Modeling Chaotic Systems

ARO (@_art_of_electronics_)

June 17, 2025

Introduction

This document presents the modeling of chaotic and hyperchaotic systems. A Simulink block diagram and Python code are also provided for simulation purposes.

1 Aizawa

1.1 Equation

$$\begin{cases} \dot{x} = x \cdot (z - \beta) - \sigma \cdot y \\ \dot{y} = \sigma \cdot x + y \cdot (z - \beta) \\ \dot{z} = \gamma + \alpha \cdot z - \frac{z^3}{3} - x^2 + \epsilon \cdot z \cdot x^3 \end{cases}$$

1.2 Python model

```python
def aizawa(state, __time__):
    x, y, z = state
    return x * (z - beta) - sigma * y, \
        sigma * x + y * (z - beta), \
        gamma + alpha * z - (z ** 3) / 3 - x ** 2 + epsilon * z *
            x ** 3
```

1.3 Simulink model

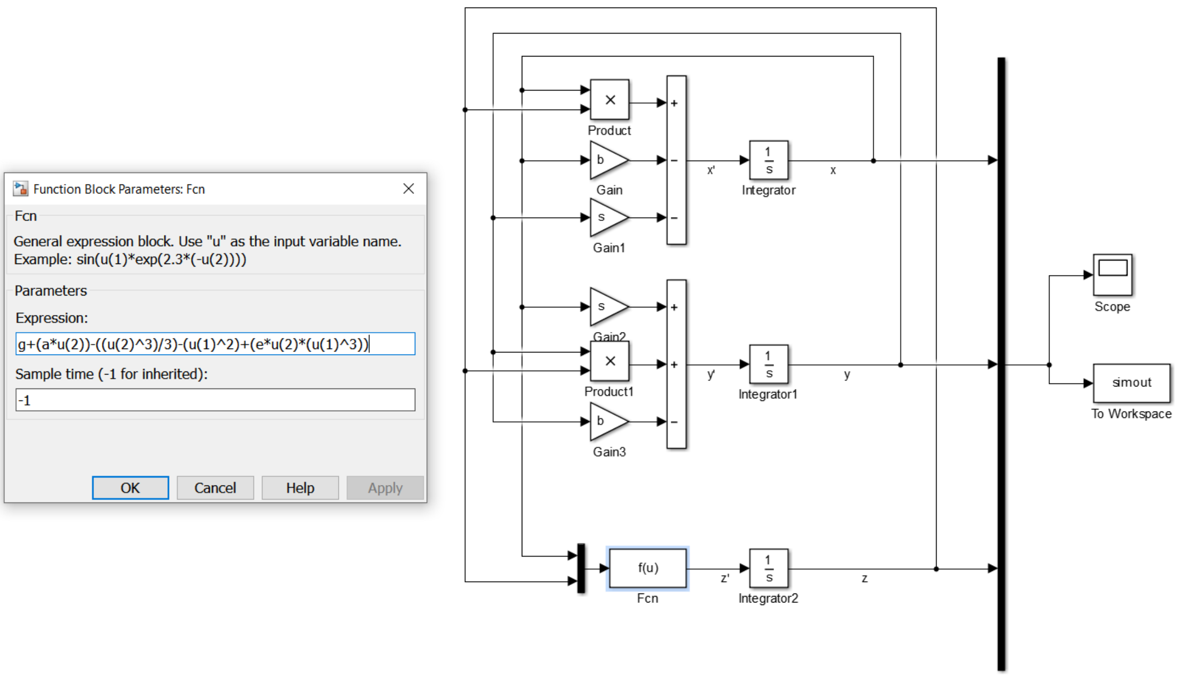

Figure 1: Aizawa Simulink model

1.4 Result

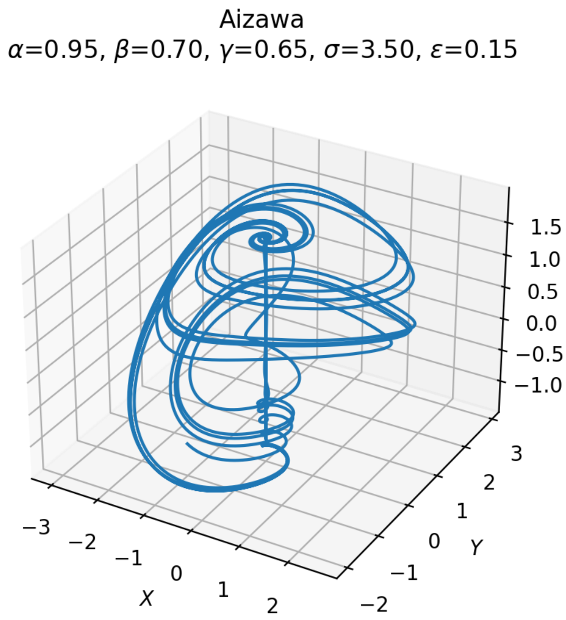

Figure 2: Aizawa system simulation results

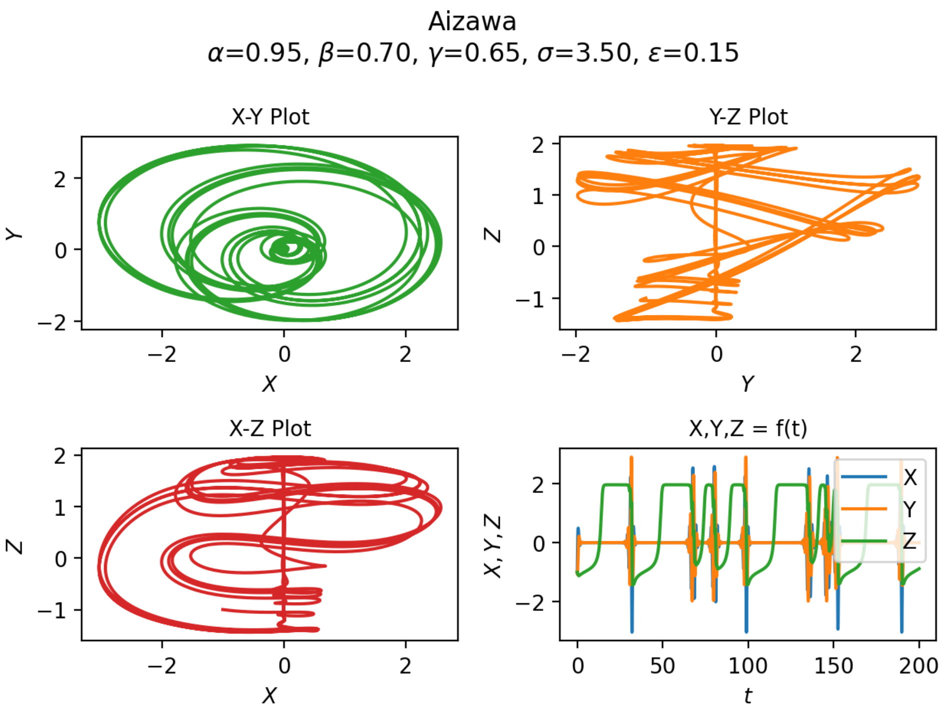

Figure 3: Aizawa system simulation results

2 Chua

2.1 Equation

$$\begin{cases} \dot{x} = a \cdot (y - x - diode) \\ \dot{y} = x - y + z \\ \dot{z} = -y \cdot b \end{cases}$$

$$diode = (m_1 \cdot x) + \left((m_0 - m_1) \cdot \frac{1}{2} \left(|x+1| - |x-1| \right) \right)$$

2.2 Python model

```python
def chua(state, __time__):
    x, y, z = state
    diode = (m[1] * x) + ((m[0] - m[1]) * (abs(x + 1) - abs(x - 1)) / 2)
    return a * (y - x - diode), x - y + z, - y * b
```

