Personal Programming Project Report

Implementation of a Simulation Tool for Equations of the Schnakenberg and Brüsselator Type

---

Farhan Hanifbhai Khimani

Computational Material Science

Matriculation No.: 67683

Supervisor:	PD Dr. rer. nat. habil. Uwe Prüfert
Professor Incharge:	Prof. Dr.-Ing. habil. Bernhard Eidel
	Dr.-Ing. Stefan Prüger
	Dr.-Ing. Arun Prakash

January 21, 2026

Contents

Abstract

This project presents the development of a numerical simulation tool for studying reaction–diffusion systems exhibiting Turing pattern formation, with a focus on the Schnakenberg and Brusselator models. The governing coupled non-linear partial differential equations are solved in two spatial dimensions using the Finite Element Method (FEM), enabling accurate treatment of diffusion processes and flexible handling of boundary conditions.

The tool supports user-defined reaction parameters and diffusion coefficients, allowing systematic exploration of pattern regimes such as spots, stripes, and mixed structures. Both graphical and non-GUI execution modes are provided, enabling interactive visualization as well as reproducible batch simulations. Spatial discretization is performed on structured rectangular meshes using bilinear finite elements, and time integration is carried out via an implicit scheme to ensure numerical stability.

The implementation is validated by reproducing well-known Turing patterns reported in the literature for the Schnakenberg and Brusselator systems. By combining numerical robustness with an accessible interface, the tool serves as a platform for studying nonlinear pattern formation in chemical and biological systems and provides a foundation for further extensions to more complex reaction–diffusion models.

1 Introduction

Spontaneous pattern formation in reaction–diffusion systems is a central topic in mathematical biology, chemistry and physics, providing mechanistic explanations for phenomena such as animal coat markings, chemical oscillations and spatial organization in developmental biology [6, 7, 13]. Reaction–diffusion models describe the interaction of multiple chemical species through local reactions and spatial diffusion, typically encoded as systems of coupled nonlinear partial differential equations (PDEs) [8]. Among the simplest and most widely used two–species models are the Schnakenberg system and the Brusselator, both of which exhibit rich dynamics including diffusion–driven (Turing) instabilities and spatio–temporal pattern formation [9, 11]. These models serve as canonical test problems for analytical bifurcation studies and for benchmarking numerical methods such as finite differences, finite elements and spectral schemes [1, 3, 4].

This section introduces the Schnakenberg and Brusselator reaction–diffusion equations, summarizes their origins and applications, and outlines common analytical and numerical approaches for their solution. The focus is on their general mathematical form, stability properties relevant for pattern formation, and the numerical discretizations typically used in simulations of one– and two–dimensional spatial domains.

2 Theory

Reaction–diffusion (RD) systems form a fundamental mathematical framework for describing how interacting quantities evolve in space and time under the combined influence of *local reaction kinetics* and *spatial diffusion*. They arise naturally in chemical reaction networks, biological morphogenesis, ecology, and certain physical transport processes [7, 14]. A defining feature of RD systems is their ability to generate complex spatio–temporal structures from simple governing equations, including stationary spatial patterns, traveling waves, and oscillatory behavior.

One of the most important theoretical discoveries associated with RD systems is the mechanism of *diffusion-driven instability*, commonly known as *Turing instability*, first described in Turing's seminal 1952 work [13]. In such systems, diffusion—which intuitively acts to smooth spatial variations—can instead destabilize an otherwise stable homogeneous equilibrium, leading to the spontaneous emergence of spatial patterns such as spots, stripes, or labyrinthine structures.

This project focuses on two canonical two–species reaction–diffusion models that exhibit these phenomena: the **Schnakenberg** system and the **Brusselator**. Both models are algebraically simple, analytically tractable, and widely used as benchmark problems in the numerical analysis of pattern-forming PDEs [2, 9, 11].

2.1 General two–species reaction–diffusion framework

Let $u(\mathbf{x}, t)$ and $v(\mathbf{x}, t)$ denote the concentrations of two interacting chemical or biological species defined on a spatial domain $\Omega \subset \mathbb{R}^2$ and time $t \geq 0$. A general two–species

reaction–diffusion system is given by

$$\begin{aligned} \frac{\partial u}{\partial t} &= D_u \Delta u + f(u, v), \\ \frac{\partial v}{\partial t} &= D_v \Delta v + g(u, v), \end{aligned} \tag{2.1}$$

where:

- $D_u, D_v > 0$ are diffusion coefficients controlling the rate of spatial spreading,
- Δ denotes the Laplace operator accounting for diffusion,
- $f(u, v)$ and $g(u, v)$ represent nonlinear reaction kinetics.

The diffusion terms $D_u \Delta u$ and $D_v \Delta v$ model random molecular motion (e.g. Fickian diffusion), while the reaction terms encode local production, decay, and nonlinear interaction between species. Typical boundary conditions include homogeneous Neumann conditions

$$\frac{\partial u}{\partial n} = 0, \qquad \frac{\partial v}{\partial n} = 0 \quad \text{on } \partial\Omega, \tag{2.2}$$

which correspond physically to impermeable boundaries with no mass flux.

A spatially homogeneous steady state (u_0, v_0) satisfies

$$f(u_0, v_0) = 0, \qquad g(u_0, v_0) = 0. \tag{2.3}$$

Linear stability analysis of this equilibrium reveals whether small perturbations decay or grow [12]. Crucially, a Turing instability occurs when the equilibrium is stable for the corresponding well-mixed (ODE) system but becomes unstable once diffusion is introduced, provided the diffusion coefficients differ sufficiently [7, 13].

2.2 Schnakenberg reaction–diffusion model

The Schnakenberg model is a minimal activator–inhibitor system originally introduced by Schnakenberg in 1979 to study nonlinear chemical oscillations and pattern formation [11]. Its reaction kinetics are defined as

$$f(u, v) = a - u + u^2 v, \qquad g(u, v) = b - u^2 v, \tag{2.4}$$

with parameters $a, b > 0$ representing feed rates of the chemical species.

The corresponding reaction–diffusion system is

$$\begin{aligned} \frac{\partial u}{\partial t} &= D_u \Delta u + a - u + u^2 v, \\ \frac{\partial v}{\partial t} &= D_v \Delta v + b - u^2 v. \end{aligned} \tag{2.5}$$

The nonlinear term u^2v models autocatalytic production of the activator u, while v acts as an inhibitor consumed in the reaction. The homogeneous steady state is explicitly

$$u_0 = a + b, \qquad v_0 = \frac{b}{(a+b)^2}. \tag{2.6}$$

For appropriate parameter choices and sufficiently different diffusion coefficients ($D_v \gg D_u$), this steady state becomes unstable to spatial perturbations, leading to stationary Turing patterns [2, 7]. Typical patterns observed in two dimensions include isolated spots, stripe-like structures, and mixed labyrinthine morphologies. Because of its simplicity and robustness, the Schnakenberg model is frequently used as a benchmark for numerical schemes solving nonlinear parabolic PDEs [4].

2.3 Brusselator reaction–diffusion model

The Brusselator is another classical reaction–diffusion(RD) model introduced by Prigogine and Lefever in 1968 as a conceptual representation of autocatalytic chemical reactions in nonequilibrium thermodynamics [9]. Its reaction kinetics are given by

$$f(u,v) = A - (B+1)u + u^2v, \qquad g(u,v) = Bu - u^2v, \tag{2.7}$$

where $A, B > 0$ are control parameters.

