Modeling Chaotic Systems

ARO (@_art_of_electronics_)

June 22, 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

```
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

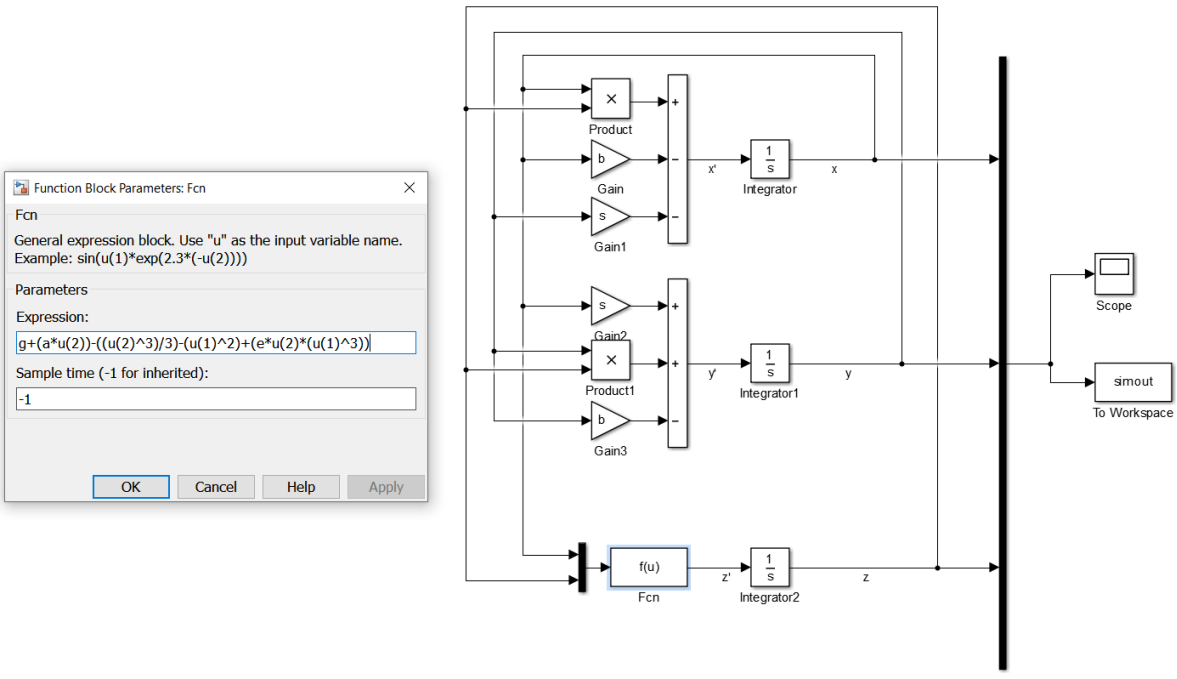

Figure 1: Aizawa Simulink model

1.4 Result

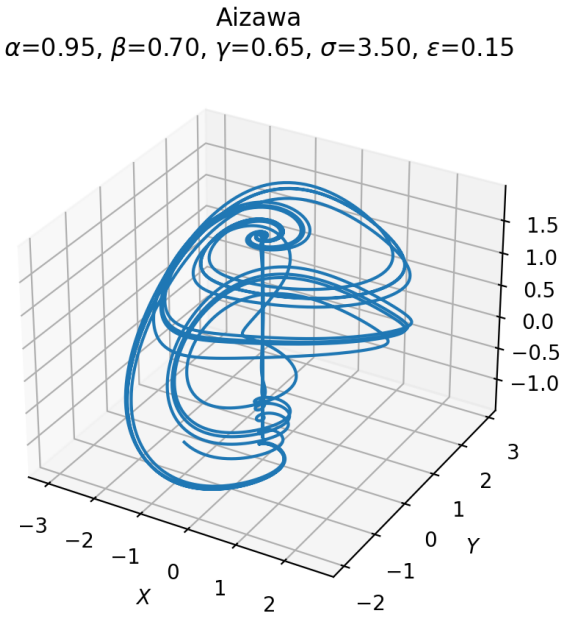

Figure 2: Aizawa system simulation results

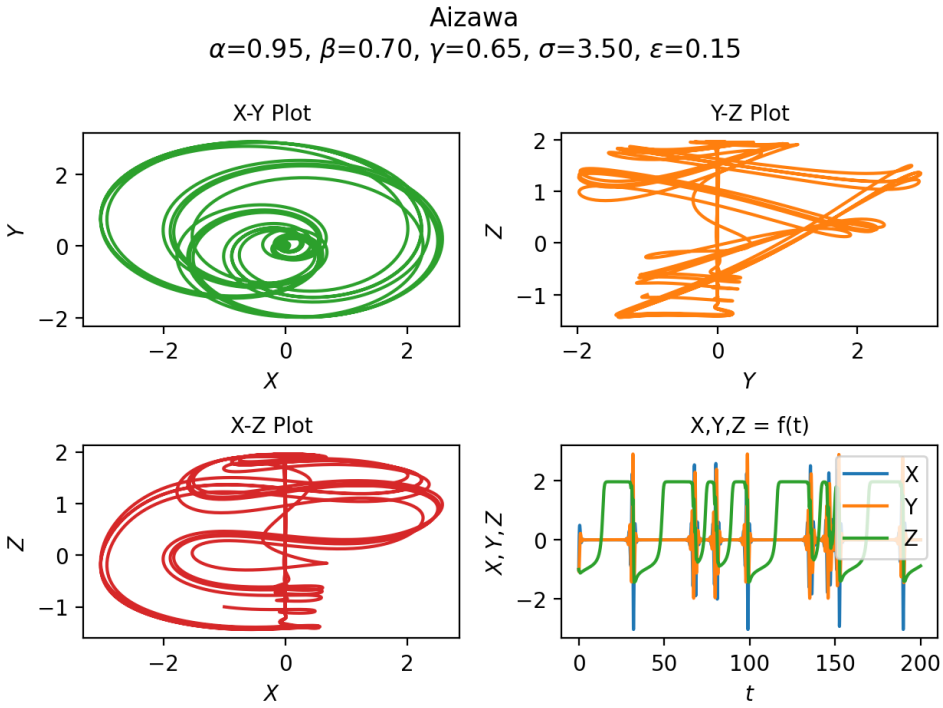

Figure 3: Aizawa system simulation results

2 Arneodo

2.1 Equation

$$\begin{cases} \dot{x} = y \\ \dot{y} = z \\ \dot{z} = -a \cdot x - b \cdot y - z + c \cdot x^3 \end{cases}$$

2.2 Python model

```
def arneodo(state, __time__):
        x, y, z = state
        return y, z, \
                -a * x - b * y - z + c * x ** 3
```

2.3 Simulink model

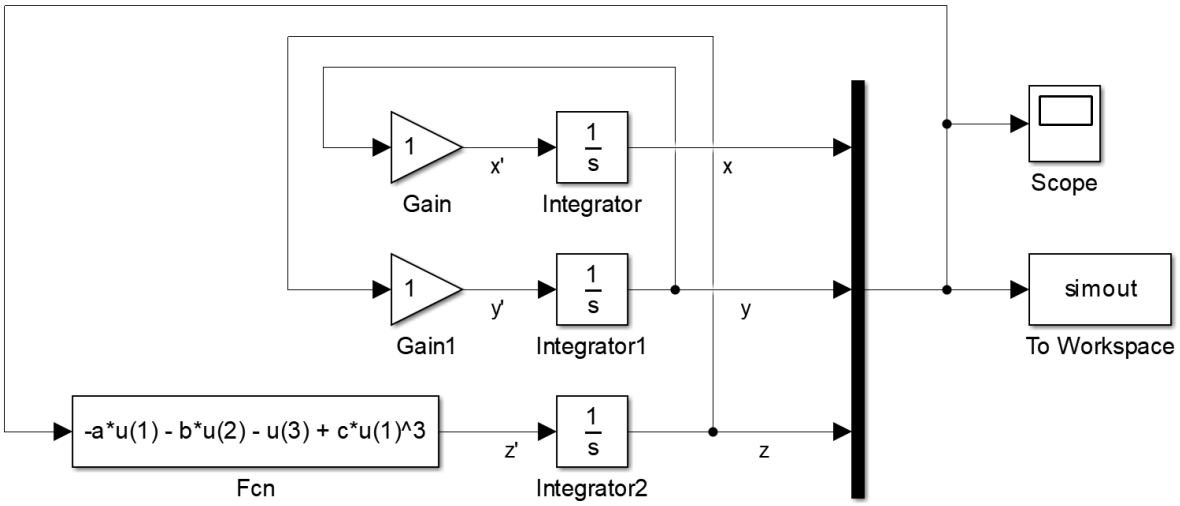

Figure 4: Arneodo Simulink model

2.4 Result

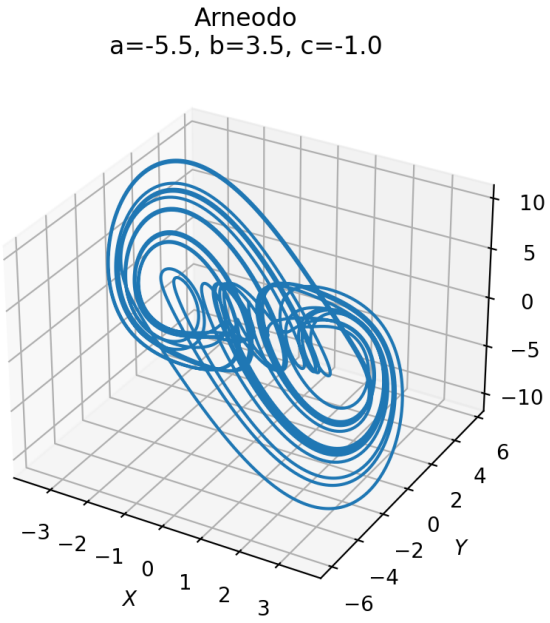

Figure 5: Arneodo system simulation results

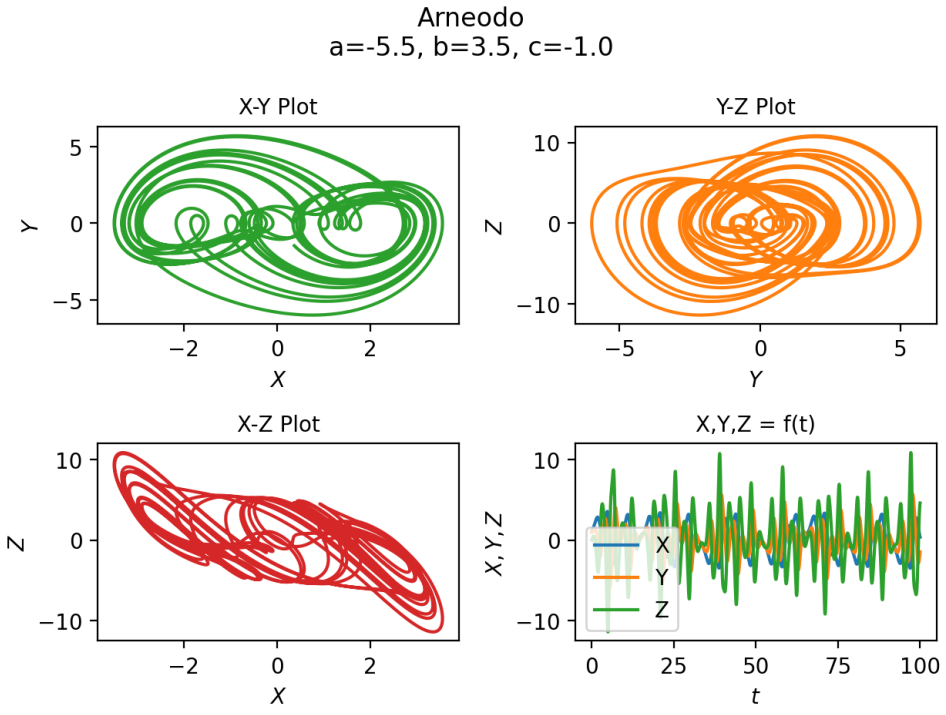

Figure 6: Arneodo system simulation results

3 Belousov-Zhabotinsky

3.1 Equation

$$\begin{cases} \dot{x} = \frac{1}{\epsilon}\left(x + q \cdot y - x^2 - x \cdot y\right) \\ \dot{y} = \frac{1}{\delta}\left(-q \cdot y + f \cdot z - x \cdot y\right) \\ \dot{z} = x - z \end{cases}$$

3.2 Python model