2.3 Simulink model

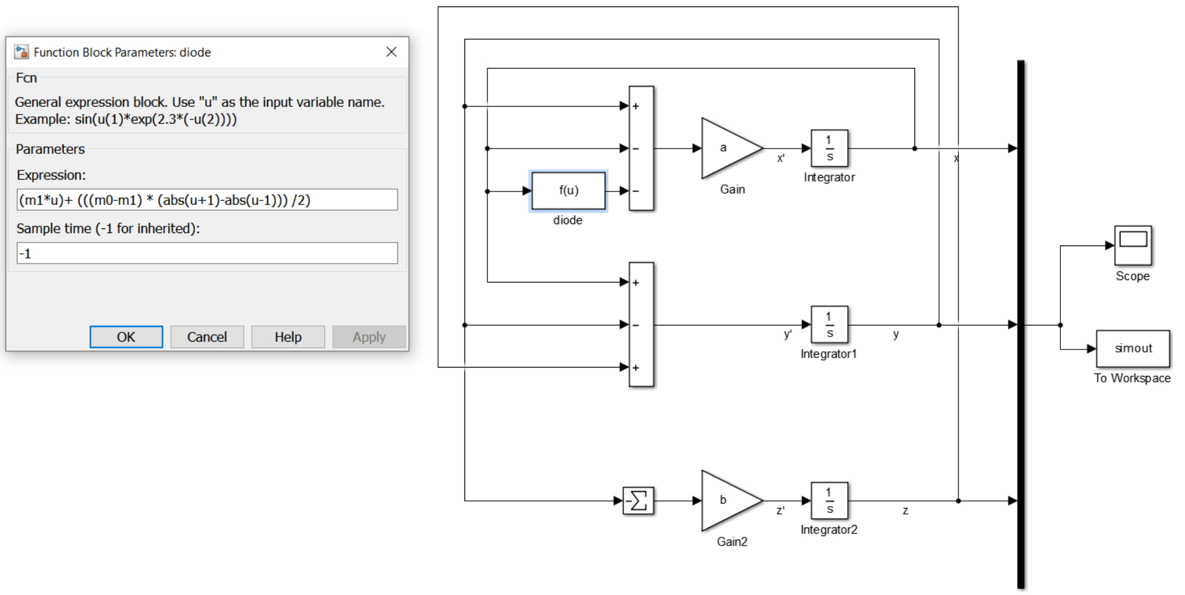

Figure 4: Chua Simulink model

2.4 Result

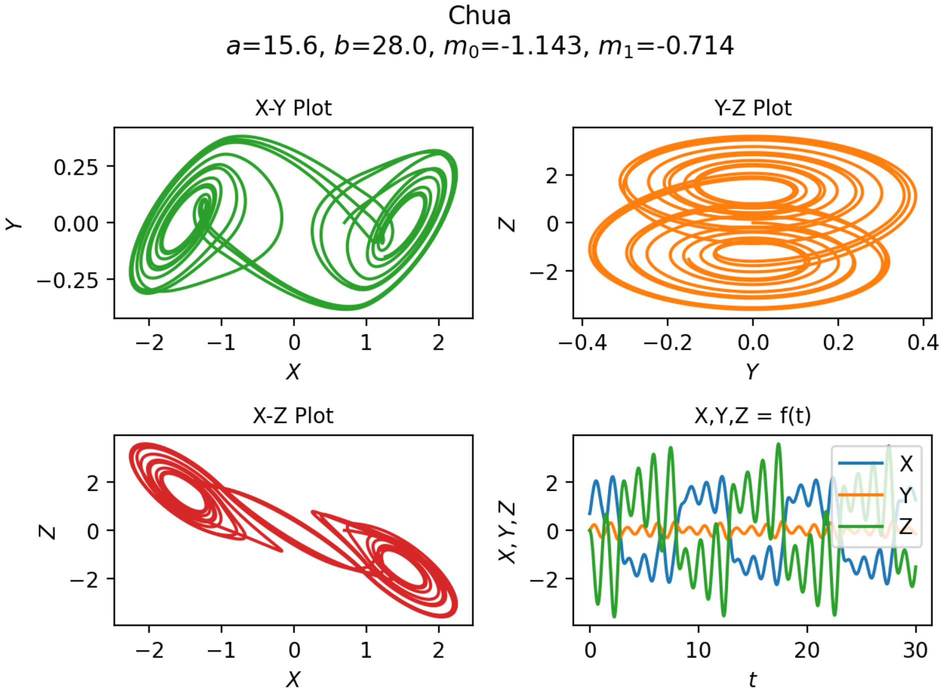

Figure 5: Chua system simulation results

2.5 Electronics Circuit

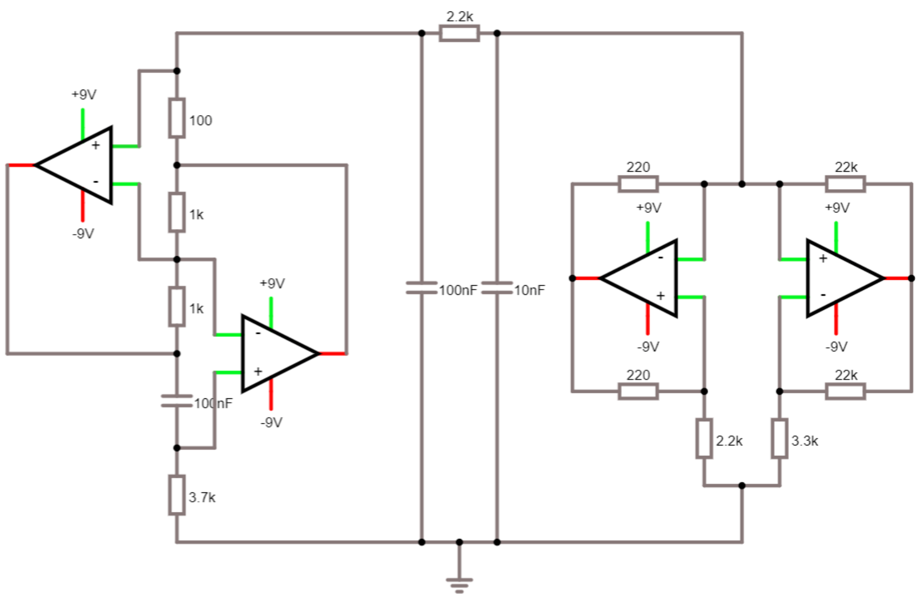

Figure 6: Chua system electronics circuit

3 Chua (Hyperchaotic)

3.1 Equation

$$\begin{cases} \dot{x} = \alpha \cdot (y - a \cdot x^3 - x \cdot (1 + c)) \\ \dot{y} = x - y + z \\ \dot{z} = -\beta \cdot y - \gamma \cdot z + w \\ \dot{w} = -s \cdot x + y \cdot z \end{cases}$$

3.2 Python model

```python
def hyper_chua(state, __time__):
    x, y, z, w = state
    return alpha * (y - a * (x ** 3) - x * (1 + c)), \
        x - y + z, \
        - beta * y - gamma * z + w, - s * x + y * z
```

3.3 Simulink model

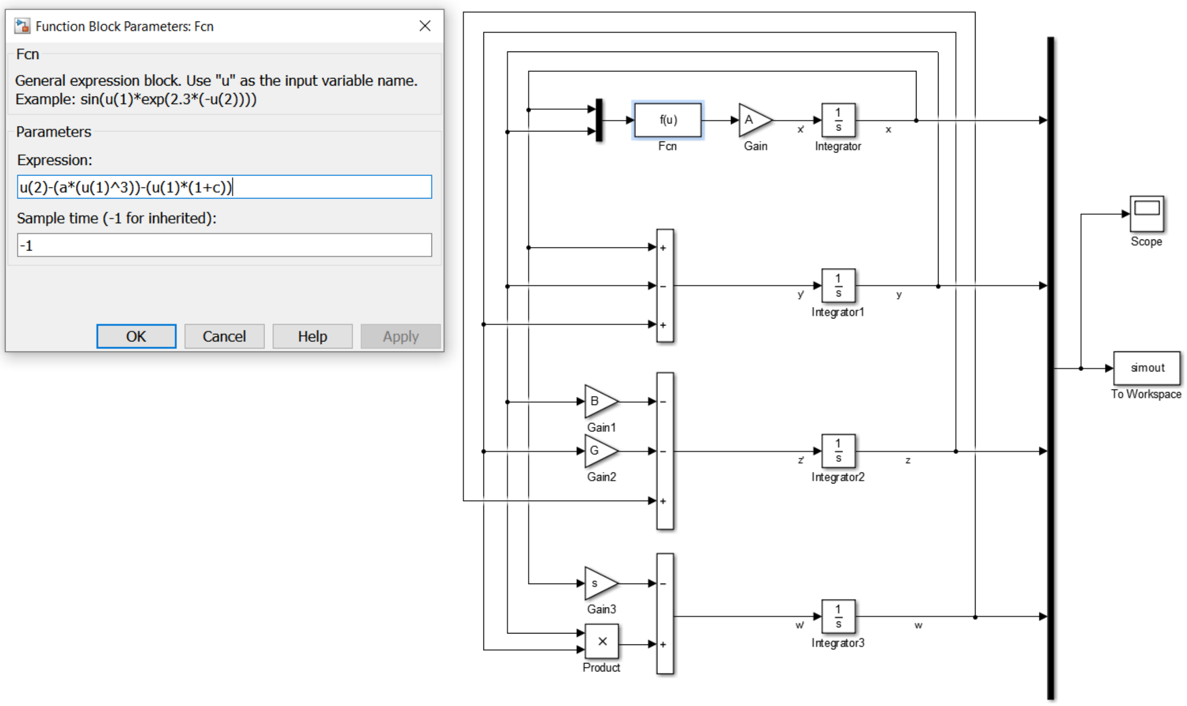

Figure 7: Hyperchaotic Chua Simulink model

3.4 Result

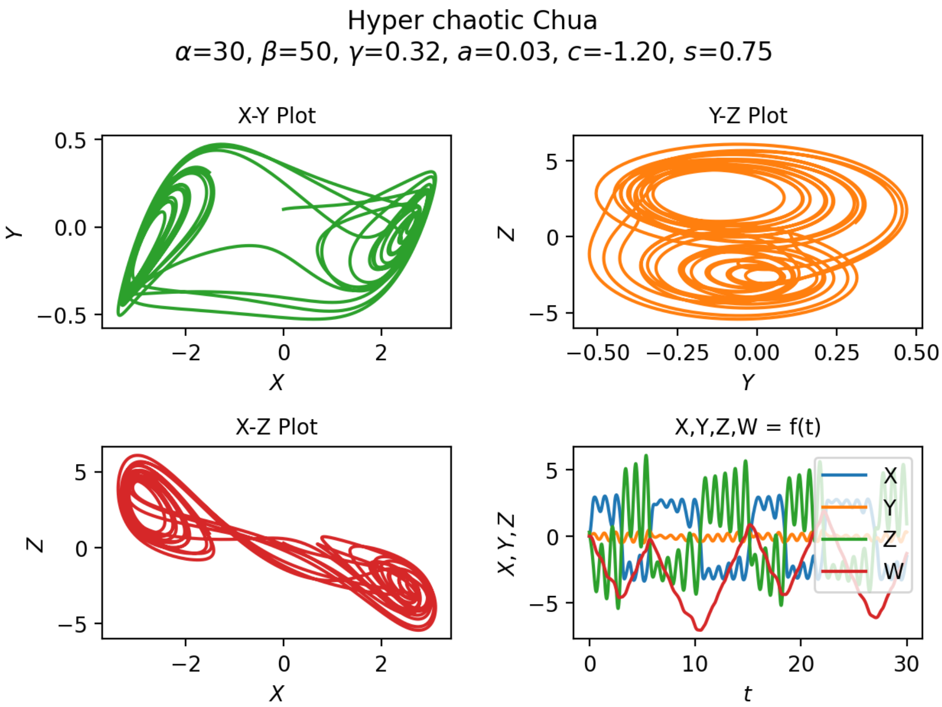

Figure 8: Hyperchaotic Chua system simulation results

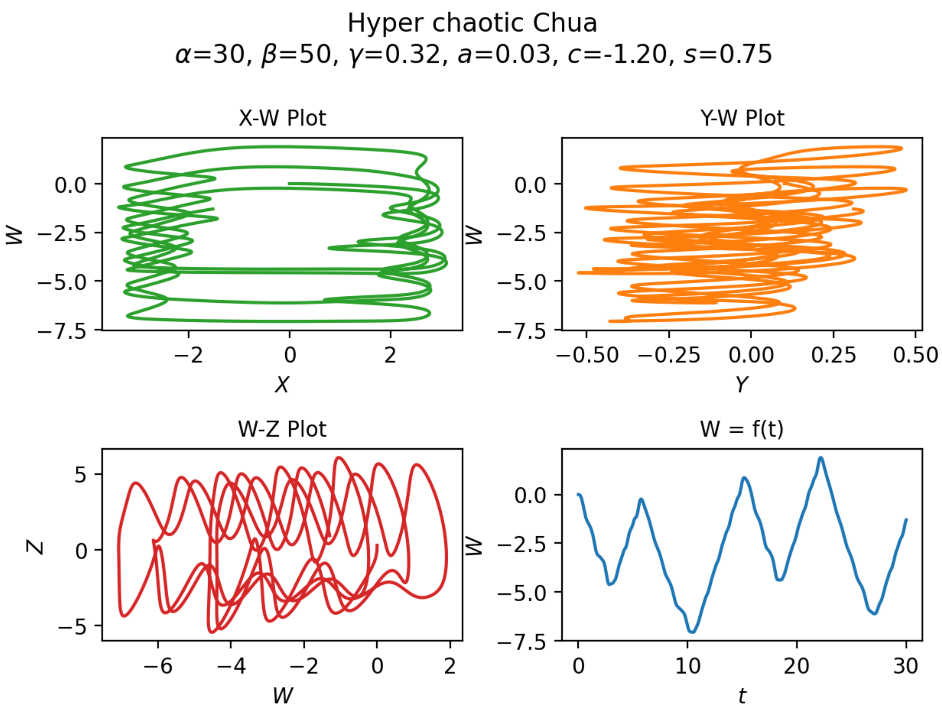

Figure 9: Hyperchaotic Chua system simulation results

4 Duffing

4.1 Equation

$$\ddot{x} = \gamma \cos \omega t - \delta \cdot \dot{x} - \beta \cdot x^3 - \alpha \cdot x$$

