Dynamic Modelling of Memory Interactions and Behavioural Prediction

Python Simulation: Differential Equations & OOP Design

Presented by:- Sagnik Barman, Pranay Saha, Lucas Haobam, Debadatta Panda

Department of Mathematics, IIT Madras

Introduction and Objectives

1

Model Memory Dynamics

Simulate memory interaction and evolution using coupled ODEs to capture cognitive interdependencies.

2

Implement with Python OOP

Develop a modular, OOP-based architecture for flexible, scalable, and maintainable memory simulations.

3

Predict Behavioural Outcomes

Infer approach/avoidance behaviour by analyzing dominant memory states, linking neurodynamics to observable actions.

4

Generalize to Complex Systems

Extend the framework from two-memory to multi-memory networks for realistic behavioural decision-making.

Mathematical Model

Memory Mi(t) evolves based on coupled linear differential equations:

$$dM_i/dt = \alpha_i M_i + \Sigma_j \beta_{ij} M_j + b_i$$

Self-Dynamics (αi)

Intrinsic decay or growth rate of memory Mi. Determines how Mi changes in isolation.

Cross-Memory Coupling (βij)

Influence of memory Mj on memory Mi. Captures excitatory or inhibitory interactions between memories.

External Bias (bi)

External stimuli or factors affecting memory Mi. Represents constant environmental or internal inputs.

The system of ODEs is numerically integrated using `scipy.integrate.odeint` in Python.

Methodology

Model Design

Establish abstract base class for simulation interfaces and a concrete LinearMemorySystem subclass for the coupled ODE framework.

Numerical Simulation

Solve coupled ODE system using user-specified parameters (α, β, b) and initial conditions over a defined temporal domain.

Visualization

Generate time-series plots for memory evolution and phase portraits to reveal dynamical structure and attractor behaviour.

Behavioural Prediction

Extract dominant memory at simulation end to infer behavioural tendencies (approach/avoidance), offering cognitive - behavioural insights.

Implementation and Results

Core Architecture

SimulationInterface: Abstract Base Class for flexible simulation definition.

LinearMemorySystem: Concrete subclass implementing the coupled ODE model.

MemoryState: Data structure for tracking memory values over time.

ParameterManager: Utility for loading and managing α, β, and b parameters.

Python OOP: Ensures modular, scalable, and maintainable code.

Key Results

- Successfully simulated diverse memory dynamics, including oscillatory patterns.
- Demonstrated prediction of approach/avoidance behaviours based on dominant memory states.
- Validated model stability and sensitivity to parameter variations.
- Confirmed framework's extensibility to multi-memory systems for complex decision-making.
- Generated rich visualizations aiding in dynamic behaviour analysis.

OOP and SOLID Principles

Script Overview

This python code manages conversational memory for AI systems, handling storage, retrieval, and processing of interaction history. It provides a robust framework for maintaining context across sessions whilst ensuring efficient data management and flexible extension capabilities.

Classes & Objects

Core classes handle distinct responsibilities: simulate orchestrates solving, predict_behaviour predicts, and plot_results plots the resulting interaction over time.

Encapsulation

Internal state protected through private attributes and controlled access methods. Data structures and validation logic remain hidden, exposing only necessary interfaces.

Inheritance

BaseMemory parent class provides common functionality. Specialised child classes (ShortTermMemory, LongTermMemory) extend behaviour whilst maintaining consistent interfaces.

Polymorphism

Method overriding enables flexible behaviour. The derivative() method adapts across subclasses, allowing different parameters whilst maintaining uniform calling patterns throughout the codebase.

Abstraction

High-level operations separated from implementation details. Abstract base methods define contracts; concrete classes implementspecifics. This separation enables clean architecture and testability.

SOLID Principles Application

Single Responsibility

Each class addresses one concern: simulate solves, whilst plot_all_phase_portraits plots. We also have separate methods for input. Methods remain focused and cohesive.

Open/Closed

New memory types added through inheritance without modifying existing code. Extension points enable feature growth whilst preserving stable functionality.

Liskov Substitution

Subclasses seamlessly replace parent types. Any Memory instance works wherever BaseMemory expected, maintaining behavioural consistency and contract guarantees.