```
def belousov_zhabotinsky(state, __time__):
    x, y, z = state
    return (x + q * y - x ** 2 - x * y) / e, \
       (-q * y + f * z - x * y) / d, \
       x - z
```

3.3 Simulink model

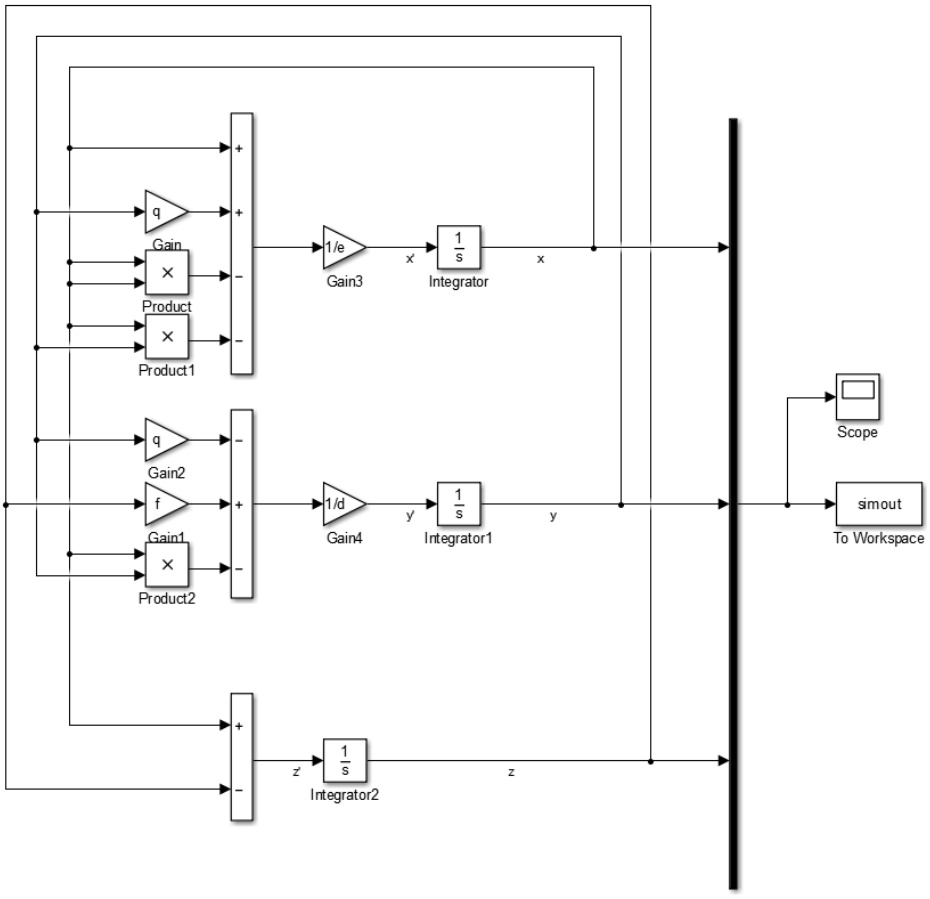

Figure 7: Belousov-Zhabotinsky Simulink model

3.4 Result

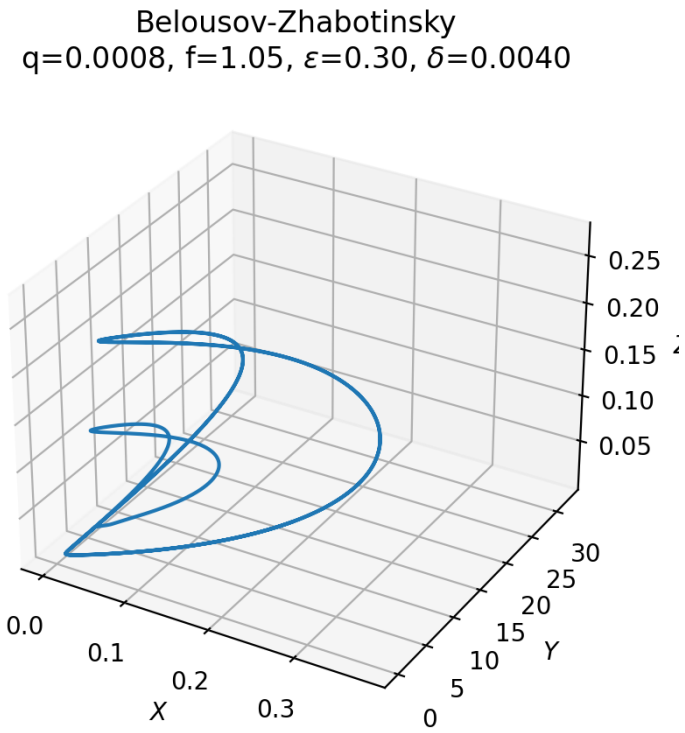

Figure 8: Belousov-Zhabotinsky system simulation results

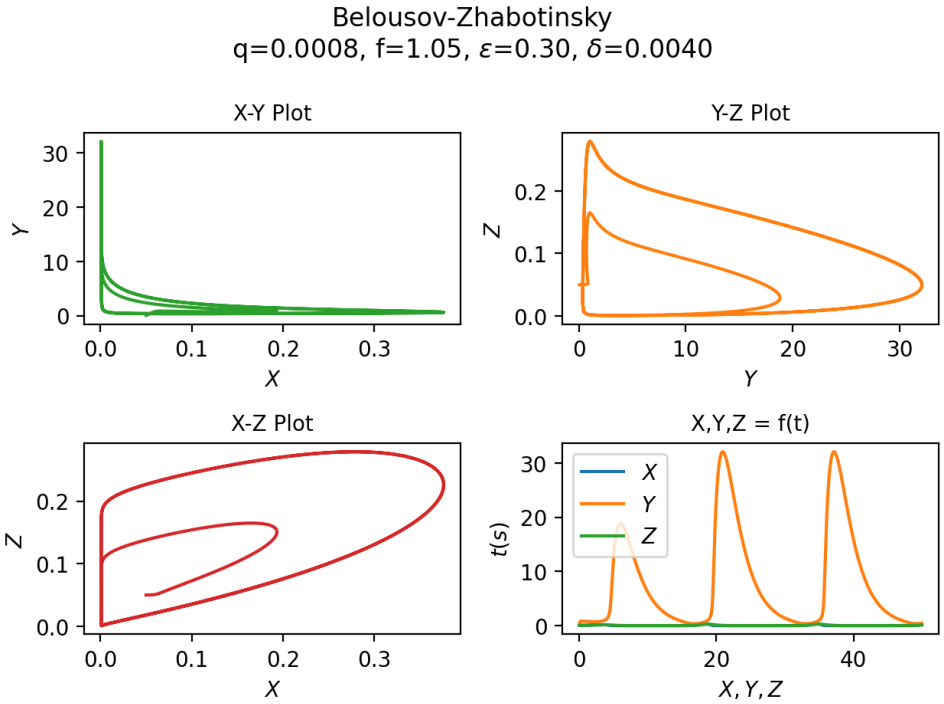

Figure 9: Belousov-Zhabotinsky system simulation results

4 Chen

4.1 Equation

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

4.2 Python model

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

4.3 Simulink model

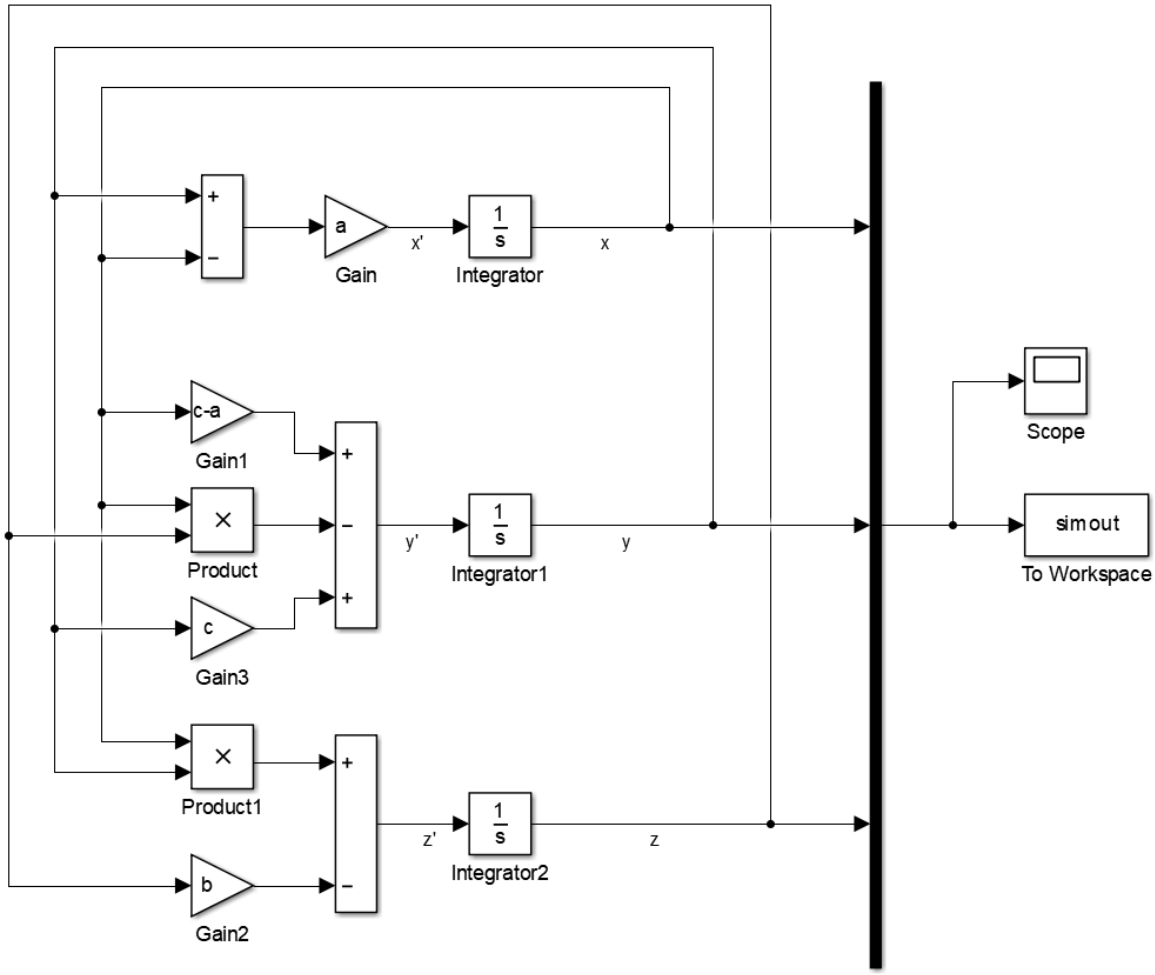

Figure 10: Chen Simulink model

4.4 Result

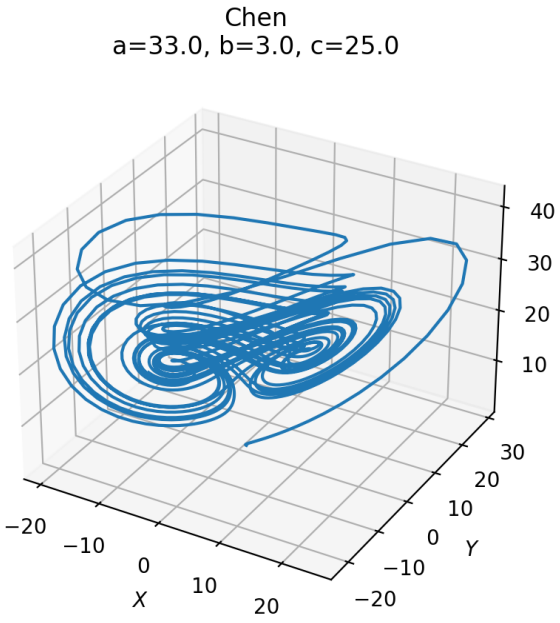

Figure 11: Chen system simulation results

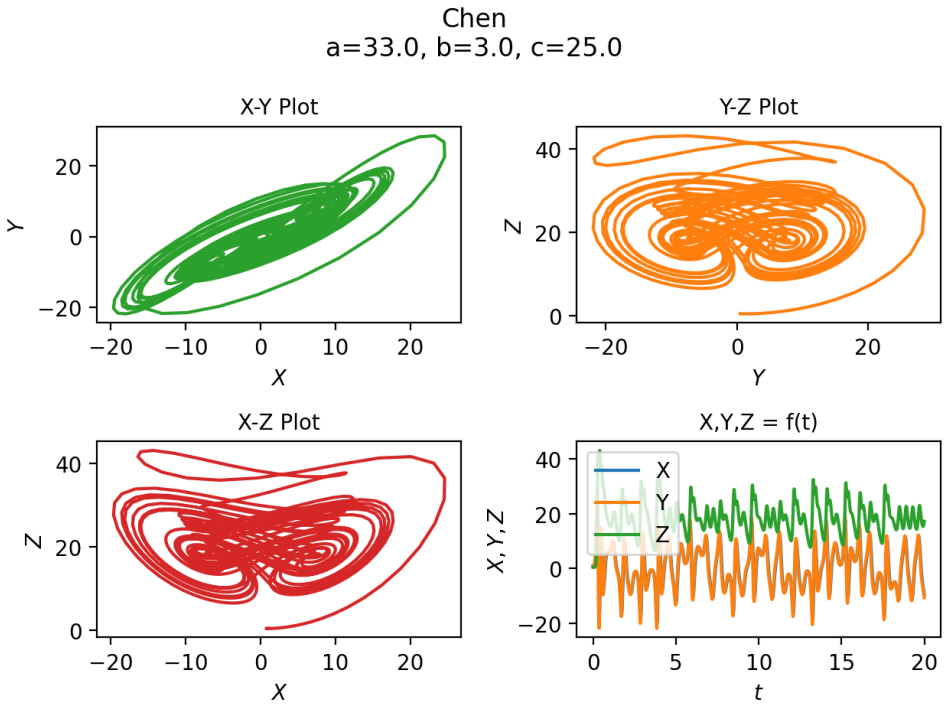

Figure 12: Chen system simulation results

5 Chua

5.1 Equation

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

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

5.2 Python model

```
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
```

5.3 Simulink model

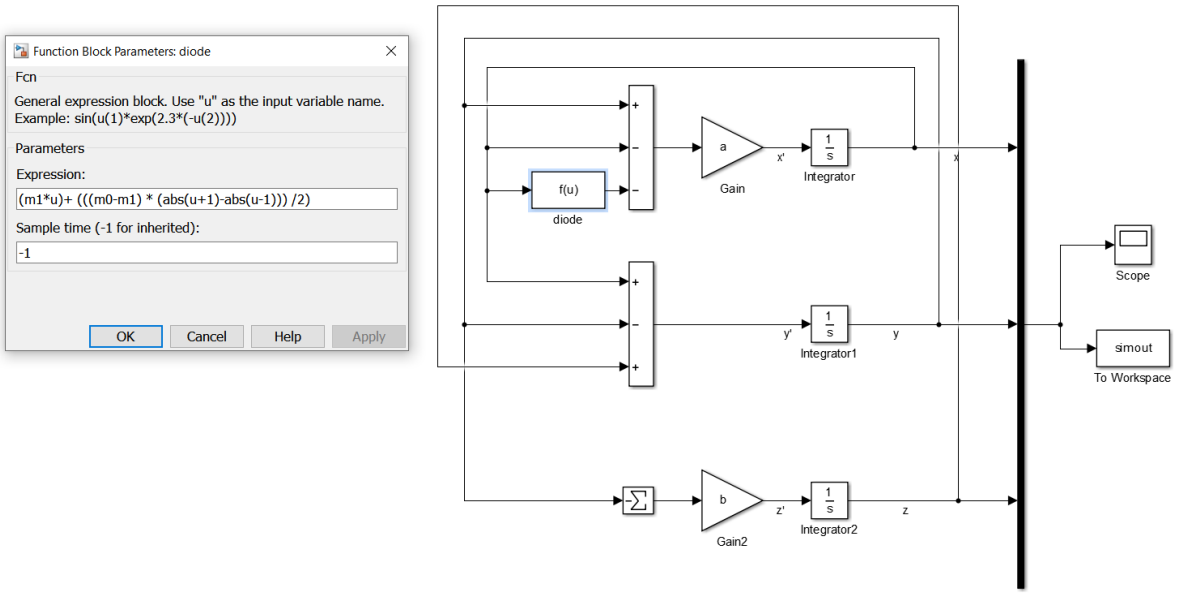

Figure 13: Chua Simulink model

5.4 Result

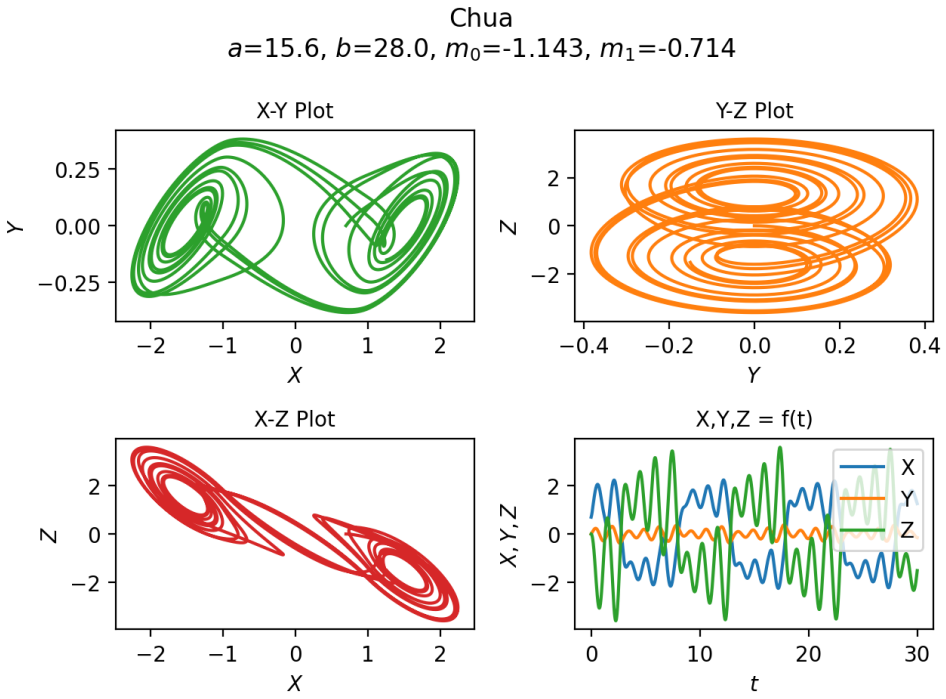

Figure 14: Chua system simulation results

5.5 Electronics Circuit

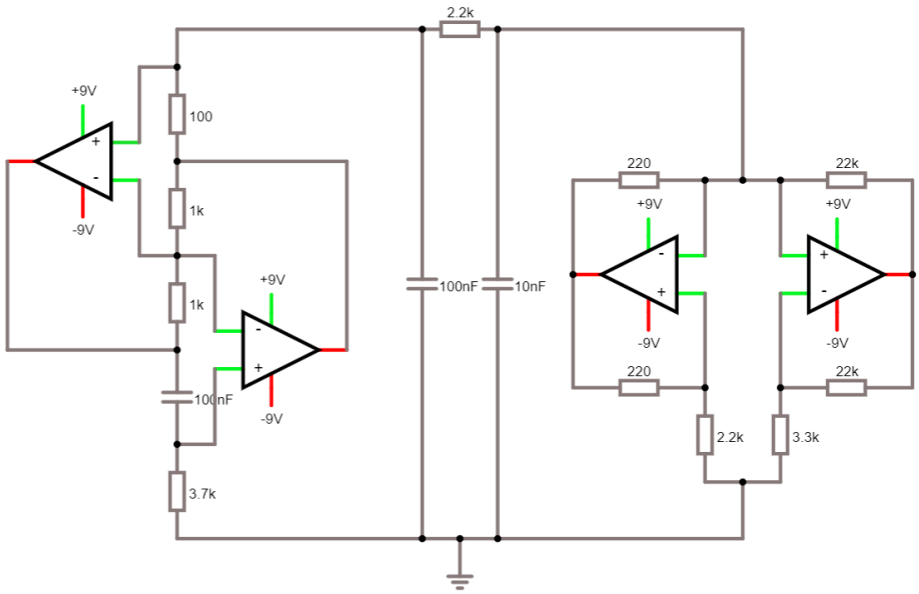

Figure 15: Chua system electronics circuit

6 Chua (Hyperchaotic)

6.1 Equation

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

6.2 Python model

```
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
```

6.3 Simulink model

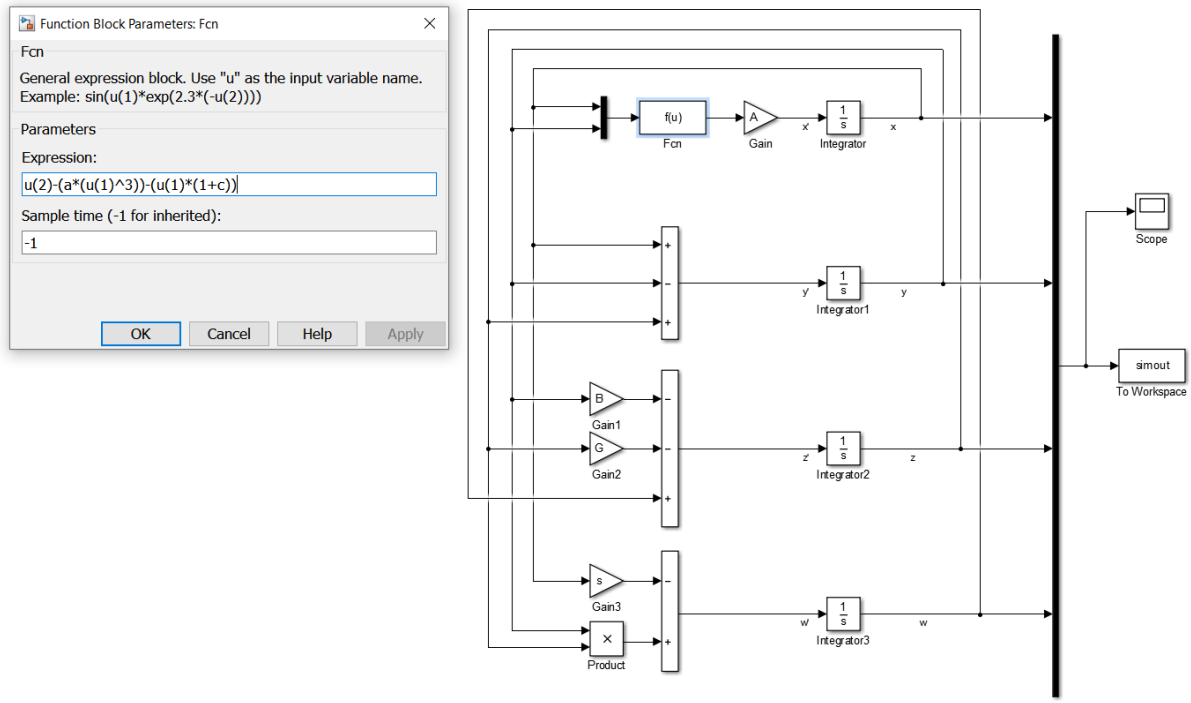

Figure 16: Hyperchaotic Chua Simulink model

6.4 Result

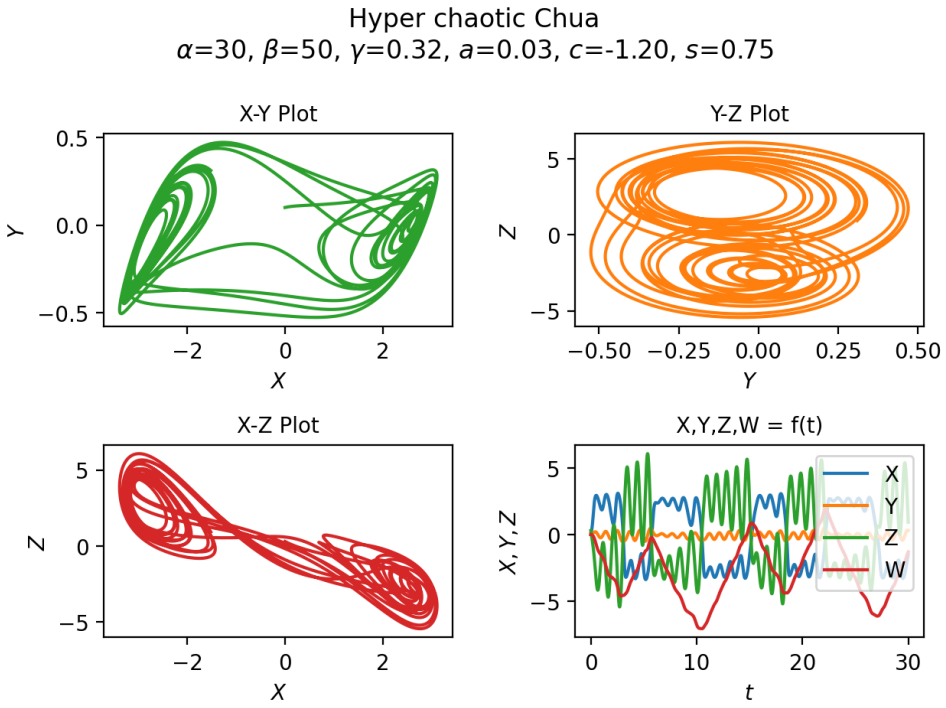

Figure 17: Hyperchaotic Chua system simulation results

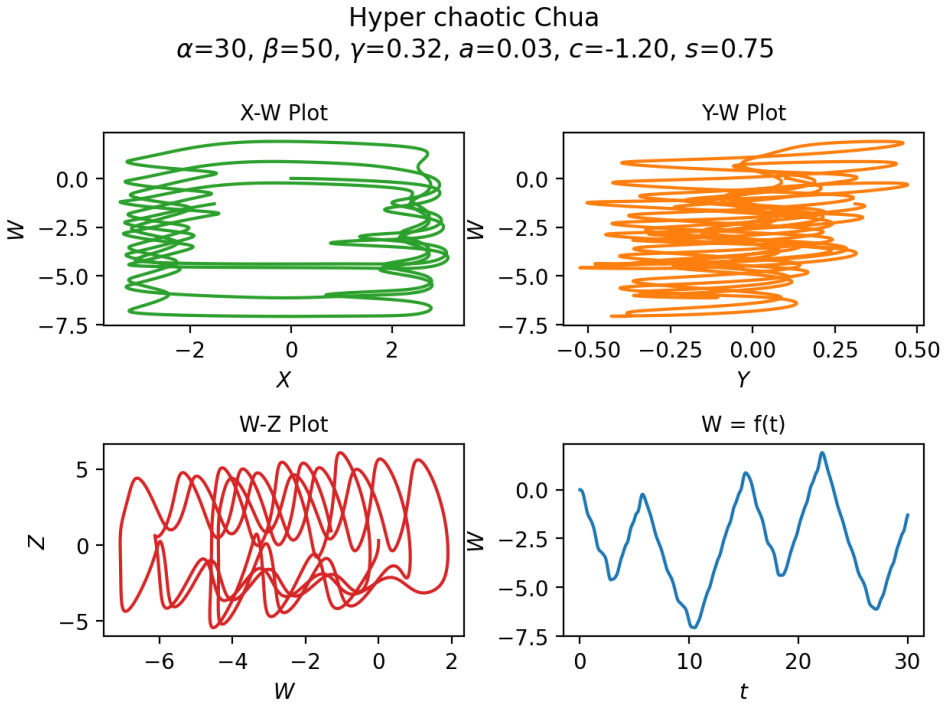

Figure 18: Hyperchaotic Chua system simulation results

7 Dadras

7.1 Equation

$$\begin{cases} \dot{x} = y - p \cdot x + o \cdot y \cdot z \\ \dot{y} = r \cdot y - x \cdot z + z \\ \dot{z} = c \cdot x \cdot y - e \cdot z \end{cases}$$

7.2 Python model