4.2 Python model

```python
def duffing(x, time):
    return x[1], \
        gamma * np.cos(omega * time) - sigma * x[1]\
        - alpha * x[0] - beta * x[0] ** 3
```

4.3 Simulink model

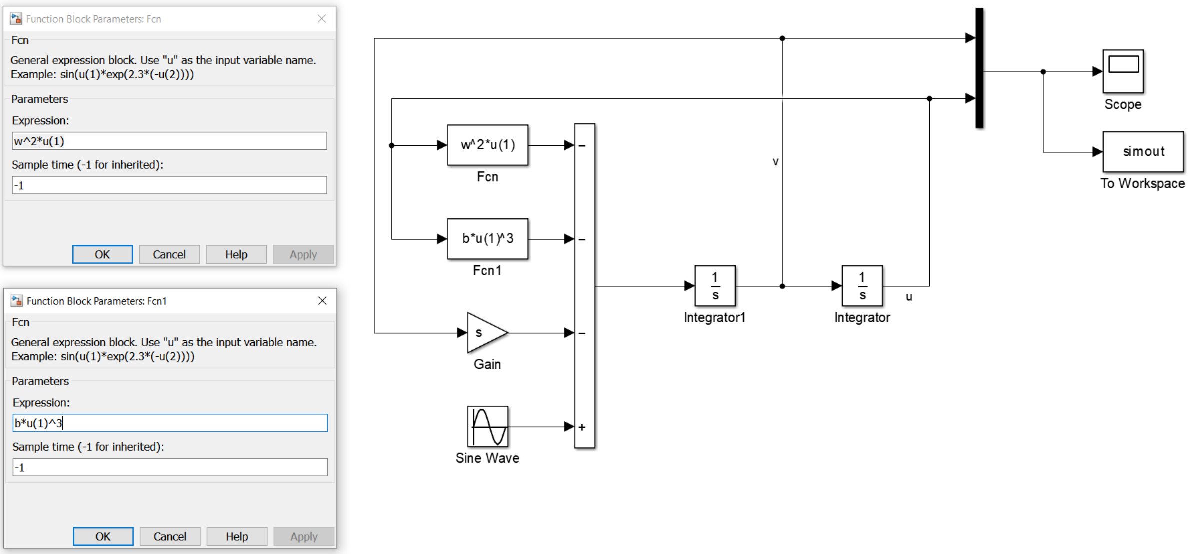

Figure 10: Duffing Simulink model

4.4 Result

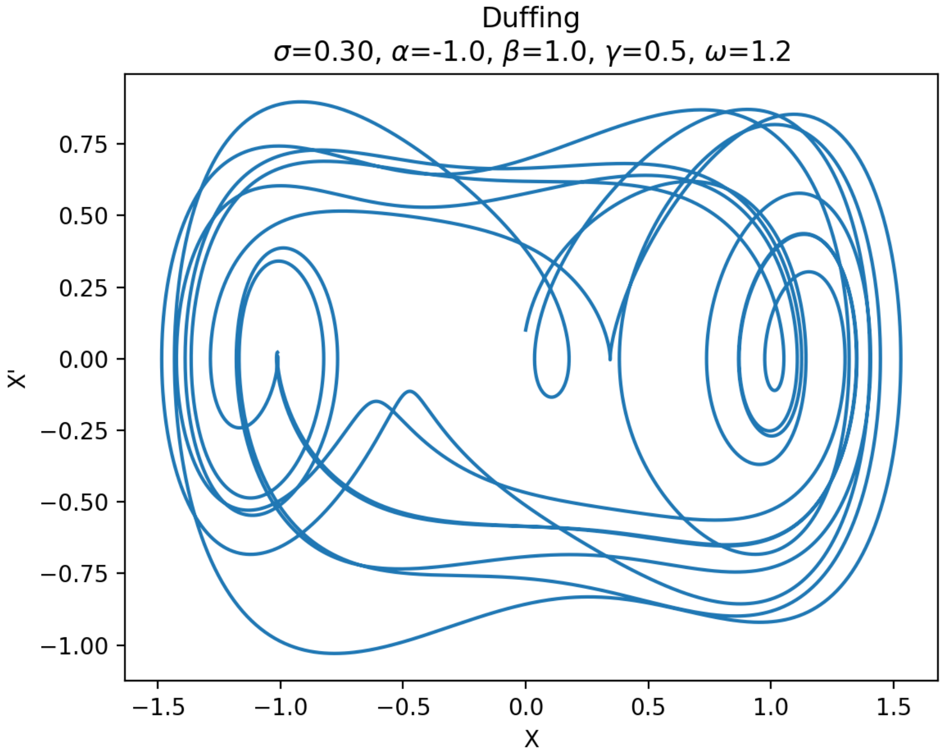

Figure 11: Duffing system simulation results

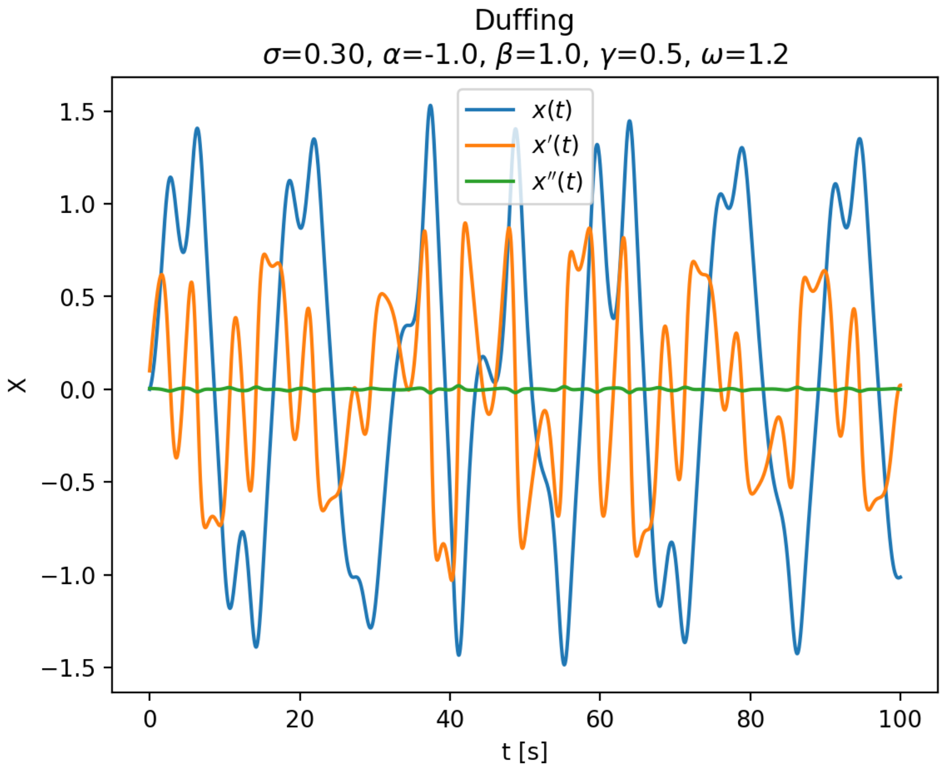

Figure 12: Duffing system simulation results

5 Henon-Heiles

5.1 Equation

$$\begin{cases} \ddot{x} = -x - 2\lambda \cdot x \cdot y \\ \ddot{y} = -y - \lambda \cdot (x^2 - y^2) \end{cases}$$

5.2 Python model

```python
def henon_heiles(x, __time__):
    _x = - x[0] - 2 * lambda_val * x[0] * x[2]
    _y = - x[2] - lambda_val * (x[0] ** 2 - x[2] ** 2)
    return x[1], _x, x[3], _y
```

