])
        mat_save_name = '{}to{}_transition_matrix.npy'.format(name_pair[0][0], name_pair[1][0])
```

```
        transition_matrix = transition_matrix_list[tind]

        fig = plt.figure(figsize=(6,5),dpi=300)
        plt.imshow(
            transition_matrix.T,
            cmap='Blues',
            origin='lower'
        )
        for i in range(transition_matrix.shape[1]):
            for j in range(transition_matrix.shape[0]):
                text = plt.text(
                    j,
                    i,
                    '{:.2}'.format(transition_matrix[j,i]),
                    ha="center",
                    va="center",
                    color="k"
                )
        plt.xticks(range(len(state_options)), labels=state_options)
        plt.yticks(range(len(state_options)), labels=state_options)
        plt.title('{} to {}'.format(name_pair[0], name_pair[1]))
        plt.xlabel('{}'.format(name_pair[0]))
        plt.ylabel('{}'.format(name_pair[1]))
        plt.colorbar(label='transition probability')
        plt.tight_layout()
        fig.savefig(fig_save_name)

        np.save(mat_save_name, transition_matrix.T)

if __name__ == '__main__':
    main()
```

Then the concentration of endocrine factors was calculated for a range of timepoints using the time/concentration data from [7].

```
def non_decreasing(L):
    return all(x<=y for x, y in zip(L, L[1:]))

def non_increasing(L):
    return all(x>=y for x, y in zip(L, L[1:]))

def monotonic(L):
    return non_decreasing(L) or non_increasing(L)

def spline_interp_conc(t, x):
    if np.size(t) != np.size(x):
        print('Sizes not equal')
        return
    if not monotonic(t):
        sortind = np.argsort(t)
        t = t[sortind]
        x = x[sortind]
    spline_interp = scipy.interpolate.CubicSpline(
        t,
        x,
        bc_type='periodic'
    )
    conc = spline_interp(np.arange(0,4.25,0.25))
    return conc

e_times = np.array([0, 1, 1.25, 1.5, 1.75, 2, 2.5, 3, 4])
e_vals = np.array([10, 20, 25, 40, 10, 5, 10, 5, 10])/1000 # convert to ng/mL (from pg/mL)
p_times = np.array([0,0.5,1.25,1.5,1.6,1.75,0.25,2.0,4.0,3.0,3.5,2.5])
```

```
p_vals = np.array([30,10,4,2,3,60,30,5,30,15,30,5])
l_times = np.array([0,1,1.1,0.9,1.5,1.75,2.25,3.0,4.0,2,1.85,1.95])
l_vals = np.array([2,2,2,2,2,40,2,2,2,5,15,10])
f_times = np.array([0,.5,1,1.5,1.75,2.15,2,2.25,3.25,3.75,4,3.5,2.5,2.4,2.3,2.6,2.7])
f_vals = np.array([100,20,20,20,375,375,375,20,20,20,100,100,20,20,20,20,20])
print(
    e_times.shape, e_vals.shape, p_times.shape,
    p_vals.shape, l_times.shape, l_vals.shape,
    f_times.shape, f_vals.shape)

estradiol = spline_interp_conc(e_times, e_vals)
progesterone = spline_interp_conc(p_times, p_vals)
luteinizing = spline_interp_conc(l_times, l_vals)
folliclestim = spline_interp_conc(f_times, f_vals)

# X needs to be (samples, features)
X_conc = np.stack([estradiol, progesterone, luteinizing, folliclestim]).T
```

The GLM of Equation 1 was implemented as

```
def fit_glm(X, y):

    n_samples, n_features = X.shape
    assert n_features == 4, "Input must have exactly 4 features per sample."