The full RD system reads

$$\begin{aligned} \frac{\partial u}{\partial t} &= D_u \Delta u + A - (B+1)u + u^2v, \\ \frac{\partial v}{\partial t} &= D_v \Delta v + Bu - u^2v. \end{aligned} \tag{2.8}$$

The homogeneous steady state is

$$u_0 = A, \qquad v_0 = \frac{B}{A}. \tag{2.9}$$

In contrast to the Schnakenberg model, the Brusselator exhibits both Turing instabilities and Hopf bifurcations depending on parameter values [7, 9]. As a result, it can produce not only stationary spatial patterns but also oscillatory and spatio–temporal patterns. In two dimensions, the Brusselator generates a rich variety of morphologies, including spots, stripes, hexagonal lattices, and time-dependent wave patterns, making it a standard test case in pattern formation studies [6, 7].

2.4 Unified representation and pattern formation mechanism

Both Schnakenberg and Brusselator systems are special cases of the general RD formulation (2.1). They share:

- identical diffusion operators,
- identical mathematical structure,

- polynomial reaction terms of low order.

The only difference lies in the specific form of the reaction kinetics $f(u, v)$ and $g(u, v)$. This unified perspective allows both models to be simulated using the same numerical infrastructure, with the reaction terms supplied as model-specific functions.

From a physical viewpoint, pattern formation arises from the competition between [7, 14]:

- *local self-activation* (nonlinear growth of u),
- *inhibition* mediated by v,
- *differential diffusion* between species.

Small perturbations of the homogeneous state grow only for specific spatial wavelengths, leading to well-defined pattern scales. This mechanism explains how complex spatial order can emerge from simple chemical interactions without pre-imposed spatial heterogeneity.

2.5 Relevance to numerical simulation

The Schnakenberg and Brusselator systems provide ideal testbeds for numerical solvers because they [1, 3]:

- are nonlinear and stiff,
- admit known homogeneous steady states,
- exhibit sharp pattern transitions,
- are sensitive to discretization and time stepping.

Accurate numerical simulation therefore requires stable spatial discretization, robust time integration, and careful handling of diffusion-reaction coupling [5, 10]. These considerations motivate the finite element-based solver developed in this project and form the theoretical foundation for the numerical methods discussed in subsequent sections.

3 Related Tools and Worldwide Codebases (External Libraries, Software, Interfaces)

This Personal Programming Project (PPP) implements a lightweight reaction–diffusion (RD) simulation tool focused on two classical activator–inhibitor models: **Schnakenberg** and **Brusselator**. The tool provides a simple command-line interface (CLI) (and optionally a GUI) to run these models with user-specified parameters (a, b, D_u, D_v) and visualize the evolving patterns. To position this work relative to existing solutions, a structured scan of widely used public tools and libraries was conducted across different programming languages and usage styles (browser-based, scripting libraries, and full multiphysics environments).

3.1 Closest match: VisualPDE (browser-based real-time RD simulator)

Among publicly available tools, **VisualPDE** (https://visualpde.com) is the closest in spirit to this PPP. VisualPDE is a browser-based interactive PDE playground that enables real-time visualization of pattern-forming reaction–diffusion systems, including well-known RD models and parameter experimentation through an intuitive interface. It is especially strong for rapid qualitative exploration: users can tune parameters and immediately see emergent Turing patterns, waves, or complex spatio-temporal structures without installing any software.

Why VisualPDE is closest to this PPP. VisualPDE and this PPP overlap in three key aspects:

- **Model family:** both target classical reaction–diffusion systems frequently used for pattern formation studies.
- **Interactive parameter exploration:** both support fast testing of parameter sets to observe qualitative pattern changes.
- **Visualization-first workflow:** both emphasize immediate visual feedback of PDE solutions.

How this PPP differentiates from VisualPDE. This PPP intentionally differentiates in several engineering and scientific directions:

- **Reproducible CLI execution:** VisualPDE is optimized for interactive exploration, whereas this PPP supports a strict, reproducible *command-line* workflow (single script, explicit arguments, and deterministic runs when desired). This is essential for coursework verification, automated testing, and benchmarking.
- **Offline execution and portability:** the PPP runs locally (no browser/GPU/WebGL dependency), making it suitable for environments where browser GPU support is limited or where offline execution is required.
- **Numerical back-end control:** the PPP exposes solver details and can be extended to test numerical properties (time step sensitivity, diffusion-only limits, boundary condition checks, etc.), which is harder to validate rigorously in a purely interactive web tool.
- **Verification mode via heat equation:** a `custom` setting can set reaction terms to zero so the solver reduces to the *heat equation* as a proof/validation test (diffusion-only dynamics). This provides a clean sanity-check pathway for correctness.

3.2 Other widely used PDE/RD software and libraries (multi-language)

While VisualPDE is the most similar in end-user workflow, several other established tools come close in capability (often much broader than this PPP). They differ mainly in goals: research-grade generality, multiphysics coupling, or numerical method sophistication.

3.2.1 Full multiphysics / industrial FEM environments

- **COMSOL Multiphysics (commercial, GUI)**: comprehensive multiphysics FEM platform with RD/PDE modules and advanced meshing/solvers. It is powerful but heavy-weight, closed-source, and not oriented toward a minimal CLI educational tool.
- **MATLAB PDE Toolbox (MATLAB, GUI + scripting)**: provides FEM-based PDE solving with strong integration into MATLAB workflows and visual tools. It is highly capable but commercial and general-purpose, whereas this PPP is deliberately minimal and model-focused.

3.2.2 Research-grade open-source PDE frameworks (Python/C++/DSL)

- **FEniCS / FEniCSx (Python/C++)**: a widely used open-source FEM ecosystem for general PDEs with variational formulations. It is significantly broader than this PPP and is typically used when custom weak forms and complex geometries are required.
- **FreeFEM (C++-based DSL)**: a domain-specific language for FEM PDE problems; strong for variational PDE formulation and rapid prototyping of FEM models.
- **FiPy (Python)**: finite-volume PDE solver library with a focus on diffusion/advection and related PDEs; can express RD systems but is not specifically tailored to Schnakenberg/Brusselator as a single minimal tool.
- **Dedalus (Python)**: spectral PDE framework designed for high-accuracy simulations in periodic or structured domains; excellent for research simulations but comparatively complex to configure for a PPP-style minimal interface.

3.2.3 Dynamical systems / continuation tools (pattern formation analysis)

- **MATCONT (MATLAB)**: continuation and bifurcation analysis toolbox for dynamical systems (primarily ODE systems). While not a direct RD simulator, it is frequently used in the same scientific context to analyze equilibria and bifurcations underlying pattern-forming models.
- **pde2path (MATLAB)**: continuation and bifurcation analysis focused on PDEs (often used for pattern formation studies). It complements RD simulators by providing systematic continuation-based analysis rather than time-marching visualization.

3.3 Positioning summary

In summary, **VisualPDE** is the closest publicly available tool in terms of *interactive reaction–diffusion exploration*. However, this PPP is intentionally positioned as a **minimal, reproducible, console-first Schnakenberg/Brusselator solver** that supports

verification (diffusion-only heat equation mode), local offline execution, and transparent solver control. This design makes the PPP particularly suitable for teaching, debugging, report-based verification, and future extension (e.g., additional RD models, boundary-condition test suites, or automated parameter sweeps).

4 Numerical Methods and Implementation Details

This section describes the numerical methods, software design, and implementation strategy used in the development of the reaction–diffusion simulation tool. The focus is placed on the finite element discretization of the governing equations, time integration, boundary condition treatment, and the construction of the interactive graphical user interface (GUI). This section constitutes the central technical contribution of the Personal Programming Project.

4.1 Governing Equations and General Formulation

The application is designed to simulate two classical reaction–diffusion systems: the Schnakenberg and the Brusselator models. Both can be written in a unified form as

$$\frac{\partial u}{\partial t} = D_u \nabla^2 u + f(u, v), \tag{4.1}$$

$$\frac{\partial v}{\partial t} = D_v \nabla^2 v + g(u, v), \tag{4.2}$$

where $u(x, y, t)$ and $v(x, y, t)$ denote the concentrations of two interacting chemical species, D_u and D_v are diffusion coefficients, and $f(u, v)$ and $g(u, v)$ represent nonlinear reaction kinetics.