5.3 Simulink model

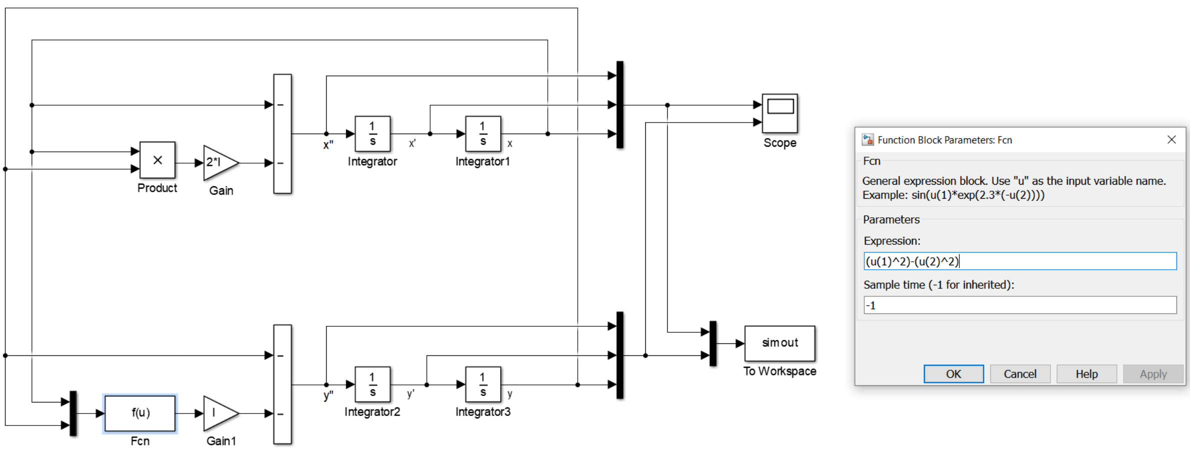

Figure 13: Henon-Heiles Simulink model

5.4 Result

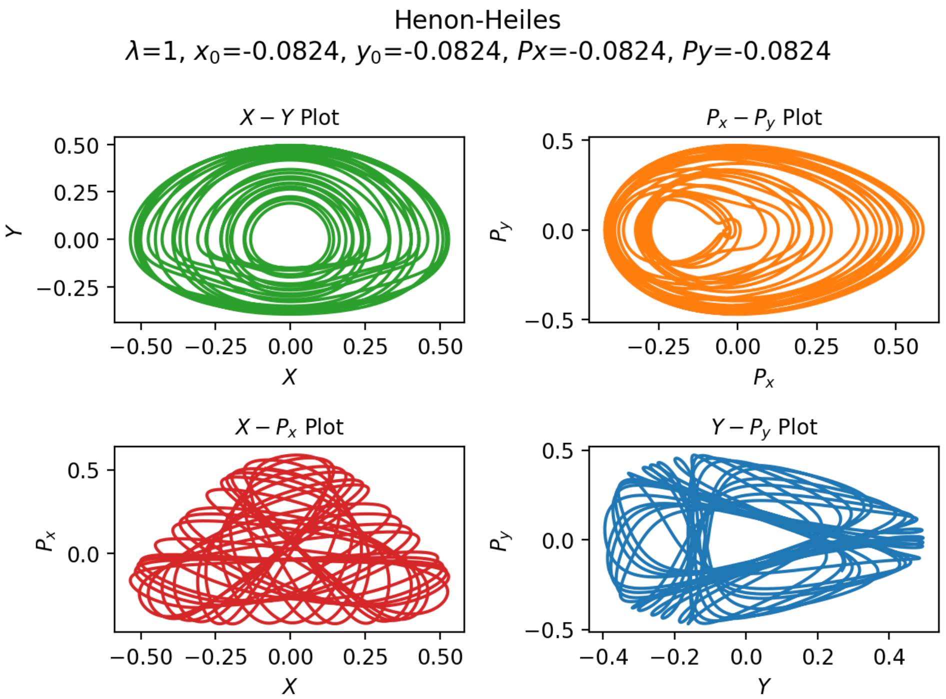

Figure 14: Henon-Heiles system simulation results

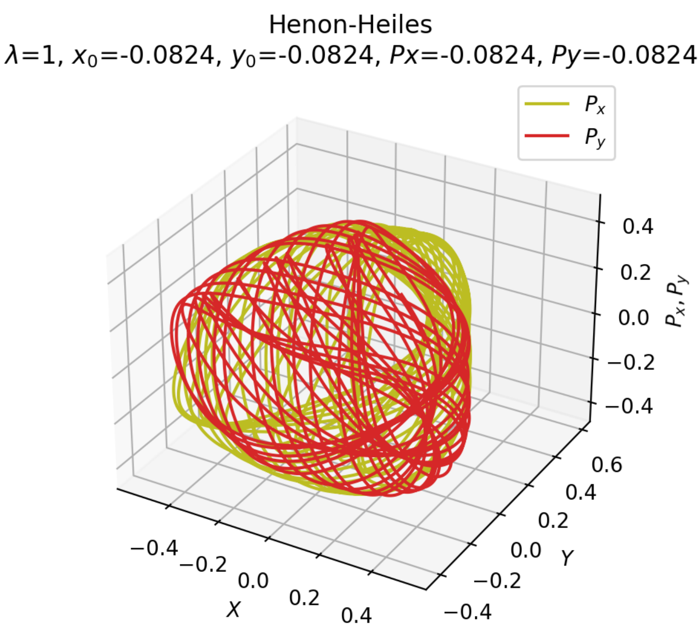

Figure 15: Henon-Heiles system simulation results

6 Hindmarsh-Rose

6.1 Equation

$$\begin{cases} \dot{x} = y + \Phi(x) - z + I \\ \dot{y} = \Psi(x) - y \\ \dot{z} = r \cdot (s \cdot (x - x_R) - z) \end{cases}$$

$$\Phi(x) = -a \cdot x^3 + b \cdot x^2$$
$$\Psi(x) = c - d \cdot x^2$$

6.2 Python model

```python
def hindmarsh_rose(state, __time__):
    x, y, z = state
    fi_x = -a * x ** 3 + b * x ** 2
    psi_x = c - d * x ** 2
    return y + fi_x - z + I, psi_x - y, r * (s * (x - xR) - z)
```

6.3 Simulink model

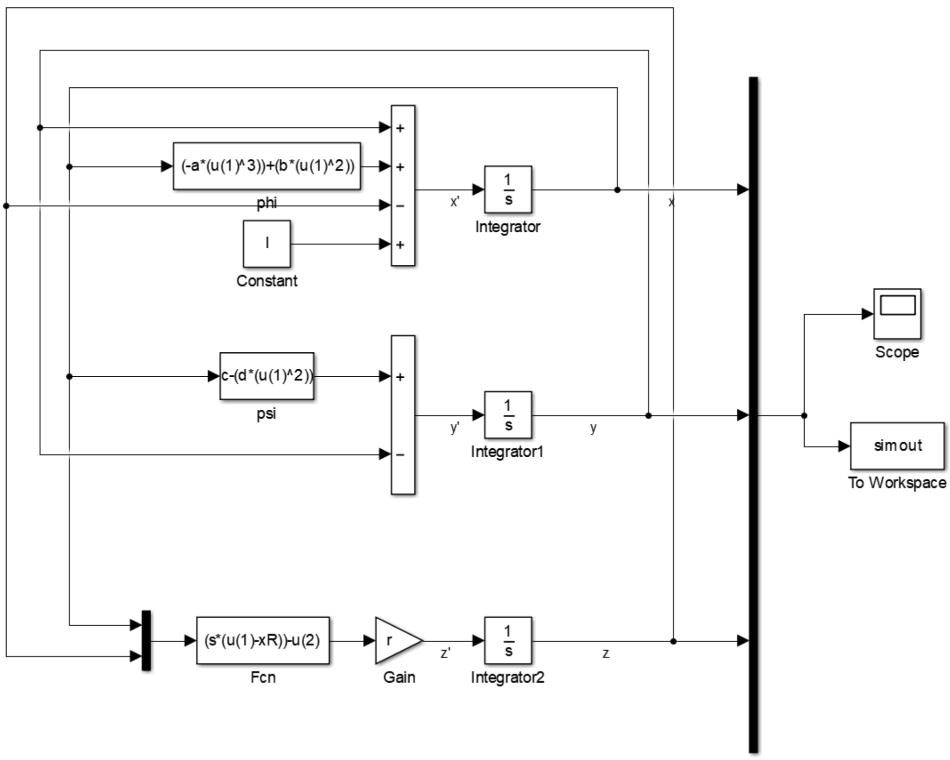

Figure 16: Hindmarsh-Rose Simulink model

6.4 Result

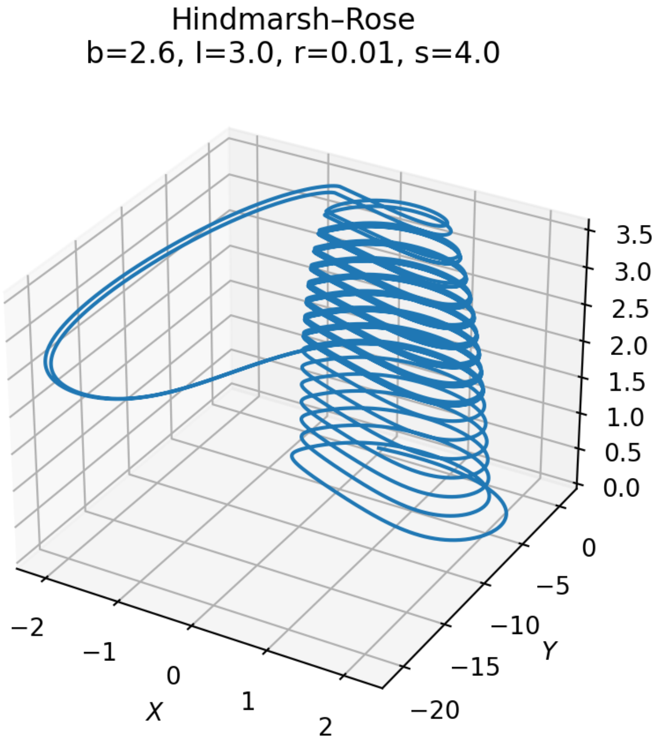

Figure 17: Hindmarsh-Rose system simulation results

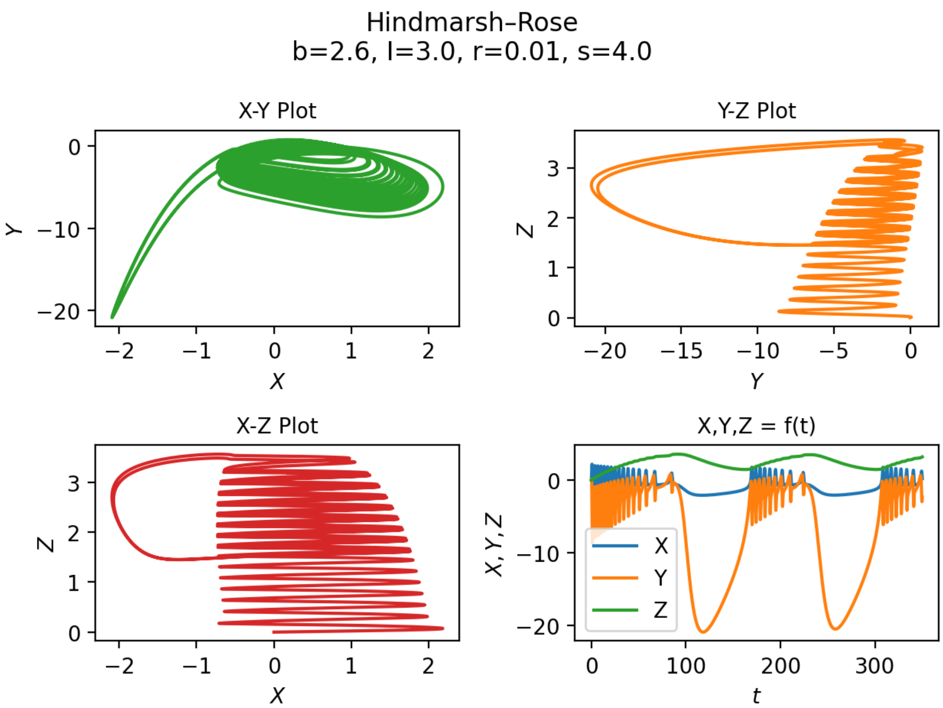

Figure 18: Hindmarsh-Rose system simulation results

7 Lorenz

7.1 Equation

$$\begin{cases} \dot{x} = \sigma \cdot (x - y) \\ \dot{y} = x \cdot (\rho - z) - y \\ \dot{z} = x \cdot y - \beta \cdot z \end{cases}$$

