

# Business-IT Alignment Dynamics: A Chaotic Systems Approach

Alessandro Aquilini

May 5, 2025

## Abstract

Lorem Ipsum

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>User Requirements</b>	<b>2</b>
2.1	Target Audience . . . . .	2
2.2	Getting Started . . . . .	2
2.2.1	Installation . . . . .	2
2.2.2	Running the Notebook . . . . .	2
<b>3</b>	<b>Model Description</b>	<b>3</b>
3.1	Core Equation . . . . .	3
3.2	Component Functions . . . . .	3
3.3	Parameter Definitions . . . . .	3
3.4	Interpretation . . . . .	3
3.5	Parameter Analysis . . . . .	4
3.5.1	Environmental Pressure Function . . . . .	4
3.5.2	IT Department Efficacy . . . . .	4
3.5.3	Organizational Adaptability . . . . .	4
<b>4</b>	<b>Implementation</b>	<b>6</b>
4.1	Technological Stack . . . . .	6
4.2	Key lines of code . . . . .	6
4.2.1	Simulating the equation . . . . .	6
4.2.2	Long term behavior . . . . .	6
4.2.3	Phase Portrait Analysis . . . . .	7
4.2.4	Bifurcation Analysis . . . . .	7
4.3	Numerical Considerations . . . . .	7
<b>5</b>	<b>Results</b>	<b>8</b>
5.1	Interactive Tools . . . . .	8
5.1.1	Time Evolution Simulation . . . . .	8
5.1.2	Phase Portrait Analysis . . . . .	8
5.1.3	Bifurcation Diagram . . . . .	9
5.2	Examples . . . . .	9
<b>A</b>	<b>Appendix: Complete Python Code</b>	<b>9</b>

# 1 Introduction

## 2 User Requirements

### 2.1 Target Audience

This interactive notebook is designed for anyone interested in Business-IT alignment dynamics. It also serves as template for modeling other complex dynamics using discrete time equations in the form:

$x_{t+1} = x_t + \Delta(t, \text{*args})$ . The user just needs to apply a few minor customization to the code.

### 2.2 Getting Started

#### 2.2.1 Installation

1. Clone the repository:

```
git clone https://github.com/Kinshale/pii.git
cd pii
```

2. Install required packages:

```
pip install numpy matplotlib ipywidgets
```

#### 2.2.2 Running the Notebook

Choose your preferred environment:

- **Local Jupyter:** Launch Jupyter Notebook and open `pii.ipynb`
- **VS Code:** Open the notebook with Jupyter extension
- **Google Colab:**
  1. Visit <https://colab.research.google.com>
  2. Select "GitHub" tab and paste the notebook URL
  3. Or open the cloned one

For optimal experience:

- Start with default parameters to observe baseline behavior
- Modify one parameter at a time to understand its effect
- Use the bifurcation tool to identify chaotic parameter regions

### 3 Model Description

#### 3.1 Core Equation

The alignment dynamics are governed by:

$$x_{t+1} = x_t + A(x_t) - B(x_t)C(x_t) \quad (1)$$

Where:

- $x_t$ : Percentage of dissatisfied users (misalignment proxy)
- $A(x_t)$ : Environmental pressure effect
- $B(x_t)$ : IT department efficacy
- $C(x_t)$ : Organizational adaptability

#### 3.2 Component Functions

$$A(x_t) = d(1 - x_t) \quad (2)$$

$$B(x_t) = \frac{ax_t(1 - x_t)^g}{1 + ahx_t} \quad (3)$$

$$C(x_t) = \frac{1}{1 + z^s} \quad \text{where} \quad z = \frac{r(1 - x_t)}{x_t(1 - r)} \quad (4)$$

#### 3.3 Parameter Definitions

Parameter	Description	Range	Default
$x_0$	Initial misalignment	[0.01, 0.99]	0.3
$d$	Environmental dynamicity	[0.01, 5]	0.5
$a$	IT department efficacy	[0.1, 10]	2
$h$	IT system rigidity	[0.1, 5]	1
$g$	IT investment propensity	[0.1, 5]	1
$r$	Action threshold	[0.01, 0.99]	0.3
$s$	Organizational flexibility	[1, 10]	3

Table 1: Model Parameters and Ranges

#### 3.4 Interpretation

Let's take a deeper look at our equation:

$$x_{t+1} = x_t + \underbrace{A(x_t)}_{\text{Environmental Pressure}} - \underbrace{B(x_t)C(x_t)}_{\text{Recovery Mechanism}} \quad (5)$$

$x_t$  – **Alignment** Measures the percentage of dissatisfied users at time  $t$ :

- $0 \rightarrow$  Complete satisfaction (perfect alignment)
- $1 \rightarrow$  Utter dissatisfaction (total misalignment)

$A(x_t)$  – **Environmental Pressure**: Increases misalignment due to external factors, representing how competitive environments and technological changes increase dissatisfaction.

