Dynamic Modeling of Memory Interactions and Behavioral Prediction

A Python-Based Simulation Using Differential Equations and OOP Design

Submitted By:

Sagnik Barman

Pranay Saha

Lucas Haobam

Debadatta Panda

Department of Mathematics

IIT Madras

November 6, 2025

Abstract

This project presents a computational model that simulates the dynamic interaction between multiple memories and their influence on human behavior. The system employs a set of coupled ordinary differential equations (ODEs) to represent memory decay and mutual interaction effects. Implemented in Python using an object-oriented paradigm, the model allows user-defined parameters and visualizes memory evolution through time-series plots and phase portraits. The simulation predicts behavioral tendencies, such as approach or avoidance, based on dominant memory strengths at the end of the simulation.

Contents

Chapter 1

Introduction

Understanding how memories interact and influence behavior is a key problem in the field of computational neuroscience and psychology. This project models memory systems as interacting dynamic entities, governed by mathematical equations that represent biological processes such as decay, reinforcement, and cross-influence.

The objective is to bridge cognitive theory and computational modeling to predict behavioral outcomes like approach or avoidance tendencies based on the relative dominance of specific memory states.

Chapter 2

Project Objectives

- To model the dynamic evolution of multiple interacting memory states using differential equations.
- To implement the model using Python's object-oriented programming (OOP) paradigm for scalability and clarity.
- To simulate and visualize memory trajectories over time.
- To predict behavior (e.g., approach vs. avoidance) from final memory strengths.
- To generalize the model for two or more interacting memory systems.

Chapter 3

Mathematical Model

Each memory $M_i(t)$ evolves over time according to:

$$\frac{dM_i}{dt} = \alpha_i M_i + \sum_{j=1}^{n} \beta_{ij} M_j + b_i$$

where:

- α_i: self-dynamics (decay or reinforcement rate) of memory i
- β_{ij}: influence of memory j on memory i
- b_i: bias term (external influence)
- n: total number of memories

The system is numerically integrated over time using `scipy.integrate.odeint`, and the resulting trajectories are analyzed to determine which memory becomes dominant.

Chapter 4

Methodology

4.1 1. Model Design

The model is designed using an abstract base class `AbstractMemorySystem` that defines:

- Parameter initialization
- Derivative computation (abstract)
- Simulation using numerical integration
- Behavior prediction logic

A concrete subclass `LinearMemorySystem` implements the linear ODE version of the model.

4.2 2. Numerical Simulation

- The system of ODEs is solved over time with a fixed time vector.
- The user inputs the number of memories, decay rates, interaction matrix, and initial conditions.
- The model integrates these values and computes trajectories for each memory.

4.3 3. Visualization

- Time-series plots show the evolution of each memory.
- Phase portraits visualize pairwise interactions between memories as vector fields.

4.4 4. Behavioral Prediction

At the end of the simulation, the dominant memory (i.e., the one with the highest strength) is identified. The output predicts:

- **Approach behavior:** if positive or rewarding memory dominates.
- **Avoidance behavior:** if stressful or negative memory dominates.

Chapter 5

Implementation in Python

5.1 Core Code

Listing 5.1: Core implementation of the memory system model.

```
class LinearMemorySystem(AbstractMemorySystem):
    def __init__(self, alpha, beta, **kwargs):
        super().__init__(alpha, beta, **kwargs)

    def derivatives(self, M, t):
        M = np.array(M)
        dMdt = self.alpha * M + self.beta @ M + self.bias
        return dMdt.tolist()
```

5.2 Simulation Execution

[1, 2] The user provides input parameters, which are used to create a `LinearMemorySystem` instance. The program then:

1. Runs the simulation (`simulate`)
2. Predicts behavior (`predict_behavior`)
3. Generates time and phase plots

5.3 Visualization Functions

- `plot_results`: Shows time evolution of all memories.
- `plot_all_phase_portraits`: Displays all pairwise vector field interactions.

Chapter 6

Results and Discussion

The simulation produces:

- Time-series plots of memory evolution.
- Phase portraits for each memory pair.
- Printed final memory strengths and predicted dominant behavior.