7.2 Python model

```python
def lorenz(state, __time__):
    x, y, z = state
    return sigma * (y - x), x * (rho - z) - y, x * y - beta * z
```

7.3 Simulink model

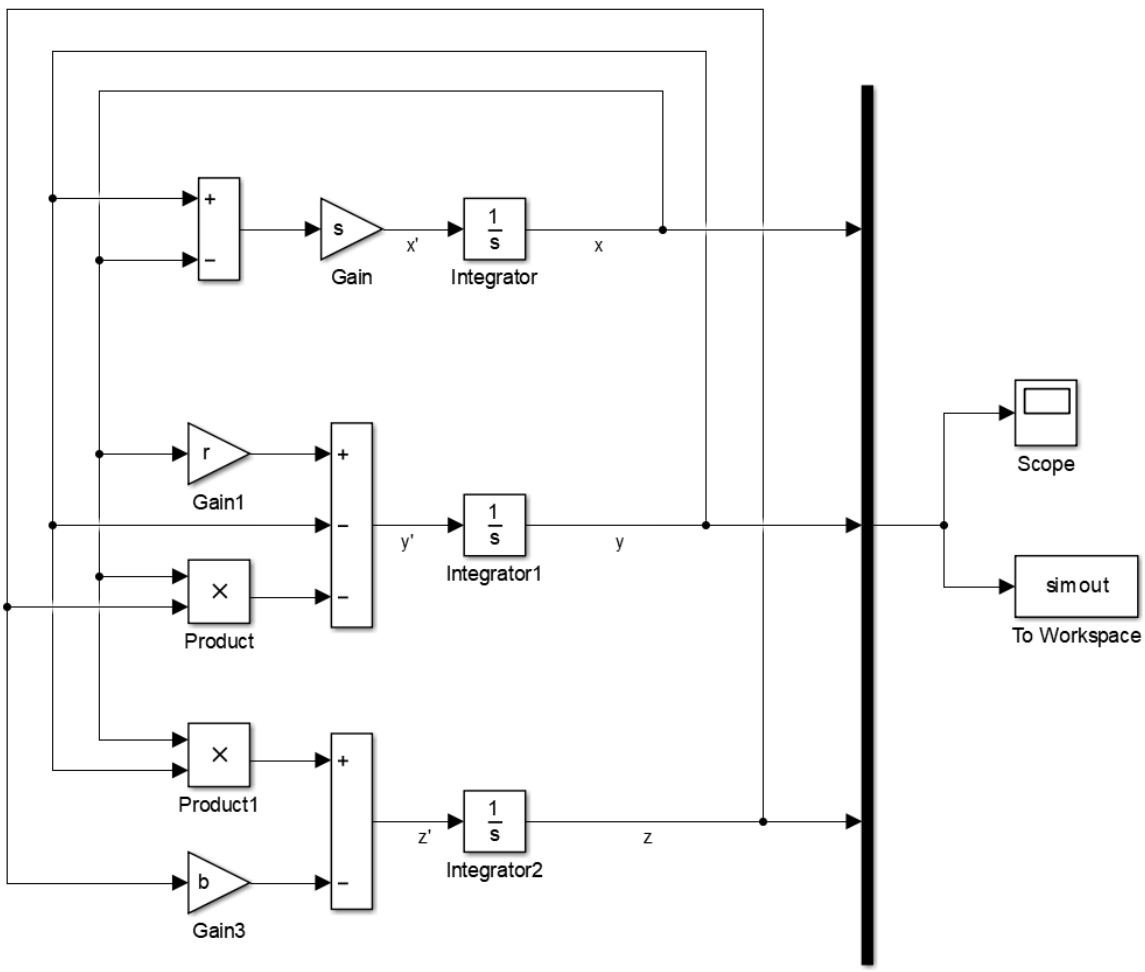

Figure 19: Lorenz Simulink model

7.4 Result

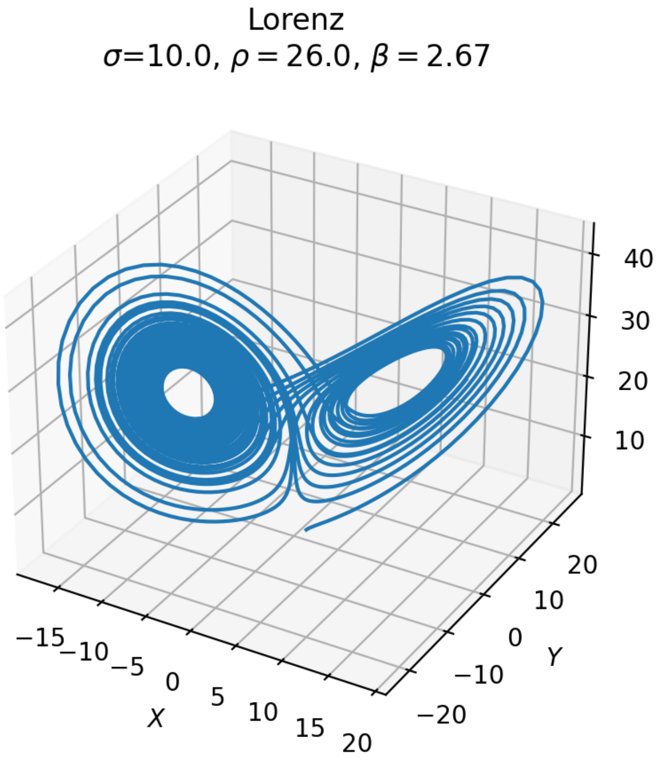

Figure 20: Lorenz system simulation results

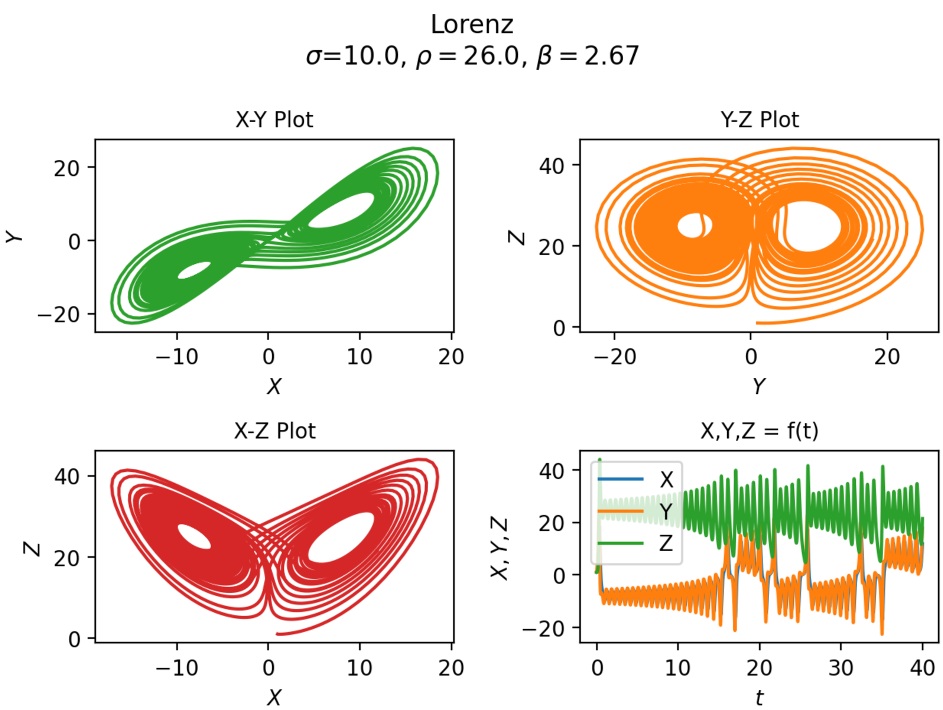

Figure 21: Lorenz system simulation results

8 Mackey-Glass

8.1 Equation

$$\dot{x} = \beta \cdot \frac{x_\tau}{1 + x_\tau^n} - \gamma \cdot x$$

8.2 Python model

```python
def mackey_glass(x, time, d):
    _x = x(time)
    _xt = x(time - d)
    return beta * _xt / (1 + _xt ** n) - _x * gamma
```