$B(x_t) \cdot C(x_t)$  – **Recovery Mechanism**: Reduces misalignment through:

- $B(x_t)$ : IT department's effectiveness.
- $C(x_t)$ : Organization's adaptability.

### 3.5 Parameter Analysis

Full interactive simulation available at: <https://www.desmos.com/calculator/qdendm1pfg>

#### 3.5.1 Environmental Pressure Function

$$A(x_t) = d(1 - x_t) \quad (6)$$

Forms a line passing through  $(1, 0)$  with slope  $-d$ .

- **$d$  (dynamicity)**: Fast changing industries (e.g., a tech startup) have a competition/innovation that rapidly renders old IT systems obsolete.
- $1 - x_t$ : As misalignment grows, environmental pressure has less "room" to worsen things.

#### 3.5.2 IT Department Efficacy

$$B(x_t) = \frac{ax_t(1 - x_t)^g}{1 + ahx_t} \quad (7)$$

This looks like a function that peaks at some  $x_a$  and then tapers off.

- **$a$  (IT efficacy)**: an higher  $a$  can more effectively reduce misalignment.
- **$x$  (current misalignment)**: the more misalignment exists, the more opportunity/pressure there is for IT to act.
- **$(1 - x_t)^g$  (Diminishing Returns)**: as satisfaction improves, the IT department's impact diminishes.
  - I haven't understood  $g$ .
- **$1 + ahx_t$  (Saturation)**: even if IT is highly capable ( $a \gg 1$ ), inflexible systems ( $h \gg 1$ ) limit its efficacy.

#### 3.5.3 Organizational Adaptability

$$C(x_t) = \frac{1}{1 + z^s} \quad \text{where} \quad z = \frac{r(1 - x_t)}{x_t(1 - r)} \quad (8)$$

This is a **sigmoid** function in disguise. Sigmoids are exploited for modeling "threshold behaviors".

- **$r$  (activation threshold)**: below  $r$ , the organization resists to change ( $C(x) \rightarrow 0$ ). But when misalignment crosses a certain threshold adaptability kicks in.
- **$s$  (flexibility)**: higher  $s$  make the sigmoid steeper (sharper transition from resistance to adaptation).