```
def dadras(state, __time__):
        x, y, z = state
        return y - p * x + o * y * z, \
                r * y - x * z + z, \
                c * x * y - e * z
```

7.3 Simulink model

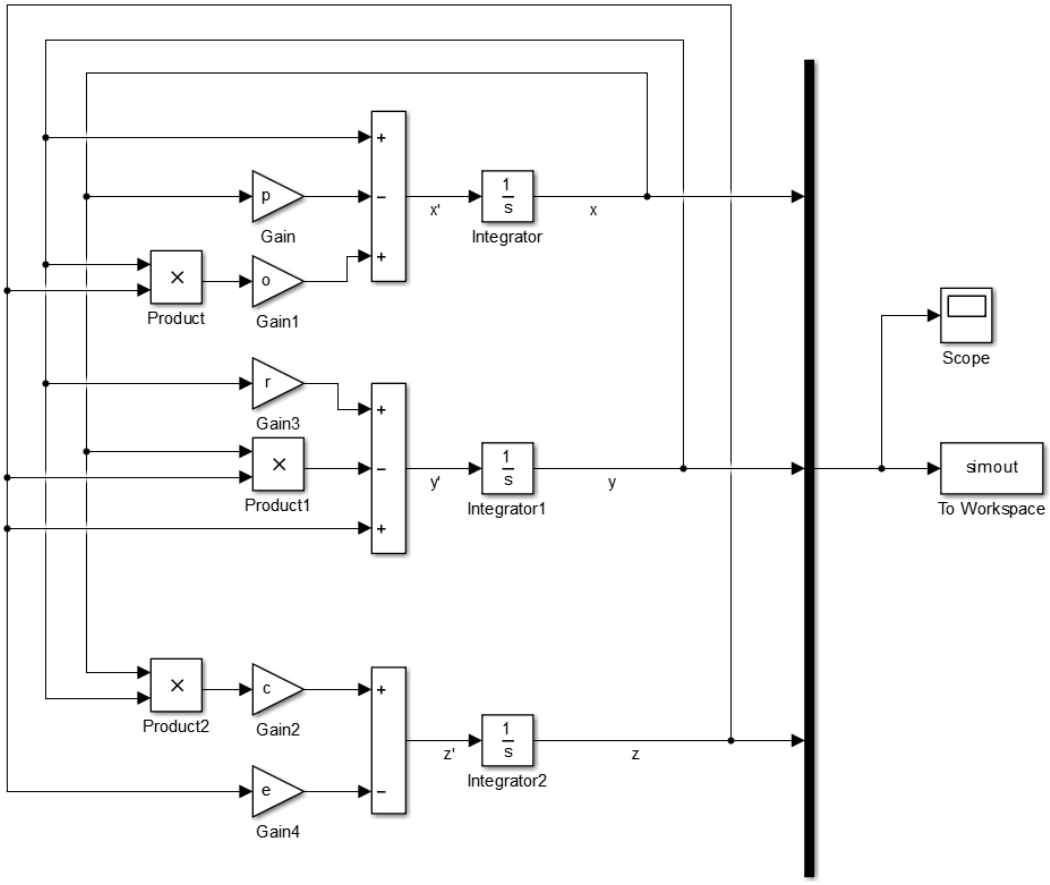

Figure 19: Dadras Simulink model

7.4 Result

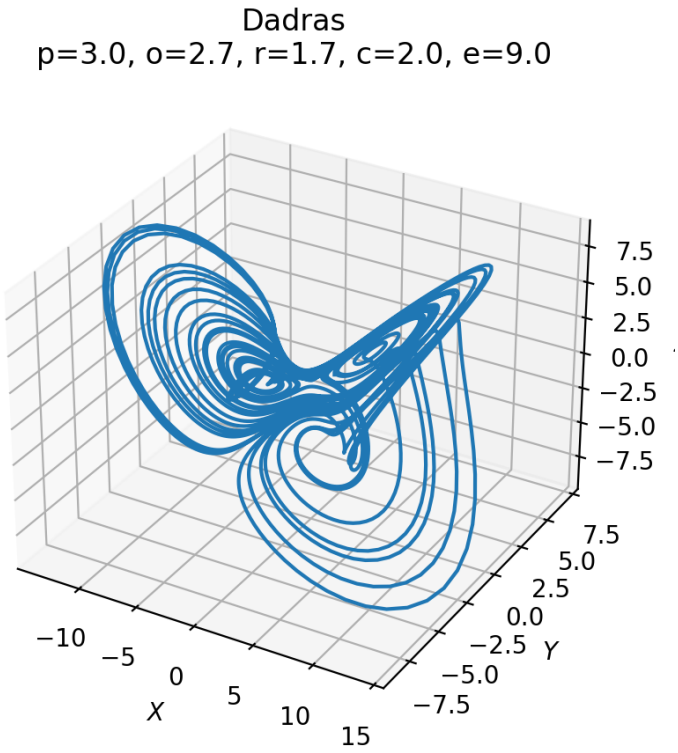

Figure 20: Dadras system simulation results

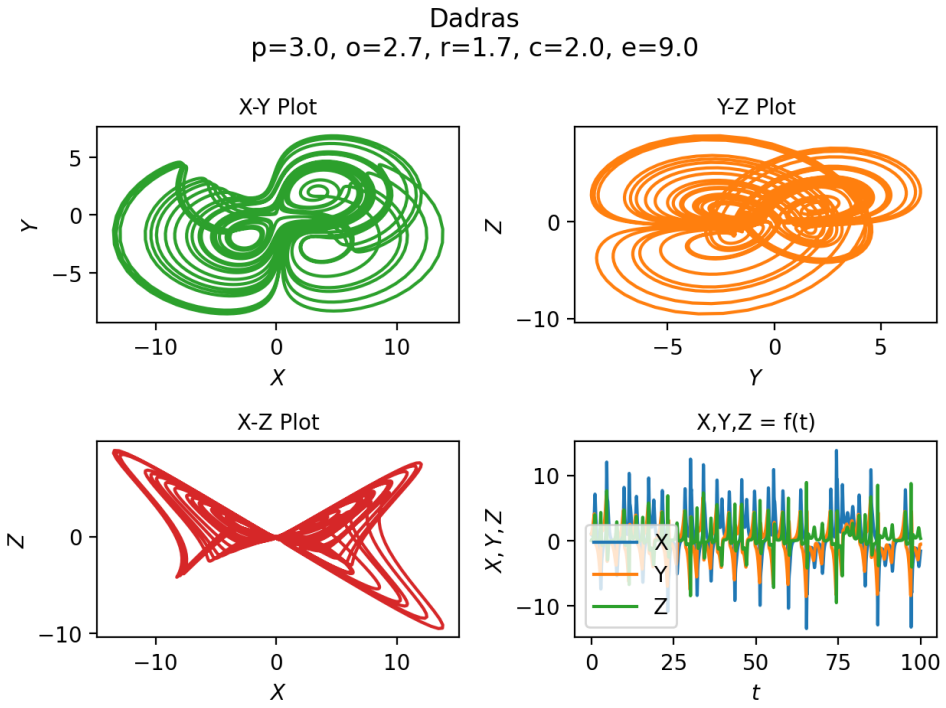

Figure 21: Dadras system simulation results

8 Duffing

8.1 Equation

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

8.2 Python model

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

8.3 Simulink model

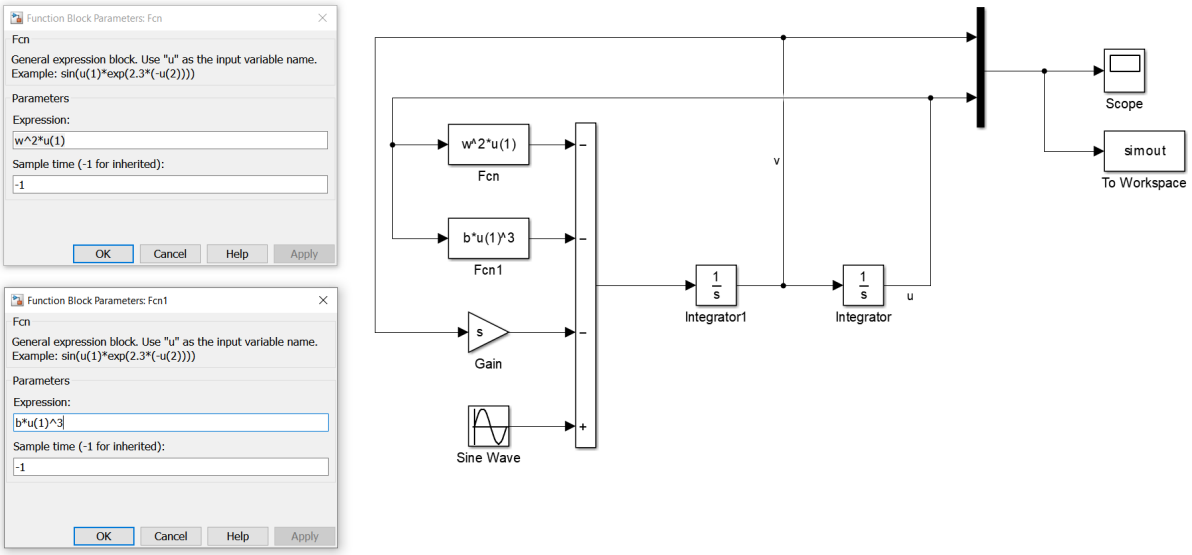

Figure 22: Duffing Simulink model

8.4 Result

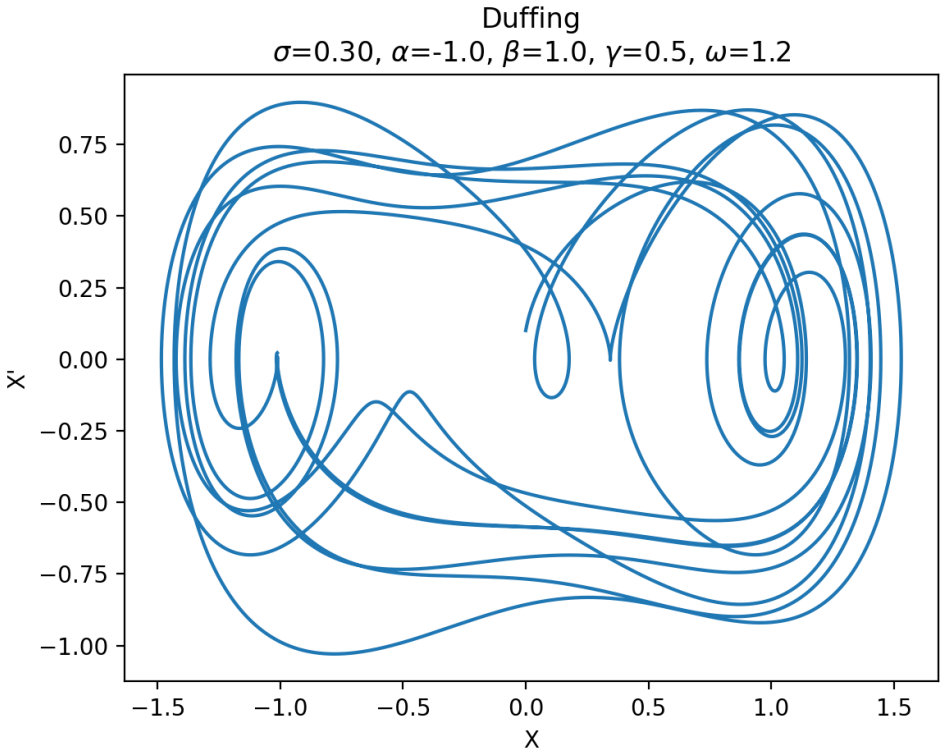

Figure 23: Duffing system simulation results

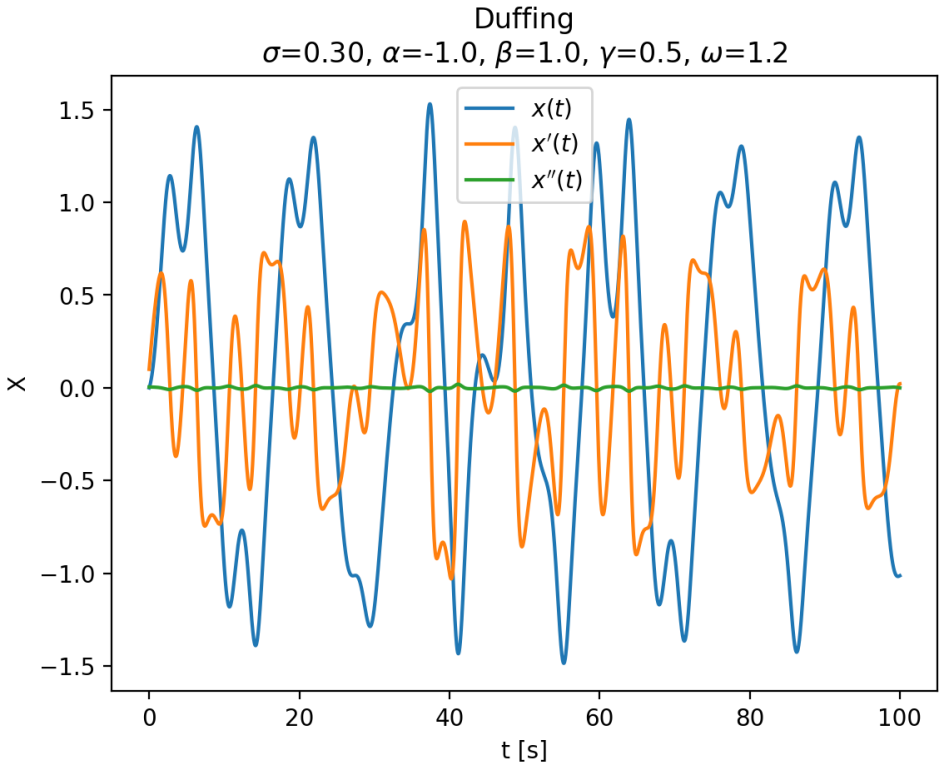

Figure 24: Duffing system simulation results

9 FitzHugh-Nagumo

9.1 Equation

$$\begin{cases} \dot{v} = v - \frac{v^3}{3} - w - I \\ \dot{w} = \epsilon \left(v + a - b \cdot w \right) \end{cases}$$

$$I = A \sin(\omega \cdot t)$$

9.2 Python model