    # Add bias (intercept) term: shape becomes (n_samples, 5)
    X_aug = np.hstack([np.ones((n_samples, 1)), X])

    # Closed-form solution: w = (X^T X)^(-1) X^T y
    XtX = X_aug.T @ X_aug
    Xty = X_aug.T @ y
    weights = np.linalg.inv(XtX) @ Xty

    return weights

transition_timepoints = np.arange(0, 4)

w = {}
for i in range(np.size(symbol_matrix,0)):
    for j in range(np.size(symbol_matrix,1)):

        y_train = np.zeros(4) * np.nan

        for stage in range(len(mats)):

            param = symbol_matrix[i,j]
            param_at_stage = mats[stage][i,j]

            y_train[stage] = param_at_stage

        symbol_name = 'w_{}'.format(symbol_matrix[i,j])

        w_ = fit_glm(
            X = X_conc[transition_timepoints.astype(int),:],
            y = y_train
        )

        w[symbol_name] = w_
```

The panels for Figure 2 were plotted as

```
fig, [ax1,ax2,ax3,ax4] = plt.subplots(4,1, figsize=(2.75,4), dpi=300)
ax1.plot(np.arange(0,4.25,0.25), X_conc[:,0], color='k')
ax2.plot(np.arange(0,4.25,0.25), X_conc[:,1], color='k')
```

```
ax3.plot(np.arange(0,4.25,0.25), X_conc[:,2], color='k')
ax4.plot(np.arange(0,4.25,0.25), X_conc[:,3], color='k')
for ax in [ax1,ax2,ax3,ax4]:
    ax.set_xlim([0,4])
    ax.vlines([1,2,3], 0, 500, color='k', alpha=0.3, ls='--', lw=1)
ax1.set_ylim([0,0.045])
ax2.set_ylim([0,65])
ax3.set_ylim([0,45])
ax4.set_ylim([0,400])
ax1.set_ylabel('estradiol (ng/mL)')
ax2.set_ylabel('progesterone (ng/mL)')
ax3.set_ylabel('LH (ng/mL)')
ax4.set_ylabel('FSH (ng/mL)')
plt.tight_layout()
plt.savefig('hormone_interp_concentrations_dt0p25.svg')

weight_array = np.zeros([5,25])
for i, k in enumerate(w.keys()):
    weight_array[:,i] = w[k]
def normalize_rows(x):
  norm = np.sqrt(np.sum(x**2, axis=1, keepdims=True))
  return x / norm
plt.imshow(normalize_rows(weight_array).T)
plt.yticks(np.arange(25), labels=symbol_matrix.flatten())
plt.xticks(np.arange(5), labels=['bias','estradiol','prog.','LH','FSH'], rotation=90)
plt.colorbar()
plt.tight_layout()
plt.savefig('glm_weights_colnorm.svg')
```

The functions to calculate the value x which solves the cumulative distribution for the poisson process to determine the inter-event time τ (SSA+ Steps #2-#4) were implemented as:

```
def nhpp(tottime, pops, param, calc_intensity, timeleft, maxT=96):

    Y = random.random()
    def f(X):
        # Integrate sum of intensities from 0 to X
        integral, _ = scipy.integrate.quad(
              lambda x: sum(calc_intensity(tottime+x % maxT, pops, param)),
              0, X, limit=200)
        return 1 - math.exp(-integral) - Y
    try:
        sol = scipy.optimize.root_scalar(f, bracket=[0, timeleft], method='bisect', xtol=1e-5)
        return sol.root
    except ValueError:
        # In case no root is found, return a value greater than timeleft
        return timeleft + 1

def calc_intensity(t, pops, params):
    # Intensity function and demographic functions combined into single func
    # Should return the values for birth*population, etc. as numbers not as rates
    # c == Current hormone concentrations (not including bias term

    alltimes = np.arange(0,4.25,0.25) # units of days
    ind, _ = find_closest_timestamp(alltimes, t)
    c = X_conc[ind,:]