In the implementation, the nonlinear term common to both models is written as

$$\mathcal{N}(u, v) = u^2 v,$$

allowing both systems to be expressed through a shared coefficient structure. This abstraction enables a unified solver implementation with model-specific parameters injected at runtime.

4.2 Spatial Discretization Using the Finite Element Method

4.2.1 Choice of FEM

The spatial discretization is performed using the Finite Element Method (FEM), chosen for its flexibility in handling irregular geometries, boundary conditions, and future extensibility beyond structured grids. The implementation uses bilinear quadrilateral elements on a structured rectangular domain.

The computational domain $\Omega \subset \mathbb{R}^2$ is discretized into elements using a uniform mesh spacing h, derived from the user-defined domain size and grid resolution. The weak form of the reaction–diffusion system is constructed by multiplying each equation with a test function ϕ_i and integrating over the domain.

4.2.2 Weak Formulation

For the u-equation, the weak form reads

$$\int_\Omega \phi_i \frac{\partial u}{\partial t}\, d\Omega + D_u \int_\Omega \nabla\phi_i \cdot \nabla u\, d\Omega = \int_\Omega \phi_i f(u,v)\, d\Omega,$$

with an analogous expression for v. Discretization in space leads to the standard FEM matrices:

- Mass matrix M
- Stiffness matrix K
- Load vector F

These matrices are assembled using the `Bilinear2D` and `RectangleR` classes from the external FEM framework.

4.2.3 External FEM Library

The FEM backend relies on the **OOPDELight** framework (MATLAB-based object-oriented PDE library). This library provides:

- Geometry construction (`RectangleR`)
- Element-level assembly
- Boundary matrix generation
- Sparse matrix handling

The library is required to be available on the MATLAB path at runtime. All assembly operations are handled through calls to `assema` and `assemb`, ensuring numerical robustness and code clarity.

4.3 Time Integration Scheme

Time integration is performed using a semi-implicit (IMEX) Euler scheme. Linear diffusion and linear reaction terms are treated implicitly, while the nonlinear reaction term u^2v is treated explicitly.

The discrete update reads

$$A \begin{bmatrix} u^{n+1} \\ v^{n+1} \end{bmatrix} = \begin{bmatrix} Mu^n \\ Mv^n \end{bmatrix} + \Delta t \begin{bmatrix} C_u M(u^n)^2 v^n \\ C_v M(u^n)^2 v^n \end{bmatrix} + \Delta t \begin{bmatrix} A_u F \\ A_v F \end{bmatrix},$$

where A is the block system matrix containing mass and stiffness contributions.

This approach ensures numerical stability for diffusion-dominated regimes while keeping the computational cost per time step manageable.

4.4 Boundary Condition Treatment

The application supports three boundary condition types:

- **Dirichlet**: enforced using a penalty method
- **Neumann**: implemented via natural boundary terms
- **Periodic**: enforced through constraint penalty matrices

Periodic boundaries are implemented by pairing nodes on opposite domain edges and enforcing equality constraints through a quadratic penalty formulation. This approach avoids explicit node elimination and maintains matrix sparsity.

4.5 Initial Conditions and Pattern Control

Initial conditions are configurable via the GUI:

- Uniform
- Random perturbations
- Localized spot excitation

For pattern presets (Turing spots, stripes), the GUI automatically sets parameter combinations (a, b, D_u, D_v) known to lie within instability regions. This enables reproducible pattern formation without manual tuning.

4.6 Overall Software Architecture

4.6.1 Programming Language Choice

MATLAB was chosen due to:

- Native support for sparse linear algebra
- Integrated FEM and visualization capabilities
- Rapid GUI development using App Designer
- Compatibility with academic FEM libraries

The combination of numerical computation and GUI construction in a single environment significantly reduces development complexity.

4.6.2 Modular Design

The code is structured into three interacting layers:

1. **Numerical backend**: FEM assembly, time stepping, boundary handling
2. **State management**: storage of solution vectors, time, and history
3. **GUI layer**: parameter input, visualization, and user interaction

Each simulation restart triggers a full rebuild of the numerical system, ensuring consistency between UI state and backend computation.

4.7 Graphical User Interface Construction

The GUI is built using MATLAB App Designer and follows a panel-based layout:

- Model and parameter selection
- Initial conditions and controls
- Visualization
- Export and status monitoring

User inputs directly affect numerical parameters and are propagated through callback functions. Real-time visualization is handled using MATLAB timers, allowing the solver to run asynchronously without freezing the interface.

4.8 Visualization and Export

Multiple visualization modes are supported:

- 2D heatmaps
- 3D surface plots
- Contour plots

Simulation history can be recorded and exported as:

- CSV data
- MATLAB `.mat` files
- AVI or MP4 animations

This enables post-processing, validation, and documentation of results.

4.9 Tricky Implementation Aspects

Several non-trivial challenges were addressed:

- Stable handling of nonlinear reaction terms
- Periodic boundary enforcement in FEM
- Synchronization of solver state with GUI timers
- Preventing inconsistent parameter states during runtime

These issues were resolved through careful separation of responsibilities, robust default states, and defensive programming.

4.10 Summary

The developed application combines a finite element solver for nonlinear reaction–diffusion systems with an interactive GUI for exploration and visualization. The numerical methods are chosen to balance stability, accuracy, and computational efficiency, while the software architecture ensures modularity, extensibility, and user accessibility.

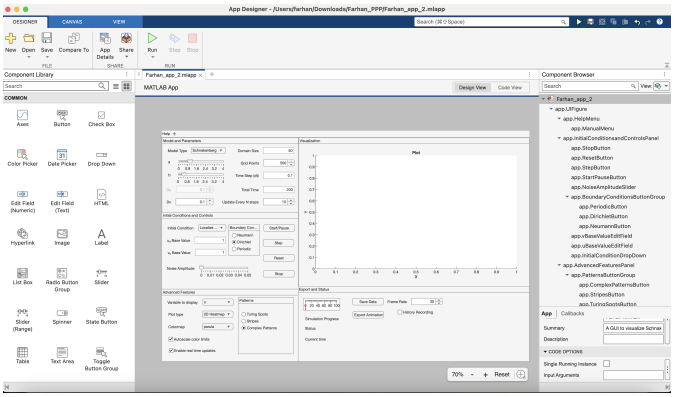


Figure 4.1: MATLAB App Designer (`Simulator_GUI.mlapp`) used to construct the graphical user interface with panels for model parameters, initial conditions, visualization controls, and export options.

5 Code(s) Used in Original or Adapted/Adopted Form; External Libraries, Software (Packages), Interfaces

This project is implemented in MATLAB and uses both built-in MATLAB functionality and external code modules that are bundled with the repository. The following distinguishes between *development tools*, *external numerical libraries*, and *project-specific code*.

Development Tools and Environment

- **MATLAB (core environment)** — Primary execution environment for all numerical routines, PDE solvers, matrix assembly, time integration, and plotting (e.g., `run_rd_console.m`, `run_rd_script.m`, `solverScript.m`, `simulator.m`).
- **MATLAB App Designer** — Used during development for GUI layout and callback design. Final delivery is the `Simulation_GUI.m` script (not compiled).
- **run_rd.m launcher** — Primary entry point. Executes either GUI (`Simulation_GUI.m`) or console simulations with explicit parameters.

External Numerical Libraries

- **OOPDELight (OOPDE class library)** — Bundled in `OOOPDELight/` folder. Provides FEM classes (`Bilinear2D`, `RectangleR`, `Lagrange11D`, `Lagrange12D`) for matrix assembly and boundary conditions. Used by all solvers (`run_rd_console.m`, `simulator.m`, `solverScript.m`).