```
def fitzhugh_nagumo(state, time):
    v, w = state
    return v - v ** 3 / 3 - w + (A * np.sin(f * time)), \
        e * (v + a - b * w)
```

9.3 Simulink model

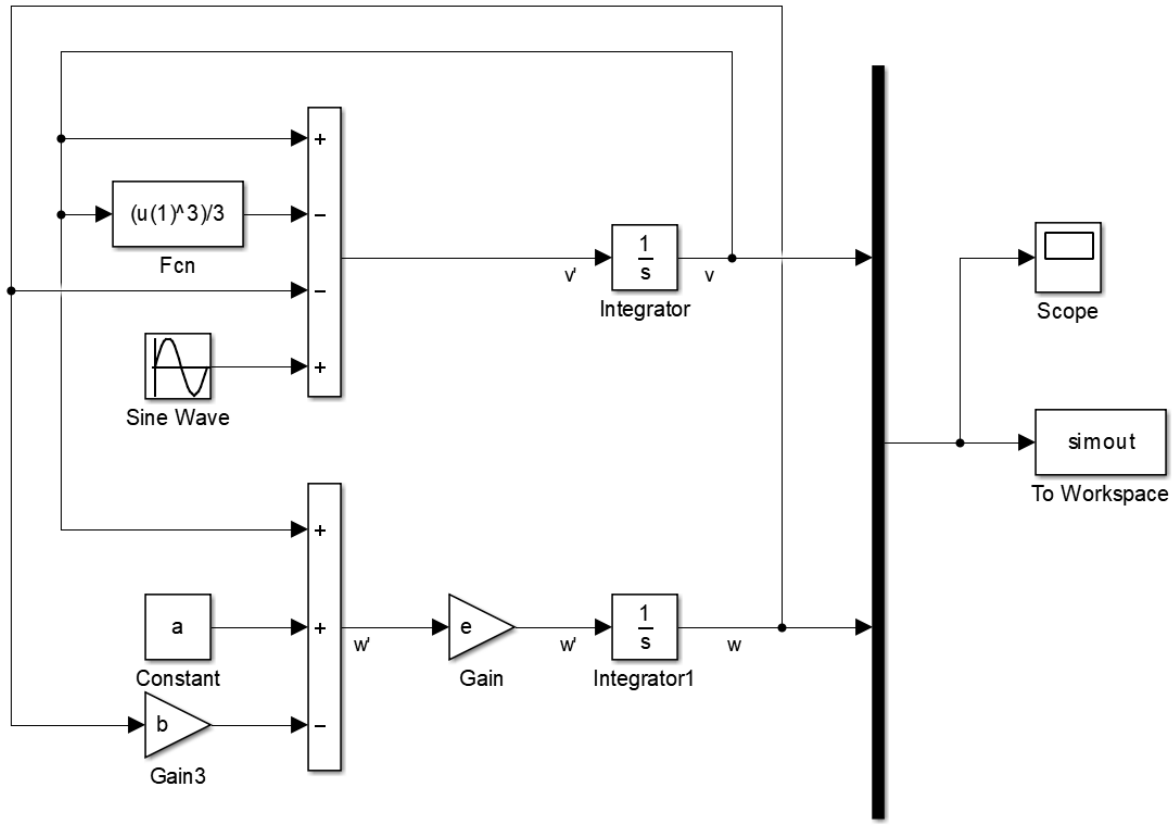

Figure 25: FitzHugh-Nagumo Simulink model

9.4 Result

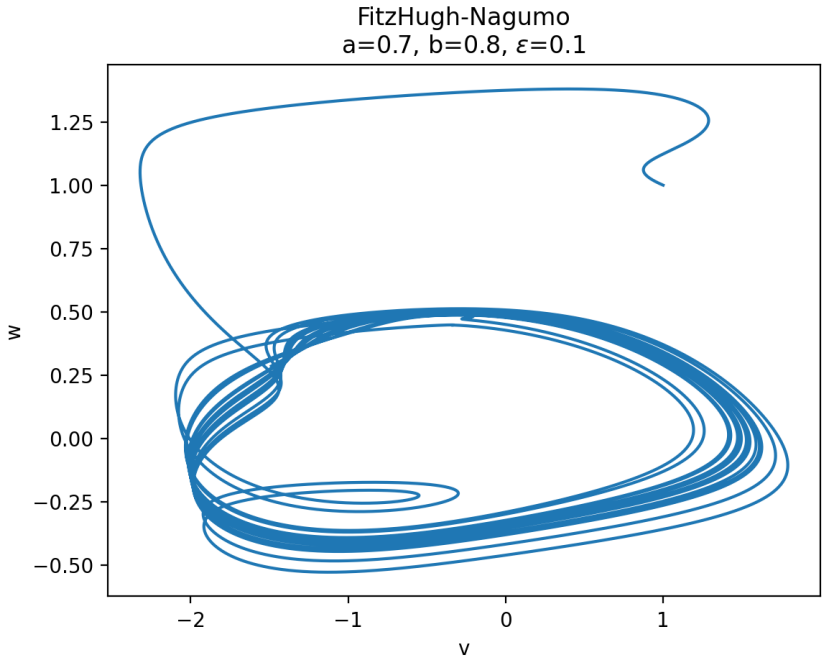

Figure 26: FitzHugh-Nagumo system simulation results

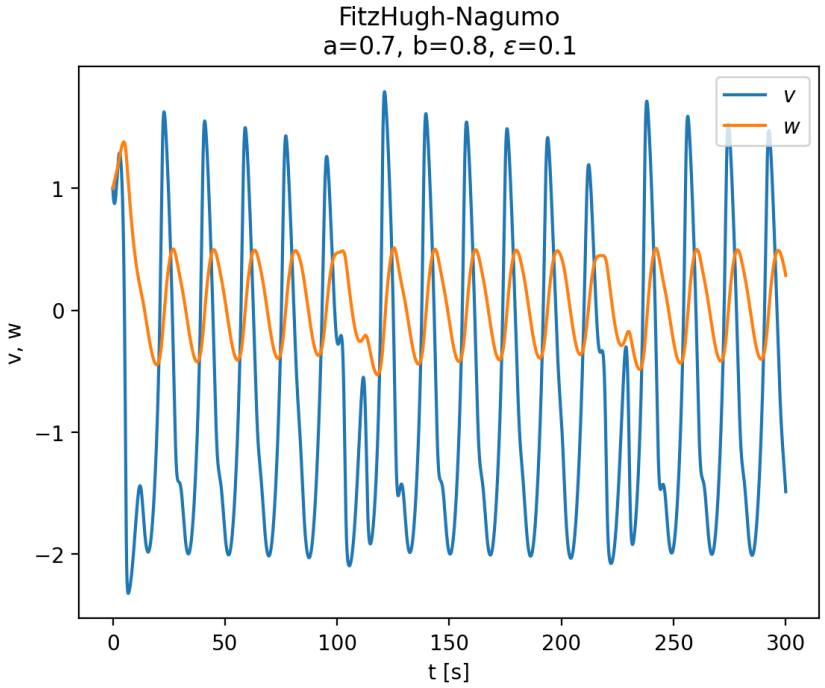

Figure 27: FitzHugh-Nagumo system simulation results

10 Halvorsen

10.1 Equation

$$\begin{cases} \dot{x} = -a \cdot x - 4 \cdot (y + z) - y^2 \\ \dot{y} = -a \cdot y - 4 \cdot (z + x) - z^2 \\ \dot{z} = -a \cdot z - 4 \cdot (x + y) - x^2 \end{cases}$$

10.2 Python model

```
def halvorsen(state, __time__):
        x, y, z = state
        return -a * x - 4 * (y + z) - y ** 2, \
                -a * y - 4 * (z + x) - z ** 2, \
                -a * z - 4 * (x + y) - x ** 2
```

10.3 Simulink model

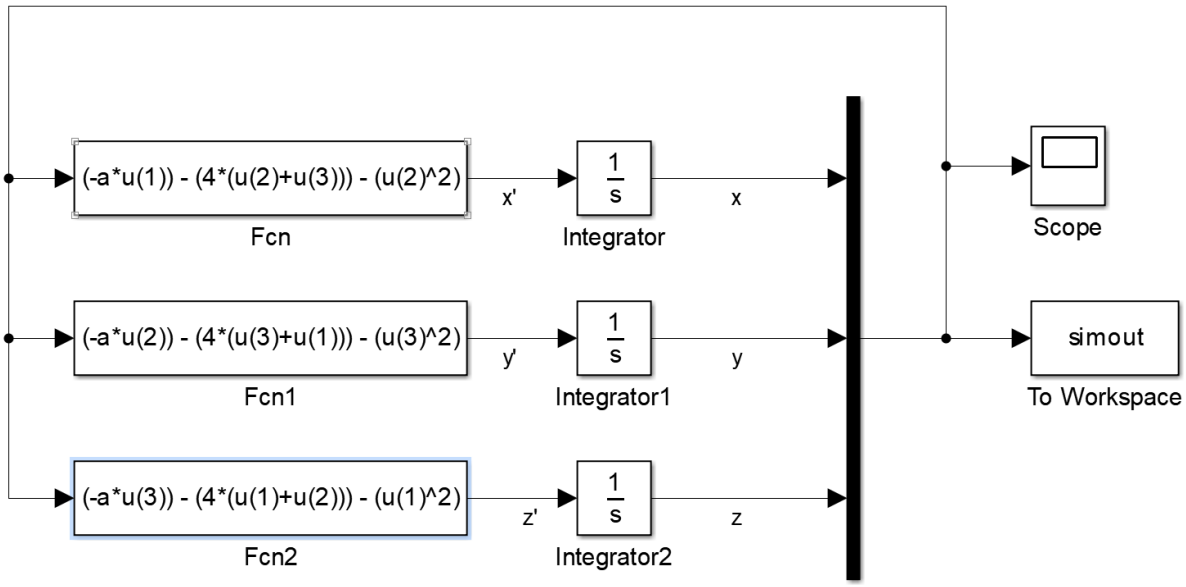

Figure 28: Halvorsen Simulink model

10.4 Result

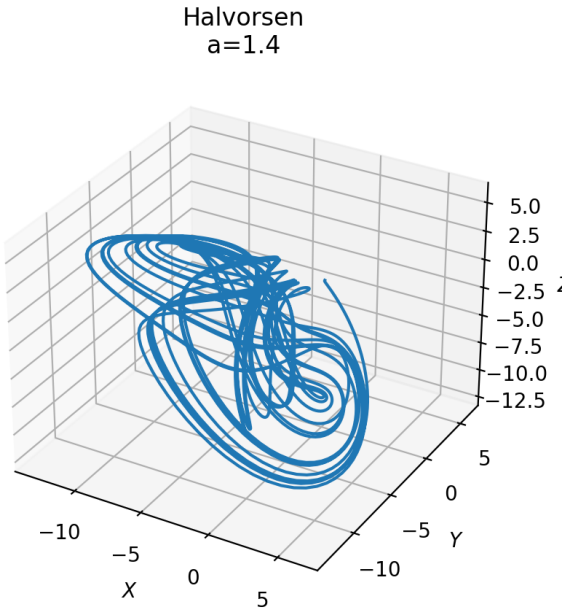

Figure 29: Halvorsen system simulation results

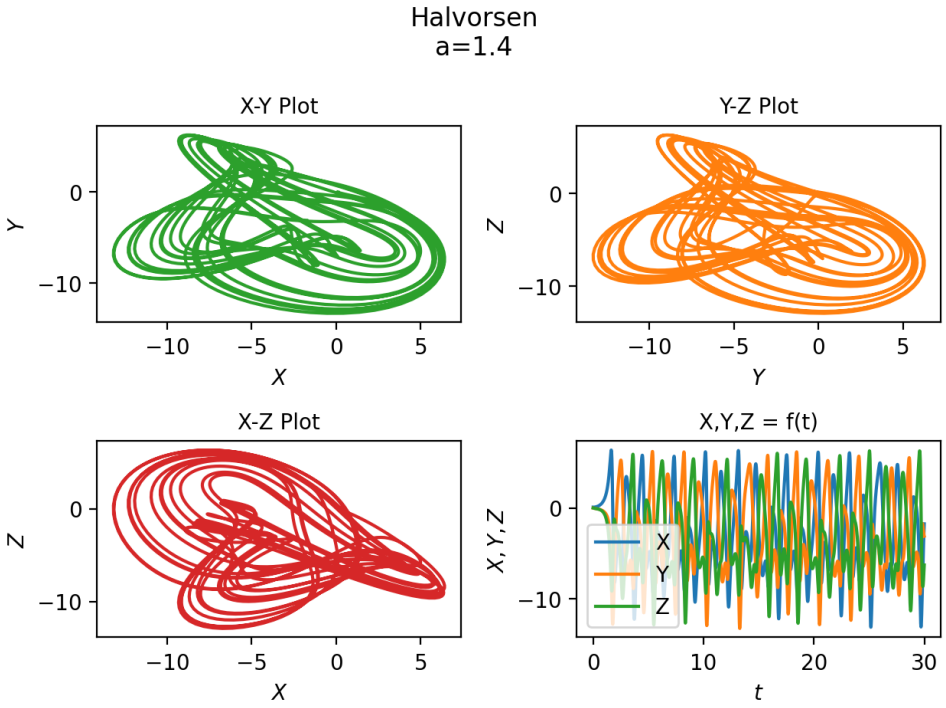

Figure 30: Halvorsen system simulation results

11 Henon-Heiles

11.1 Equation

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

11.2 Python model

```
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
```

11.3 Simulink model

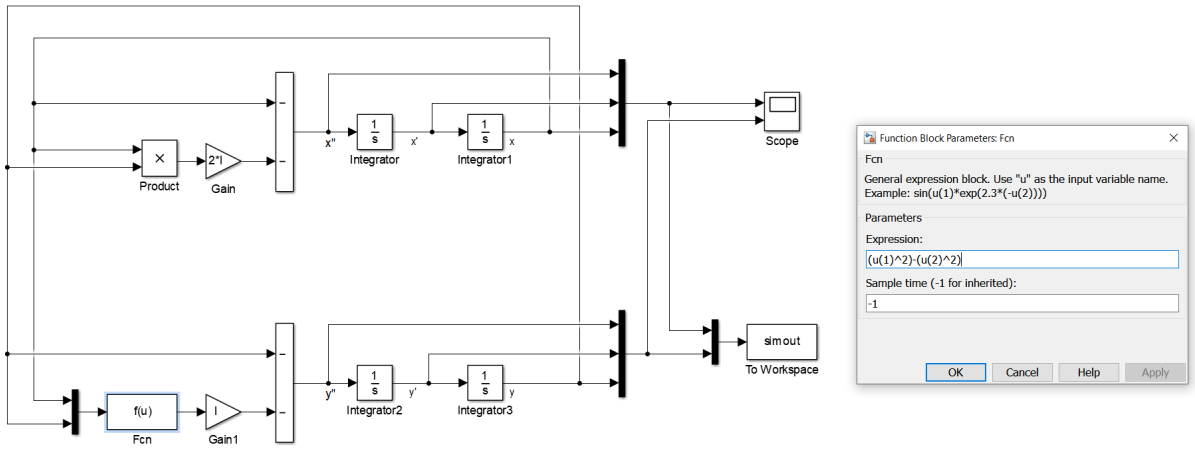

Figure 31: Henon-Heiles Simulink model

11.4 Result

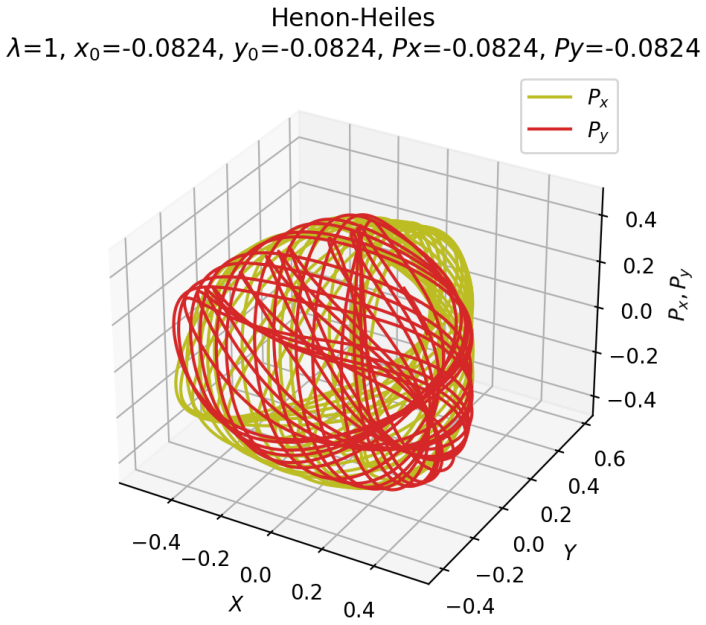

Figure 32: Henon-Heiles system simulation results

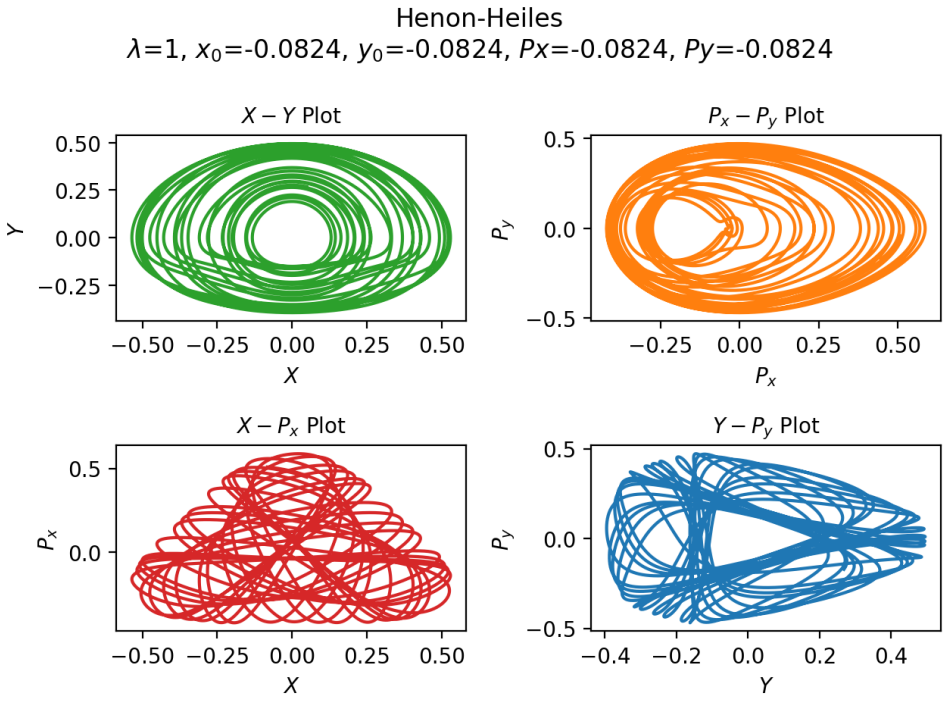

Figure 33: Henon-Heiles system simulation results

12 Hindmarsh-Rose

12.1 Equation

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

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

12.2 Python model