    # augment concentrations by inserting a 1.0 to use full bias term
    c_aug = np.insert(c, 0, 1.0)

    # Current populations
    F, H, S, M = pops

    # Ignore lambda values because these represent stability not a real transition
```

```
    # The population intensities, lambda(t), of the point process are the expected rate
    # of occurrence of events at a particular time t
    ppintens = np.array([
        params['w_betaF'] @ c_aug,              # betaF
        params['w_betaH'] @ c_aug,              # betaH
        params['w_betaS'] @ c_aug,              # betaS
        params['w_betaM'] @ c_aug,              # betaM
        (params['w_deltaF'] @ c_aug) * F,       # deltaF
        (params['w_deltaH'] @ c_aug) * H,       # deltaF
        (params['w_deltaS'] @ c_aug) * S,       # deltaF
        (params['w_deltaM'] @ c_aug) * M,       # deltaF
        (params['w_gammaH2F'] @ c_aug) * H,     # gammaHF
        (params['w_gammaS2F'] @ c_aug) * S,     # gammaSF
        (params['w_gammaM2F'] @ c_aug) * M,     # gammaMF
        (params['w_gammaF2H'] @ c_aug) * F,     # gammaFH
        (params['w_gammaF2S'] @ c_aug) * F,     # gammaFS
        (params['w_gammaF2M'] @ c_aug) * F,     # gammaFM
        (params['w_gammaS2H'] @ c_aug) * S,     # gammaSH
        (params['w_gammaM2H'] @ c_aug) * M,     # gammaMH
        (params['w_gammaH2S'] @ c_aug) * H,     # gammaHS
        (params['w_gammaM2S'] @ c_aug) * M,     # gammaMS
        (params['w_gammaH2M'] @ c_aug) * H,     # gammaHM
        (params['w_gammaS2M'] @ c_aug) * S,     # gammaSM
    ])
    return ppintens
```

and called from the simulation function

```
params = w.copy()

def gillespie_plus(init, times, calc_intensity, nhpp_func):

    pproc = get_state_change_mat()

    tottime = times[0]
    tinc = len(times)
    pops = np.array(init, dtype=float)
    results = np.zeros((tinc, len(pops)))
    results[0, :] = pops.copy()

    i = 1
    while i < tinc:

        results[i, :] = results[i-1, :].copy()

        while tottime <= times[i]:

            tau = nhpp_func(tottime, pops, params, calc_intensity, times[-1]-tottime)
            tottime += tau

            # Recalculate intensities for the new time.
            intentemp = calc_intensity(tottime, pops, params)

            # Handle negative intensities (before normalizing by the sum) by shifting
            # up so the lowest intensity is zero. Once scaled by the sum, they'll still sum
            # to 1.
            if np.nanmin(intentemp) < 0:
                intentemp += - np.nanmin(intentemp)

            probabilities = np.array(intentemp) / np.sum(intentemp)

            _choice = np.arange(pproc.shape[0])
            _probs = probabilities.flatten()
            event_index = np.random.choice(_choice, p=_probs)
```

```
            if tottime > times[i]:
                results[i, :] = pops.copy()
                pops = pops + pproc[event_index, :]
                break
            else:
                pops = pops + pproc[event_index, :]

            pops[pops<0] = 0
        i += 1

    return np.column_stack((times, results))
```

The simulation was called for the initial conditions, run 100 times with those same initial conditions, and outputs were saved as a ".yaml" file.

```
# Initial population sizes
init = [1,6,23,2]

times = np.arange(0, 20.5, 0.25)

# Run simulation using gillespie_plus (nonhomogeneous Poisson process sampler)
res_list = {}
for i in tqdm(range(100)):
    res_gillespie_plus = gillespie_plus(init, times, calc_intensity, nhpp)
    res_list[i] = res_gillespie_plus

savepath = 'res_gillespie_plus_100r_dt_0p25_init_1_6_23_2.yaml'
with open(savepath, 'w') as outfile:
    yaml.dump(res_list, outfile, default_flow_style=False)
```

The results of the simulation were plotted for Figure 5 as