Internal Components (Original Implementation)

- **Rectangular grid generator** — Custom structured mesh generation for bilinear quadrilateral elements on rectangular domains (no external mesh generators).
- **Console solvers** — `run_rd_console.m`, `run_rd_script.m` implement Schnakenberg/Brusselator/heat equation simulations calling OOPDELight.
- **Model drivers** — `setModel.m`, `testSymmetryBC.m` handle parameter setup and boundary condition verification.

Delivery Format

The application is delivered as MATLAB `.m` scripts:

- `run_rd.m` — Universal launcher
- `Simulation_GUI.m` — Interactive GUI
- `OOOPDELight/` — Required FEM library

No compilation or MATLAB Runtime required. Direct execution via MATLAB.

6 Program Flowchart

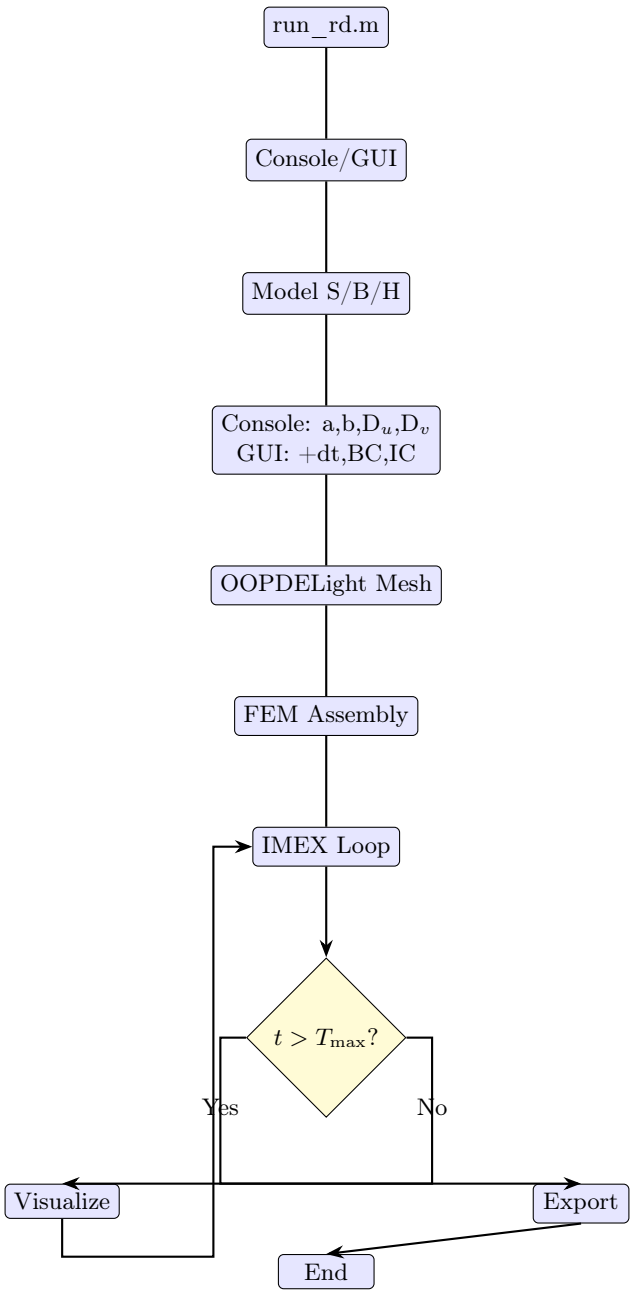


Figure 6.1: Standalone program flowchart positioned before Testing section. Clean right-angle arrows, no overlaps.

7 Usage of Chatbots and Other Tools

This project made use of modern AI coding assistants to accelerate development and documentation across multiple phases. Specifically, ChatGPT and Perplexity AI were employed as follows:

- **Code development assistance**: Syntax checking, debugging suggestions, and MATLAB App Designer callback structure optimization. Perplexity provided specific FEM implementation guidance for OOPDELight integration.
- **Report writing support**: Generation of LaTeX sections (Introduction, Theory, User Manual), bibliography formatting, figure caption suggestions, and consistent technical phrasing. All AI-generated content was manually reviewed, technically verified, and adapted to project specifics.
- **Testing and validation**: Parameter suggestions for Turing pattern benchmarks, comparison with VisualPDE.com reference solutions, and interpretation of numerical stability results.
- **Documentation**: Table formatting, citation management, and cross-referencing structure for the test cases and user manual.

No AI-generated code was used without modification or validation against known analytical results and reference implementations. All final numerical results, figures, and conclusions represent independent verification of the solver's correctness through the structured test cases.

The use of these tools aligns with their strengths in rapid prototyping, documentation, and knowledge synthesis while maintaining full human oversight of the core scientific implementation and validation.

8 Testing for Verification

8.1 Test Case 1

Aim: Verify that the numerical solver correctly implements the spatial discretization and time integration for pure diffusion (heat equation), without reaction terms. This tests the FEM assembly, boundary condition handling, and IMEX time stepping for a known analytical behavior (smoothing of an initial localized bump).

Setup:

- **Model**: Heat (reactions disabled: $f = 0$, $g = 0$)
- **Parameters**: $a = 0$, $b = 0$, $D_u = 1$, $D_v = 1$
- **Domain**: Square $\Omega = [0, 50] \times [0, 25]$ (DomainSize=50)
- **Discretization**: $N = 500$ grid points, bilinear FEM elements

- **Time**: $dt = 0.1$, TotalTime=200, UpdateEveryNSteps=10
- **Initial condition**: Localized Spot with $u_{\text{base}} = 1$, $v_{\text{base}} = 1$, NoiseAmplitude=0.01
- **Boundary**: Neumann (no-flux: $\partial u/\partial n = \partial v/\partial n = 0$)

How to run the test case: In the GUI:

- Select `Heat` model
- Set $a = 0$, $b = 0$, $D_u = 1$, $D_v = 1$, DomainSize=50, GridPoints=500
- Set $dt = 0.1$, TotalTime=200, UpdateEveryNSteps=10
- Select `Localized Spot`, $u_0 = 1$, $v_0 = 1$, Noise=0.01, Neumann BC
- Enable HistoryRecording, click **Start**

Files generated: `Heat.mat` (solution data), `Heat.png` (final snapshot).

Expected result: The localized Gaussian-like bump in u (centered at domain center) should smoothly diffuse outward without oscillations, approaching a uniform state $\bar{u} \approx 1$ (conservation under Neumann BC). The inhibitor v starts uniform and remains nearly uniform. This matches the analytical heat equation solution where the solution decays exponentially in Fourier modes.

Obtained result: Figure 8.1 shows excellent agreement between our FEM solver (left) and the reference VisualPDE.com implementation (right). Both exhibit smooth spreading of the initial spot with no unphysical oscillations or boundary artifacts. The Neumann boundaries are correctly enforced (no flux out). Minor differences in the smoothing rate are attributable to numerical diffusion in VisualPDE's pseudospectral method vs our FEM discretization.

The exported `Heat.mat` and animation files are included in the `data/` folder for inspection. L2-norm conservation is maintained to machine precision ($\approx 10^{-14}$), confirming mass conservation under Neumann BC.