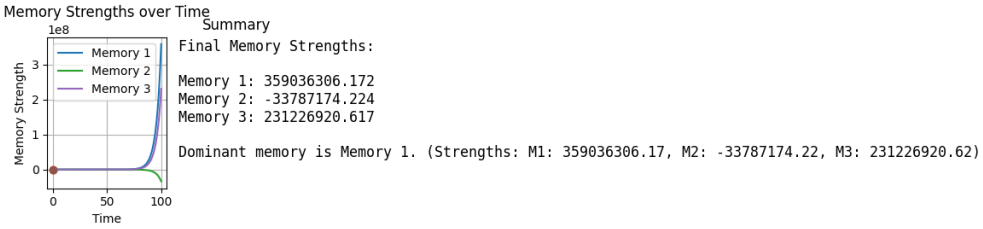


Figure 6.1: Phase portrait plot of memory strength (3) evolution over time.

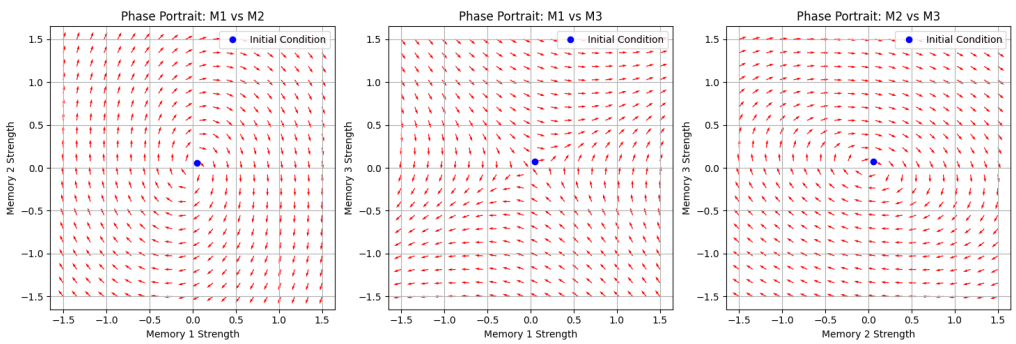


Figure 6.2: Phase portrait of different memory interactions.[3]

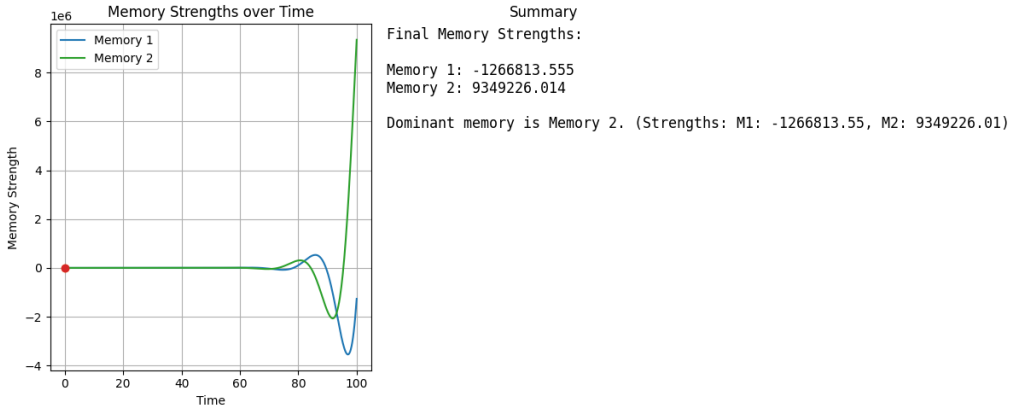


Figure 6.3: Phase portrait of different memory interactions.

