Networks and Heterogeneous Media

https://www.aimspress.com/journal/nhm

NHM, 20(4): 1292–1332.
DOI: 10.3934/nhm.2025056
Received: 28 March 2025
Revised: 07 November 2025
Accepted: 21 November 2025
Published: 05 December 2025

Research article

Kinetic modeling approach for a heterogeneous neuronal network activity using adjacency matrices

M. Menale[1,*], C. Tribuzi[2], R. Shah[1], C. A. Lupascu[3] and A. Marasco[1,3]

[1] Department of Mathematics and Applications, University of Naples Federico II, Naples, Italy
[2] Nova Analysis, Brescia, Italy
[2] Institute of Biophysics, National Research Council, Palermo, Italy

* **Correspondence:** Email: marco.menale@unina.it.

Abstract: The heterogeneity of neuronal networks plays a crucial role in shaping emergent dynamics. In this work, we introduced a kinetic modeling approach to describe the activity of heterogeneous neuronal networks through transition probabilities and adjacency matrices. The model explicitly accounts for both structural and functional heterogeneity by considering two interacting neuronal populations—excitatory pyramidal neurons and inhibitory interneurons—distributed across network slices. The transition probabilities encode the binary stochastic interactions between neurons, capturing both the neuronal types involved (excitatory or inhibitory) and the connectivity structure within and between slices. Complementarily, adjacency matrices define the weighted connections among neurons, specifying the structural organization of each slice and the interactions across slices. Together, these two components characterize the functional and the structural heterogeneity of the system. From this framework, we derived a system of nonlinear ordinary differential equations describing the mesoscopic dynamics of the network. First, for the one-slice model, we provided analytical results on the existence and stability of equilibrium states. Then, we presented numerical simulations for two- and four-slice networks to investigate the role of functional and structural heterogeneity. In particular, after defining the excitatory-, inhibitory-, and balanced count regimes and introducing an a priori criterion for their identification, we demonstrated how heterogeneity influences both the short- and long-term dynamics of the network. Our findings revealed that increasing heterogeneity not only alters the proportion of active neurons but also induces more complex dynamical behaviors, potentially driving shifts between excitatory-count- and inhibitory-count-dominated regimes.

Keywords: mathematical modeling; stochastic interactions; network connectivity; inhibitory- and excitatory-count-dominated network regimes; external inputs

Supplementary 1.

In Figure S1, we show the dynamics of active interneurons and pyramidal neurons across the whole network for each stimulation scenario described in Table 4 of the main text and corresponding to panels A–D of Figure 13. For each case, the fraction $\eta_{I,E}$ of inactive interneurons and pyramidal neurons that become active at each time t_h within the interval [0, 10] is shown as green stem plots. The percentages of newly activated interneurons and pyramidal neurons (red and blue stems, respectively), together with the percentage of active neurons at each time step (black stems), are reported for each stimulation scenario.

These figures also include a comparison between the intrinsic network dynamics (in the absence of external input, dashed lines) and the externally driven dynamics (solid lines), in terms of the fraction of active interneurons (red dashed and continuous lines) and pyramidal neurons (black dashed and continuous lines).