```
def calc_mean_trace(res_list, ind):
    mean_trace = np.zeros_like(res_list[0][:,1])
    for i in range(len(res_list)):
        mean_trace += res_list[i][:,ind] / len(res_list)
    return mean_trace

sum_trace = np.zeros_like(res_list[0][:,1])
fig, [ax1,ax2,ax3,ax4,ax5] = plt.subplots(5, 1, dpi=300, figsize=(5,7))
for i in range(len(res_list)):
    ax1.plot(res_list[i][:,0], res_list[i][:,1], alpha=0.1)
    ax2.plot(res_list[i][:,0], res_list[i][:,2], alpha=0.1)
    ax3.plot(res_list[i][:,0], res_list[i][:,3], alpha=0.1)
    ax4.plot(res_list[i][:,0], res_list[i][:,4], alpha=0.1)
    ax5.plot(
        res_list[0][:,0],
        (res_list[i][:,1]+res_list[i][:,2]+res_list[i][:,3]+res_list[i][:,4]),
        alpha=0.1
    )
    sum_trace+=(res_list[i][:,1]+res_list[i][:,2]+res_list[i][:,3]+res_list[i][:,4])/len(res_list)
ax1.plot(res_list[0][:,0], calc_mean_trace(res_list,1), color='tab:blue')
ax2.plot(res_list[0][:,0], calc_mean_trace(res_list,2), color='tab:red')
ax3.plot(res_list[0][:,0], calc_mean_trace(res_list,3), color='tab:orange')
ax4.plot(res_list[0][:,0], calc_mean_trace(res_list,4), color='tab:green')

for ax in [ax1,ax2,ax3,ax4,ax5]:
    ax.vlines(np.arange(2.1, 20, 4), 0, 60, ls='--', lw=1, alpha=0.3, color='gray')
    ax.set_xlim([0,20])
for ax in [ax1,ax2,ax3,ax4]:
    ax.set_xticks(np.arange(0,24,4), labels=[])
ax1.set_ylim([0,13])
ax2.set_ylim([0,25])
ax3.set_ylim([0,33])
```

```
ax4.set_ylim([0,25])
ax5.set_ylim([0,55])
ax5.plot(res_list[0][:,0], sum_trace, color='k')
ax1.set_ylabel('F')
ax2.set_ylabel('H')
ax3.set_ylabel('S')
ax4.set_ylabel('M')
ax5.set_ylabel('F+H+S+M')
ax5.set_xticks(np.arange(0,24,4))
ax5.set_xlabel('time (days)')
fig.tight_layout()
fig.savefig('sim_results_v2.svg')
```

The values of the coefficient matrix C from Equation 4 and vector b from Equation 5 was calculated as

```
DtoP = np.load('spinePopModels/transition_mats/DtoP_transition_matrix.npy')
PtoE = np.load('spinePopModels/transition_mats/PtoE_transition_matrix.npy')
EtoM = np.load('spinePopModels/transition_mats/EtoM_transition_matrix.npy')
MtoD = np.load('spinePopModels/transition_mats/MtoD_transition_matrix.npy')

def compute_C_from_P(P):

    P = np.array(P, dtype=float)

    # Extract the 4x4 submatrix (rows 0-3, cols 1-4) which contains the gamma
    # parameters (off-diagonal)
    # Diagonal deltas are in P[i, i] (0 to 3)
    # We'll build a 4x4 matrix of gammas, with diagonal zeros.

    A = np.zeros((4,4))

    for i in range(4):
        delta_i = P[i, i]          # diagonal delta
        # gamma params in row i, columns 1 to 4 (index 1 to 4)
        # but only columns 1 to 4, excluding diagonal column i
        # diagonal in terms of gammas: we want to skip P[i, i] but i and column indices don't
        # align because gamma start at col 1
        # So build row gamma vector (cols 1 to 4)
        gamma_row = P[i, 1:5]  # length 4

        # The diagonal of A corresponds to gamma_row element where col = i
        # Because for i=0 -> skip gamma_row[0], i=1 -> skip gamma_row[1], etc.
        denom = delta_i + np.sum(np.delete(gamma_row, i))

        for j in range(4):
            if j == i:
                A[i,j] = 0.0
            else:
                A[i,j] = gamma_row[j] / denom

    beta_F = P[4, 1]
    beta_H = P[4, 2]
    beta_S = P[4, 3]
    beta_M = P[4, 4]

    # Order: [M, S, H, F]
    B = np.array([beta_M, beta_S, beta_H, beta_F])

    return A, B

A_DP, B_DP = compute_C_from_P(DtoP)
A_PE, B_PE = compute_C_from_P(PtoE)
A_EM, B_EM = compute_C_from_P(EtoM)
```