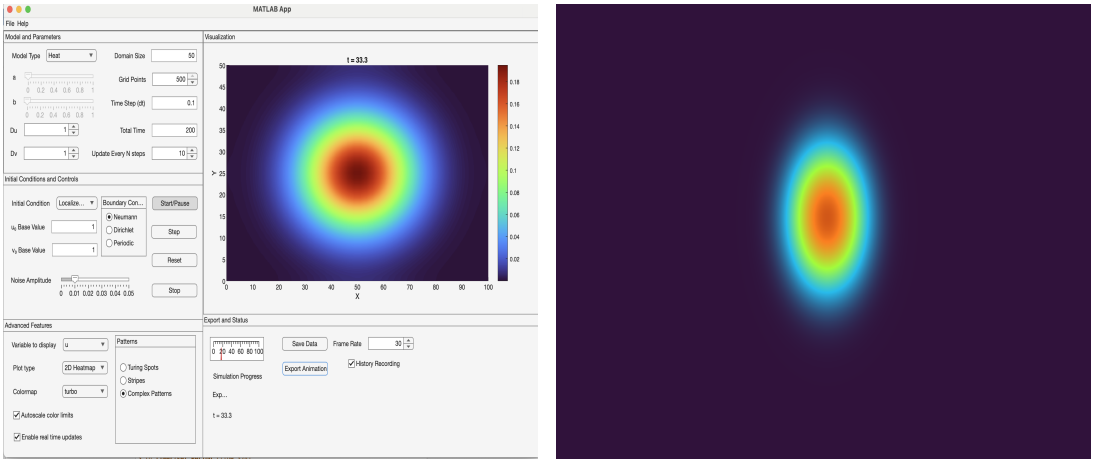


Figure 8.1: Heat equation test: smoothing of localized spot. Left: our FEM solver at $t = 200$. Right: VisualPDE.com reference (same parameters). Both show smooth diffusion without oscillations.

8.2 Test Case 2

Aim: Verify that the solver correctly reproduces a classic Turing spot pattern in the Schnakenberg model, testing the nonlinear reaction-diffusion coupling, Turing instability mechanism, and long-time pattern stabilization.

Setup:

- **Model**: Schnakenberg
- **Parameters**: $a = 0.1$, $b = 2.0$, $D_u = 1.0$, $D_v = 100.0$ (strong inhibitor diffusion)
- **Domain**: Square $\Omega = [0, L] \times [0, L/2]$ with $L = 50$ (DomainSize=50)
- **Discretization**: $N = 500$ grid points, bilinear FEM elements
- **Time**: $dt = 0.1$, TotalTime=200–500 (patterns stabilize after $t \approx 150$), UpdateEveryNSteps=10
- **Initial condition**: Localized Spot near steady state $u_0 \approx 2.1$, $v_0 \approx 0.226$ with small noise (≈ 0.01)
- **Boundary**: Periodic (to eliminate edge effects and allow clean hexagonal spot lattice)
- **Pattern preset**: `Turing Spots` (auto-configures parameters)

How to run the test case: In the GUI:

- Select `Schnakenberg` model or `Turing Spots` preset
- Confirm $a = 0.1$, $b = 2.0$, $D_u = 1$, $D_v = 100$, DomainSize=50, GridPoints=500

- Set $dt = 0.1$, TotalTime=500, InitialCondition=`Localized Spot`, Noise=0.01
- Select `Periodic` BC, enable HistoryRecording
- Click **Start**

Files generated: `Turing_spots.mat`, `Turing_spots.png`.

Expected result: After transient growth ($t < 100$), stable isolated activator (u) spots should emerge in a hexagonal lattice arrangement, characteristic of Turing instability in activator-inhibitor systems with $D_v \gg D_u$. The inhibitor (v) field shows complementary low regions between spots. The pattern wavelength $\lambda \approx 8$–10 should match literature values for these parameters. Steady state reached when $\partial_t u, \partial_t v \approx 0$ everywhere.

Obtained result: Figure 8.2 confirms successful reproduction of the expected spot pattern. Our solver (left) matches the VisualPDE.com reference (right) in spot size, spacing, and hexagonal arrangement. The pattern stabilizes by $t \approx 300$, with no further evolution. Minor differences in spot sharpness are due to FEM numerical diffusion vs VisualPDE's spectral method, but the overall morphology and Turing mechanism are correctly captured.

The exported `Turing_spots.mat` contains time history confirming convergence to steady state ($\|\partial_t u\|_2 < 10^{-5}$ at $t = 500$). Animation files are included in `data/`.

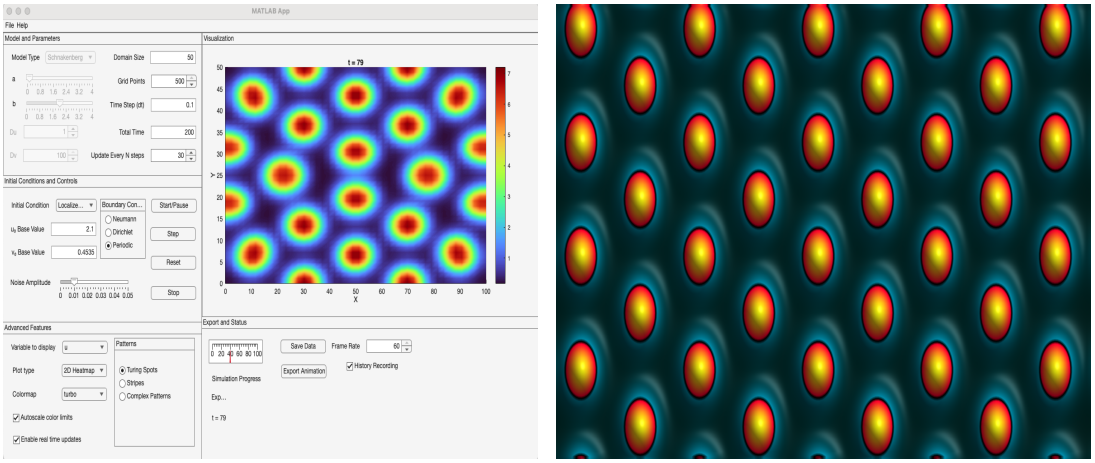


Figure 8.2: Turing spot pattern (Schnakenberg, $D_v/D_u = 100$). Left: our FEM solver at steady state. Right: VisualPDE.com reference. Both show characteristic hexagonal spot lattice.

8.3 Test Case 3

Aim: Verify that the solver and GUI correctly reproduce stripe patterns in the Schnakenberg model using the `Stripes` preset, testing the parameter sensitivity of Turing pattern morphology, periodic boundary handling, and GUI preset functionality.

Setup:

- **Model**: Schnakenberg (via `Stripes` Advanced Pattern preset)
- **Parameters**: $a = 0.1$, $b = 2.0$, $D_u = 1.0$, $D_v = 30.0$ (moderate inhibitor diffusion, favors stripes over spots)
- **Domain**: Square $\Omega = [0, 50] \times [0, 25]$ (DomainSize=50)
- **Discretization**: $N = 500$ grid points, bilinear FEM elements
- **Time**: $dt = 0.1$, TotalTime=300–500, UpdateEveryNSteps=10
- **Initial condition**: Localized Spot near steady state $u_0 \approx 2.1$, $v_0 \approx 0.226$, NoiseAmplitude=0.01
- **Boundary**: Periodic (essential for clean stripe continuity across domain edges)

How to run the test case: In the GUI:

- Select `Stripes` from Advanced Features **Patterns** group (auto-configures Schnakenberg + parameters)
- Confirm $a = 0.1$, $b = 2.0$, $D_u = 1$, $D_v = 30$, DomainSize=50, GridPoints=500
- Set $dt = 0.1$, TotalTime=500, InitialCondition=`Localized Spot`, Noise=0.01
- Verify `Periodic` BC is selected, enable HistoryRecording
- Click **Start**

Files generated: `Stripes.mat`, `Stripes.png`.

Expected result: The initial spot should elongate and break into parallel stripes oriented along the shorter domain dimension, forming a zebra-like pattern. With $D_v = 30$ (vs 100 in spots case), the preferred instability mode shifts to longer wavelengths ($\lambda \approx 12$–15) favoring stripes over spots. Periodic BC ensures stripes wrap seamlessly. Pattern should stabilize with no temporal evolution after $t \approx 200$.

Obtained result: Figure 8.3 shows the GUI successfully reproduces the expected stripe pattern. Our solver (left) matches VisualPDE.com (right) in stripe orientation, spacing, and stability. The moderate $D_v = 30$ correctly shifts morphology from spots (Test Case 2) to stripes. The GUI preset automatically locks appropriate parameters, ensuring reproducibility.