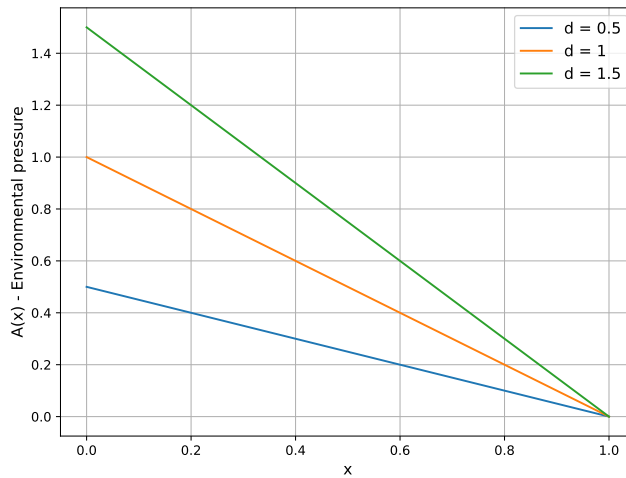


Figure 1: Environmental pressure function for varying  $d$  values

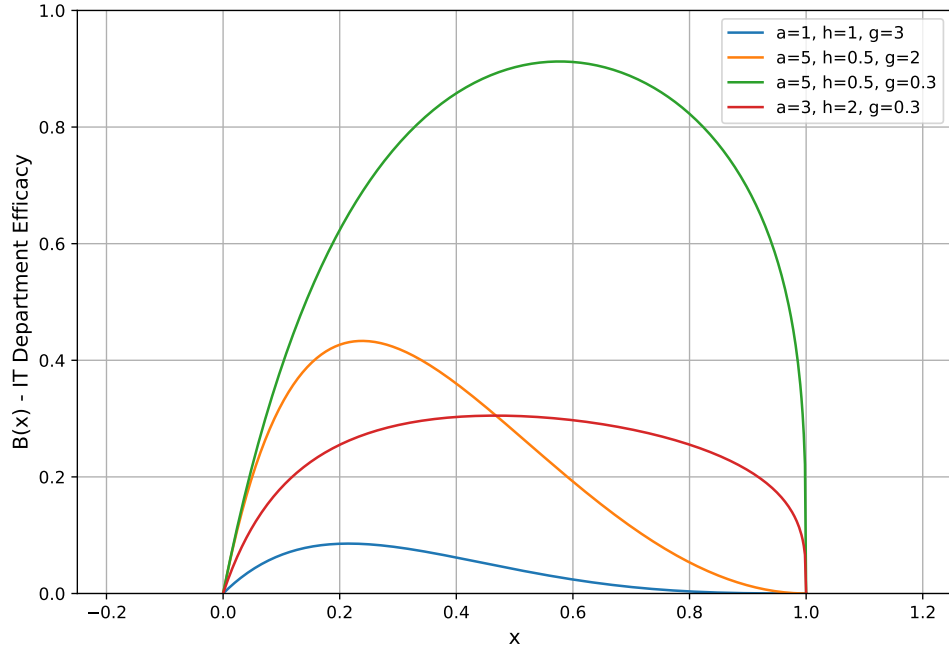


Figure 2: IT efficacy function showing the effect of parameters  $a$ ,  $h$ , and  $g$

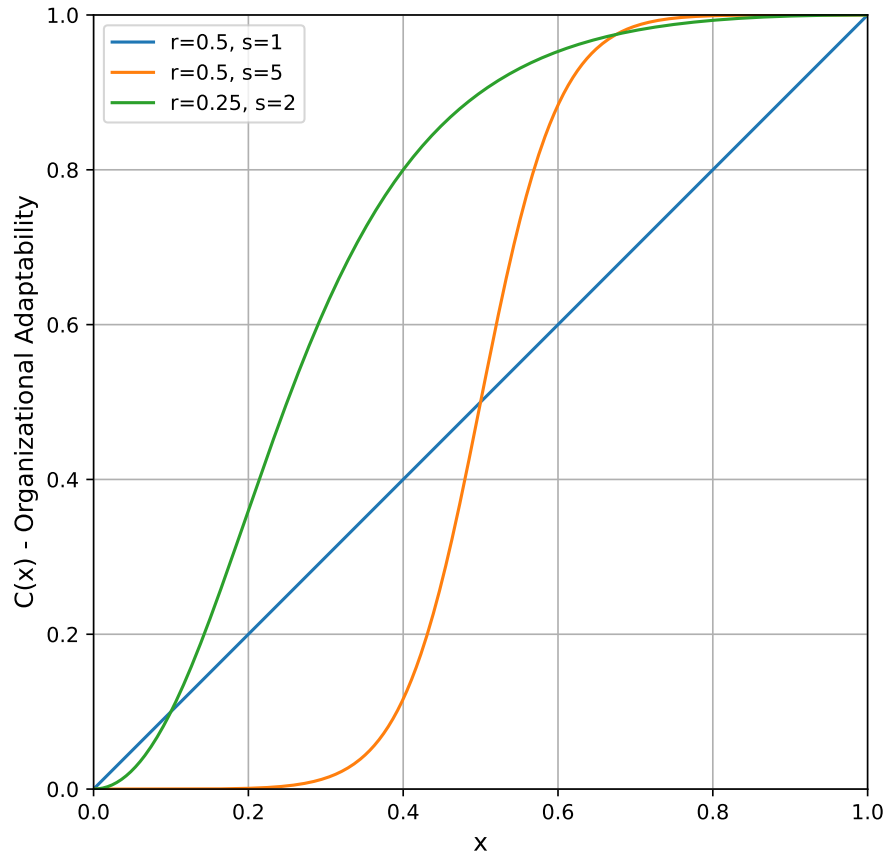


Figure 3: Organizational adaptability function demonstrating threshold behavior

## 4 Implementation

### 4.1 Technological Stack

- Python 3.10.12 (managed via Conda)
- Jupyter Notebook for interactive exploration, with markdown explanations attached
- Core dependencies (Python Libraries)
  - NumPy for numerical computations
  - Matplotlib for visualization graphs
  - ipywidgets for parameter sliders

