

Modeling Chaotic Systems

ARO (@art_of_electronics_)

June 18, 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

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

1.3 Simulink model

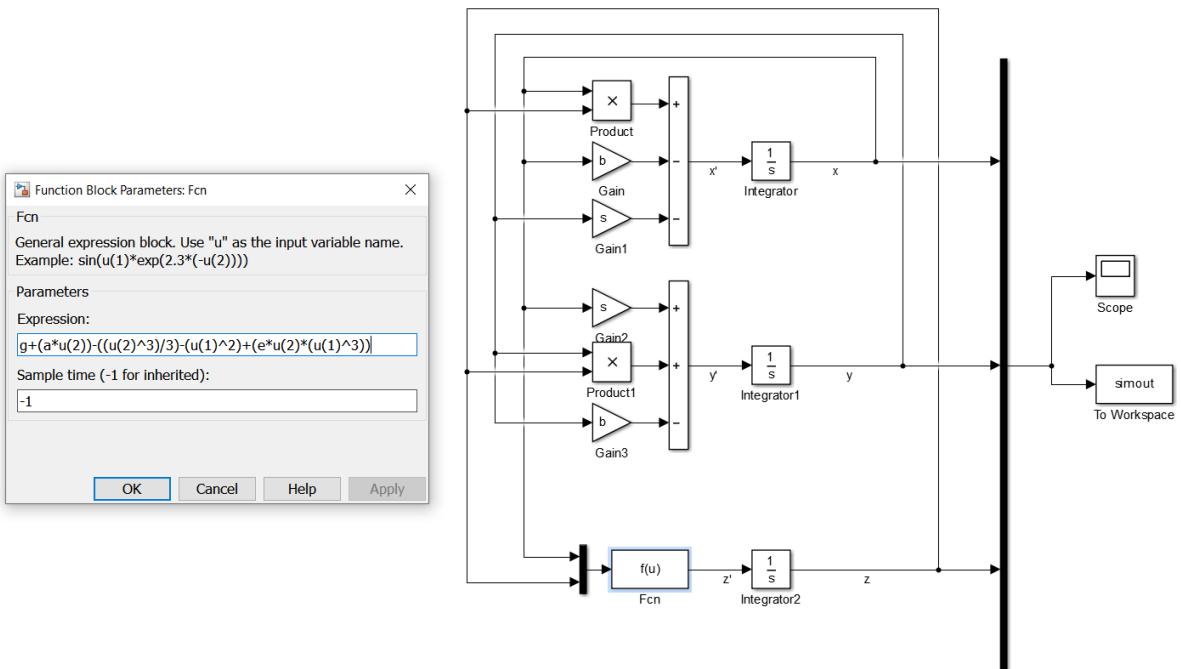


Figure 1: Aizawa Simulink model

1.4 Result

Aizawa
 $\alpha=0.95, \beta=0.70, \gamma=0.65, \sigma=3.50, \varepsilon=0.15$

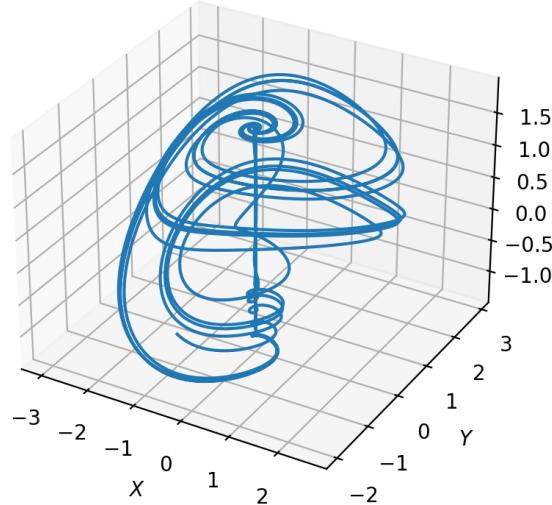


Figure 2: Aizawa system simulation results

Aizawa
 $\alpha=0.95, \beta=0.70, \gamma=0.65, \sigma=3.50, \varepsilon=0.15$

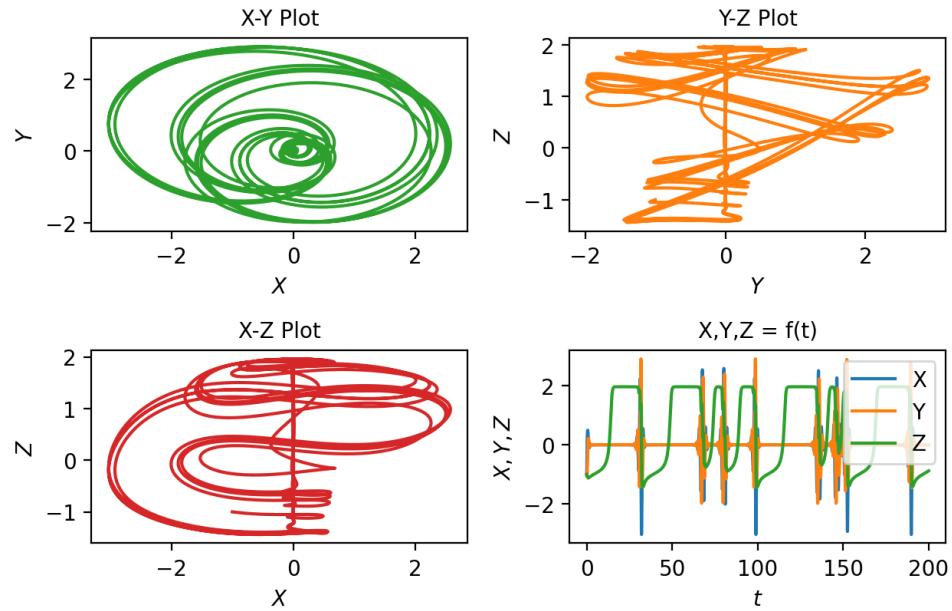


Figure 3: Aizawa system simulation results