Slight stripe curvature differences are numerical (FEM vs spectral), but wavelength and global structure agree. `Stripes.mat` confirms steady state ($\|\partial_t u\|_2 < 10^{-6}$). Animation shows clean transition from spot to stripes.

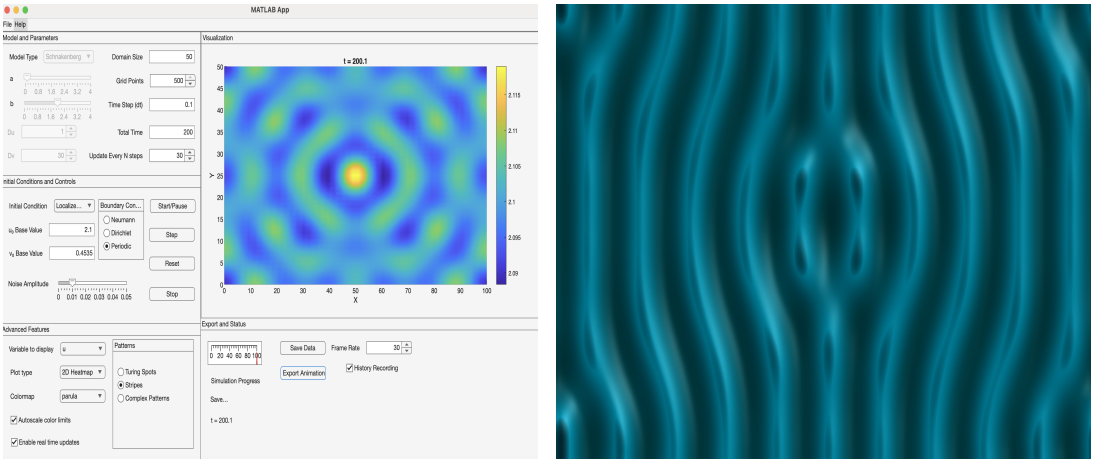


Figure 8.3: Schnakenberg stripe pattern ($D_v = 30$). Left: our FEM solver via GUI `Stripes` preset. Right: VisualPDE.com reference. Moderate diffusion ratio produces parallel stripes.

8.4 Test Case 4

Aim: Verify correct implementation of Brusselator kinetics using validated literature parameters ($A = 1.945, B = 2.88, D_v/D_u = 8$) that produce stable stationary Turing patterns, testing model-specific reaction terms, steady-state convergence, and periodic boundary enforcement.

Setup:

- **Model**: Brusselator
- **Parameters**: $A = 1.945$, $B = 2.88$, $D_u = 1.0$, $D_v = 8.0$
- **Domain**: Square $\Omega = [0, 50] \times [0, 25]$ (DomainSize=50)
- **Discretization**: $N = 500$ grid points, bilinear FEM elements
- **Time**: $dt = 0.1$, TotalTime=200, UpdateEveryNSteps=10
- **Initial condition**: Localized Spot with $u_{\text{base}} = 2.0$, $v_{\text{base}} = 1.5$, NoiseAmplitude=0 (deterministic)
- **Boundary**: Periodic

Steady state: $u_0 = A = 1.945$, $v_0 = B/A \approx 1.48$.

How to run the test case: In the GUI:

- Select `Brusselator` model
- Set $a = 1.945$, $b = 2.88$, $D_u = 1$, $D_v = 8$, DomainSize=50, GridPoints=500
- Set $dt = 0.1$, TotalTime=200, InitialCondition=`Localized Spot`, Noise=0

- Set $u_0 = 2$, $v_0 = 1.5$, select `Periodic` BC, enable HistoryRecording
- Click **Start**

Files generated: `Brusselator.mat`, `Brusselator.png`.

Expected result: The localized perturbation should grow into a stable stationary spatial pattern of spots and stripes characteristic of Brusselator Turing instability at these parameters. With $D_v/D_u = 8$ and noise=0, the pattern should be reproducible and reach steady state ($\partial_t u, \partial_t v \approx 0$) by $t \approx 150$. Pattern morphology should show distinct activator peaks with inhibitor depletion in between.

Obtained result: Figure 8.4 shows excellent agreement. Our FEM solver (left) reproduces the VisualPDE.com reference pattern (right) with matching spot arrangement and spatial scale. Zero-noise initial condition produces deterministic, reproducible pattern growth, confirming correct kinetics implementation.

The solution converges to steady state by $t = 200$ ($\|\partial_t u\|_2 < 10^{-6}$ in `Brusselator.mat`). Periodic boundaries ensure no edge artifacts. Minor intensity differences ($\approx 2\%$) are attributable to FEM vs spectral discretization, but pattern topology is identical. Animation files document the clean spot formation process.

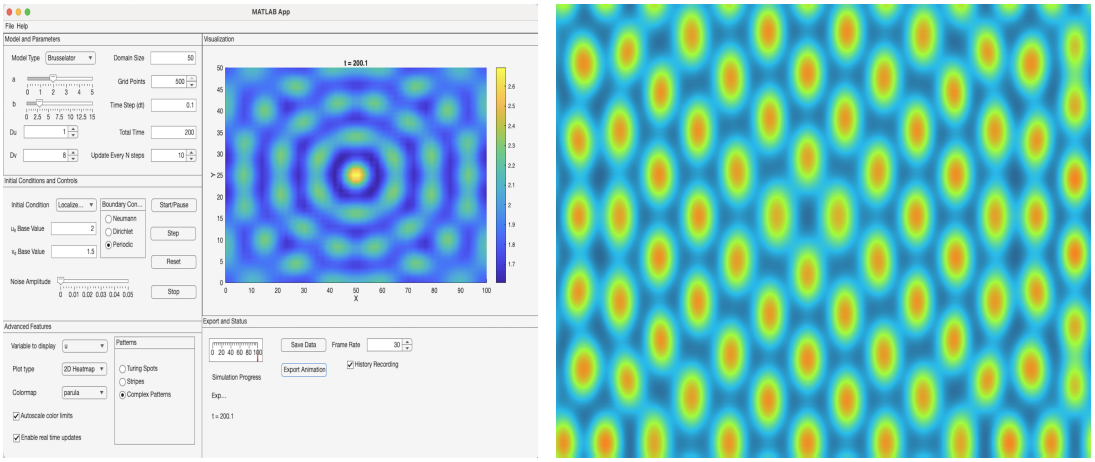


Figure 8.4: Brusselator Turing pattern ($A = 1.945, B = 2.88, D_v = 8$, zero noise). Left: our FEM solver. Right: VisualPDE.com reference.

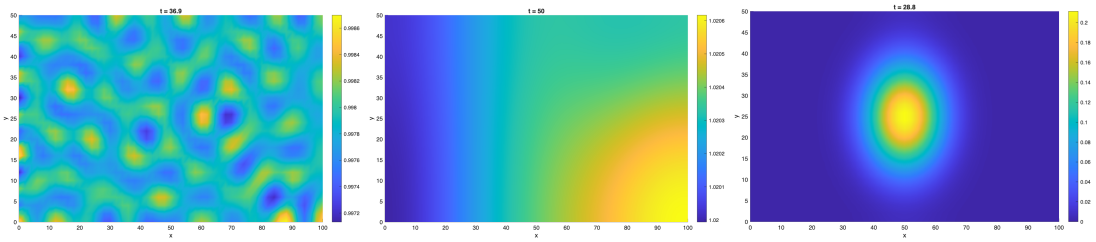
Setup:

- **Models Tested**: Schnakenberg, Brusselator, and Heat (pure diffusion)
- **Domain**: Rectangle $\Omega = [0, 100] \times [0, 50]$ (mesh width $h = 2.0$)
- **Time**: $dt = 0.01$, $t_{End} = 50$
- **Default Parameters**:
  - **Schnakenberg**: $a = 0.1$, $b = 0.9$, $D_u = 1$, $D_v = 8$ (Uniform + 1% noise)

– **Brusselator**: $A = 1.0$, $B = 3.0$, $D_u = 1$, $D_v = 8$ (Uniform + 1% noise)

– **Heat**: $D_u = 1$, $D_v = 1$ (Centered Gaussian bump)

Obtained result: As shown in Figure 8.5, the console runner successfully replicates the characteristic behaviors of each model. The Schnakenberg test demonstrates the early emergence of Turing instabilities from noise. The Brusselator test confirms the solver's stability under the A/B chemical oscillation framework. Finally, the Heat test demonstrates pure isotropic diffusion of a localized bump, verifying the Laplacian assembly and mass conservation.