8.3 Simulink model

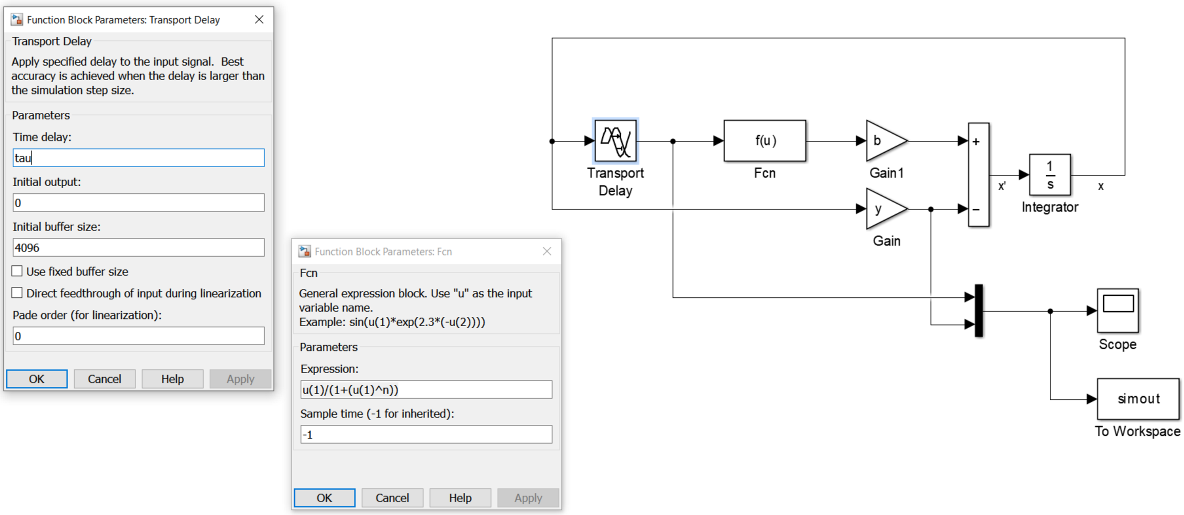

Figure 22: Mackey-Glass Simulink model

8.4 Result

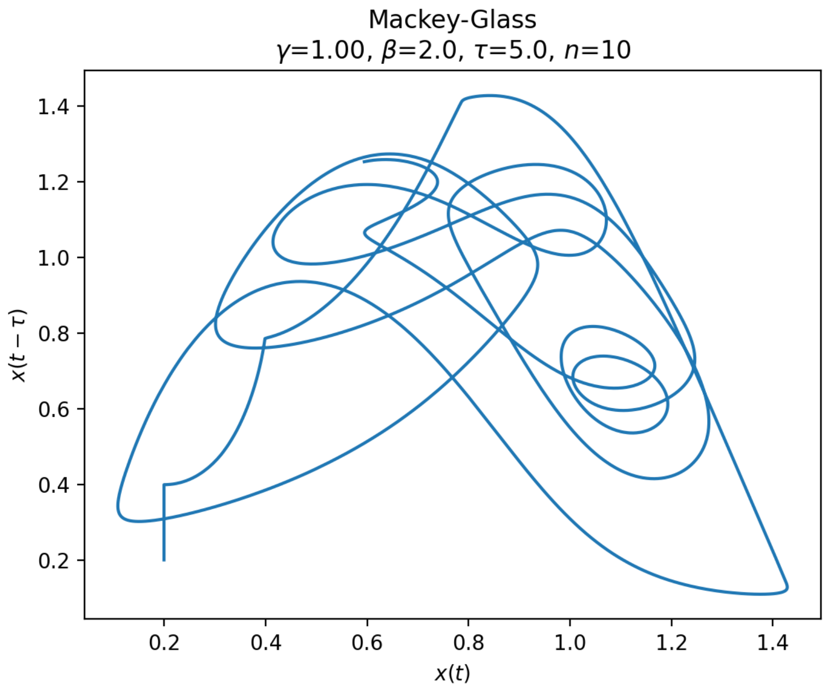

Figure 23: Mackey-Glass system simulation results

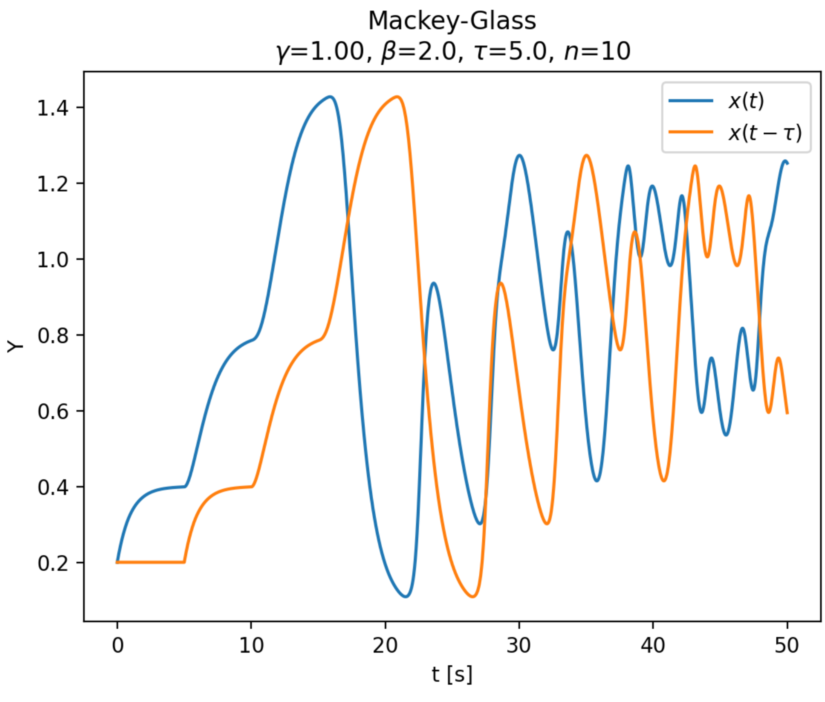

Figure 24: Mackey-Glass system simulation results

9 Rössler

9.1 Equation

$$\begin{cases} \dot{x} = -y - z \\ \dot{y} = x + a \cdot y \\ \dot{z} = b + z \cdot (x - c) \end{cases}$$

9.2 Python model

```python
def roessler(state, __time__):
    x, y, z = state
    return - y - z, x + (a * y), b + z * (x - c)
```