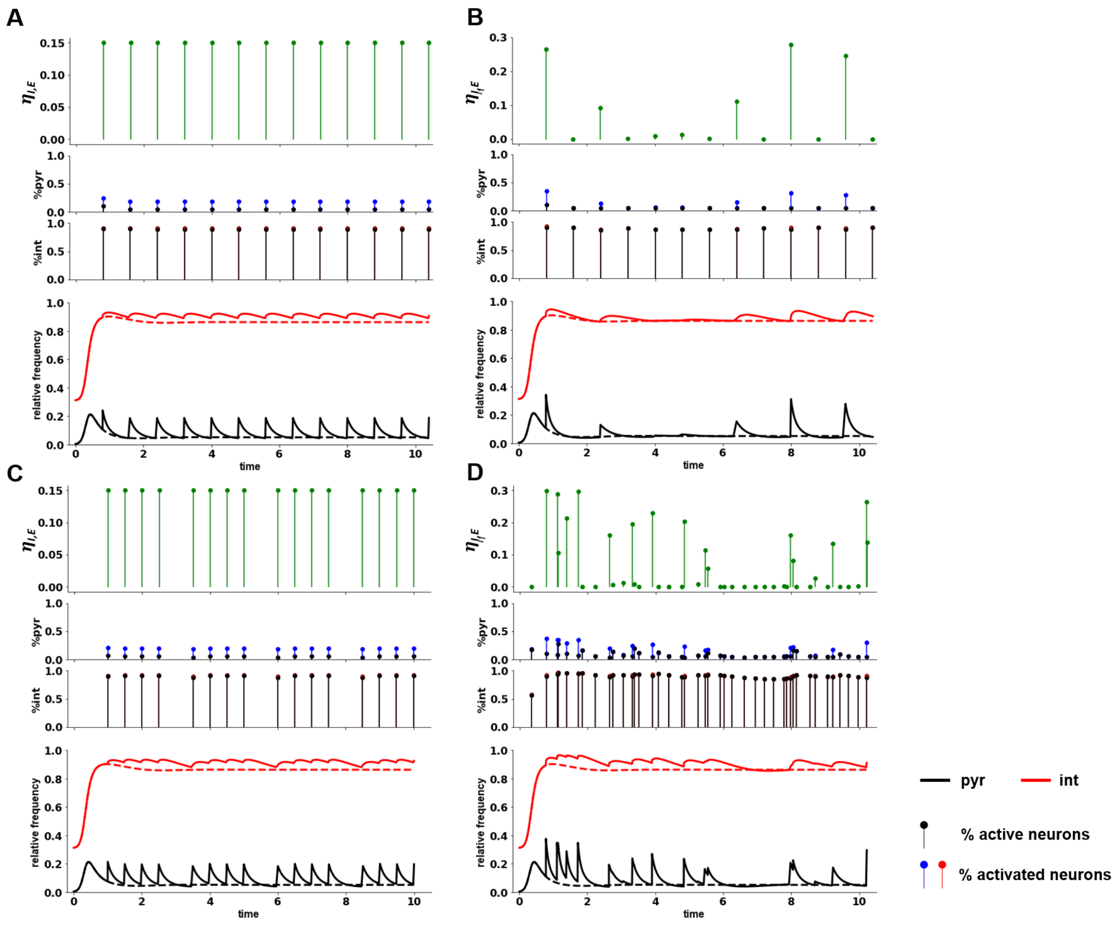


Figure S1. Oscillatory network dynamics induced by external inputs. Different oscillatory regimes emerge under external stimulation, ranging from constant and time-varying periodic inputs (panels A and B, respectively) to periodic constant and randomly time-varying bursting activity (panels C and D, respectively). Owing to the connectivity structure and interaction probabilities, the external inputs propagate throughout the network, giving rise to distinct oscillatory patterns.

Oscillatory network dynamics induced by external inputs in the four-slice model

In Figures S2–S5, we report the dynamics of active interneurons and pyramidal neurons for the whole network (left panels) and for each slice (middle and right panels) under the external inputs summarized in Table 4 of the main text. These simulations correspond to the scenarios highlighted in the insets of panels A–D in Figure 16 (as detailed, for each case, in the upper-left subpanel of Figures S2–S5).

For each case, the fraction $\eta_{I,E}$ of inactive interneurons and pyramidal neurons that become active at each time t_h within the interval [0, 10] is shown as green stem plots in each left panel. Each figure also displays the stem plots of the percentage of active interneurons and pyramidal neurons (black stems), together with the percentage of newly activated interneurons and pyramidal neurons (red and blue stems, respectively), both at the whole-network level and within each slice.

Across all panels of Figures S2–S5, only slice 1 is directly affected by the external stimulation. In contrast, slices 2–4 show no externally induced activation of either interneurons or pyramidal neurons (as evidenced by the absence of external-activation markers in the corresponding stem plots). The activity observed in these slices is therefore entirely driven by their connectivity with slice 1.