```
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
           )
```

12.3 Simulink model

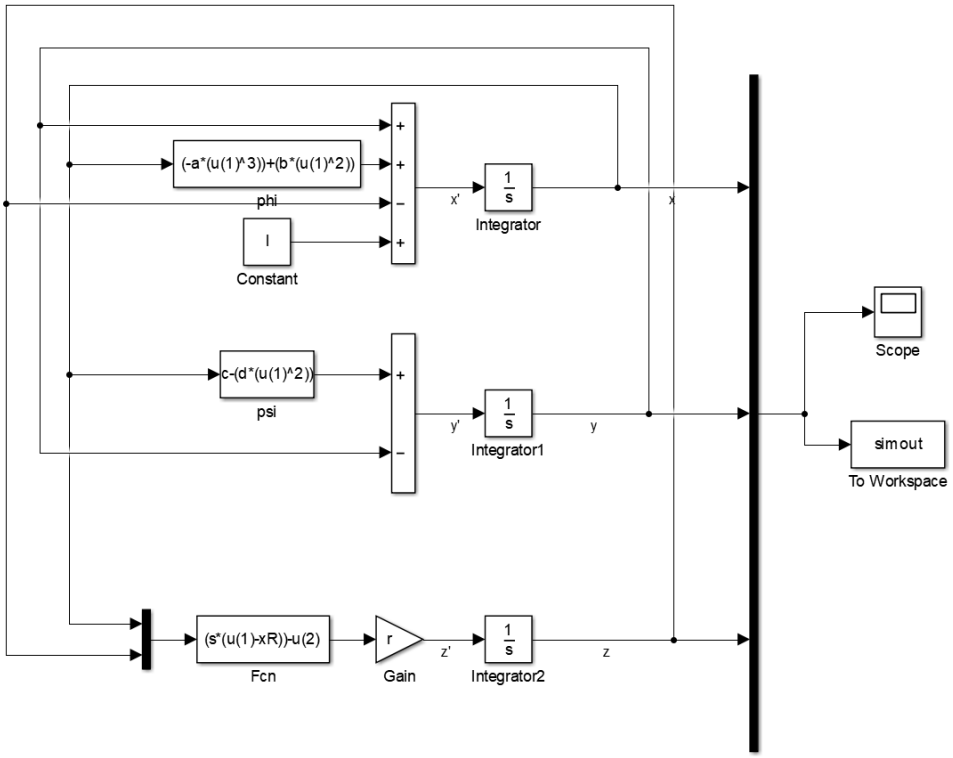

Figure 34: Hindmarsh-Rose Simulink model

12.4 Result

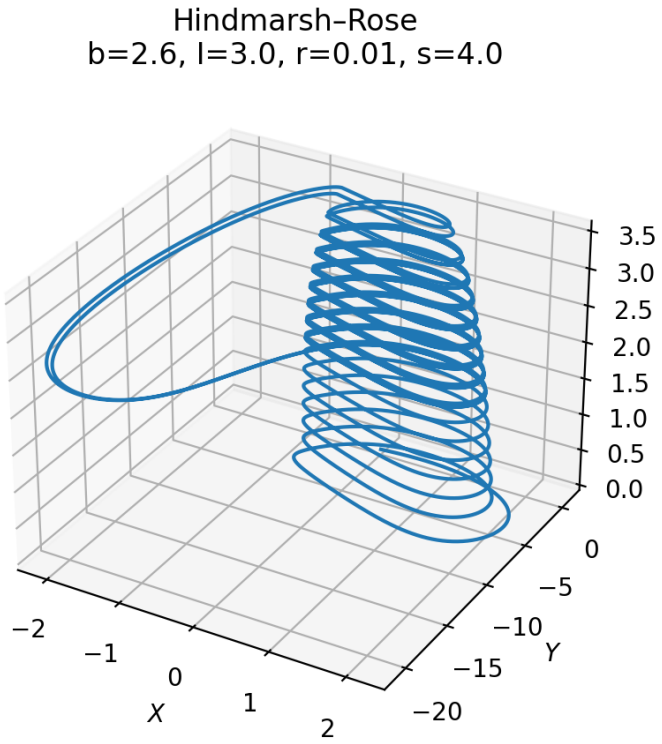

Figure 35: Hindmarsh-Rose system simulation results

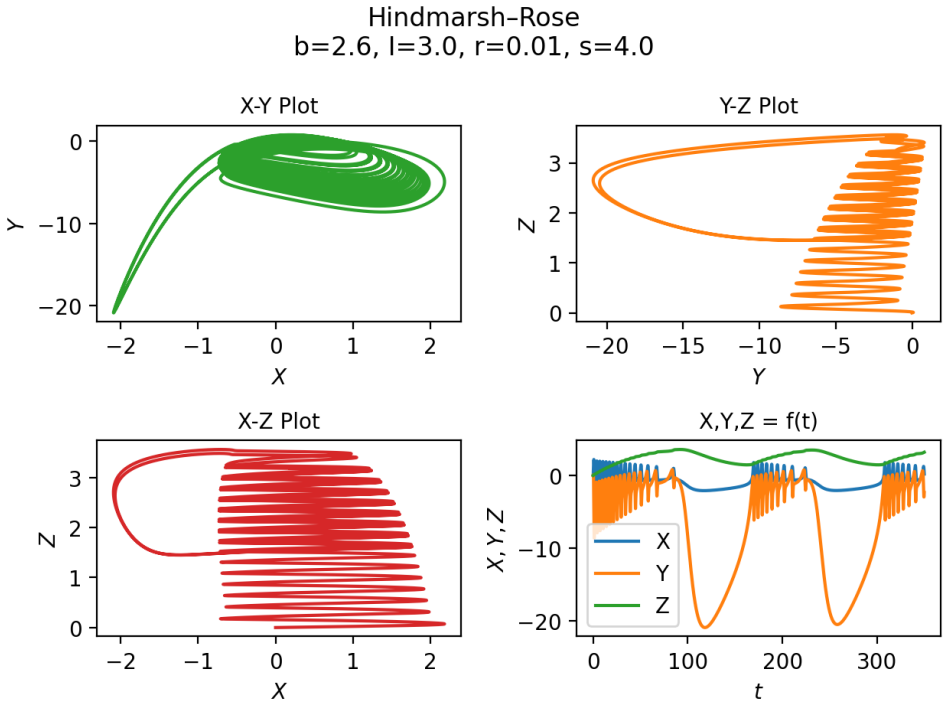

Figure 36: Hindmarsh-Rose system simulation results

13 Lorenz

13.1 Equation

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

13.2 Python model

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

13.3 Simulink model

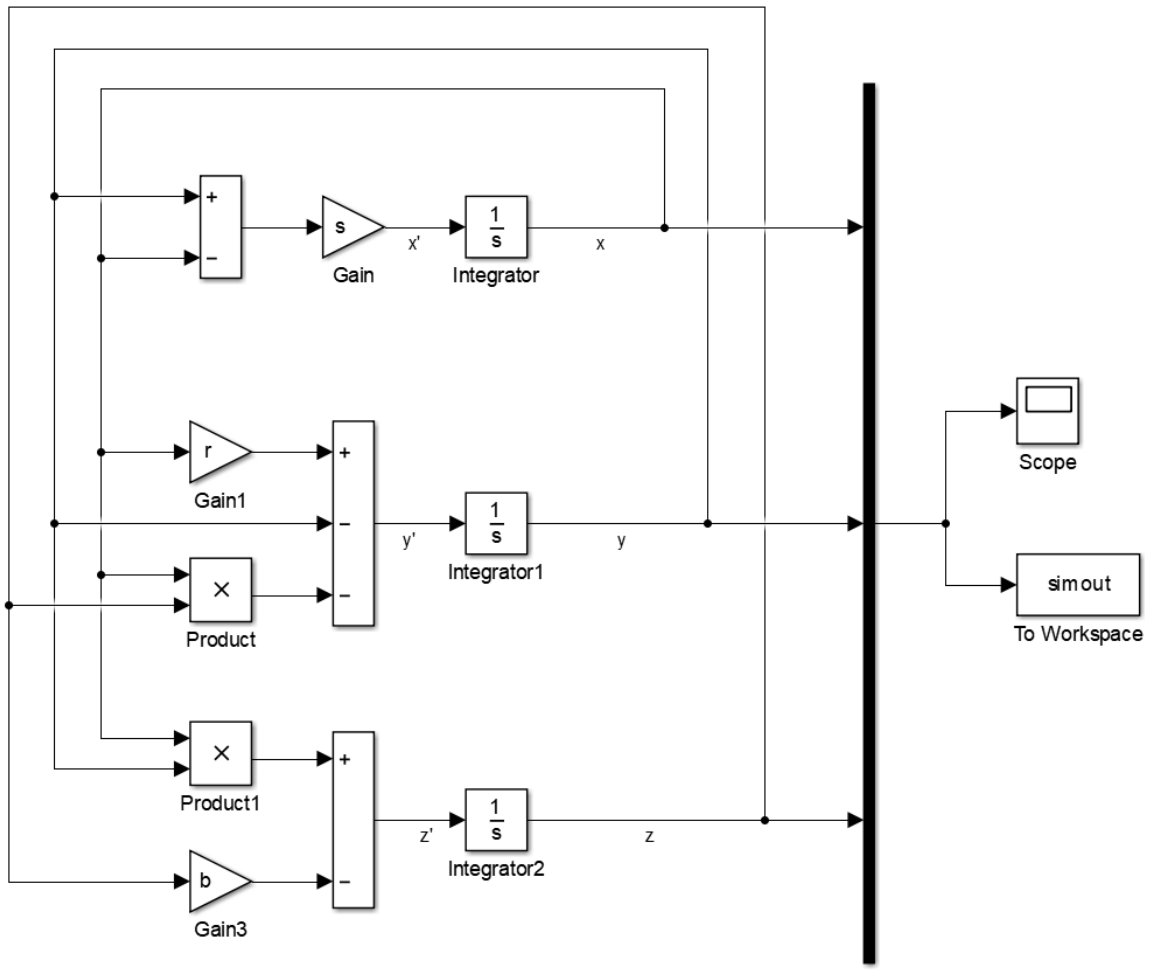

Figure 37: Lorenz Simulink model

13.4 Result

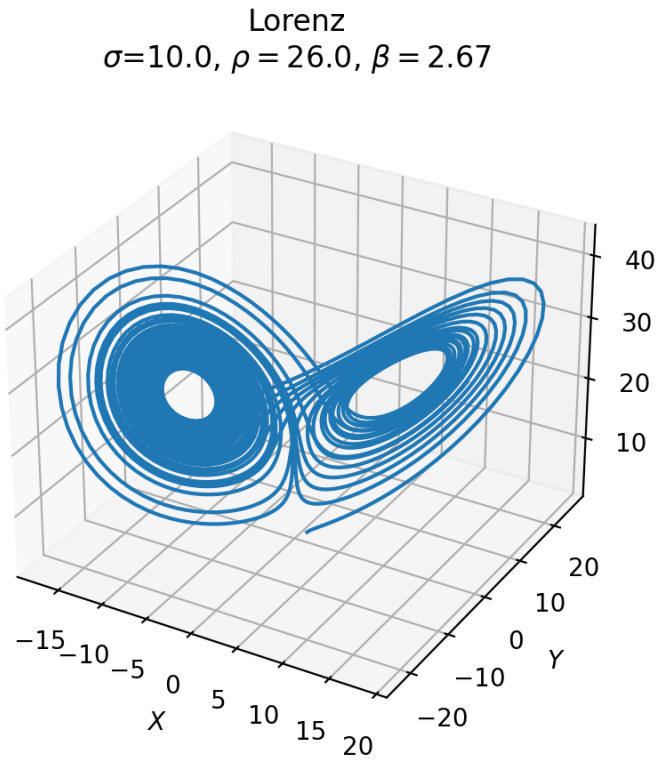

Figure 38: Lorenz system simulation results

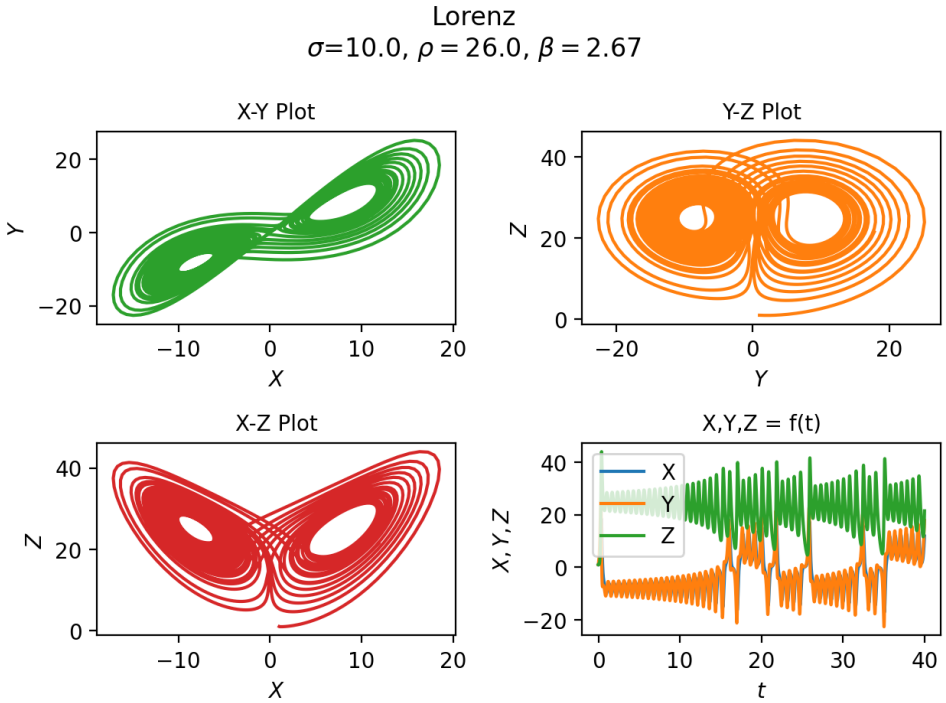

Figure 39: Lorenz system simulation results

14 Lu

14.1 Equation

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

14.2 Python model

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

14.3 Simulink model

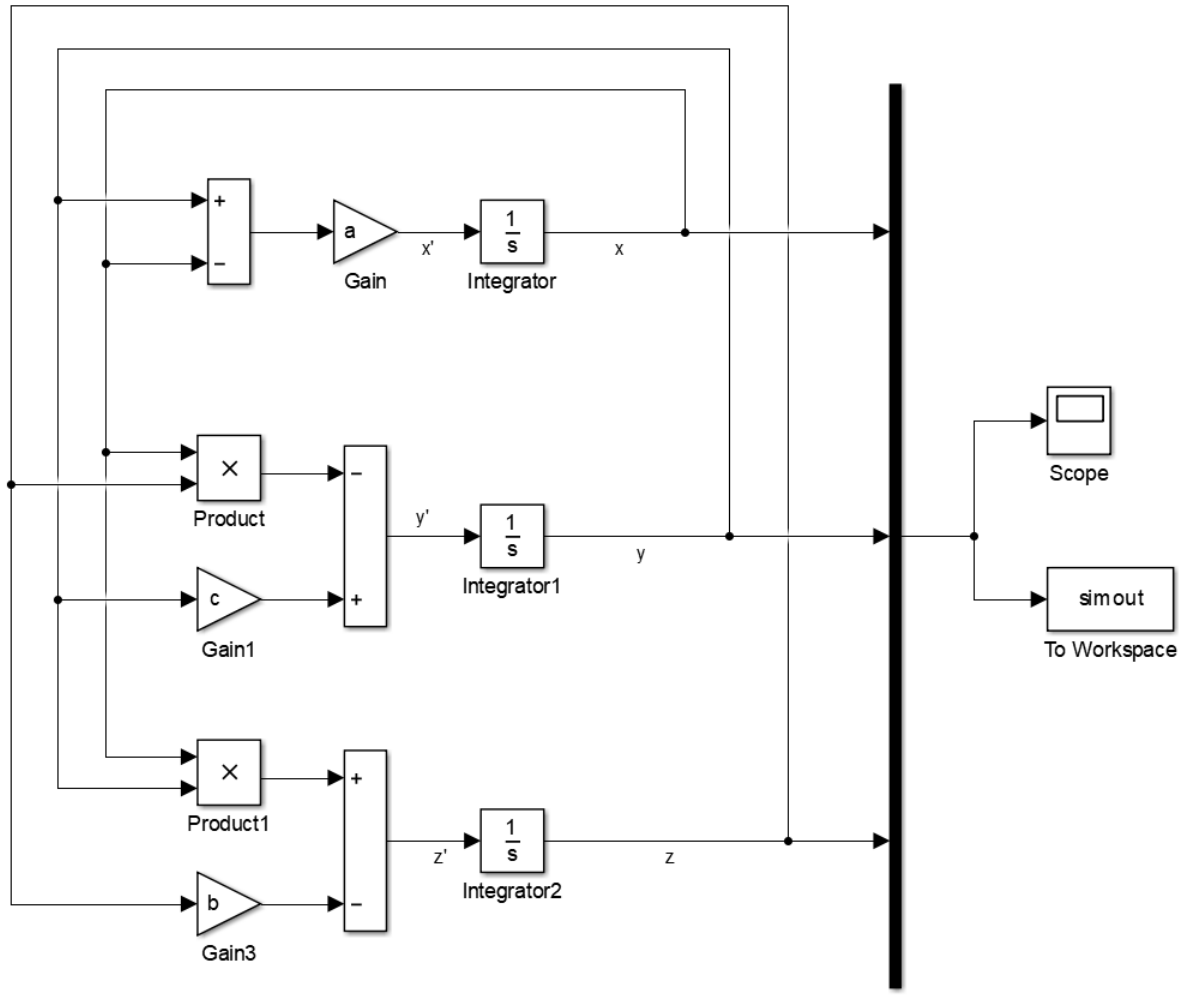

Figure 40: Lu Simulink model

14.4 Result

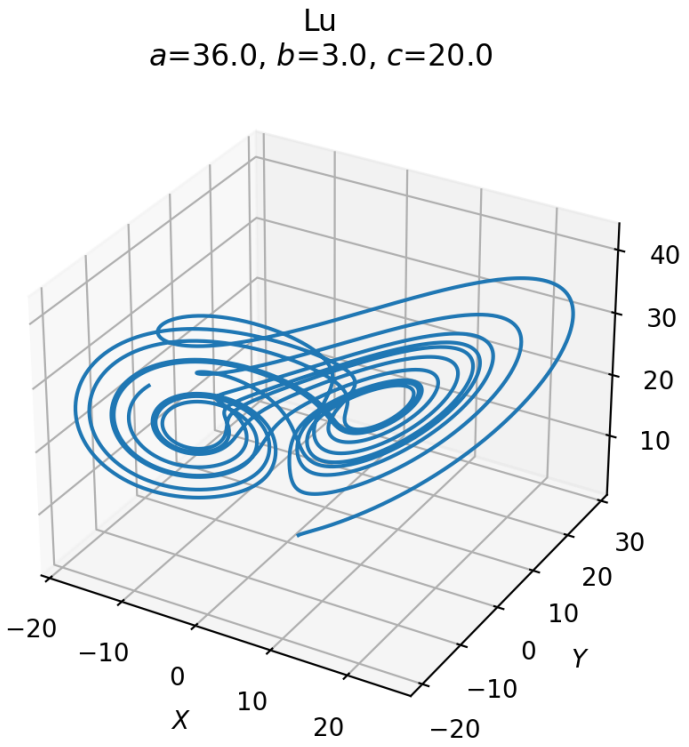

Figure 41: Lu system simulation results

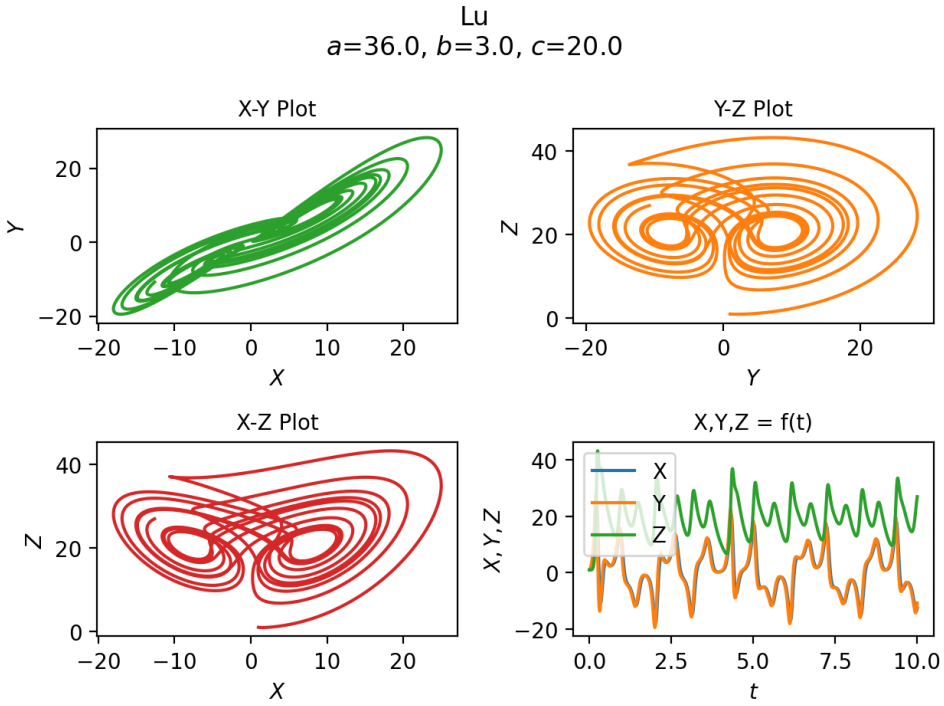

Figure 42: Lu system simulation results

15 Mackey-Glass

15.1 Equation

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

15.2 Python model

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

15.3 Simulink model

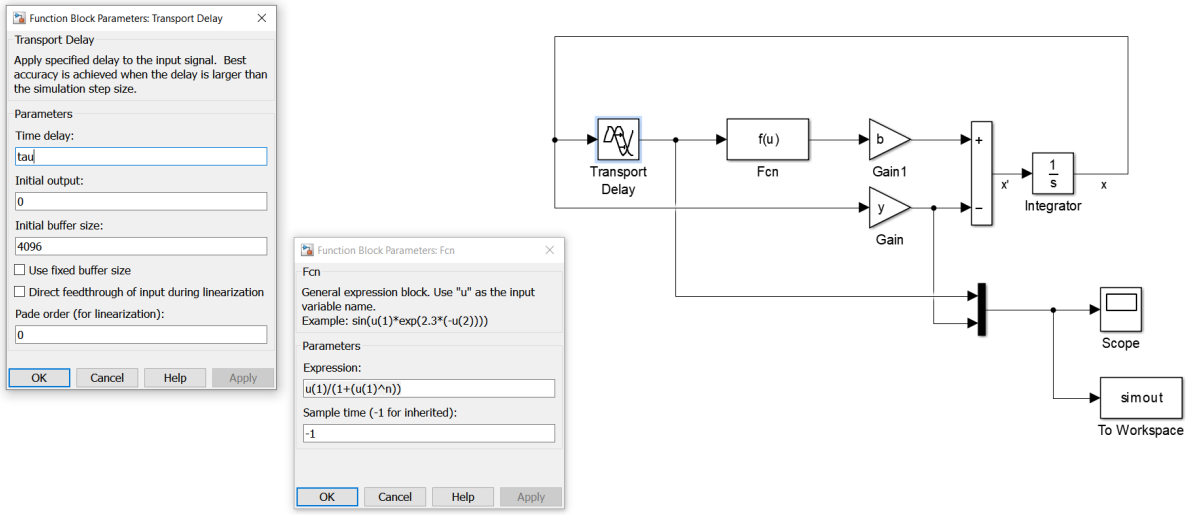

Figure 43: Mackey-Glass Simulink model

15.4 Result

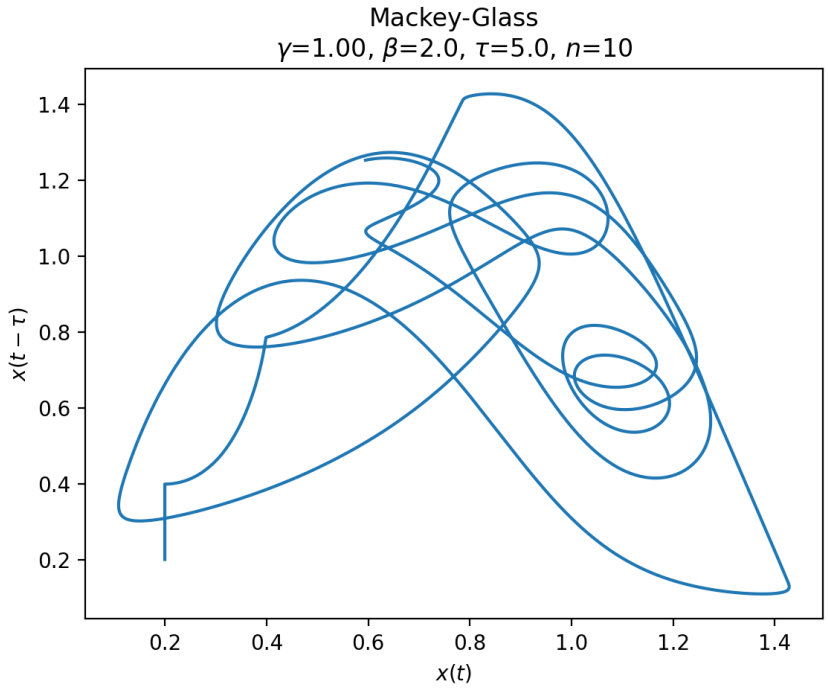

Figure 44: Mackey-Glass system simulation results

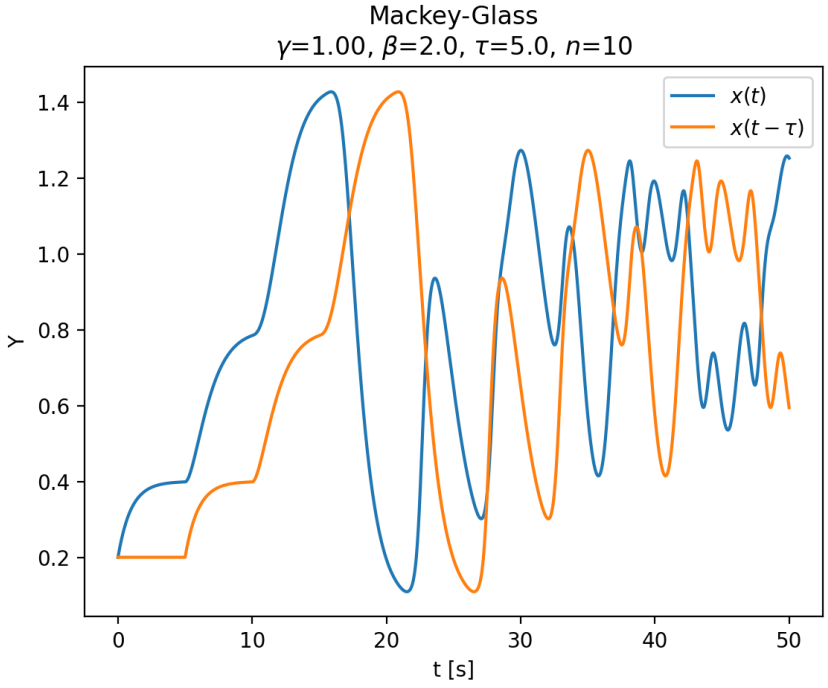

Figure 45: Mackey-Glass system simulation results

16 Nose-Hoover

16.1 Equation

$$\begin{cases} \dot{x} = y \\ \dot{y} = -x - y \cdot z \\ \dot{z} = 1 - y^2 \end{cases}$$

16.2 Python model

