Stochastic Reaction–Diffusion Lotka–Volterra

This repository implements a stochastic reaction–diffusion version of the classic Lotka–Volterra predator–prey model in 2D, using **JAX** and **Diffrax** for numerical integration.

Description

In the classic Lotka–Volterra model, prey (u) and predator (v) interact according to:

$$\begin{cases} \dfrac{du}{dt} = \alpha u - \beta uv, \\ \dfrac{dv}{dt} = \delta uv - \gamma v. \end{cases}$$

This repository extends the system to include:

1. **Spatial diffusion** of the fields u and v over a 2D domain.
2. **Stochastic fluctuations** in the interaction rates, controlled by two noise amplitude fields σ_β and σ_δ, which themselves diffuse in space.
3. **Independent white-noise processes** (η_β and η_δ) driving the predator–prey interactions.

Hence, the reaction–diffusion PDEs become:

$$\begin{aligned} \frac{\partial u}{\partial t} &= D_u \nabla^2 u + \alpha u - \Big[\overline{\beta} + \sigma_\beta(x, y, t)\eta_\beta(t)\Big] uv, \\ \frac{\partial v}{\partial t} &= D_v \nabla^2 v + \Big[\overline{\delta} + \sigma_\delta(x, y, t)\eta_\delta(t)\Big] uv - \gamma v, \\ \frac{\partial \sigma_\beta}{\partial t} &= D_{\sigma_\beta} \nabla^2 \sigma_\beta, \\ \frac{\partial \sigma_\delta}{\partial t} &= D_{\sigma_\delta} \nabla^2 \sigma_\delta. \end{aligned}$$

Where:

- u: prey density
- v: predator density
- $\sigma_\beta, \sigma_\delta$: fields determining the local noise intensity for β and δ, respectively
- $\alpha, \overline{\beta}, \overline{\delta}, \gamma$: primary Lotka–Volterra parameters (prey growth, mean interaction rates, predator death)

- $D_u, D_v, D_{\sigma_\beta}, D_{\sigma_\delta}$: diffusion coefficients
- η_β and η_δ: independent white-noise processes (or Brownian motions)

By combining spatial diffusion with stochastic forcing, this model can produce rich spatiotemporal patterns, including traveling waves, patchy predator–prey distributions, or chaotic-looking fluctuations depending on parameter choices.

Getting Started

Requirements

- Python 3.8+ (recommended)
- JAX (for array operations on CPU/GPU)
- Diffrax (for stochastic differential equation solvers)
- NumPy, Matplotlib, etc.

Install these dependencies, for example:

```
pip install jax jaxlib diffrax matplotlib numpy
```

If you have GPU/TPU access, ensure you install the appropriate JAX version (e.g., `jax[cuda]`).

Code Organization

1. `stochastic_lv.py` The main Python script that sets up:
   - The domain and grid
   - The reaction–diffusion PDEs + noise
   - The solver (Heun or Euler–Maruyama) from Diffrax
   - The initial/boundary conditions
   - Time stepping and saving results
2. `README.md` This file: explains the project and how to use it.
3. **(Optional) Notebooks** You can add Jupyter or Colab notebooks to visualize simulations, explore parameter variations, or generate figures.

Running the Code

Local Machine

1. Clone this repository:

   ```
   git clone https://github.com/YourUsername/stochastic-lotka-volterra.git
   cd stochastic-lotka-volterra
   ```

2. Install the requirements (see above).
3. Run the main script:

```
python stochastic_lv.py
```

The script will generate output to your console and (optionally) show plots.

In the Cloud (e.g., GitHub Codespaces / Colab)

- Open your repository in Codespaces or upload the `.py` file(s) to a Google Colab notebook.
- Install necessary packages if not already available.
- Execute the script or code cells to run the model.

Important Notes

- **Stability and Time Step** Stochastic PDEs can require small time steps for numerical stability. Adjust the solver's step-size settings (e.g., in Diffrax) if you see divergence or overly "noisy" solutions.
- **Boundary Conditions** The code is set up with periodic boundary conditions in the 2D plane. Adjust `laplacian_2d()` or the PDE implementation if you need different boundary types (e.g., no-flux or Dirichlet).
- **Parameter Tuning** Lotka–Volterra dynamics are sensitive to parameters. Spatial wave patterns and noise-induced fluctuations may depend heavily on these parameters.
- **High Dimensional Noise** The included example might use global noise processes. True space-time white noise requires one Brownian process per grid cell, which can be memory-intensive.

Future Extensions

- **Multiple Species** Extend to more predator/prey species or other compartments.
- **Heterogeneous Domain** Introduce spatially varying coefficients (e.g., $\alpha(x, y)$).
- **Alternative Noise Models** Use Ornstein–Uhlenbeck processes or multiplicative noise.