In all scenarios, the figures showing the dynamics of the whole network and of each individual slice also include a comparison between the intrinsic dynamics (in the absence of external input, dashed lines) and the externally driven dynamics (solid lines), in terms of the fraction of active interneurons and pyramidal neurons.

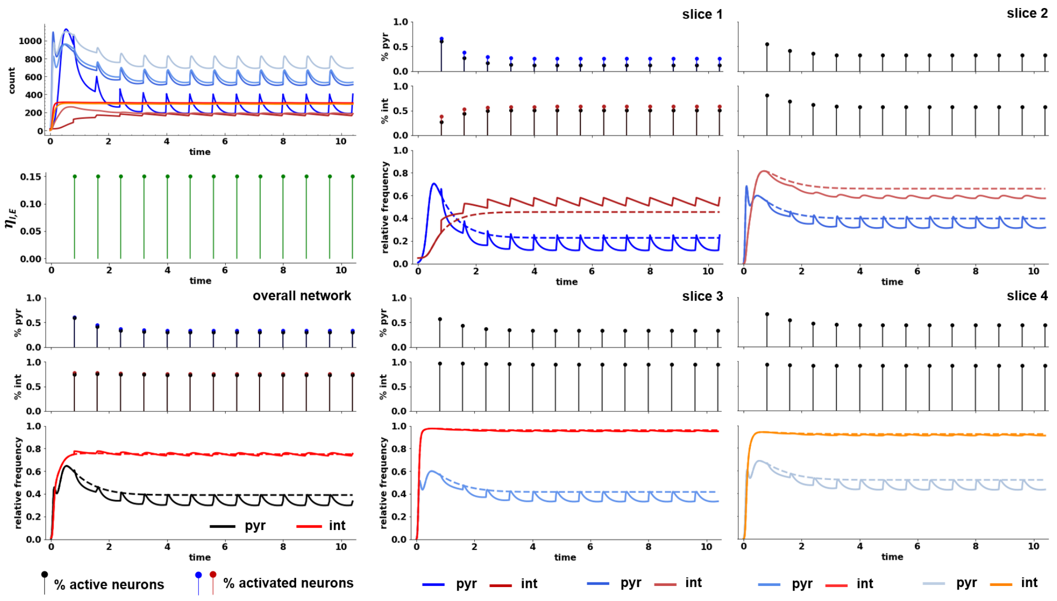


Figure S2. Oscillatory dynamics induced by constant and periodic external inputs targeting slice 1 (panel A of Figure 14). Oscillations corresponding to panel A of Figure 14 arise when a constant fraction of inactive interneurons and pyramidal neurons in slice 1 are externally activated at times $0.8h$, $h \in \mathbb{N}$. Due to the connectivity structure and interaction probabilities, this periodic stimulation propagates throughout the network, giving rise to oscillatory dynamics across all slices.