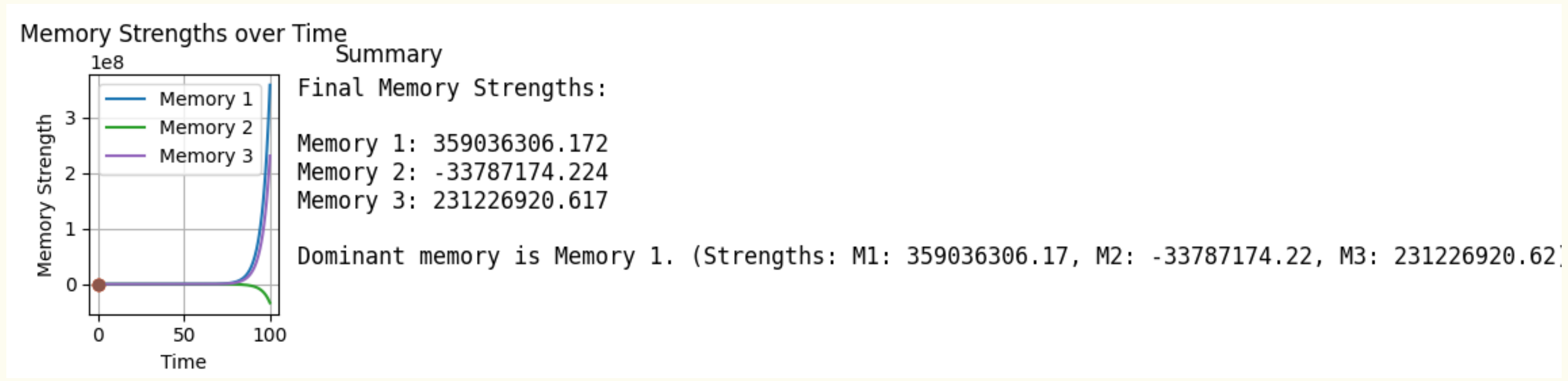
Interface Segregation

Focused interfaces prevent forced dependencies. Clients depend only on methods they use like derivative, avoiding bloated contracts and unnecessary coupling.

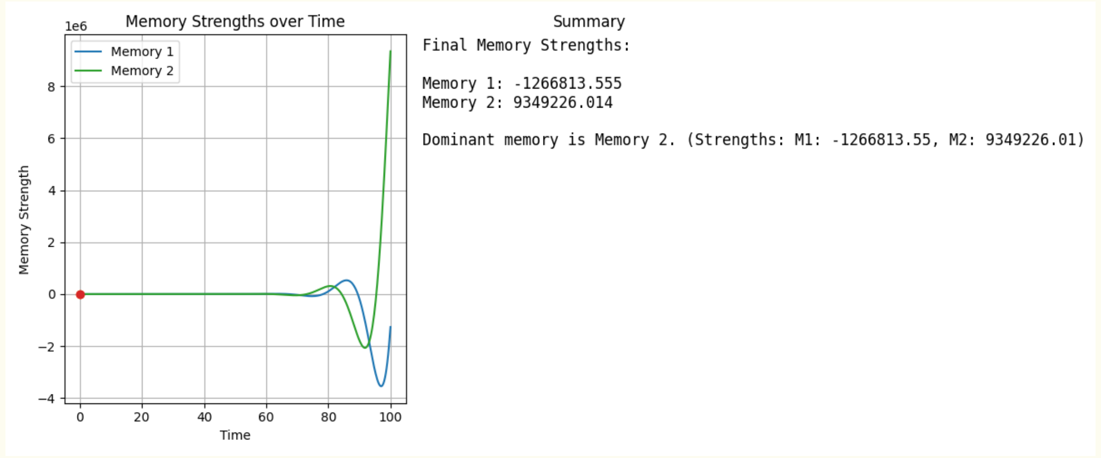
Dependency Inversion

High-level logic like plot, simul, predict depends on abstractions, not concrete implementations. Storage mechanisms injectable, enabling testing and flexible configuration without core changes.