```
def nose_hoover(state, __time__):
        x, y, z = state
        return y, -x + y * z, 1 - y ** 2
```

16.3 Simulink model

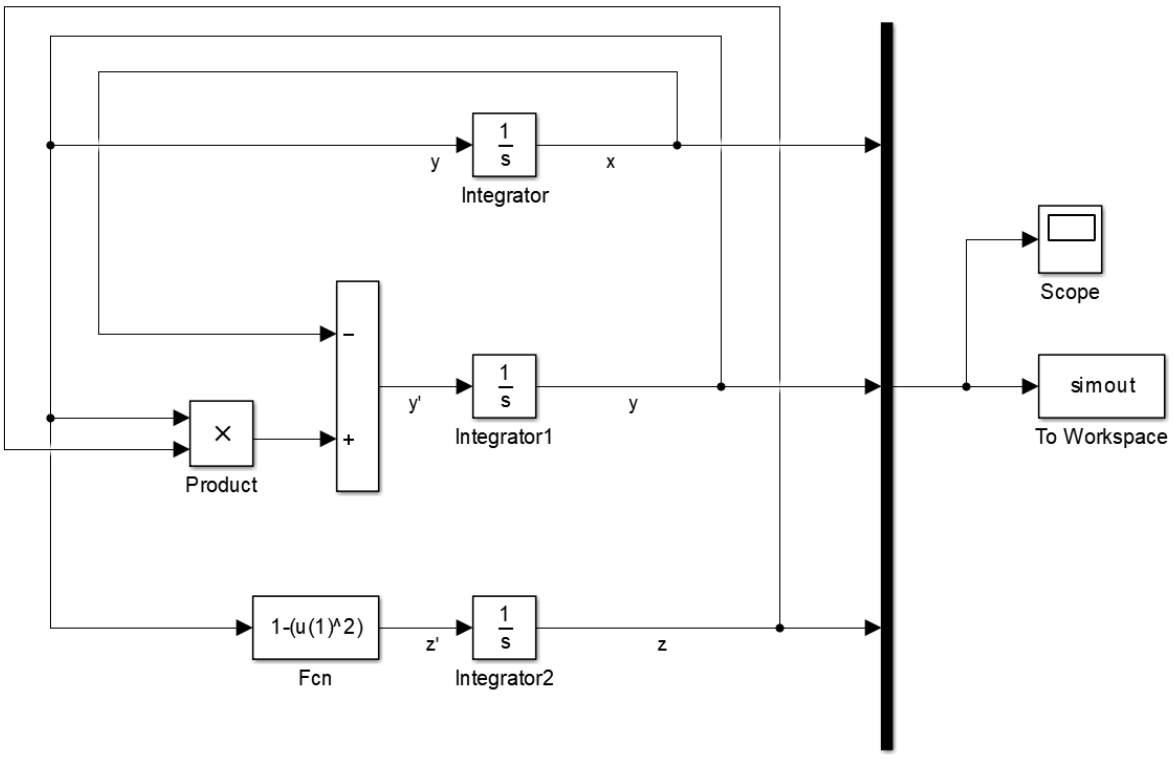

Figure 46: Nose-Hoover Simulink model

16.4 Result

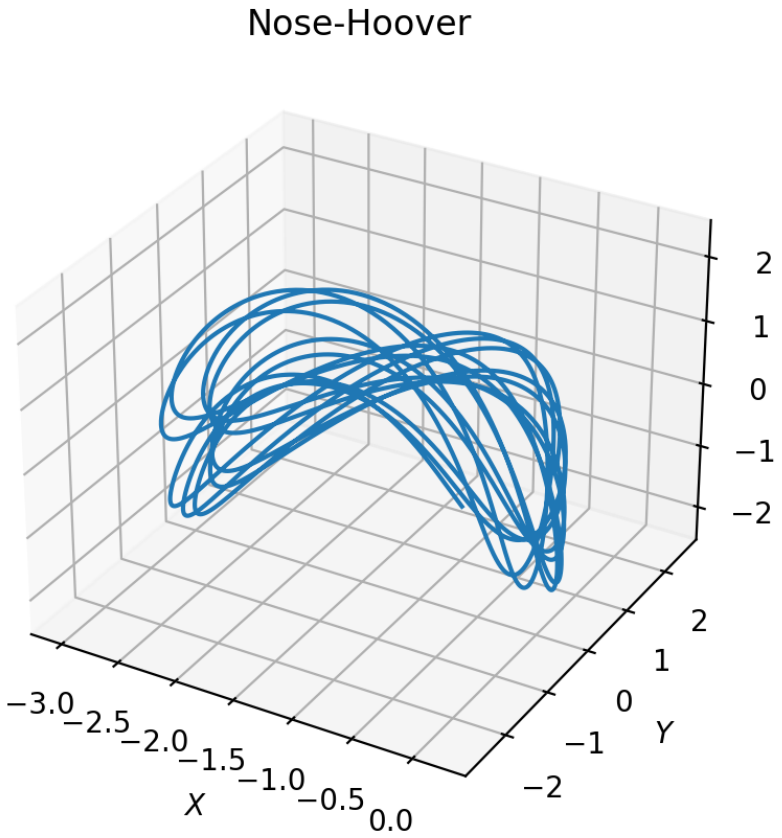

Figure 47: Nose-Hoover system simulation results

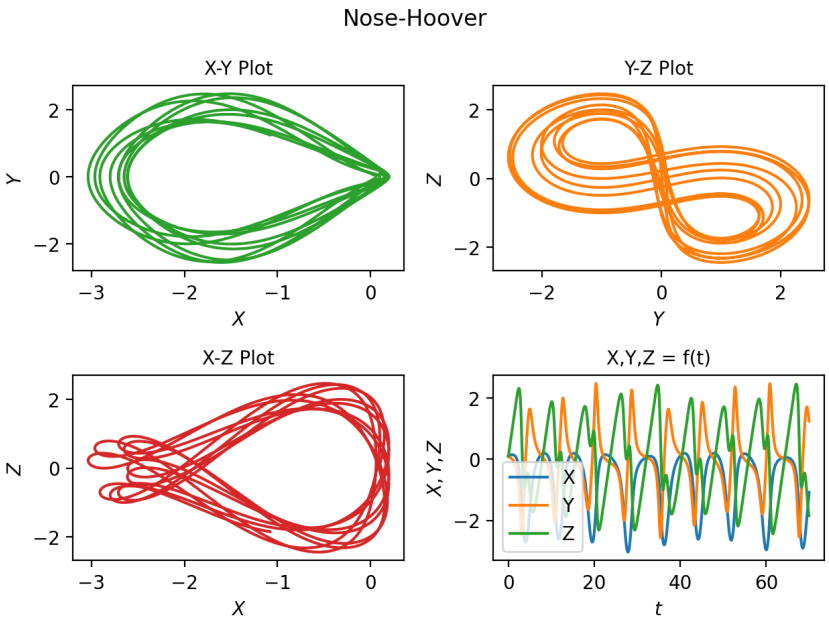

Figure 48: Nose-Hoover system simulation results

17 Qi (Hyperchaotic)

17.1 Equation

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

17.2 Python model

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

17.3 Simulink model

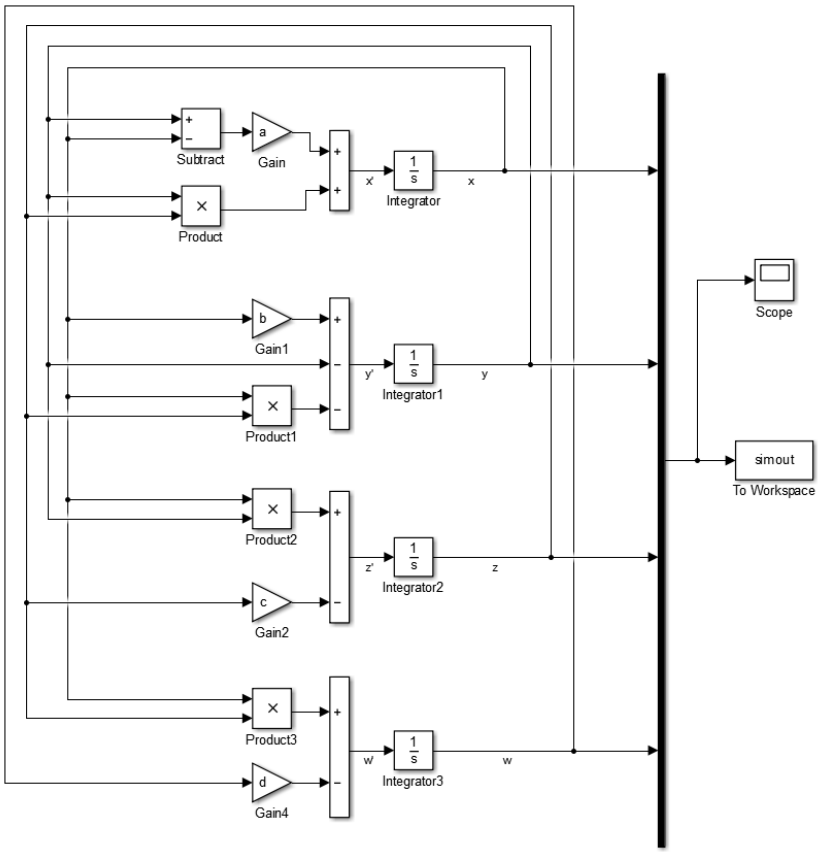

Figure 49: Hyperchaotic Qi Simulink model

17.4 Result

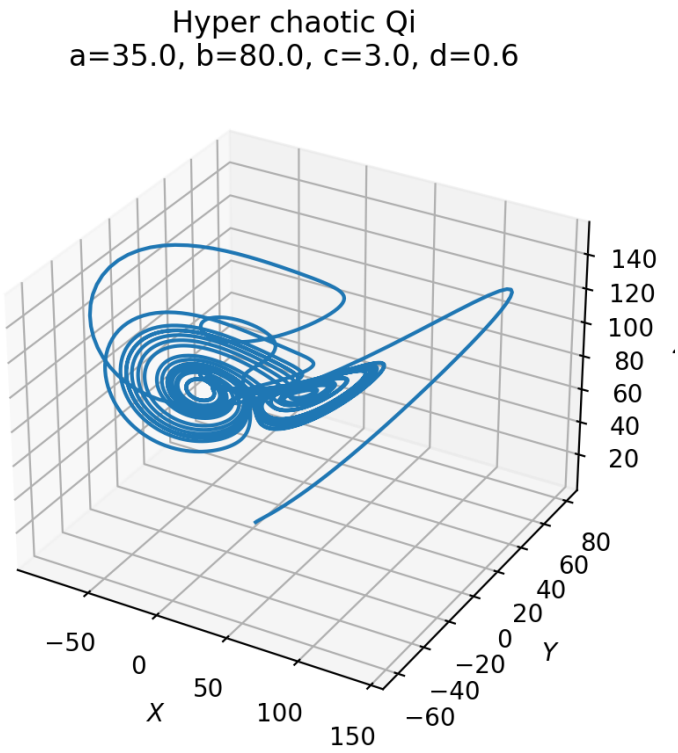

Figure 50: Hyperchaotic Qi system simulation results

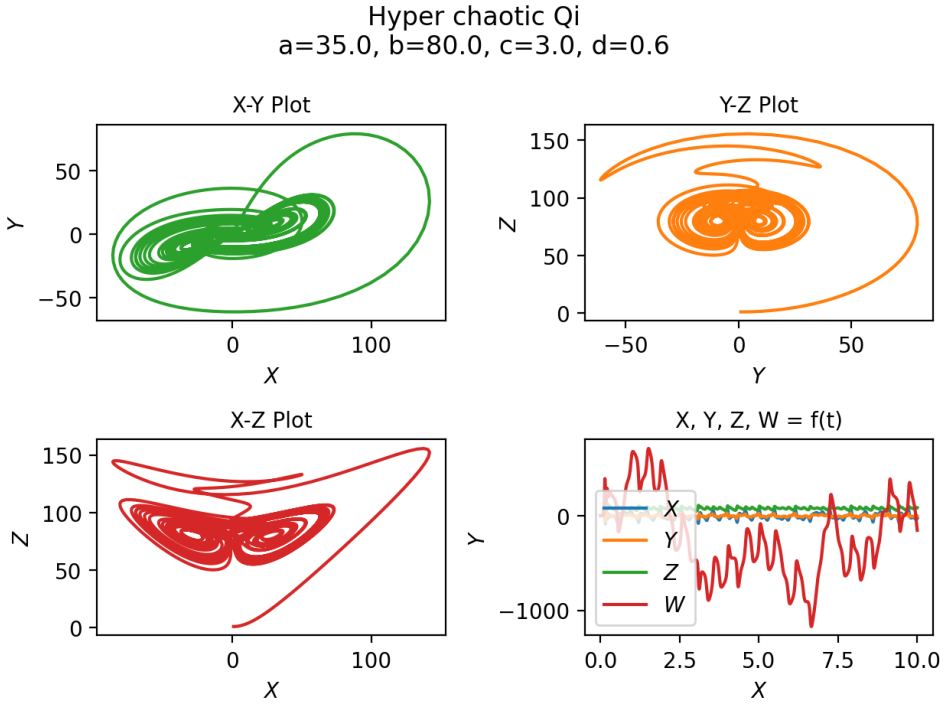

Figure 51: Hyperchaotic Qi system simulation results

18 Rabinovich-Fabrikant

18.1 Equation

$$\begin{cases} \dot{x} = y \cdot (z - 1 + x^2) + \gamma \cdot x \\ \dot{y} = x \cdot (3 \cdot z + 1 - x^2) + \gamma \cdot y \\ \dot{z} = -2 \cdot z \, (\alpha + x \cdot y) \end{cases}$$

18.2 Python model