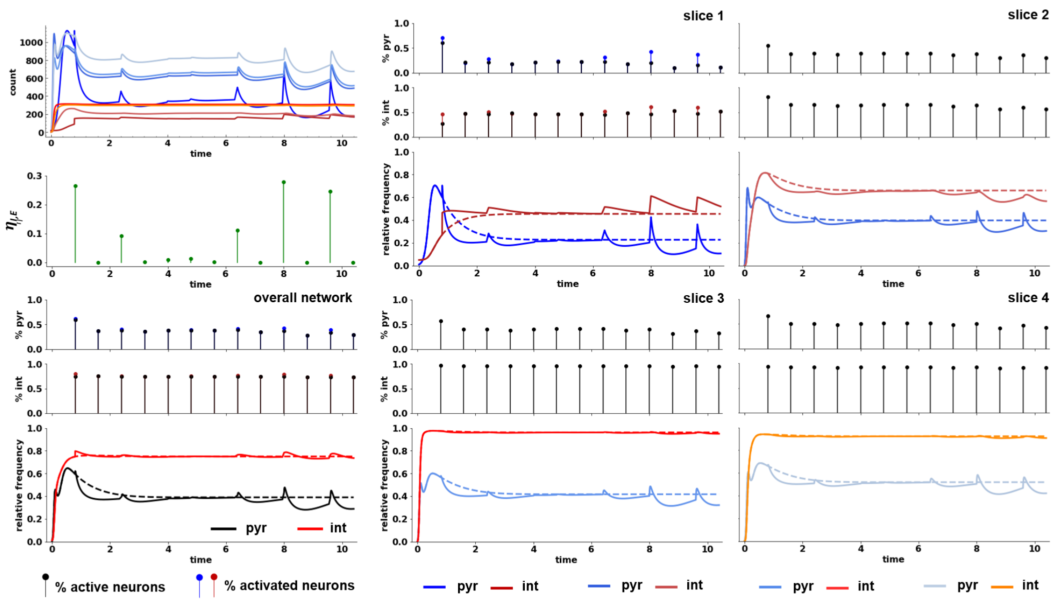


Figure S3. Oscillatory dynamics induced by time-varying periodic external inputs targeting slice 1 (panel B of Fig. 14). Oscillations corresponding to panel B of Fig. 14 arise when a time-dependent fraction of inactive interneurons and pyramidal neurons in slice 1 are externally activated at times $0.8h$, $h \in \mathbb{N}$. Owing to the connectivity structure and interaction probabilities, this periodic stimulation propagates throughout the network, leading to distinct oscillatory patterns across all slices.

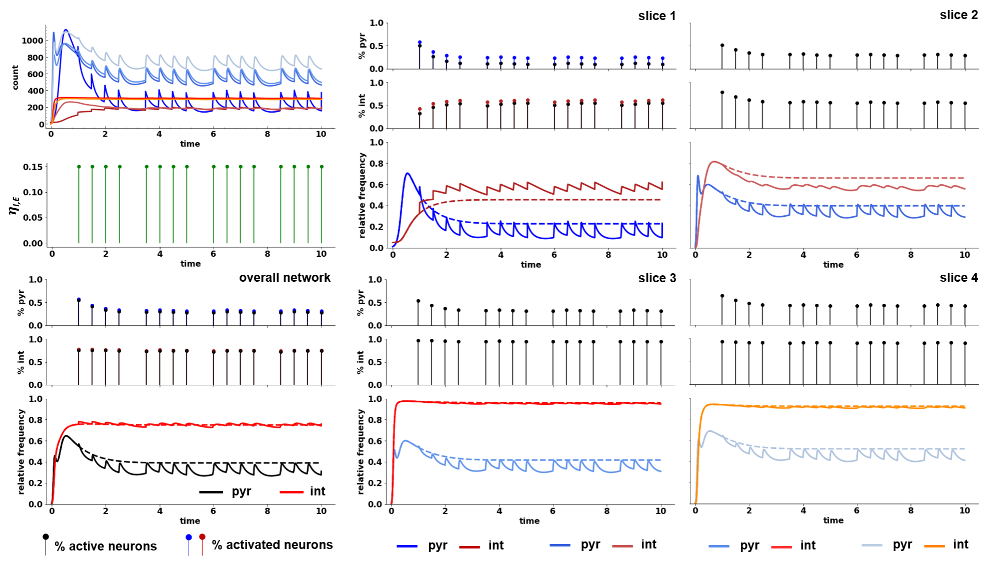


Figure S4. Oscillatory dynamics induced by regular bursting external inputs targeting slice 1 (panel C of Figure 14). Oscillations corresponding to panel C of Figure 14 emerge when a constant fraction of inactive interneurons and pyramidal neurons in slice 1 are externally activated following a regular bursting regime. Owing to the connectivity structure and interaction probabilities, this rhythmic input propagates throughout the network, resulting in regular bursting activity both at the whole-network level and within each slice.