9.3 Simulink model

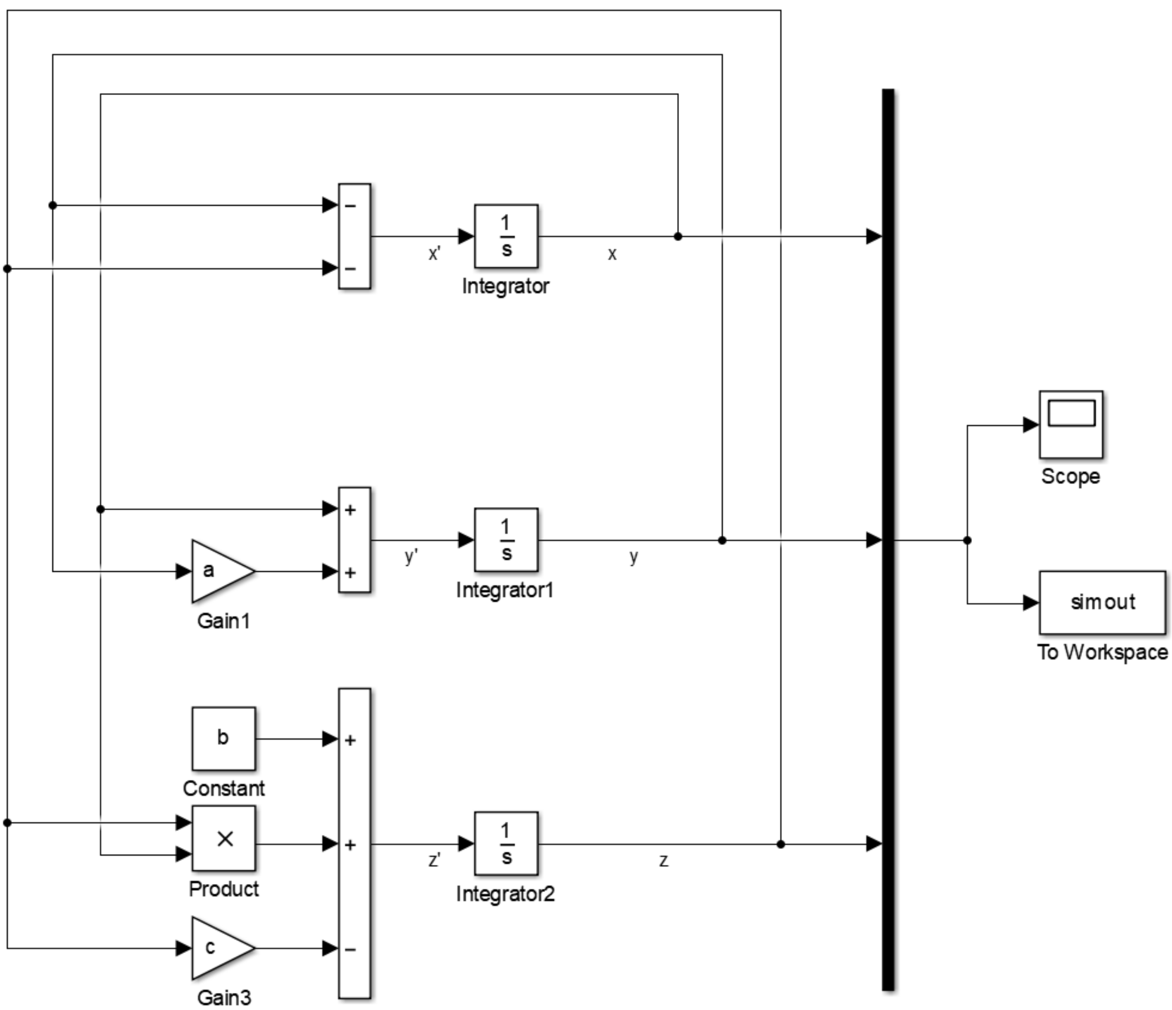

Figure 25: Rössler Simulink model

9.4 Result

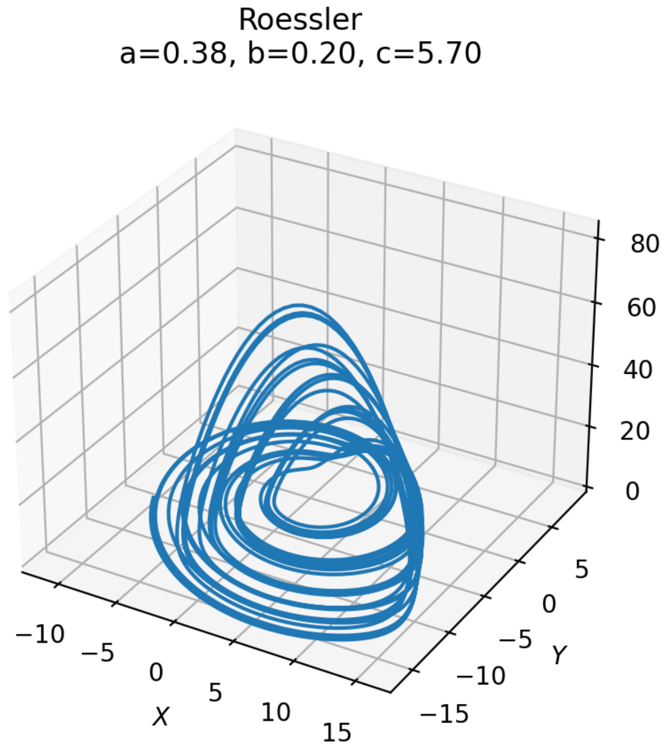

Figure 26: Rössler system simulation results

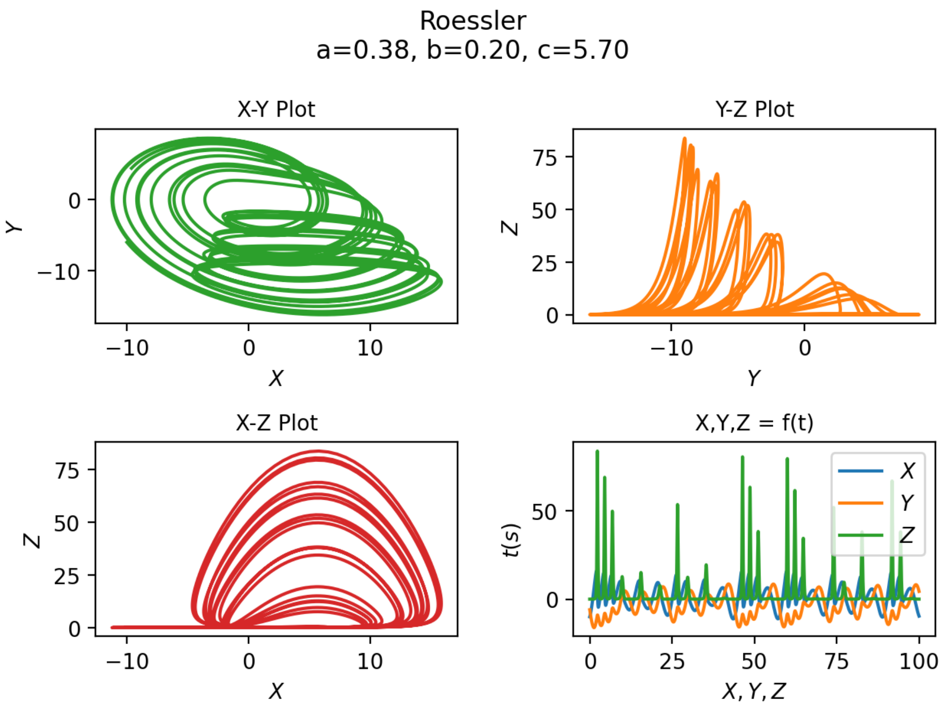

Figure 27: Rössler system simulation results

10 Rössler (Hyperchaotic)

10.1 Equation

$$\begin{cases} \dot{x} = -y - z \\ \dot{y} = x + a \cdot y + w \\ \dot{z} = b + z \cdot x \\ \dot{w} = -c \cdot z + d \cdot w \end{cases}$$

10.2 Python model

```python
def hyper_roessler(state, __time__):
    x, y, z, w = state
    return - y - z, x + (a * y) + w, b + (x * z), - (c * z) + (d
        * w)
```

10.3 Simulink model

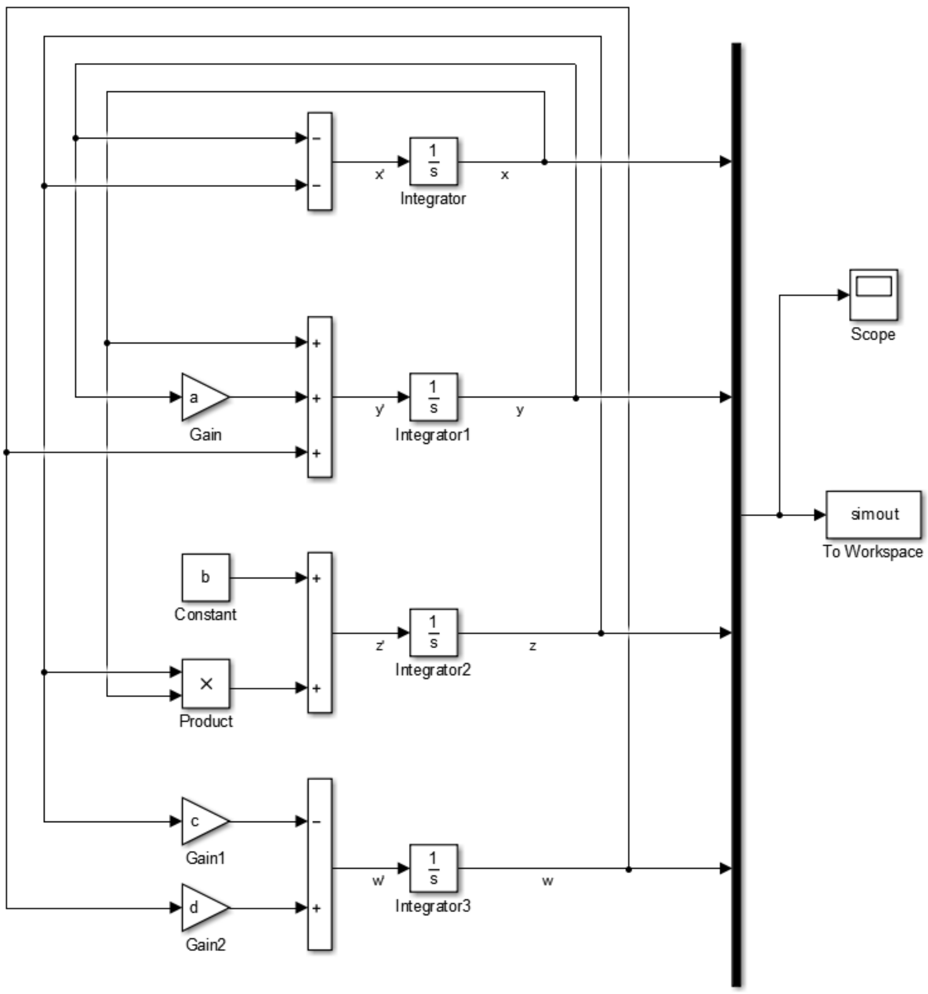

Figure 28: Hyperchaotic Rössler Simulink model

10.4 Result

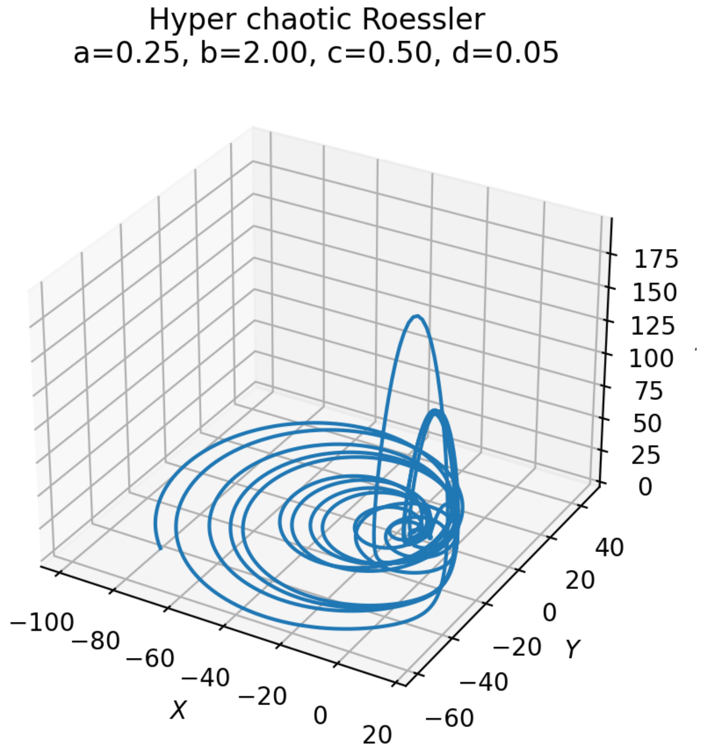

Figure 29: Hyperchaotic Rössler system simulation results

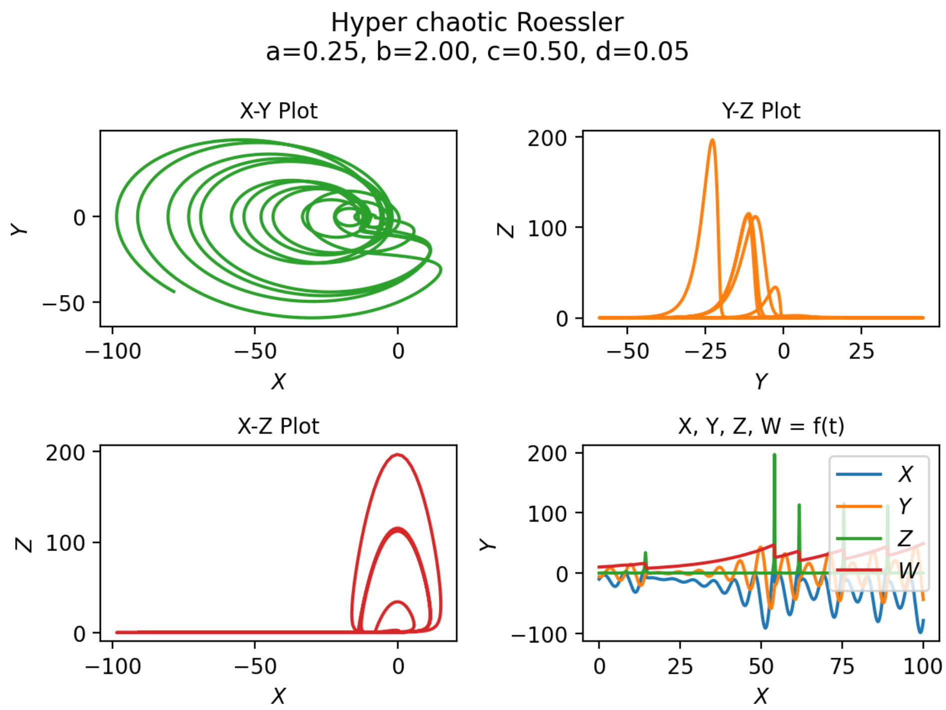

Figure 30: Hyperchaotic Rössler system simulation results

11 Thomas' Cyclically Symmetric Attractor

11.1 Equation

$$\begin{cases} \dot{x} = \sin(y) - \beta \cdot x \\ \dot{y} = \sin(z) - \beta \cdot y \\ \dot{z} = \sin(x) - \beta \cdot z \end{cases}$$

11.2 Python model

```
def thomas(state, __time__):
    x, y, z = state
    return np.sin(y) - b * x, np.sin(z) - b * y, np.sin(x) - b *
        z
```