### 4.2 Key lines of code

#### 4.2.1 Simulating the equation

The equation is hardcoded:

```
def A(x, d):  
    return d * (1 - x)  
  
def B(x, a, h, g):  
    return (a * x * (1 - x) ** g) / (1 + a * h * x)  
  
def C(x, r, s):  
    if x == 0: # Avoid division by zero  
        return 0  
    z = (r * (1 - x)) / (x * (1 - r))  
    return 1 / (1 + z ** s)  
  
def delta(x, d, a, h, g, r, s):  
    return A(x, d) - B(x, a, h, g) * C(x, r, s)  
  
def simulate(x0, d, a, h, g, r, s, steps=100):  
    x = np.zeros(steps)  
    x[0] = x0  
    for t in range(steps - 1):  
        x[t + 1] = np.clip(x[t] + delta(x[t], d, a, h, g, r, s), 0, 1)
```

#### 4.2.2 Long term behavior

Here is the key logic behind classifying the equation.

```
x = simulate(x0, d, a, h, g, r, s, steps)  
  
last_values = x[-10:]  
  
if np.std(last_values) < 0.001: # Stable state  
    final_val = np.mean(last_values)  
    if final_val < 0.1:  
        return x, "ALIGNED"  
    elif final_val > 0.9:  
        return x, "MISALIGNED"  
    else:  
        return x, "PARTIAL_ALIGNMENT"  
else: # Dynamic state  
    if len(np.unique(np.round(last_values, 2))) > 3:  
        return x, "CHAOTIC"  
    else:  
        return x, "OSCILLATING"
```

### 4.2.3 Phase Portrait Analysis

The phase space visualization algorithm:

```
def phase_portrait(d, a, h, g, r, s, n_points=200):
    x = np.linspace(0, 1, n_points)
    dx = np.array([delta(xi, d, a, h, g, r, s) for xi in x])

    # Arrow placement logic
    arrow_indices = np.linspace(0, len(x)-1, 20, dtype=int)
    norm = Normalize(vmin=0, vmax=np.max(np.abs(dx)))

    for xi, dxi in zip(x[arrow_indices], dx[arrow_indices]):
        if dxi > 0: # Right arrow
            plt.arrow(xi, 0, 0.02, 0, ...)
        elif dxi < 0: # Left arrow
            plt.arrow(xi, 0, -0.02, 0, ...)

    plt.plot(x, dx, 'k-') # Main curve
    plt.axhline(0, color='black', linestyle=':') # Zero line
```

### 4.2.4 Bifurcation Analysis

The chaotic regime detection algorithm:

```
def bifurcation_analysis(param, p_min, p_max, fixed_params, n_points=500):
    param_values = np.linspace(p_min, p_max, n_points)
    n_transient = 200 # Skip initial transient
    n_samples = 100 # Points to plot per parameter

    for p in param_values:
        params = fixed_params.copy()
        params[param] = p

        x = 0.3 # Initial value
        # Burn-in phase
        for _ in range(n_transient):
            x = np.clip(x + delta(x, **params), 0, 1)

        # Sample stable points
        x_vals = []
        for _ in range(n_samples):
            x = np.clip(x + delta(x, **params), 0, 1)
            x_vals.append(x)

        plt.plot([p]*n_samples, x_vals, 'k.', markersize=0.5)
```

## 4.3 Numerical Considerations

- State clipping ensures  $x_t \in [0, 1]$  remains meaningful
- All floating-point operations use NumPy's float64 precision
- The bifurcation analysis skips 200 transient iterations to focus on long-term behavior
- Phase portrait arrows are normalized to avoid cluttering

## 5 Results

### 5.1 Interactive Tools

The notebook provides three powerful ways to explore alignment dynamics.

#### 5.1.1 Time Evolution Simulation

The time evolution simulation displays how alignment changes over successive iterations. You can tweak the initial condition ( $x_0$ ), the number of iterations and all the parameters of the equation through the **sliders**. The diagram will update in real-time.