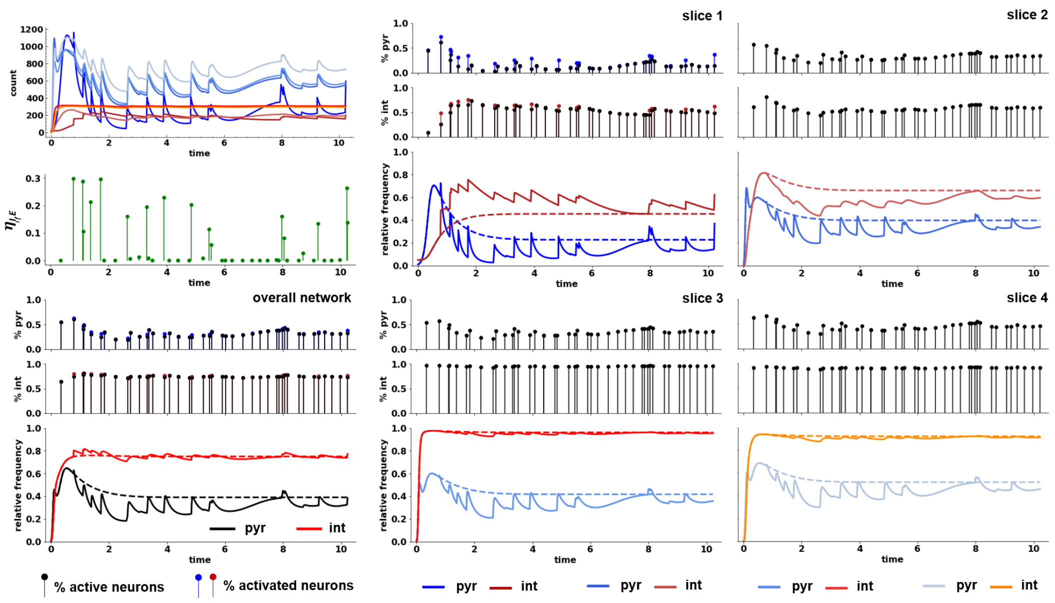


Figure S5. Oscillatory dynamics induced by intermittent bursting external inputs targeting slice 1 (panel D of Figure 14). Oscillations corresponding to panel D of Figure 14 emerge when a time-varying fraction of inactive interneurons and pyramidal neurons in slice 1 are randomly activated by external inputs. Owing to the connectivity structure and interaction probabilities, this intermittent stimulation propagates throughout the network, giving rise to irregular bursting activity both at the whole-network level and within each slice.

Supplementary 2.

The Python code, available in the ModelDB section of the Senselab database (`https://modeldb.science/2020339`), consists of four files that together implement the full workflow for running simulations of the kinetic model and visualizing the results.