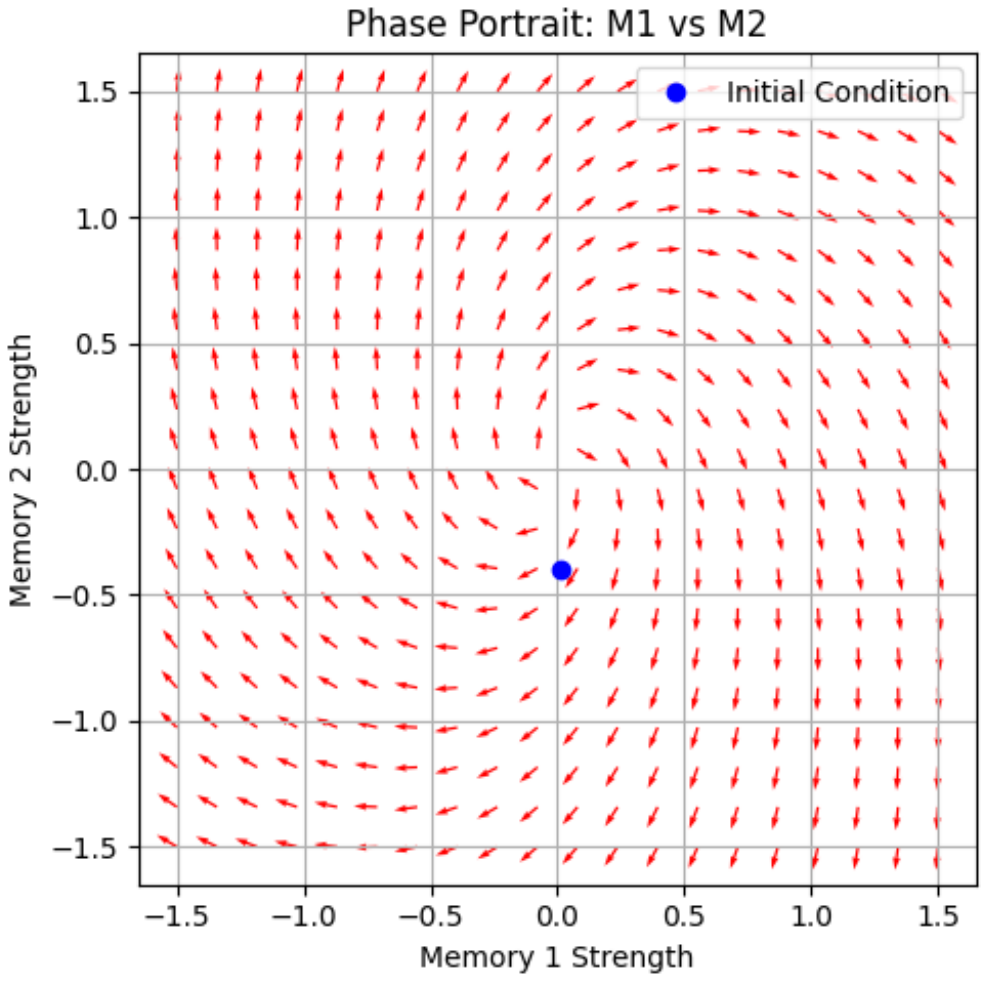


Figure 6.4: Example plot of memory strength evolution over time.[4]

From the results, one can observe [5] which memory state stabilizes at a higher strength, indicating its dominance and the corresponding behavioral tendency.

Chapter 7

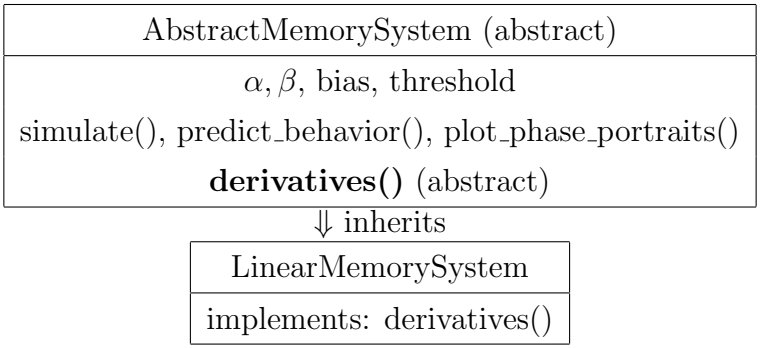
Model Class Design

The code was written using object oriented design.

7.1 OOP Concepts used

- **Abstraction** — we created an abstract class that expresses the concept "memory system" without implementation details.
- **Inheritance** — `LinearMemorySystem` inherits from `AbstractMemorySystem`. We reuse everything and only override the derivative logic.
- **Polymorphism** — `derivatives()` is defined ABSTRACT in the base class and concretely implemented in subclasses. Two different memory systems could implement completely different dynamics while the `simulate()` function remains unchanged.
- **Encapsulation** — we bundle *alpha*, *beta*, time simulation and plotting behavior inside a single object. Internal state is hidden inside that instance.

7.2 Class Diagram (conceptual)



Chapter 8

Conclusion

This project successfully demonstrates how memory interactions can be mathematically and computationally modeled using differential equations. The use of OOP ensures scalability for simulating systems with multiple interacting memories. By connecting cognitive theory, numerical modeling, and behavioral inference, this work provides a foundation for further studies in cognitive neuroscience and adaptive behavioral systems.

8.1 Conclusion

This architecture makes it easy to later introduce:

- non-linear memory dynamics
- learning / reinforcement
- context dependent gates

And the OOP structure allows all of this without touching the simulation workflow.

Bibliography

[1] "Mit opencourseware: System dynamics self-study."

[2] "Computational models of memory," 2021.

[3] "System dynamics society: What is system dynamics?."

[4] "Anylogic big book on simulation modeling."

[5] "Iisc: Neurobiology of learning and memory."