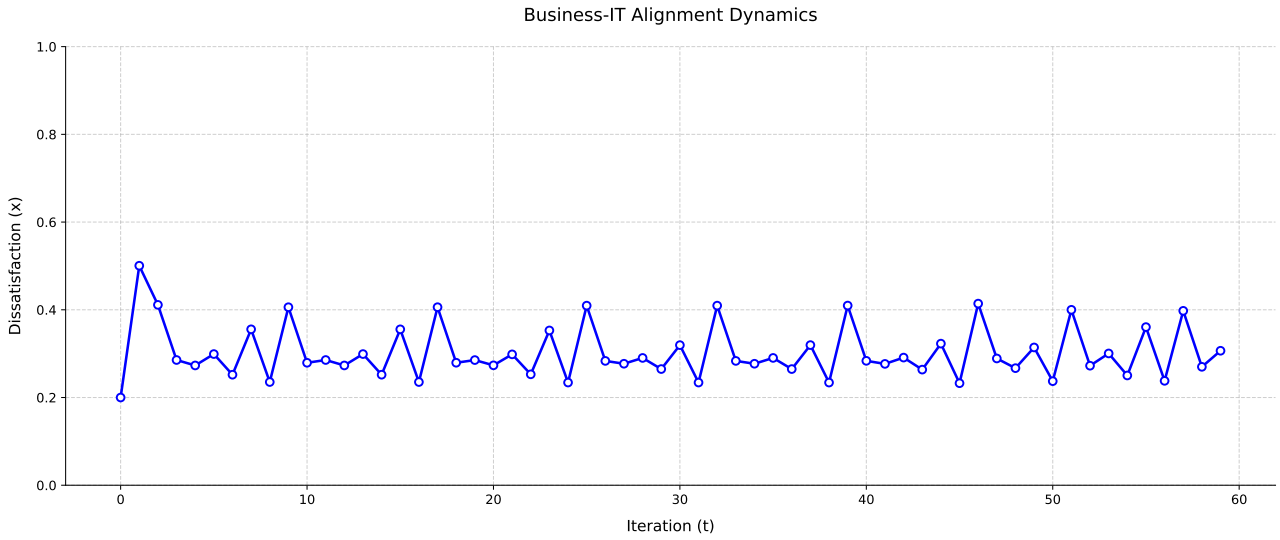


Figure 4: Time evolution showing chaotic behaviour ( $d = 0.5$ ,  $a = 6$ ,  $h = 0.4$ ,  $g = 2$ ,  $r = 0.25$ ,  $s = 5$ )

#### 5.1.2 Phase Portrait Analysis

The phase portrait examines the system's underlying dynamics through multiple visual cues. Arrow directions indicate whether misalignment tends to increase or decrease at each state point, while a color gradient represents the rate of change intensity.

**Note:** Stable equilibrium points occur where the curve crosses zero with a negative slope - these represent self-correcting alignment levels.

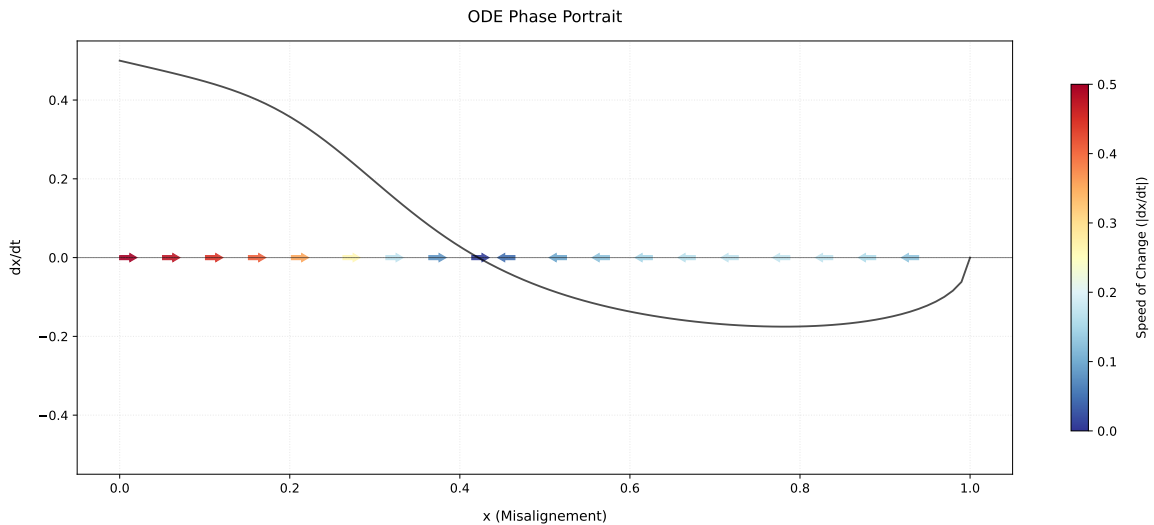


Figure 5: Phase portrait ( $d = 0.5$ ,  $a = 2$ ,  $h = 1$ ,  $g = 0.5$ ,  $r = 0.3$ ,  $s = 3$ )



### 5.1.3 Bifurcation Diagram

Users can select any parameter for the x-axis via a dropdown menu and focus on specific ranges of interest. Adjust the sliders, and you may get the characteristic period-doubling bifurcations and the emergence of chaotic behaviour.