```
A_MD, B_DM = compute_C_from_P(MtoD)
```

the equalibrium values were calculated as

```
ABs = [[A_DP, B_DP], [A_PE, B_PE], [A_EM, B_EM], [A_MD, B_DM]]
for i in range(4):
    A, B = ABs[i]
    x = np.linalg.solve((np.eye(4)-A), B)
    print(i+1, x)
```

and the Jacobians $J_{s_1 \to s_2}$ and associated eigenvales and eigenvectors were calculated as

```
def compute_jacobian(P):
    # Unpack parameters from matrix P
    delta_M = P[0, 0]
    gamma_M_to_F = P[0, 1]
    gamma_M_to_H = P[0, 2]
    gamma_M_to_S = P[0, 3]

    delta_S = P[1, 0]
    gamma_S_to_F = P[1, 1]
    gamma_S_to_H = P[1, 2]
    gamma_S_to_M = P[1, 4]

    delta_H = P[2, 0]
    gamma_H_to_F = P[2, 1]
    gamma_H_to_S = P[2, 3]
    gamma_H_to_M = P[2, 4]

    delta_F = P[3, 0]
    gamma_F_to_H = P[3, 2]
    gamma_F_to_S = P[3, 3]
    gamma_F_to_M = P[3, 4]

    # Denominator terms for each row
    D_F = delta_F + gamma_F_to_H + gamma_F_to_S + gamma_F_to_M
    D_H = delta_H + gamma_H_to_F + gamma_H_to_S + gamma_H_to_M
    D_S = delta_S + gamma_S_to_F + gamma_S_to_H + gamma_S_to_M
    D_M = delta_M + gamma_M_to_F + gamma_M_to_H + gamma_M_to_S

    # Build Jacobian matrix
    J = np.array([
        [-1,gamma_H_to_F / D_F, gamma_S_to_F / D_F, gamma_M_to_F / D_F],#dF/d*
        [gamma_F_to_H / D_H, -1,gamma_S_to_H / D_H, gamma_M_to_H / D_H],#dH/d*
        [gamma_F_to_S / D_S, gamma_H_to_S / D_S, -1, gamma_M_to_S / D_S],#dS/d*
        [gamma_F_to_M / D_M, gamma_H_to_M / D_M, gamma_S_to_M / D_M, -1]# dM/d*
    ])

    return J

DtoP = np.load('spinePopModels/transition_mats/DtoP_transition_matrix.npy')
PtoE = np.load('spinePopModels/transition_mats/PtoE_transition_matrix.npy')
EtoM = np.load('spinePopModels/transition_mats/EtoM_transition_matrix.npy')
MtoD = np.load('spinePopModels/transition_mats/MtoD_transition_matrix.npy')

for i,mat in enumerate([DtoP,PtoE,EtoM,MtoD]):
    J = compute_jacobian(mat)
    print(i+1)
    print()J)
    eigenvalues, eigenvectors = np.linalg.eig(J)
    print('vals:')
    print(eigenvalues.round(4))
    print('vecs:')
    for i in range(4):
```

```
        print(eigenvectors[:,i].round(4))
```

The phase portraits in Figure 4 were calculated as

```
def plot_phase_portrait(P, init_conditions, plot_vars=['F', 'H'], ax=None):
    # Extract parameters from P matrix
    delta_M, gMF, gMH, gMS = P[0, 0], P[0, 1], P[0, 2], P[0, 3]
    delta_S, gSF, gSH, gSM = P[1, 0], P[1, 1], P[1, 2], P[1, 4]
    delta_H, gHF, gHS, gHM = P[2, 0], P[2, 1], P[2, 3], P[2, 4]
    delta_F, gFH, gFS, gFM = P[3, 0], P[3, 2], P[3, 3], P[3, 4]
    beta_F, beta_H, beta_S, beta_M = P[4, 1], P[4, 2], P[4, 3], P[4, 4]

    # Variable mapping
    var_idx = {'F': 0, 'H': 1, 'S': 2, 'M': 3}
    all_vars = ['F', 'H', 'S', 'M']

    # Determine free and fixed variables
    free_vars = plot_vars
    fixed_vars = [v for v in all_vars if v not in free_vars]