```
def rabinovich_fabrikant(state, __time__):
        x, y, z = state
        return y * (z - 1 + x ** 2) + InputParams.gamma * x, \
                x * (3 * z + 1 - x ** 2) + InputParams.gamma * y,
                    \
                -2 * z * (InputParams.alpha + x * y)
```

18.3 Simulink model

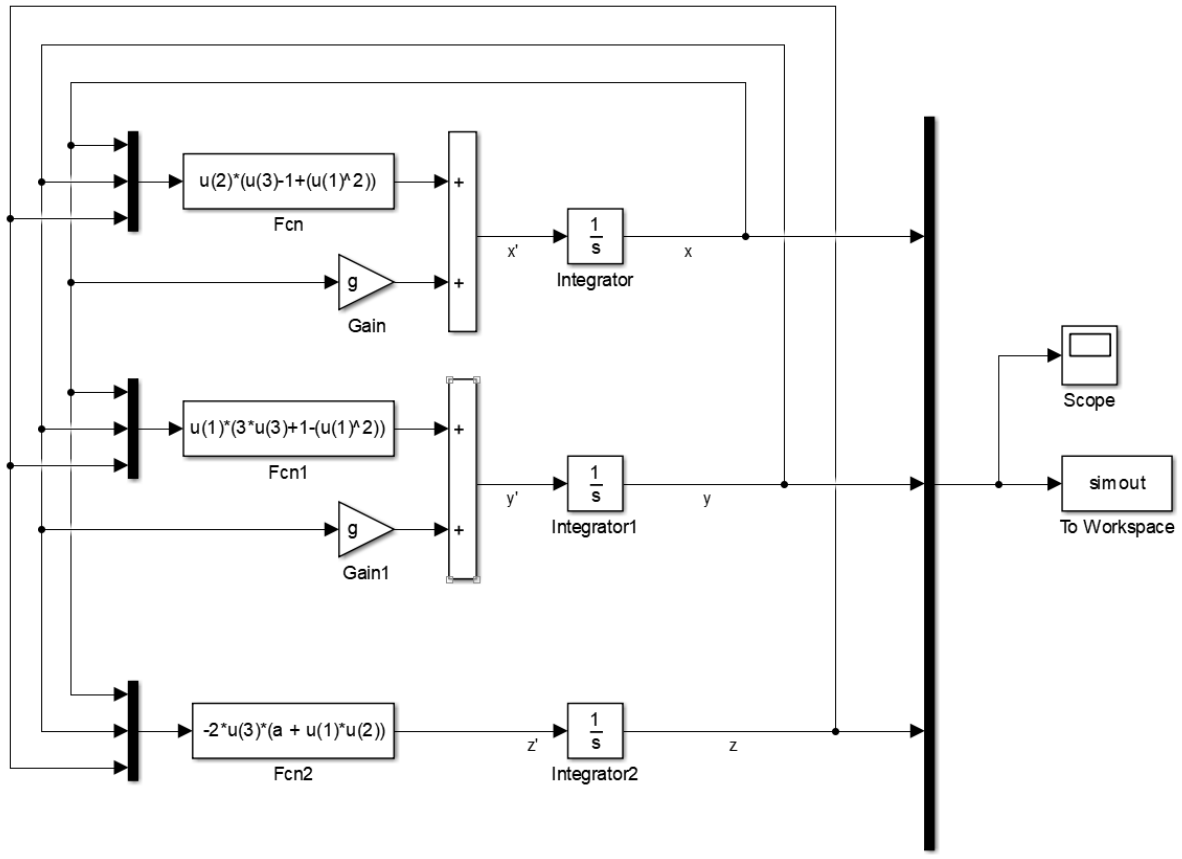

Figure 52: Rabinovich-Fabrikant Simulink model

18.4 Result

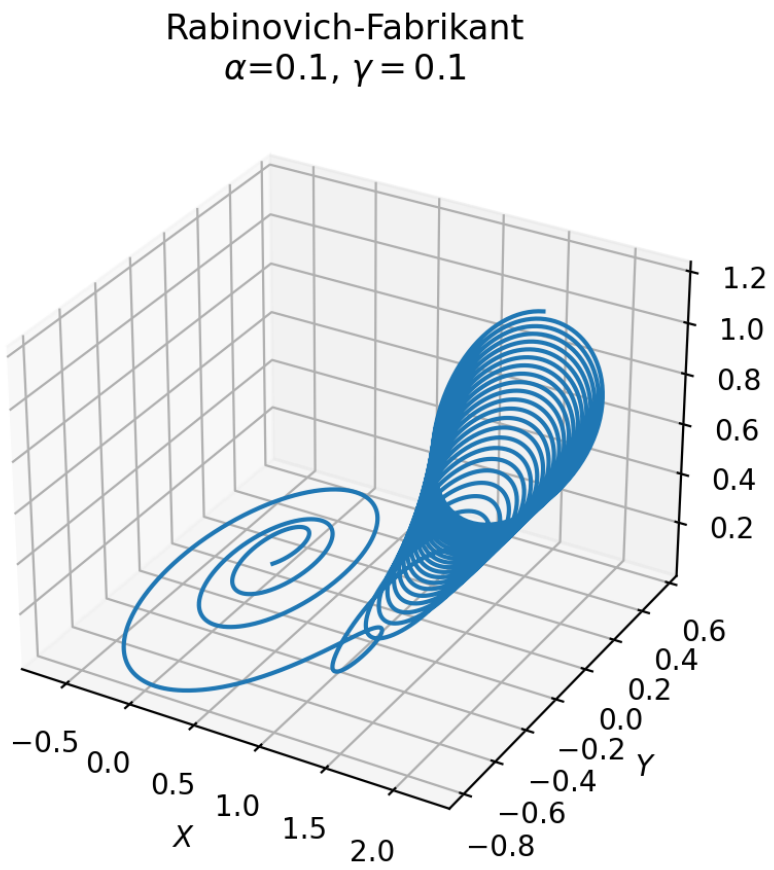

Figure 53: Rabinovich-Fabrikant system simulation results

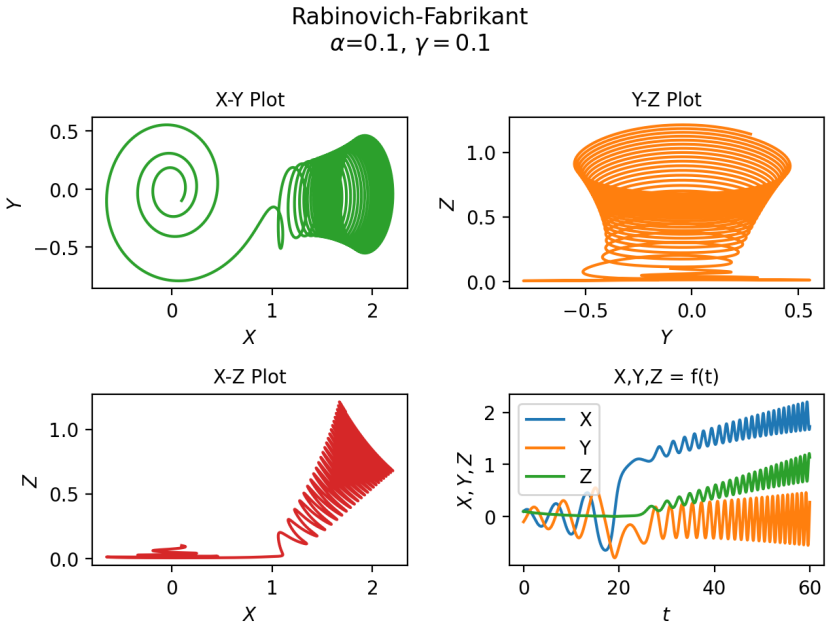

Figure 54: Rabinovich-Fabrikant system simulation results

19 Rössler

19.1 Equation

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

19.2 Python model

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

19.3 Simulink model

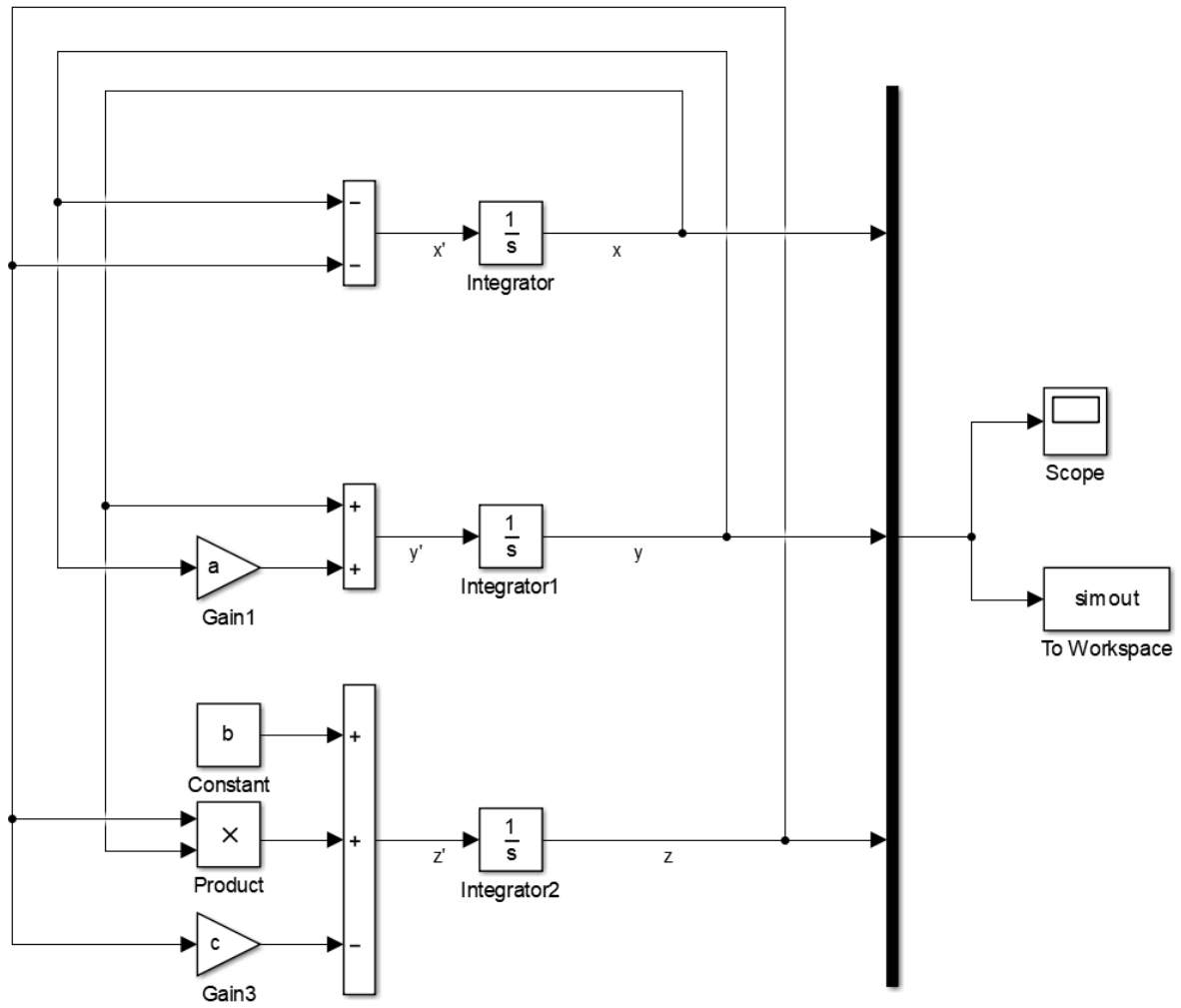

Figure 55: Rössler Simulink model

19.4 Result

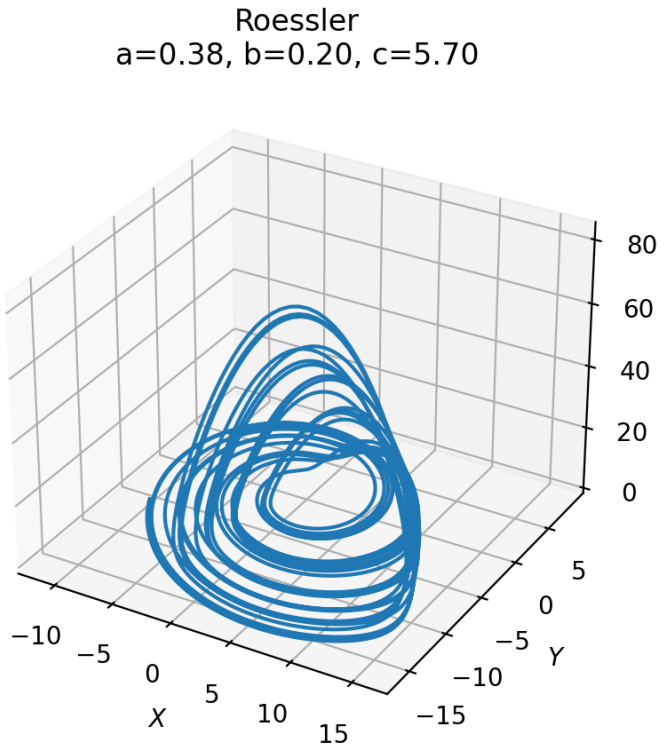

Figure 56: Rössler system simulation results

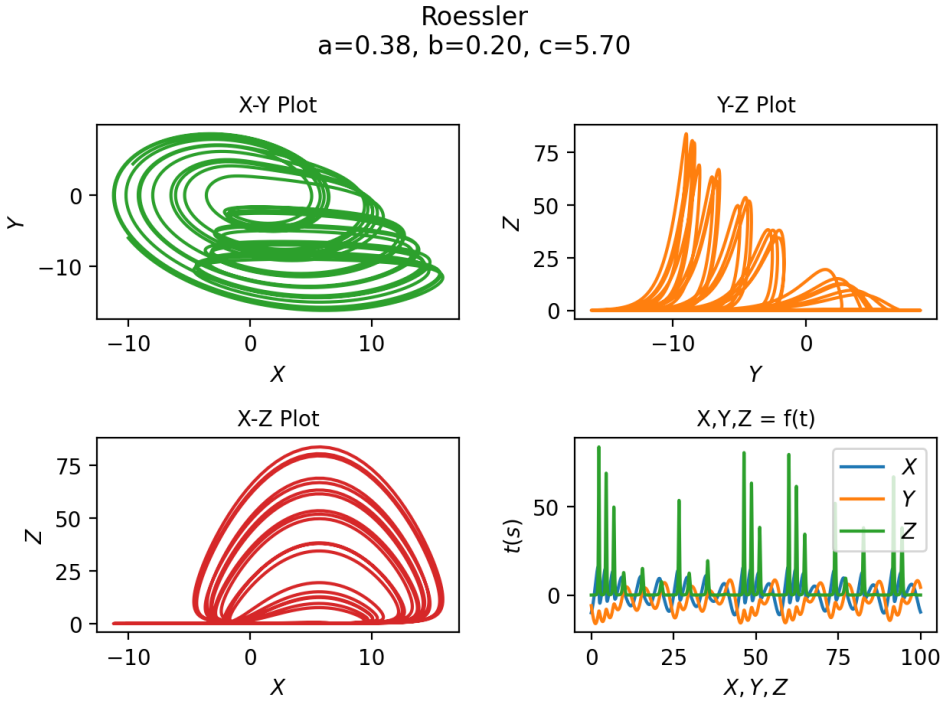

Figure 57: Rössler system simulation results

20 Rössler (Hyperchaotic)

20.1 Equation

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

20.2 Python model

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

20.3 Simulink model

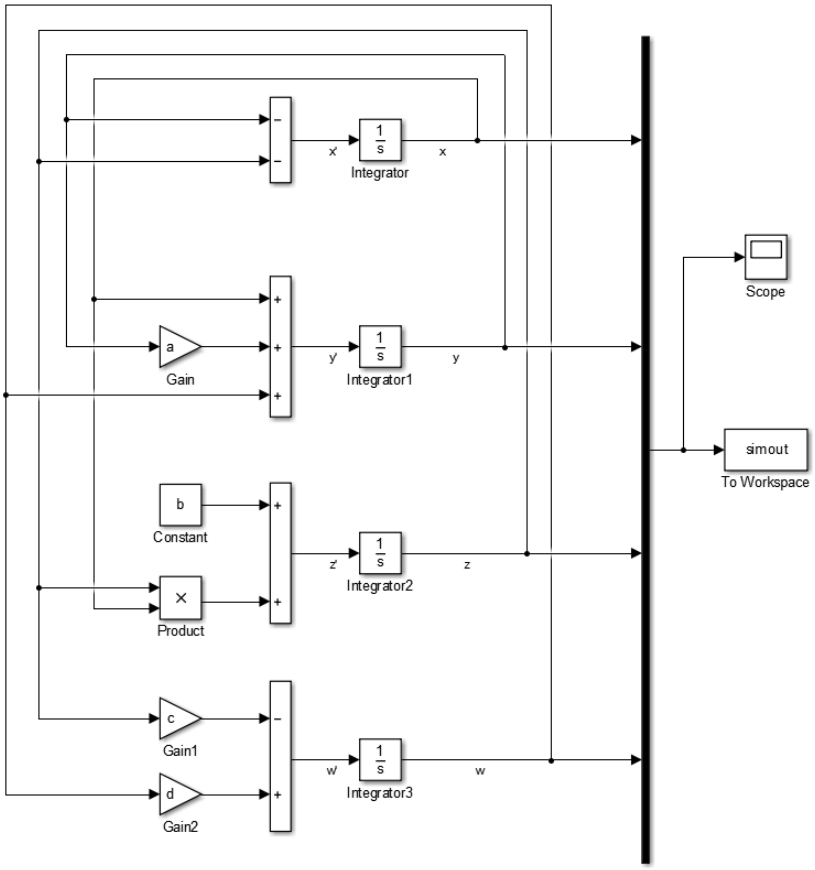

Figure 58: Hyperchaotic Rössler Simulink model

20.4 Result

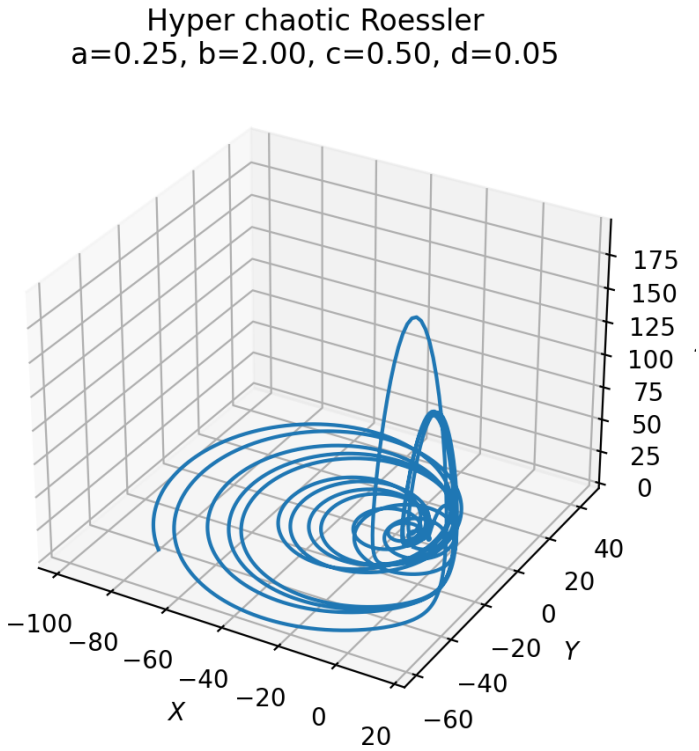

Figure 59: Hyperchaotic Rössler system simulation results

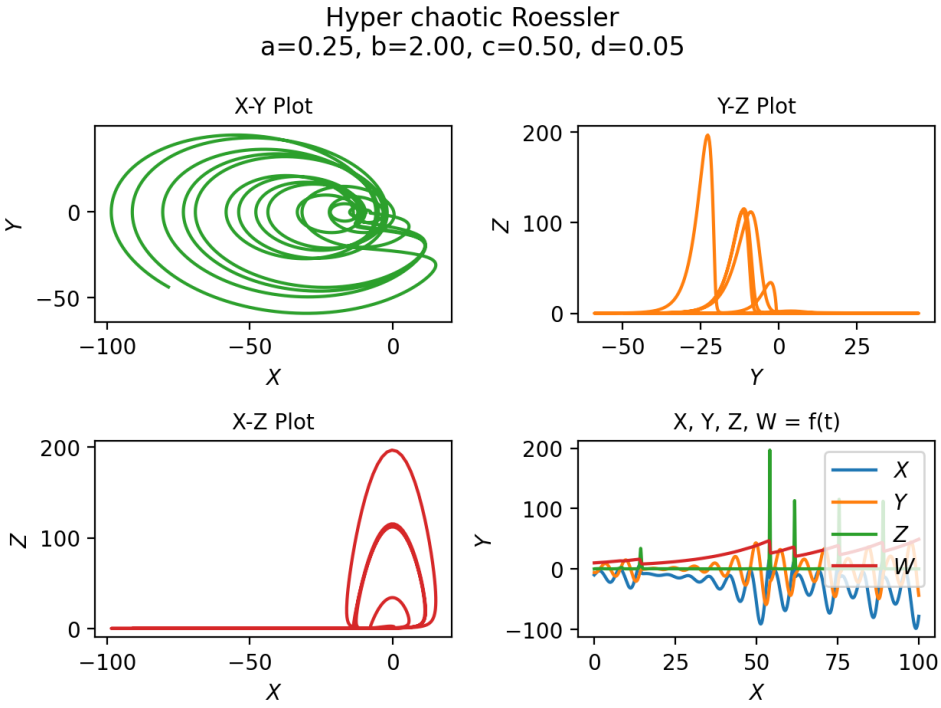

Figure 60: Hyperchaotic Rössler system simulation results

21 Sprott B

21.1 Equation

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

21.2 Python model