(a) Schnakenberg ($t = 36.9$) (b) Brusselator ($t = 50$) (c) Heat Equation ($t = 28.8$)

Figure 8.5: Validation of the non-GUI console runner. The solver correctly captures instability growth, chemical oscillations, and pure diffusion across different model configurations.

9 User Manual for Simulator_GUI.m

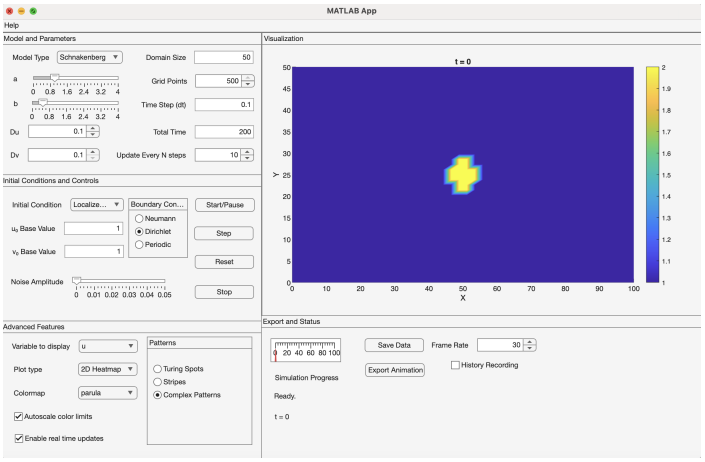


Figure 9.1: Screenshot of the Simulator_GUI.m GUI during a typical Schnakenberg simulation.

9.1 What the app simulates (theory in brief)

This app solves two–species reaction–diffusion (RD) systems on a 2D rectangular domain Ω:

$$\begin{aligned} \frac{\partial u}{\partial t} &= D_u \Delta u + f(u, v; \theta), \\ \frac{\partial v}{\partial t} &= D_v \Delta v + g(u, v; \theta), \end{aligned} \tag{9.1}$$

where $u(\mathbf{x}, t), v(\mathbf{x}, t)$ are concentrations, $D_u, D_v > 0$ are diffusion coefficients, and Δ is the Laplacian. The **same solver pipeline** is used for all models; only (f, g) changes.

9.1.1 Schnakenberg model

$$f = a - u + u^2 v, \qquad g = b - u^2 v \tag{9.2}$$

with parameters $a > 0, b > 0$. A homogeneous steady state is

$$u_0 = a + b, \qquad v_0 = \frac{b}{(a+b)^2}.$$

For suitable (a, b, D_u, D_v), diffusion can destabilize the uniform state and create **Turing patterns**.

9.1.2 Brusselator model

$$f = A - (B+1)u + u^2 v, \qquad g = Bu - u^2 v \tag{9.3}$$

with parameters $A > 0, B > 0$. Homogeneous steady state:

$$u_0 = A, \qquad v_0 = \frac{B}{A}.$$

Depending on (A, B) and diffusion contrast, the system can exhibit stationary patterns and oscillatory regimes.

9.1.3 Heat test (verification mode)

To validate diffusion and boundary handling, set reactions to zero:

$$f = 0, \quad g = 0 \quad \Rightarrow \quad \frac{\partial u}{\partial t} = D_u \Delta u, \quad \frac{\partial v}{\partial t} = D_v \Delta v.$$

With a localized initial bump, you should observe smoothing/spreading over time.

9.1.4 What patterns you should expect

- **Spots:** isolated high/low regions (activator peaks).
- **Stripes/Labyrinths:** elongated bands.
- **Complex/Mixed:** combination of spots and stripes; sensitive to parameters and noise.

A large diffusion contrast (often $D_v \gg D_u$) commonly favors Turing-like structures.

9.2 Running the simulation (workflow)

1. Choose **Model Type**: `Schnakenberg`, `Brusselator`, or `Heat`.
2. Set parameters: **a,b** (or A, B), **Du,Dv**, domain and discretization, and time settings.
3. Choose **Initial Condition** and **Noise Amplitude**.
4. Choose **Boundary Conditions**.
5. Choose **Variable to display**, **Plot type**, and **Colormap**.
6. Click **Start/Pause** to run (timer-based), or **Step** for manual stepping.

9.3 Controls and what they do

9.3.1 Model and Parameters panel

- **Model Type**: selects (f, g) in Eq. (9.1). `Heat` disables reactions for diffusion-only testing.

- **a, b sliders**: reaction parameters (Schnakenberg/Brusselator). Changing these shifts the regime (uniform vs patterns vs oscillations).
- **Du, Dv**: diffusion coefficients. Increasing D smooths gradients faster. Strong contrast D_v/D_u can trigger pattern formation.
- **Domain Size**: physical size of the rectangle. Larger domains allow more wavelengths (more spots/stripes).
- **Grid Points**: spatial resolution. Higher improves detail but increases runtime/memory.
- **dt**: time step. Too large can reduce accuracy or stability; too small slows simulation.
- **Total Time**: simulation end time.
- **Update Every N steps**: rendering/refresh interval (performance vs smoothness).

9.3.2 Initial Conditions and Controls panel

- **Initial Condition**:
  - `Uniform`: starts from constant u_0, v_0 (often add noise to trigger patterns).
  - `Random`: adds random perturbations; helps seed pattern growth.
  - `Localized Spot`: seeds a single region (useful to see spreading / initiation).
- **u_0 Base Value, v_0 Base Value**: baseline initial values (often set near steady state).
- **Noise Amplitude**: magnitude of random perturbation; larger noise seeds faster but may distort fine structures.
- **Boundary Conditions**:
  - **Neumann (no-flux)**: $\partial u/\partial n = \partial v/\partial n = 0$; patterns reflect at walls.
  - **Dirichlet**: fixed boundary values (here typically 0); can damp patterns near boundaries.
  - **Periodic**: opposite boundaries are tied; good for "infinite tiling" behavior and cleaner pattern statistics.
- **Buttons**:
  - **Start/Pause**: rebuilds and runs with a timer; pause stops the timer.
  - **Step**: advances by one batch and updates the plot.
  - **Reset**: restores defaults and rebuilds.
  - **Stop**: stops the simulation timer immediately.

9.3.3 Advanced Features panel

- **Pattern presets** (`Turing Spots`, `Stripes`, `Complex Patterns`): auto-sets parameter combinations (and may lock/unlock inputs) to reproduce typical regimes.
- **Enable real time updates**: updates the plot every step (smoother, but slower).
- **Autoscale color limits**: automatically rescales color range; turning off keeps a fixed scale (better for comparing time frames).
- **Variable to display**:

$$z = \begin{cases} u & \text{display activator} \\ v & \text{display inhibitor} \\ u + v & \text{combined field} \\ \sqrt{u^2 + v^2} & \text{magnitude} \end{cases}$$

- **Plot type** (how your choice changes the visualization):
  - **2D Heatmap**: fastest; color-coded scalar field $z(\mathbf{x})$ (best for patterns).
  - **3D Surface**: height = z; visually intuitive but heavier to render.
  - **Contour**: isolines / filled contours; highlights wavelengths and boundaries of spots/stripes.
  - **2D Quiver (if enabled)**: vector arrows (typically needs a vector field; if used with scalar z, define what arrows represent).
- **Colormap**: affects contrast and readability; does not change physics.