    # Generate grid for free variables
    grid_size = 10
    v1_vals = np.linspace(0, 20, grid_size)
    v2_vals = np.linspace(0, 20, grid_size)
    V1, V2 = np.meshgrid(v1_vals, v2_vals)

    # Initialize derivative arrays
    dV1 = np.zeros_like(V1)
    dV2 = np.zeros_like(V2)

    # Loop over grid points
    for i in range(grid_size):
        for j in range(grid_size):
            # Initialize full state vector from initial conditions
            state = init_conditions.copy()
            state[var_idx[free_vars[0]]] = V1[i, j]
            state[var_idx[free_vars[1]]] = V2[i, j]

            F, H, S, M = state

            # Compute time derivatives (df/dt, etc.)
            dF = beta_F - delta_F * F + gHF * H + gSF * S + gMF * M - F * (gFH + gFS + gFM)
            dH = beta_H - delta_H * H + gFH * F + gSH * S + gMH * M - H * (gHF + gHS + gHM)
            dS = beta_S - delta_S * S + gFS * F + gHS * H + gMS * M - S * (gSF + gSH + gSM)
            dM = beta_M - delta_M * M + gFM * F + gHM * H + gSM * S - M * (gMF + gMH + gMS)

            derivatives = [dF, dH, dS, dM]

            dV1[i, j] = derivatives[var_idx[free_vars[0]]]
            dV2[i, j] = derivatives[var_idx[free_vars[1]]]

    # Normalize vectors for quiver plot
    magnitude = np.sqrt(dV1**2 + dV2**2)
    dV1 /= (magnitude + 1e-8)
    dV2 /= (magnitude + 1e-8)

    J = compute_jacobian(P)
    eigvals, eigvecs = np.linalg.eig(J)

    if ax is None:
        fig, ax = plt.subplots(1, 1, dpi=300, figsize=(3,3))

    ax.quiver(V1, V2, dV1, dV2, angles='xy')
    ax.set_xlabel(free_vars[0])
```

```
    ax.set_ylabel(free_vars[1])
    ax.set_xlim([0,20])
    ax.set_ylim([0,20])
    ax.axis('equal')
    plt.tight_layout()

fig, [[ax1,ax2,ax3],[ax4,ax5,ax6]] = plt.subplots(2, 3, figsize=(6,4), dpi=300)

plot_phase_portrait(DtoP, [1,6,23,2], ['F','H'], ax=ax1)
plot_phase_portrait(DtoP, [1,6,23,2], ['F','S'], ax=ax2)
plot_phase_portrait(DtoP, [1,6,23,2], ['F','M'], ax=ax3)
plot_phase_portrait(DtoP, [1,6,23,2], ['H','S'], ax=ax4)
plot_phase_portrait(DtoP, [1,6,23,2], ['H','M'], ax=ax5)
plot_phase_portrait(DtoP, [1,6,23,2], ['S','M'], ax=ax6)

fig.suptitle('D to P')
fig.tight_layout()
fig.savefig('DtoP_vecfields.svg')

fig, [[ax1,ax2,ax3],[ax4,ax5,ax6]] = plt.subplots(2, 3, figsize=(6,4), dpi=300)

plot_phase_portrait(PtoE, [1,6,23,2], ['F','H'], ax=ax1)
plot_phase_portrait(PtoE, [1,6,23,2], ['F','S'], ax=ax2)
plot_phase_portrait(PtoE, [1,6,23,2], ['F','M'], ax=ax3)
plot_phase_portrait(PtoE, [1,6,23,2], ['H','S'], ax=ax4)
plot_phase_portrait(PtoE, [1,6,23,2], ['H','M'], ax=ax5)
plot_phase_portrait(PtoE, [1,6,23,2], ['S','M'], ax=ax6)

fig.suptitle('P to E')
fig.tight_layout()
fig.savefig('PtoE_vecfields.svg')

fig, [[ax1,ax2,ax3],[ax4,ax5,ax6]] = plt.subplots(2, 3, figsize=(6,4), dpi=300)