High-level overview

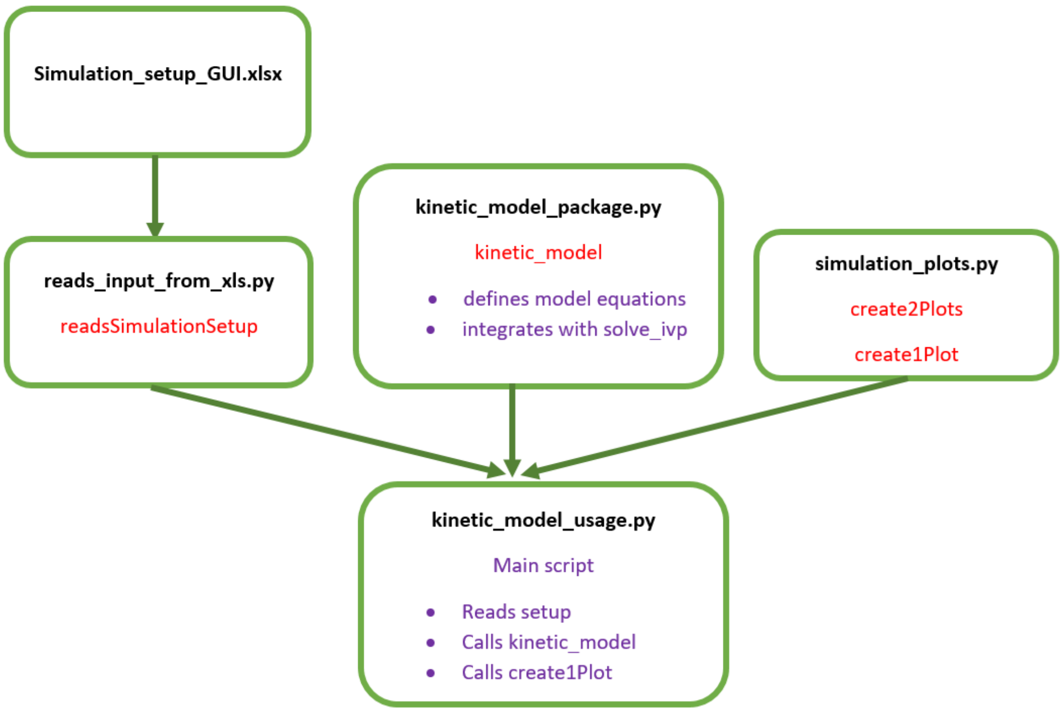


Figure S6. Workflow of the Python kinetic model simulation.

Step-by-step workflow

1. Input reading
File: `reads_input_from_xls.py`
Function: `readsSimulationSetup(filepath, sourceFileName)`

- Reads the Excel file `Simulation_setup_GUI.xlsx`.
- Extracts:
  - `listS`: number of neurons per slice.
  - `conteggioPYR`: pyramidal neuron counts.
  - Weight matrices: `W_INT_INT`, `W_INT_PYR`, `W_PYR_INT`, `W_PYR_PYR`.
  - Probability dictionaries: `dictProbab = {'p1','p2','q1','q2'}`.
  - Initial conditions: `initialConditions`.
- Returns all parameters as Pandas DataFrames, dictionaries, and lists.

2. Model definition and integration
File: `kinetic_model_package.py`
Function: `kinetic_model(...)`

- Defines the kinetic model.
- Steps:
  - Reads the initial conditions.
  - Reads the transition probabilities and computes their average values (`C_prob`, `D_function`).
  - Constructs matrices for each type of binary interaction (int-int, int–pyr, pyr-int, pyr-pyr).
  - Builds the system of ODEs in `M_rhs(t, NL)`.
  - Solves ODEs using SciPy's `scipy.integrate.solve_ivp` with the `BDF` method.
  - Returns the complete solution object `sol`.

The Python code employs the `solve_ivp` function from the SciPy library to solve the initial value problem associated with the system of ODEs derived from the kinetic model. The integration method used is the Backward Differentiation Formula (BDF), an implicit multi-step variable-order (1 to 5) scheme based on a backward differentiation formula for the derivative approximation. The implementation follows the algorithm described by Shampine and Reichelt in [1]. A quasi-constant step size is adopted, with accuracy enhanced through the Numerical Differentiation Formulas (NDF) modification. The simulation uses the default time-step configuration, meaning that the step size is not bounded and is determined adaptively by the solver.

3. Plotting and results
File: `simulation_plots.py`
Functions:

- `create1Plot(listS, scalatura, sol, simName, outputFolder)`
- `create2Plots(listS, cap, sol, simName, outputFolder)`

Purpose: Visualizes the simulated network activity.

- Separates interneuron and pyramidal neuron outputs (`create2Plots`) or plots them together (`create1Plot`).
- Uses Matplotlib for data visualization.
- Saves the resulting figures as PNG files into the output folder.

4. Model execution
File: `kinetic_model_usage.py`
Purpose: Main script that orchestrates the workflow.

- Imports the other three modules.

```
import kinetic_model_package
from kinetic_model_package import kinetic_model
from simulation_plots import create1Plot, create2Plots
import reads_input_from_xls
from reads_input_from_xls import readsSimulationSetup
```

- Define external input characteristics:
  - frequency: list of frequencies at which external input acts on the network.
  - noiseTypeString : specifies the type of external input:
    - (a) `'none'` – no external input is applied;
    - (b) `'constant'` – external input is defined by a constant value;
    - (c) `'sin712'` – external input is generated using a sine function as defined in Eq (39) of the main text.
- Reads the simulation setup:

```
(netName, listS, conteggioPYR, W_INT_INT, W_INT_PYR,
  W_PYR_INT, W_PYR_PYR, dictProbab, ic) = readsSimulationSetup(...)
```