```
def sprott_b(state, __time__):
        x, y, z = state
        return a * y * z, \
                x - b * y, \
                c - x * y
```

21.3 Simulink model

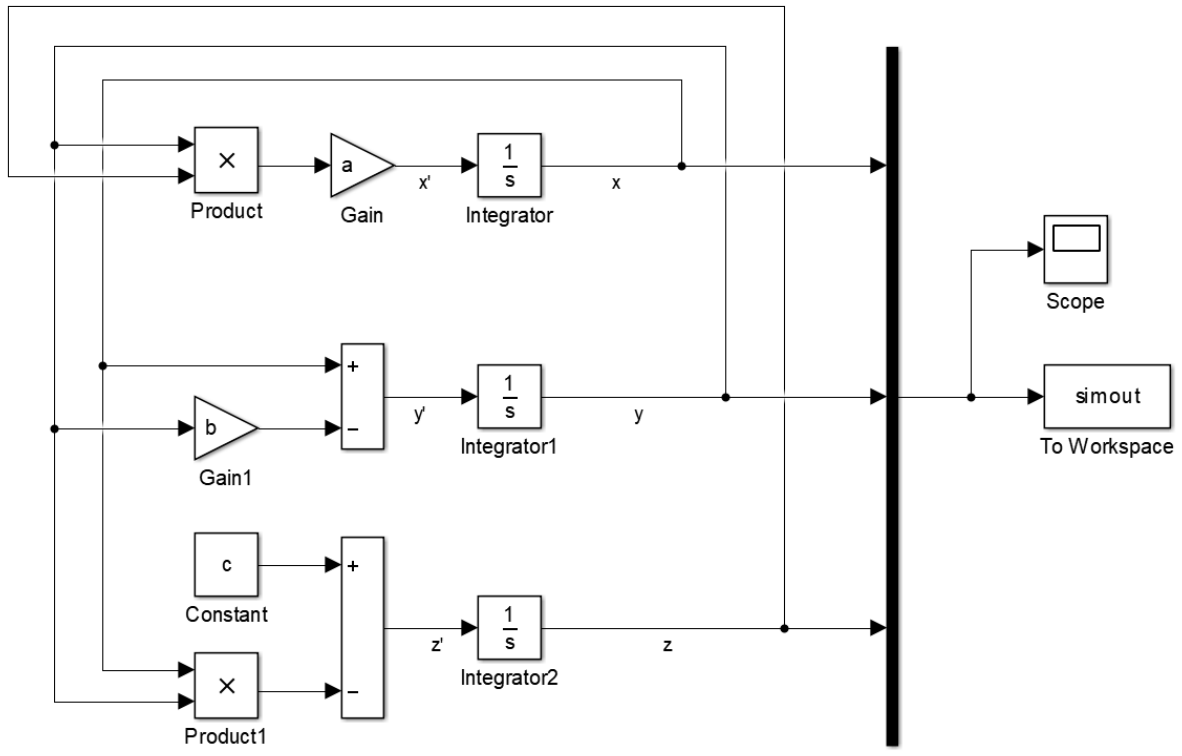

Figure 61: Sprott B Simulink model

21.4 Result

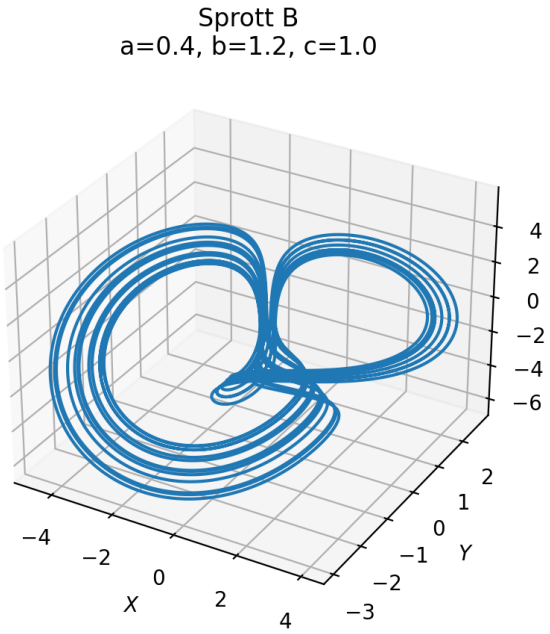

Figure 62: Sprott B system simulation results

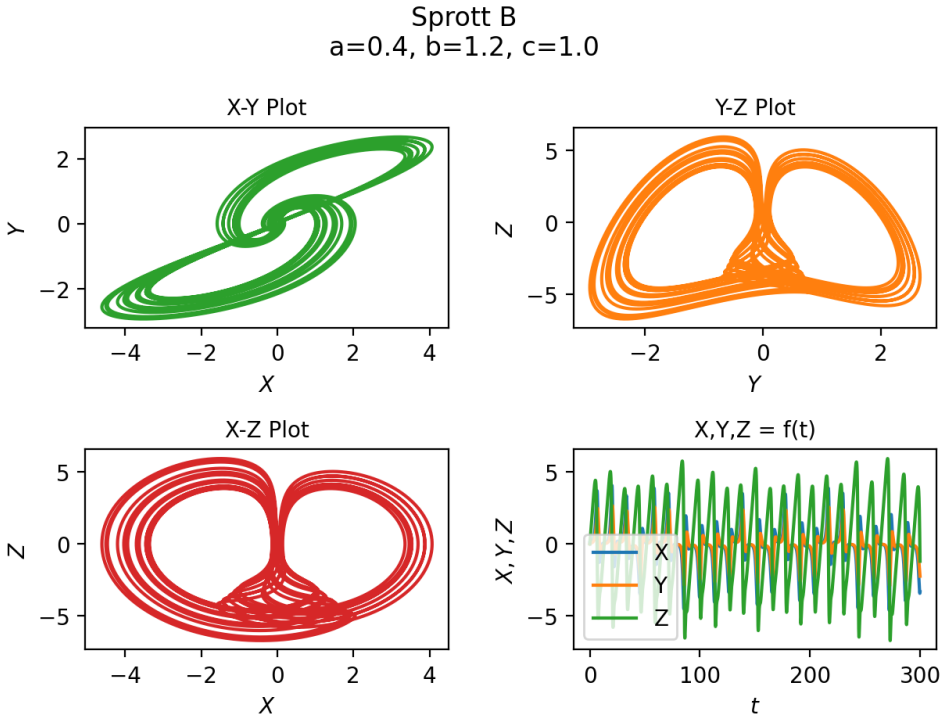

Figure 63: Sprott B system simulation results

22 Thomas' Cyclically Symmetric Attractor

22.1 Equation

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

22.2 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
```

22.3 Simulink model

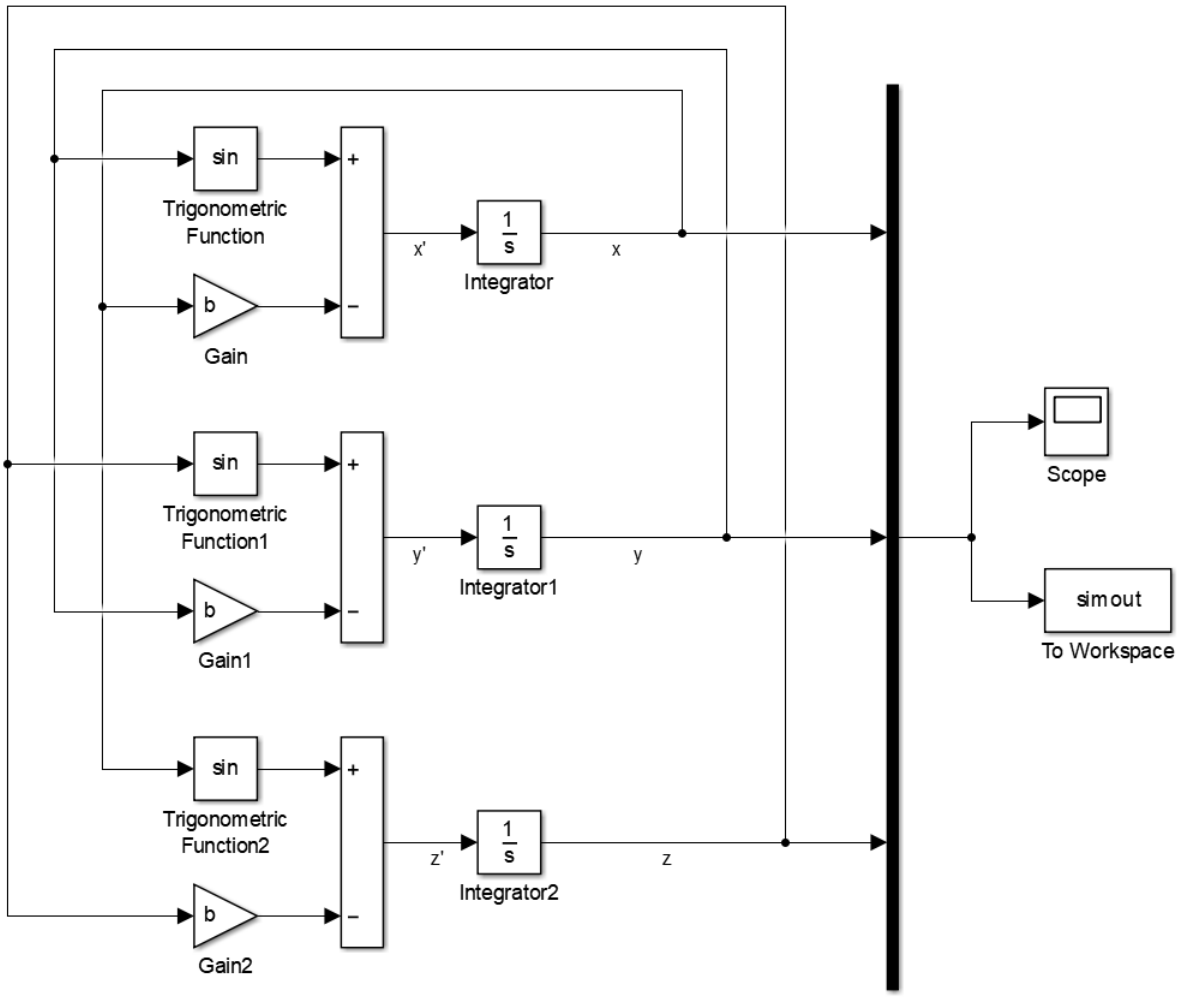

Figure 64: Thomas' attractor Simulink model

22.4 Result

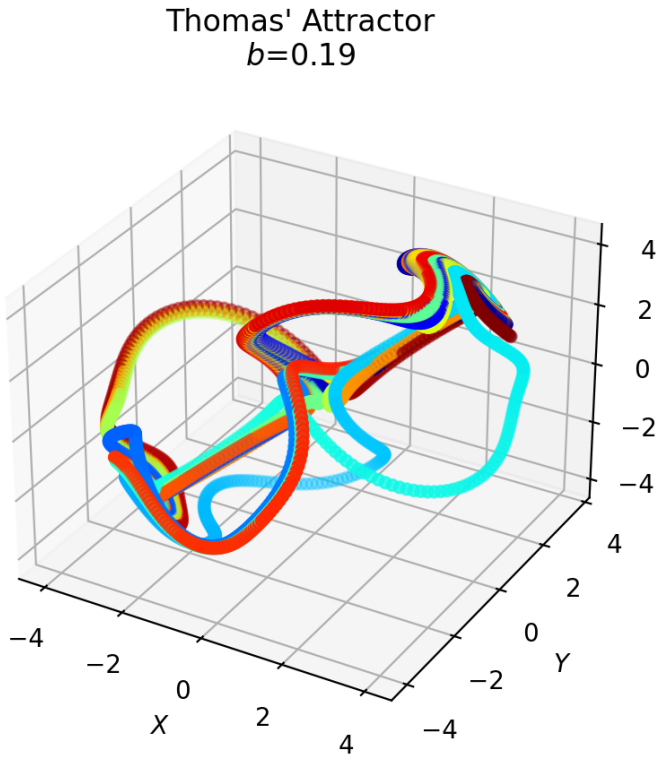

Figure 65: Thomas' cyclically symmetric attractor simulation results

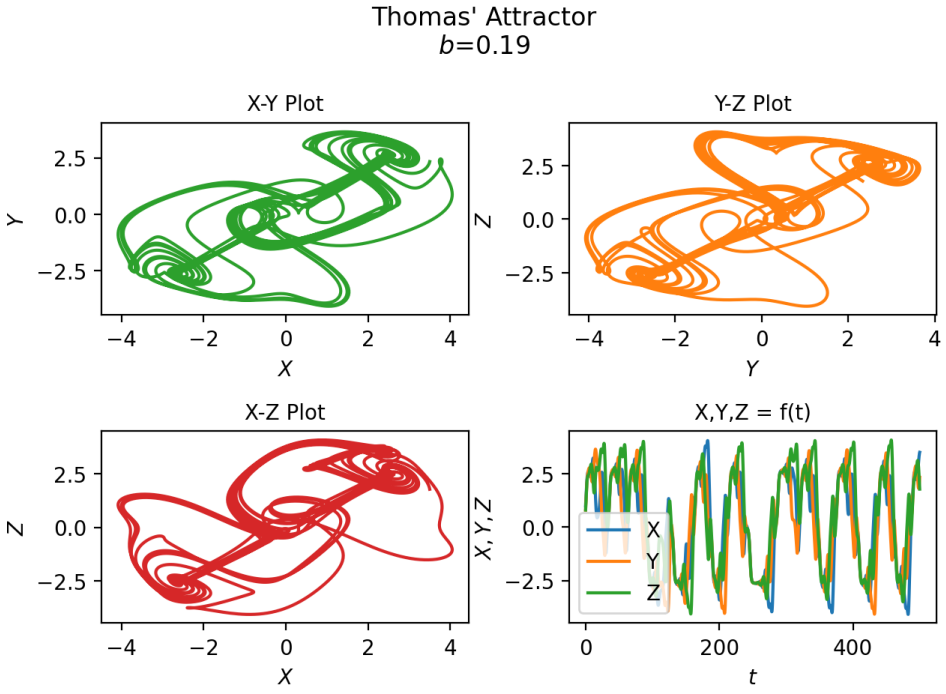

Figure 66: Thomas' cyclically symmetric attractor simulation results

23 Van der Pol

23.1 Equation

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

23.2 Python model

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

23.3 Simulink model

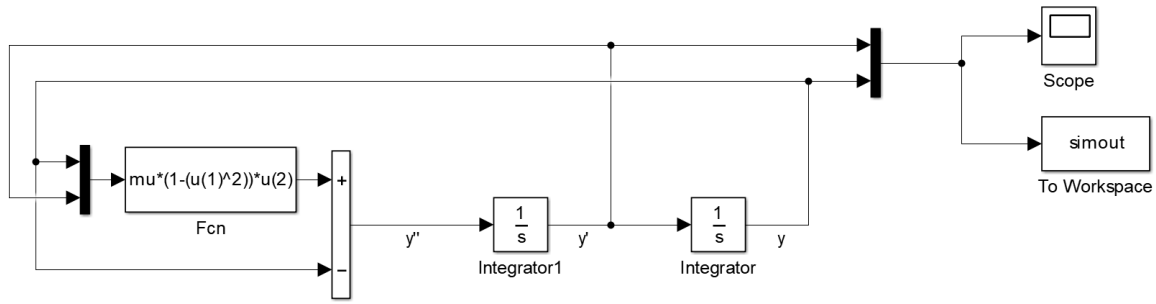

Figure 67: Van der Pol system simulation results

23.4 Result

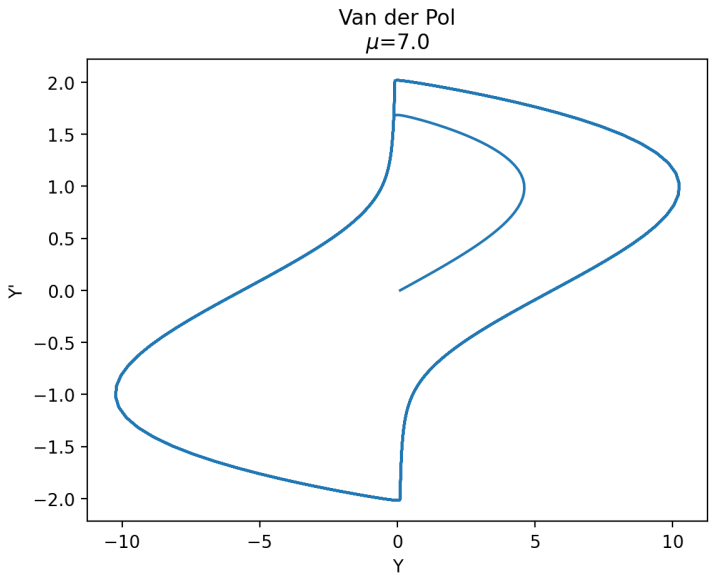

Figure 68: Van der Pol Simulink model

Conclusion

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