11.3 Simulink model

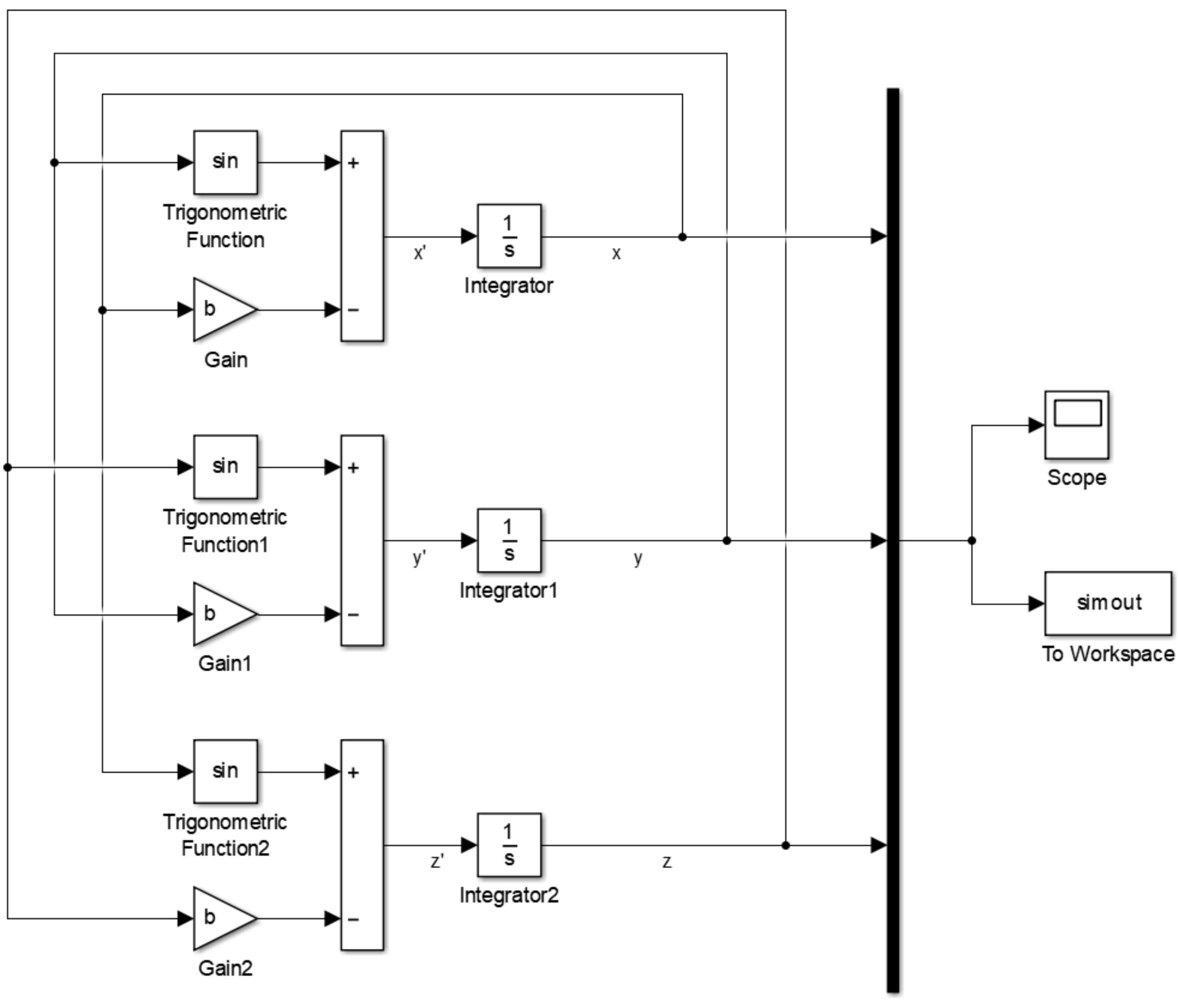

Figure 31: Thomas' attractor Simulink model

11.4 Result

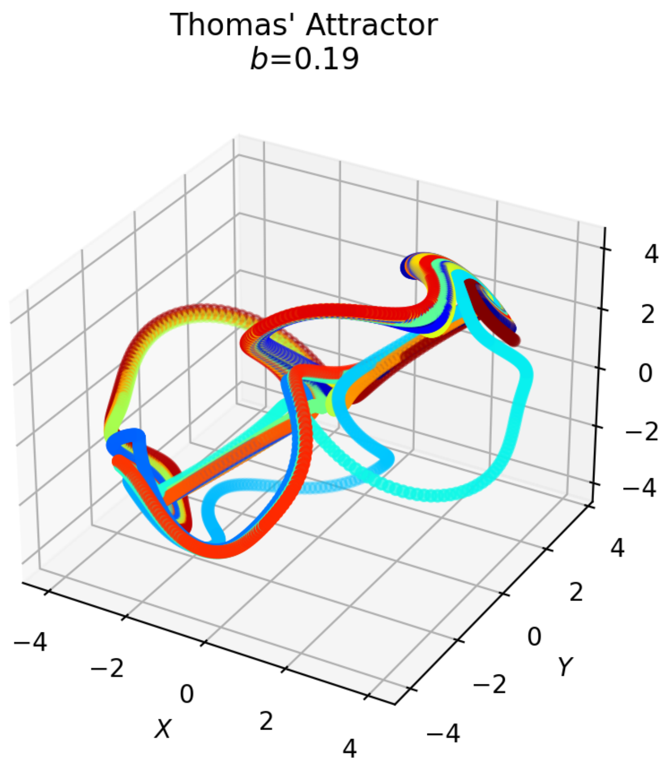

Figure 32: Thomas' cyclically symmetric attractor simulation results

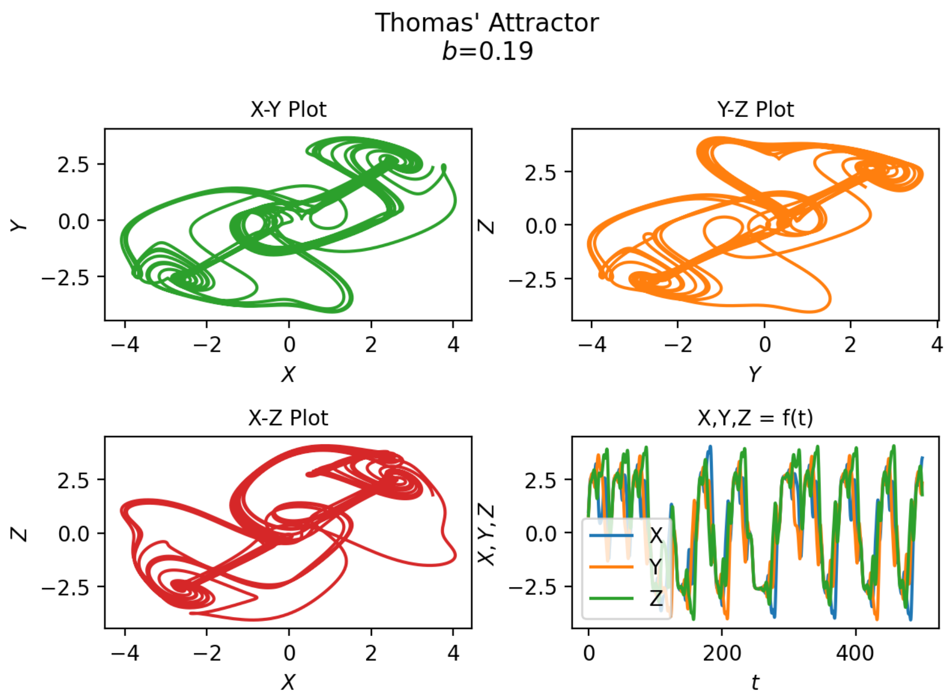

Figure 33: Thomas' cyclically symmetric attractor simulation results

12 Van der Pol

12.1 Equation

$$\ddot{y} = \mu \cdot \left(1 - y^2\right) \cdot \dot{y} - y$$

12.2 Python model

```python
def van_der_pol(y, __time__):
    return y[1], mu * (1 - y[0] ** 2) * y[1] - y[0]
```

12.3 Simulink model

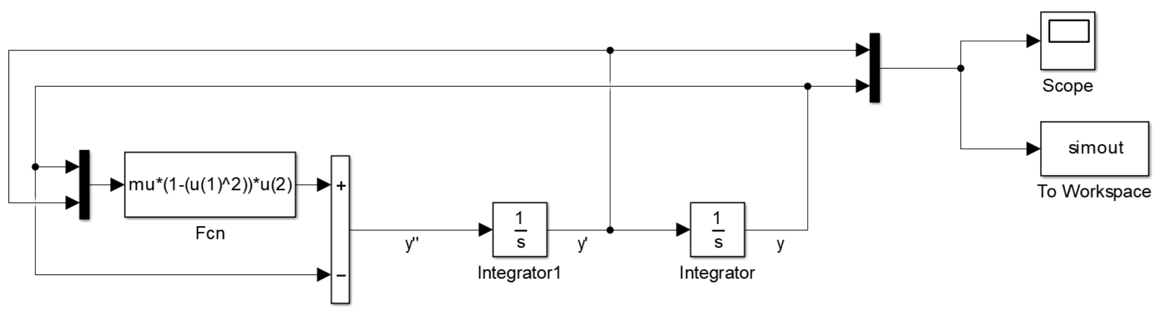

Figure 34: Van der Pol system simulation results

12.4 Result

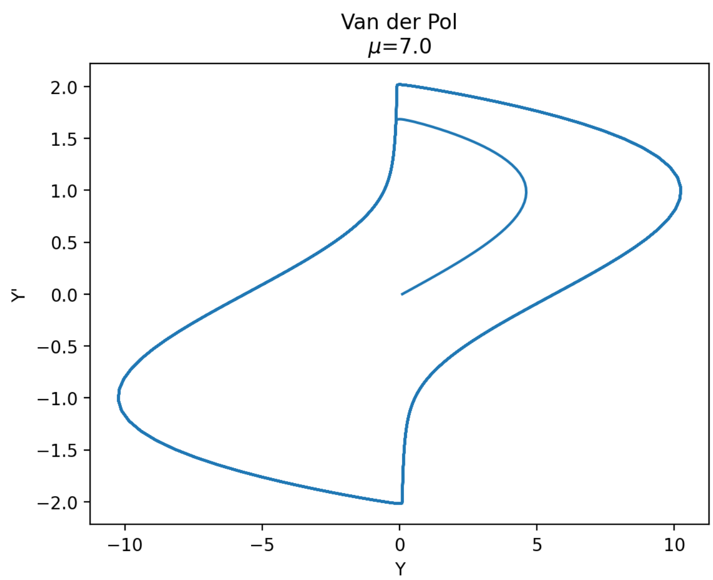

Figure 35: Van der Pol Simulink model

Conclusion

This document demonstrates equations for various chaotic systems and provides a modeling solution in Python.