- Transposes the weight matrices.
- Computes the time scaling factor, derived from the network population size as

$$t_s = 1/(10^{\lfloor \log_{10}(n_{\text{pyr}}+n_{\text{int}}) \rfloor})^2,$$

  where $\lfloor\cdot\rfloor$ denotes the floor function.
  This time scaling factor is then used to rescale the time axis in the plots.
- Calls kinetic_model to execute the simulation using the defined setup.

```
sol = kinetic_model(listS, conteggioPYR, dictProbab, np.array(ic),
W_INT_INT, W_INT_PYR, W_PYR_INT, W_PYR_PYR, maxTime,
frequency, noiseTypeString)
```

- Generates the simulation plots:

```
create1Plot(listS, scalatura, sol, simName, outputFolder)
```

Execution flow summary

- **Start** →`kinetic_model_usage.py`
- **Read input** → `reads_input_from_xls.py`
- **Run model** → `kinetic_model_package.py`
- **Generate plots** → `simulation_plots.py`
- **End** → Saves simulation figures as `.png` files in the results folder

How to run the code

This section describes the procedures to run the Python scripts. The user interface is designed to simulate the four-slice network model (see Figure S7).

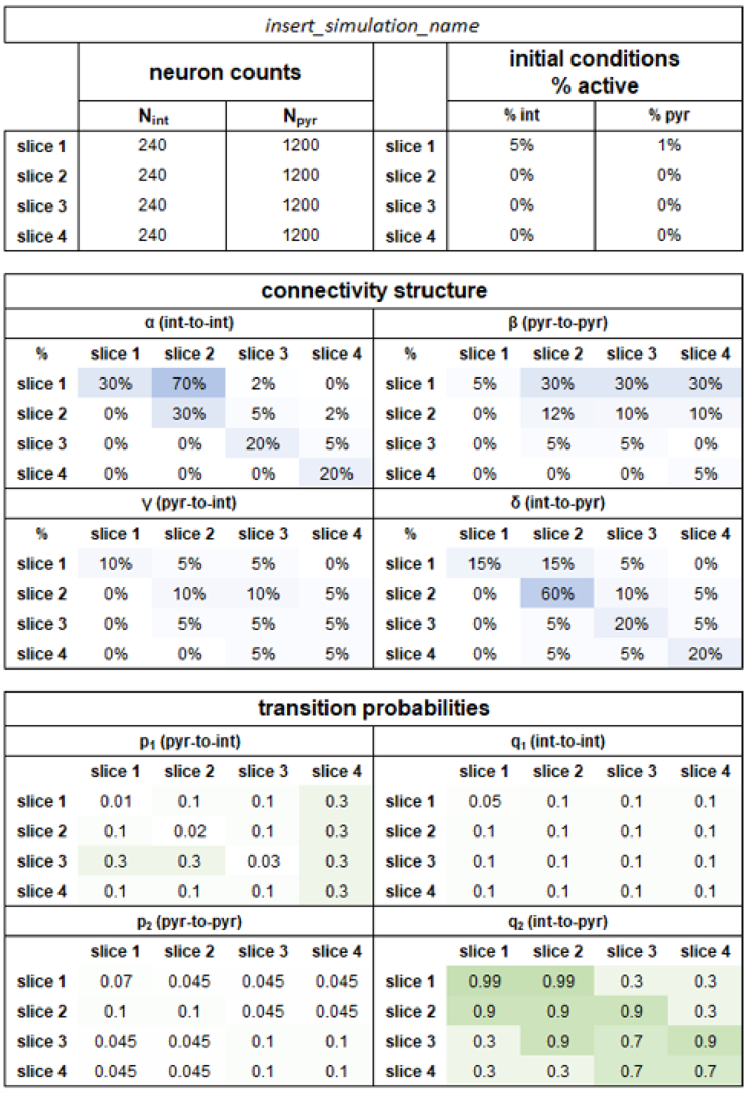
1. Define the network properties in the `Simulation_setup_GUI.xlsx` file.
   - Enter the simulation name, which will be used when generating plot images;
   - specify the neuron count per slice, the initial conditions per slice, the network connectivity, and the transition probabilities;
   - save the file
2. Configure external inputs in the `kinetic_model_usage.py` script:

(a) define the type of external input using the `noiseTypeString` variable:
- `'none'`:– no external input is applied.
- `'constant'`– constant external input is applied (see Eq (3.20) in the main text).
- `'sin712'`: sinusoidal external input is applied (see Eq (3.20) in the main text).

(b) select the frequency for external inputs:
- `'none'`: no external input is applied;
- `'constant'`: assign a constant frequency (see Eq (3.20) and Table 4 panels A and B);
- `'burst'`: assign a burst frequency (see Eq (3.20) and Table 4 panel C);
- `'random'`: assign a pseudo-random frequency (see Eq (3.20) and Table 4 panel D)

(c) Run the code.

3. Saved figures can be accessed in the results folder.



insert_simulation_name

	neuron counts			initial conditions % active	
	N_{int}	N_{pyr}		% int	% pyr
slice 1	240	1200	slice 1	5%	1%
slice 2	240	1200	slice 2	0%	0%
slice 3	240	1200	slice 3	0%	0%
slice 4	240	1200	slice 4	0%	0%

connectivity structure

α (int-to-int)					β (pyr-to-pyr)				
%	slice 1	slice 2	slice 3	slice 4	%	slice 1	slice 2	slice 3	slice 4
slice 1	30%	70%	2%	0%	slice 1	5%	30%	30%	30%
slice 2	0%	30%	5%	2%	slice 2	0%	12%	10%	10%
slice 3	0%	0%	20%	5%	slice 3	0%	5%	5%	0%
slice 4	0%	0%	0%	20%	slice 4	0%	0%	0%	5%

γ (pyr-to-int)					δ (int-to-pyr)				
%	slice 1	slice 2	slice 3	slice 4	%	slice 1	slice 2	slice 3	slice 4
slice 1	10%	5%	5%	0%	slice 1	15%	15%	5%	0%
slice 2	0%	10%	10%	5%	slice 2	0%	60%	10%	5%
slice 3	0%	5%	5%	5%	slice 3	0%	5%	20%	5%
slice 4	0%	0%	5%	5%	slice 4	0%	5%	5%	20%

transition probabilities

p_1 (pyr-to-int)					q_1 (int-to-int)				
	slice 1	slice 2	slice 3	slice 4		slice 1	slice 2	slice 3	slice 4
slice 1	0.01	0.1	0.1	0.3	slice 1	0.05	0.1	0.1	0.1
slice 2	0.1	0.02	0.1	0.3	slice 2	0.1	0.1	0.1	0.1
slice 3	0.3	0.3	0.03	0.3	slice 3	0.1	0.1	0.1	0.1
slice 4	0.1	0.1	0.1	0.3	slice 4	0.1	0.1	0.1	0.1

p_2 (pyr-to-pyr)					q_2 (int-to-pyr)				
	slice 1	slice 2	slice 3	slice 4		slice 1	slice 2	slice 3	slice 4
slice 1	0.07	0.045	0.045	0.045	slice 1	0.99	0.99	0.3	0.3
slice 2	0.1	0.1	0.045	0.045	slice 2	0.9	0.9	0.9	0.3
slice 3	0.045	0.045	0.1	0.1	slice 3	0.3	0.9	0.7	0.9
slice 4	0.045	0.045	0.1	0.1	slice 4	0.3	0.3	0.7	0.7

Figure S 7. `Simulation_setup_GUI.xlsx` file. User interface for running the four-slice model.

Reference

1. L. F. Shampine, M. W. Reichelt, The matlab ode suite, *SIAM J. Sci. Comput.*, **18** (1997), 1–22. https://doi.org/10.1137/S1064827594276424