plot_phase_portrait(EtoM, [1,6,23,2], ['F','H'], ax=ax1)
plot_phase_portrait(EtoM, [1,6,23,2], ['F','S'], ax=ax2)
plot_phase_portrait(EtoM, [1,6,23,2], ['F','M'], ax=ax3)
plot_phase_portrait(EtoM, [1,6,23,2], ['H','S'], ax=ax4)
plot_phase_portrait(EtoM, [1,6,23,2], ['H','M'], ax=ax5)
plot_phase_portrait(EtoM, [1,6,23,2], ['S','M'], ax=ax6)

fig.suptitle('E to M')
fig.tight_layout()
fig.savefig('EtoM_vecfields.svg')

fig, [[ax1,ax2,ax3],[ax4,ax5,ax6]] = plt.subplots(2, 3, figsize=(6,4), dpi=300)

plot_phase_portrait(MtoD, [1,6,23,2], ['F','H'], ax=ax1)
plot_phase_portrait(MtoD, [1,6,23,2], ['F','S'], ax=ax2)
plot_phase_portrait(MtoD, [1,6,23,2], ['F','M'], ax=ax3)
plot_phase_portrait(MtoD, [1,6,23,2], ['H','S'], ax=ax4)
plot_phase_portrait(MtoD, [1,6,23,2], ['H','M'], ax=ax5)
plot_phase_portrait(MtoD, [1,6,23,2], ['S','M'], ax=ax6)

fig.suptitle('M to D')
fig.tight_layout()
fig.savefig('MtoD_vecfields.svg')
```

Finally, the transition probabilities as a function of hormone concentration, visualized in Figure 3, was created as

```
def plot_transition_vs_E2(matrices, x_data=None, y_data=None, y_label='estradiol (ng/mL)'):

    num_values = 100
```

```
assert len(matrices) == 4, "Input must be a list of four 5x5 matrices"
for mat in matrices:
    assert mat.shape == (5, 5), "All matrices must be 5x5"

fig, axes = plt.subplots(5, 5, figsize=(6,5), dpi=300)#, sharex=True, sharey=True)

if x_data is None:
    x_data = np.array([0, 1, 1.25, 1.5, 1.75, 2, 2.5, 3, 4])
if y_data is None:
    y_data = np.array([10, 20, 25, 40, 10, 5, 10, 5, 10])
estradiol_level = spline_interp_conc(x_data, np.linspace(0,4,num_values), y_data)
timepoints_f = np.linspace(0,4,num_values)

plt.figure()
plt.plot(timepoints_f, estradiol_level)
plt.ylabel(y_label)

symbol_matrix = np.array([
    [r'$\delta_M$', r'$\gamma_{M \to F}$', r'$\gamma_{M \to H}$',
          r'$\gamma_{M \to S}$', r'$-\alpha_M$'],
    [r'$\delta_S$', r'$\gamma_{S \to F}$', r'$\gamma_{S \to H}$',
          r'$-\alpha_S$', r'$\gamma_{S \to M}$'],
    [r'$\delta_H$', r'$\gamma_{H \to F}$', r'$-\alpha_H$',
          r'$\gamma_{H \to S}$', r'$\gamma_{H \to M}$'],
    [r'$\delta_F$', r'$-\alpha_F$', r'$\gamma_{F \to H}$',
          r'$\gamma_{F \to S}$', r'$\gamma_{F \to M}$'],
    [r'$0$', r'$\beta_F$', r'$\beta_H$', r'$\beta_S$', r'$\beta_M$']
])

new_colors = cm.get_cmap('hsv')(np.linspace(0, 1, num_values))
for i in range(5):
    for j in range(5):
        values = [mat[i, j] for mat in matrices]
        ax = axes[i,j]