**Pro tip:** Set  $h$  (IT rigidity) to a low value for observing beautiful patterns.

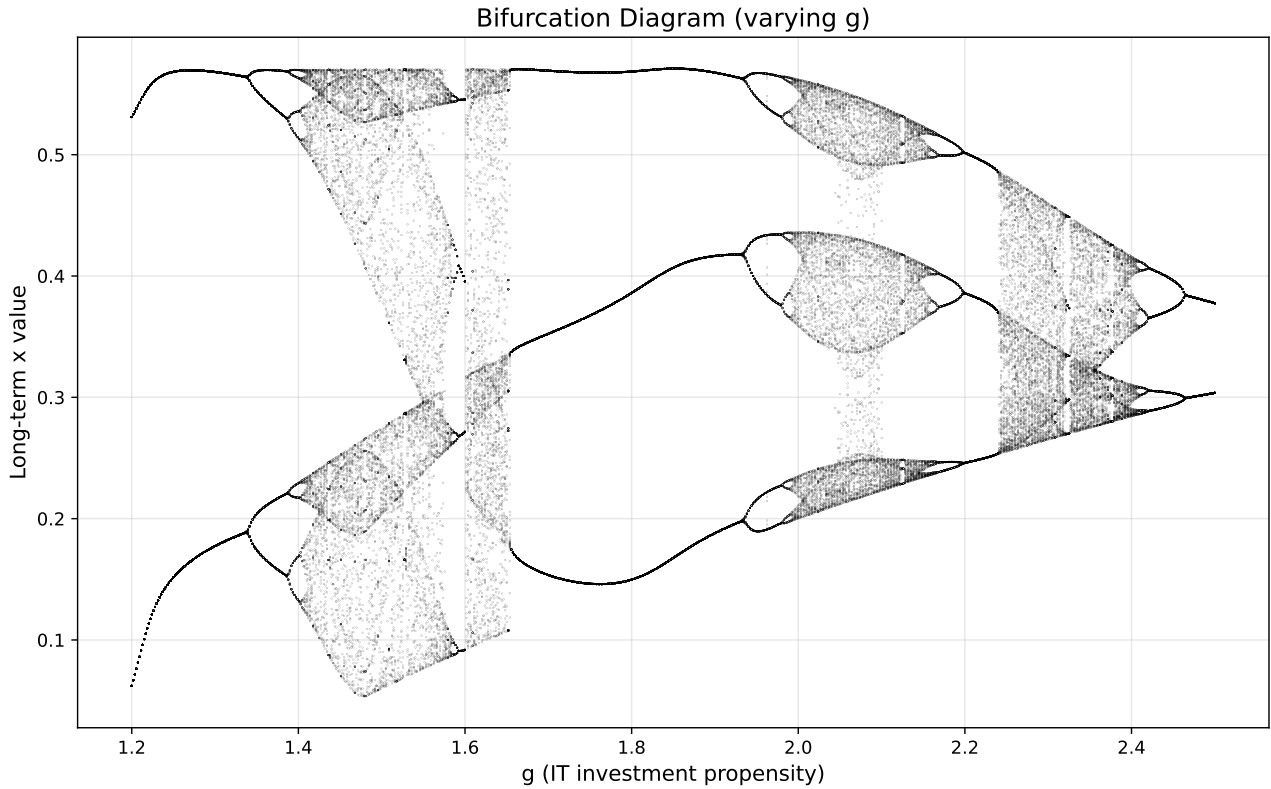


Figure 6: Shouldn't this be exposed at Louvre? ( $d = 0.5$ ,  $a = 7$ ,  $h = 0.3$ ,  $r = 0.3$ ,  $s = 5.1$ )

## References

## References

- [1] Strogatz, S. H. (2018). *Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering*. CRC Press.
- [2] Luftman, J. (2003). *Assessing IT/business alignment*. Information Systems Management, 20(4), 9-15.
- [3] Hunter, J. D. (2007). *Matplotlib: A 2D graphics environment*. Computing in science & engineering, 9(3), 90-95.

## 5.2 Examples

## A Appendix: Complete Python Code

The full implementation is available at: <https://github.com/Kinshale/pii>