Turing Pattern Analysis Framework

This framework provides a comprehensive set of tools for analyzing Turing patterns in reaction-diffusion systems, with a specific focus on the Schnakenberg model. It is designed to help researchers explore pattern formation, particularly in cases where the activator's diffusion constant is higher than the inhibitor's.

Overview

The framework consists of several integrated components:

1. **Simulation Engine**: A C++ implementation of the Schnakenberg reaction-diffusion model with parameter input capabilities.
2. **Pattern Quantification**: Tools for measuring pattern characteristics such as size, density, and wavelength.
3. **Bifurcation Analysis**: Methods for creating bifurcation diagrams and analyzing pattern formation conditions.
4. **Linear Stability Analysis**: Tools for performing linear stability analysis and understanding pattern selection mechanisms.
5. **Parameter Space Exploration**: Methods for systematically exploring parameter space and analyzing pattern transitions.
6. **Integration Framework**: A unified interface that ties all components together.

Installation

1. Clone or download this repository to your local machine.
2. Ensure you have the following dependencies installed:
3. C++ compiler (g++ with C++11 support)
4. Python 3.6 or higher
5. NumPy
6. SciPy
7. Matplotlib
8. Pandas

9. Seaborn (optional, for enhanced visualizations)

10. Install Python dependencies: `pip install numpy scipy matplotlib pandas seaborn tqdm`

11. Compile the C++ simulation code: `python integrated_framework.py` This will automatically compile the necessary C++ files.

Usage

The framework can be used through the integrated command-line interface:

```
python integrated_framework.py [command] [options]
```

Available Commands

- **run**: Run a single simulation with specified parameters `python integrated_framework.py run --params "alpha=0.1,beta=0.9,DA=1.0,DB=100.0"`
- **sweep**: Run a parameter sweep `python integrated_framework.py sweep --params "alpha=0.1:0.5,beta=0.5:1.5,DA=1.0,DB=100.0" --resolution 5`
- **analyze**: Analyze simulation results `python integrated_framework.py analyze --dir "path/to/simulation/results" --type all`
- **quantify**: Quantify pattern characteristics from a pattern file `python integrated_framework.py quantify --file "path/to/pattern.csv" --output "metrics.csv"`
- **bifurcation**: Create bifurcation diagrams `python integrated_framework.py bifurcation --params "alpha=0.1,beta=0.9,DA=1.0,DB=100.0"`
- **stability**: Perform linear stability analysis `python integrated_framework.py stability --params "alpha=0.1,beta=0.9,DA=1.0,DB=100.0"`
- **explore**: Explore parameter space `python integrated_framework.py explore --params "alpha=0.1:0.5,beta=0.5:1.5,DA=0.5:5.0,DB=0.5:5.0" --resolution 5`

Framework Components

1. Simulation Engine (sweep_solver_param.cpp)

The C++ simulation engine implements the Schnakenberg reaction-diffusion model:

```
∂u/∂t = DA∇²u + α - u + u²v
∂v/∂t = DB∇²v + β - u²v
```

Where: - u is the activator concentration - v is the inhibitor concentration - DA is the diffusion coefficient of the activator - DB is the diffusion coefficient of the inhibitor - α and β are feed rates

The simulation uses a finite difference method with periodic boundary conditions.

2. Pattern Quantification (pattern_quantification.py)

This module provides tools for quantifying pattern characteristics:

- **analyze_pattern**: Compute basic metrics like standard deviation and contrast
- **analyze_wavelength**: Estimate pattern wavelength using Fourier analysis
- **analyze_features**: Identify and measure pattern features (spots, stripes)
- **compute_power_spectrum**: Calculate the power spectrum of patterns
- **visualize_pattern**: Create visualizations of patterns and their metrics

3. Bifurcation Analysis (bifurcation_diagram.py)

This module provides tools for bifurcation analysis:

- **calculate_steady_state**: Calculate homogeneous steady state
- **check_turing_conditions**: Check conditions for Turing instability
- **calculate_dispersion_relation**: Calculate dispersion relation
- **plot_dispersion_relation**: Plot dispersion relation
- **plot_bifurcation_diagram**: Create bifurcation diagrams for different parameters

4. Linear Stability Analysis (linear_stability_analysis.py)

This module provides tools for linear stability analysis:

- **analyze_jacobian**: Analyze the Jacobian matrix at steady state
- **calculate_eigenvalues**: Calculate eigenvalues of the linearized system
- **analyze_stability_regions**: Analyze stability regions in parameter space
- **analyze_wavelength_scaling**: Analyze how wavelength scales with parameters
- **analyze_nontraditional_mechanisms**: Analyze mechanisms for non-traditional Turing patterns

5. Parameter Space Exploration (parameter_space_exploration.py)

This module provides tools for exploring parameter space:

- **create_parameter_grid**: Create a grid of parameter combinations
- **parallel_parameter_sweep**: Run multiple simulations in parallel
- **create_parameter_heatmaps**: Create heatmaps of pattern metrics

- **explore_diffusion_ratio_transition**: Explore transition between diffusion regimes
- **analyze_wavelength_scaling**: Analyze wavelength scaling with parameters
- **create_phase_diagram**: Create phase diagram of pattern types

6. Integration Framework (integrated_framework.py)

This script provides a unified interface to all components:

- **setup_directories**: Set up directory structure
- **compile_simulation_code**: Compile C++ simulation code
- **run_simulation**: Run a single simulation
- **analyze_simulation_results**: Analyze simulation results
- **create_bifurcation_diagram**: Create bifurcation diagrams
- **perform_stability_analysis**: Perform linear stability analysis
- **explore_parameter_space**: Explore parameter space

Research Applications

This framework is particularly useful for investigating:

1. **Non-Traditional Turing Patterns**: Exploring pattern formation when DA > DB, which challenges traditional Turing theory.
2. **Wavelength Scaling Laws**: Investigating how pattern wavelength scales with diffusion ratio.
3. **Pattern Type Transitions**: Understanding transitions between spots, stripes, and labyrinthine patterns.
4. **Parameter Space Mapping**: Creating comprehensive maps of pattern types across parameter space.

Example Workflow

1. **Theoretical Analysis**: `python integrated_framework.py bifurcation --params "alpha=0.1,beta=0.9,DA=1.0,DB=100.0" python integrated_framework.py stability --params "alpha=0.1,beta=0.9,DA=1.0,DB=100.0"`

2. **Run Simulations**: `python integrated_framework.py run --params "alpha=0.1,beta=0.9,DA=1.0,DB=100.0"`

3. **Analyze Results**: `python integrated_framework.py analyze --dir "simulations/single/sim_20250415_123456" --type all`

4. **Explore Parameter Space**: `python integrated_framework.py explore --params "alpha=0.1:0.5,DA=0.5:5.0,DB=0.5:5.0" --resolution 5`

Directory Structure

The framework creates the following directory structure:

```
integrated_framework/
├── simulations/      # Simulation results
│   ├── single/       # Single simulation results
│   └── sweep/        # Parameter sweep results
├── results/          # Analysis results
├── figures/          # Generated figures
│   ├── bifurcation/  # Bifurcation diagrams
│   ├── stability/    # Stability analysis figures
│   ├── patterns/     # Pattern visualizations
│   └── exploration/  # Parameter exploration figures
└── data/             # Miscellaneous data files
```

Extending the Framework

The modular design of this framework makes it easy to extend:

1. **New Pattern Metrics**: Add new functions to pattern_quantification.py
2. **Different Reaction Models**: Modify sweep_solver_param.cpp to implement different reaction terms
3. **Additional Analysis Methods**: Add new modules or extend existing ones
4. **Custom Visualizations**: Add new visualization functions to any module

Troubleshooting

- **Compilation Errors**: Ensure you have a C++11 compatible compiler
- **Runtime Errors**: Check parameter values are within reasonable ranges
- **Memory Issues**: For large simulations, reduce grid size or use a machine with more RAM
- **Visualization Problems**: Ensure matplotlib and other visualization dependencies are correctly installed

References

1. Turing, A. M. (1952). The chemical basis of morphogenesis. Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences, 237(641), 37-72.
2. Schnakenberg, J. (1979). Simple chemical reaction systems with limit cycle behaviour. Journal of theoretical biology, 81(3), 389-400.
3. Murray, J. D. (2003). Mathematical Biology II: Spatial Models and Biomedical Applications. Springer-Verlag.