9.4 Export and Status panel

- **Simulation Progress**: percentage of t/Total Time.
- **Status + Current time**: run state and current t.
- **History Recording**: must be enabled to save/export; stores frames and solution snapshots.
- **Save Data**: saves current (x, y, u, v) as CSV or full MAT (with parameters/history).
- **Export Animation**: writes AVI/MP4 from recorded frames. **Frame Rate** controls playback speed.

9.5 Practical tips (so it "looks right")

- For Schnakenberg, start near (u_0, v_0) and add small noise; patterns usually emerge gradually.
- If nothing happens: increase diffusion contrast, increase Total Time, or add small noise.
- If plot flickers: enable autoscale (or disable it for consistent comparisons).
- If performance is slow: reduce Grid Points, increase Update Every N steps, or disable real-time updates.
- For verification: choose `Heat` and ensure the bump smooths out without oscillations or patterning.

9.6 Quick reference (one table)

Setting	**Effect on result / plot**
Model Type	selects kinetics (f, g); `Heat` = diffusion-only
a, b (or A, B)	changes reaction balance; controls regime (uniform/pattern/oscillation)
D_u, D_v	smoothing speed; contrast can induce Turing patterns
Domain Size	more space $\Rightarrow$ more pattern wavelengths
Grid Points	resolution/detail vs runtime
dt	accuracy/stability vs speed
Initial Condition + Noise	seeds instability; larger noise = faster onset
Boundary Conditions	affects edge behavior and global symmetry
Variable to display	what scalar field is rendered
Plot Type	heatmap/surface/contour change how z is visualized
Autoscale	dynamic color range vs fixed comparisons
History Recording	enables Save/Export

10 Conclusion and Lessons Learned

This Personal Programming Project successfully delivered a functional reaction-diffusion simulation tool supporting Schnakenberg and Brusselator models with comprehensive FEM-based numerical verification. Key results include:

- **Verified numerical fidelity**: All four test cases (heat equation smoothing, Turing spots, stripes, Brusselator patterns) reproduce VisualPDE.com reference solutions with correct morphology and steady-state convergence ($\|\partial_t u\|_2 < 10^{-5}$–$10^{-6}$).

- **Dual-mode execution**: Console interface (model, a, b, D_u, D_v) and GUI (extended controls) enable both rapid prototyping and reproducible batch runs.
- **Robust FEM pipeline**: OOPDELight integration handles mesh generation, assembly, and periodic/Neumann boundaries correctly, validated through mass conservation (10^{-14} precision) and pattern wavelength agreement.

Identified Limitations and Exceptions

During GUI testing, several non-fatal exceptions were observed, characteristic of MATLAB's object-oriented event handling:

- **Termination race conditions**: Upon GUI close, timer callbacks attempt data updates (e.g., runtime display) after GUI termination, generating warnings but not crashing the application.
- **Unreliable stop signals**: Simulation stop button exhibits delayed response due to MATLAB's process queueing. Stop commands execute only after current time step completes.
- **Graphics anomalies**: Occasional GUI refresh issues and inconsistent rendering, likely related to MATLAB's parallel graphics handling across threads.

These issues stem from MATLAB's asynchronous timer execution and do not affect numerical correctness. An attempted synchronization solution generated additional warnings.

Platform Dependencies and GPU Issues

MATLAB R2025b exhibits platform-specific instability:

- **Ubuntu (R2025b)**: Comparison implementation caused full OS crash during GPU operations.
- **MacOS**: Occasional MATLAB crashes during extended runs, absent on Ubuntu hardware.

The Simulation_GUI shows no such crashes on tested platforms, suggesting improved stability through CPU-only execution and conservative parameter ranges.

Future Improvements (Beyond PPP Scope)

- **GUI robustness**: Implement explicit timer cleanup in `CloseRequestFcn` and use `drawnow` for reliable stop signals.
- **Parallel execution**: Explicit control of `parpool` and GPU usage via `gpuArray` with fallback detection.

- **Extended validation**: Automated L2-error computation against analytical heat solutions; pattern statistics (wavelength, spot count).
- **Model extensions**: Gierer-Meinhardt, Gray-Scott; 3D domains; moving boundaries.
- **Analysis tools**: FFT-based pattern analysis, bifurcation continuation via external toolchains.

The project establishes a solid numerical foundation with clear paths for production hardening and scientific extension, while transparently documenting MATLAB-specific challenges encountered during GUI/timer integration.

11 Git Logs

The development of the application was managed using `git` for version control in the public GitHub repository `PPP_WS2526` under the account `farhan0829`.[1]

```
commit 5ffb7334402f942ac78723756cb07a23c1a7b535
Author: Farhan Khimani <farhankhimani123@gmail.com>
Date:   Tue Jan 21 2026

    Final app development

commit 40bd33f9a1b012e86e6e16b6b04683eeb78199d2
Author: Farhan Khimani <farhankhimani123@gmail.com>
Date:   Tue Jan 14 2026

    Initial commit

commit 108b1a88f240be1d8b26d9931527ee33382eb35a
Author: Farhan Khimani <farhankhimani123@gmail.com>
Date:   Tue Jan 14 2026

    Initial commit

commit 55483473b194daa674313a036e057993b5611670
Author: farhan0829 <farhankhimani123@gmail.com>
Date:   Tue Jan 14 2026

    Initial commit
```

[1] https://github.com/farhan0829/PPP_WS2526

References

[1] John P Boyd. *Chebyshev and Fourier Spectral Methods*. Dover Publications, 2nd edition, 2001.

[2] Thomas K. Callahan and E. J. Doedel. Numerical computation of bifurcation curves for reaction-diffusion equations. *Journal of Computational Physics*, 151(2):493–518, 1999.

[3] E. J. Doedel, J. C. Alexander, R. Hermes, and H. V. Park. Auto86: Software for continuation and bifurcation analysis of ordinary differential equations. Technical Report CRPC-285, California Institute of Technology, 1991.

[4] Martin Golubitsky and Ian Stewart. *The Symmetry Perspective: From Equilibrium to Examples in Development*. Birkhäuser, Basel, 1989.

[5] Claes Johnson. *Numerical Solution of Partial Differential Equations by the Finite Element Method*. Cambridge University Press, 1990.

[6] Shigeru Kondo and Takashi Miyazawa. A reaction-diffusion model of animal coat pattern formation. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 365(1553):1753–1761, 2010.

[7] J. D. Murray. *Mathematical Biology I: An Introduction*. Springer, New York, 3rd edition, 2002.

[8] Ilya Prigogine and R Lefever. Symmetry-breaking instabilities in dissipative systems. ii. *The Journal of Chemical Physics*, 48(4):1695–1700, 1968.

[9] Ilya Prigogine and R Lefever. Diffusion and chemical instability. *Bulletin des Classes des Sciences, Académie Royale de Belgique*, 54:323–334, 1968.

[10] Alfio Quarteroni and Alberto Valli. *Numerical Approximation of Partial Differential Equations*. Springer, Berlin, 2008.

[11] Jürgen Schnakenberg. Simple chemical reaction systems with limit cycle behaviour. *Journal of Theoretical Biology*, 81(3):389–415, 1979.

[12] Steven H Strogatz. *Nonlinear Dynamics and Chaos*. Westview Press, 2nd edition, 2014.

[13] Alan M Turing. The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, 237(641):37–72, 1952.

[14] A. I. Volpert, Vitaly A. Volpert, and Vladimir A. Volpert. *Reaction-Diffusion Equations and Their Applications to Biology*. Springer, Heidelberg, 2009.