        # linear interpolation
        # fit_over_finer_grid = np.interp(timepoints_f, np.arange(4)*24, values)

        # circular interpolation
        x_ = np.arange(5)
        y_ = np.append(values, values[0])
        spline_interp = scipy.interpolate.CubicSpline(x_, y_, bc_type="periodic")
        fit_over_finer_grid = spline_interp(timepoints_f)
        fit_over_finer_grid[fit_over_finer_grid<0] = 0

        ax.scatter(
            estradiol_level[np.arange(num_values)],
            fit_over_finer_grid,
            s=2.5, color=new_colors)
        ax.set_title(symbol_matrix[i,j])
        ax.set_ylabel('')
        ax.set_xlabel('')
        if 'delta' in symbol_matrix[i,j] or symbol_matrix[i,j]=='$0$':
            ax.set_ylabel('Prob')
        if 'beta' in symbol_matrix[i,j] or symbol_matrix[i,j]=='$0$':
            ax.set_xlabel(y_label)

fig.tight_layout()
fig.show()
savename = 'probability_evolution_vs_{}.svg'.format(y_label.split(' ')[0])
print(savename)
fig.savefig(savename)
```

```
e_times = np.array([0, 1, 1.25, 1.5, 1.75, 2, 2.5, 3, 4])
e_vals = np.array([10, 20, 25, 40, 10, 5, 10, 5, 10])/1000 # convert to ng/mL (from pg/mL)
p_times = np.array([0,0.5,1.25,1.5,1.6,1.75,0.25,2.0,4.0,3.0,3.5,2.5])
p_vals = np.array([30,10,4,2,3,60,30,5,30,15,30,5])
l_times = np.array([0,1,1.1,0.9,1.5,1.75,2.25,3.0,4.0,2,1.85,1.95])
l_vals = np.array([2,2,2,2,2,40,2,2,2,5,15,10])
f_times = np.array([0,.5,1,1.5,1.75,2.15,2,2.25,3.25,3.75,4,3.5,2.5,2.4,2.3,2.6,2.7])
f_vals = np.array([100,20,20,20,375,375,375,20,20,20,100,100,20,20,20,20,20])

plot_transition_vs_E2([DtoP,PtoE,EtoM,MtoD], e_times, e_vals, 'estradiol ng/mL')
plot_transition_vs_E2([DtoP,PtoE,EtoM,MtoD], p_times, p_vals, 'progesterone ng/mL')
plot_transition_vs_E2([DtoP,PtoE,EtoM,MtoD], l_times, l_vals, 'LH ng/mL')
plot_transition_vs_E2([DtoP,PtoE,EtoM,MtoD], f_times, f_vals, 'FSH ng/mL')
```

References

[1] E.A. Nimchinsky, Sabatini B.L., and K. Svoboda. Structure and function of dendritic spines. *Ann. Rev. Phys.*, 64:313–353, 2002.

[2] C. Bird and N. Burgess. The hippocampus and memory: insights from spatial processing. *Nat. Rev. Neurosci.*, 9:182–194, 2008.

[3] C.S. Woolley, E. Gould, M. Grankfurt, and B.S. McEwen. Naturally occurring fluctuation in dendritic spine density on adult hippocampal pyramidal neurons. *J. Neurosci.*, 10:4035–4039, 1990.

[4] N.S. Wolcott, W.T. Redman, Karpinska M., E.G. Jacobs, and M.J. Goard. The estrous cycle modulates hippocampal spine dynamics, dendritic processing, and spatial coding. *Neuron*, 12:31–48, 2019.

[5] D. Vardalaki, K. Chung, and M.T. Harnett. Filopodia are a structural substrate for silent synapses in adult neocortex. *Nat.*, 612:323–327, 2022.

[6] N.S. Wolcott, K.K. Sit, G. Raimondi, T. Hodges, R.M. Shansky, L.A.M. Galea, L.E. Ostroff, and M.J. Goard. Automated classification of estrous stage in rodents using deep learning. *Sci. Rep.*, 12:17685, 2022.

[7] N.C. Donner and C.A. Lowry. Sex differences in anxiety and emotional behavior. *Pflugers Arch - Eur. J. Physiol.*, 465:601–626, 2013.

[8] D.T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem.*, 81:2340—-2361, 1977.

[9] G. Legault and B.A. Melbourne. Accounting for environmental change in continuous-time stochastic population models. *Theor. Ecol.*, 12:31–48, 2019.