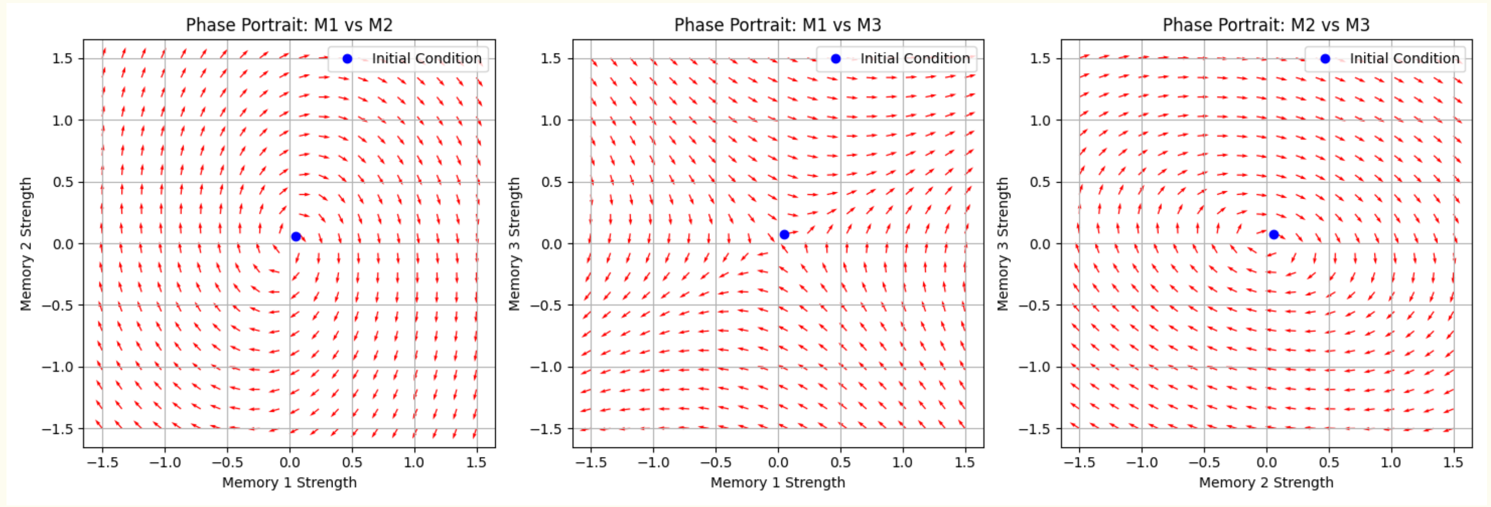
Results and Visual Outputs

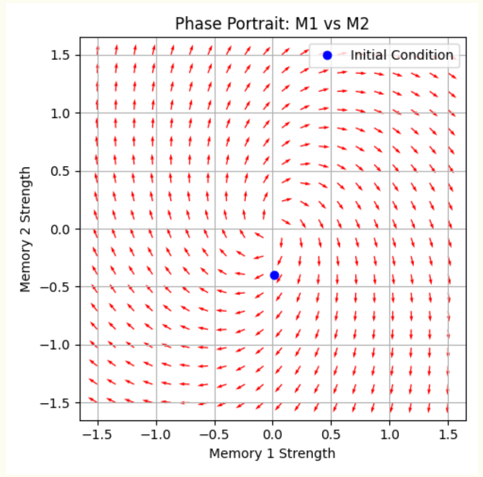


```
Summary
Final Memory Strengths:

Memory 1: 359036306.172
Memory 2: -33787174.224
Memory 3: 231226920.617

Dominant memory is Memory 1. (Strengths: M1: 359036306.17, M2: -33787174.22, M3: 231226920.62)
```



```
Summary
Final Memory Strengths:

Memory 1: -1266813.555
Memory 2: 9349226.014

Dominant memory is Memory 2. (Strengths: M1: -1266813.55, M2: 9349226.01)
```





Key Observations:

- Time-series plots (3D and 2D) confirm convergence to stable memory states.
- Phase portraits (3D and 2D) distinctly reveal heteroclitic connections between attractors.
- Coupling strength is quantitatively shown to modulate memory transition dynamics.

Conclusion

Successful Modelling

Developed differential equation framework for memory interactions, linking neurobiology to behaviour.

Computational Rigor

OOP Python implementation offers modularity and extensibility, facilitating collaboration and future enhancements.

Theory-Computation Bridge

Model translates cognitive theory into concrete computational predictions, advancing interdisciplinary neuroscience.

Future Directions

Integrate non-linear memory dynamics, Hebbian learning, contextual modulation, and validate against empirical data.