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

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

2.3 Simulink model

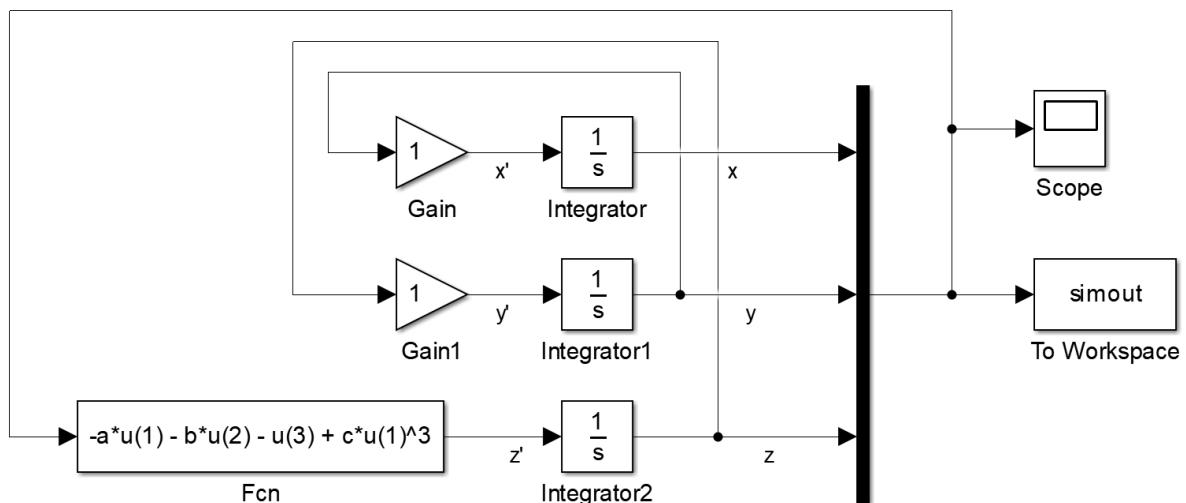


Figure 4: Arneodo Simulink model

2.4 Result

Arneodo
 $a=-5.5, b=3.5, c=-1.0$

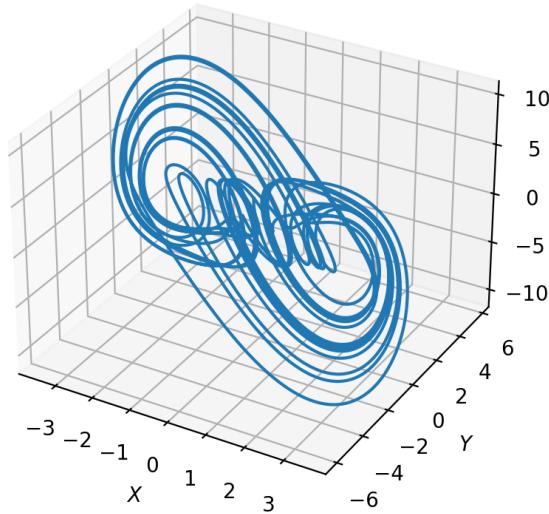


Figure 5: Arneodo system simulation results

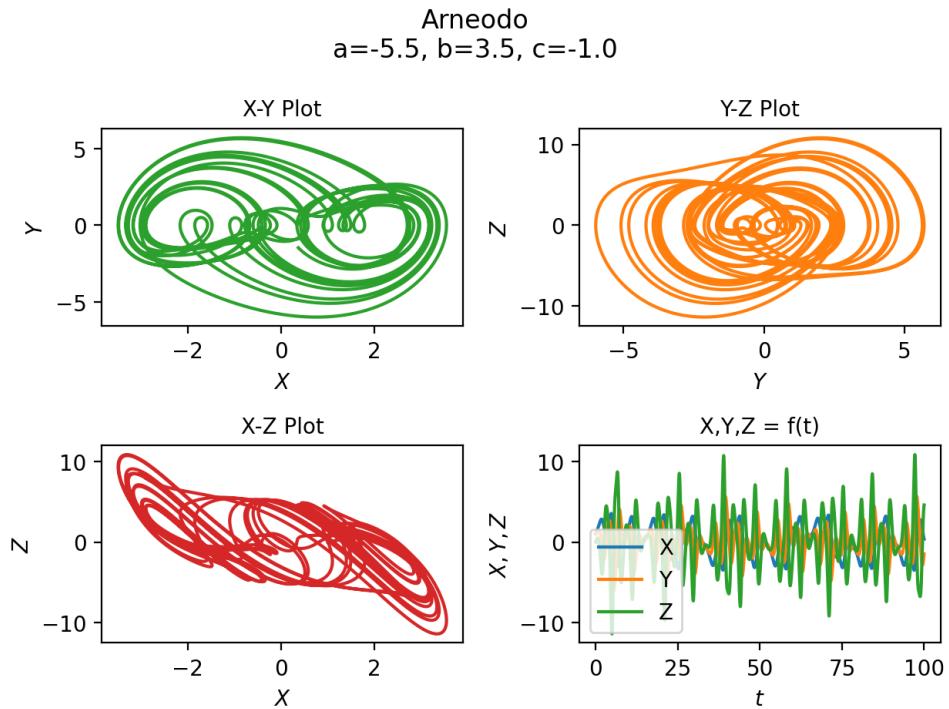


Figure 6: Arneodo system simulation results

3 Chen

3.1 Equation

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

3.2 Python model

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

3.3 Simulink model

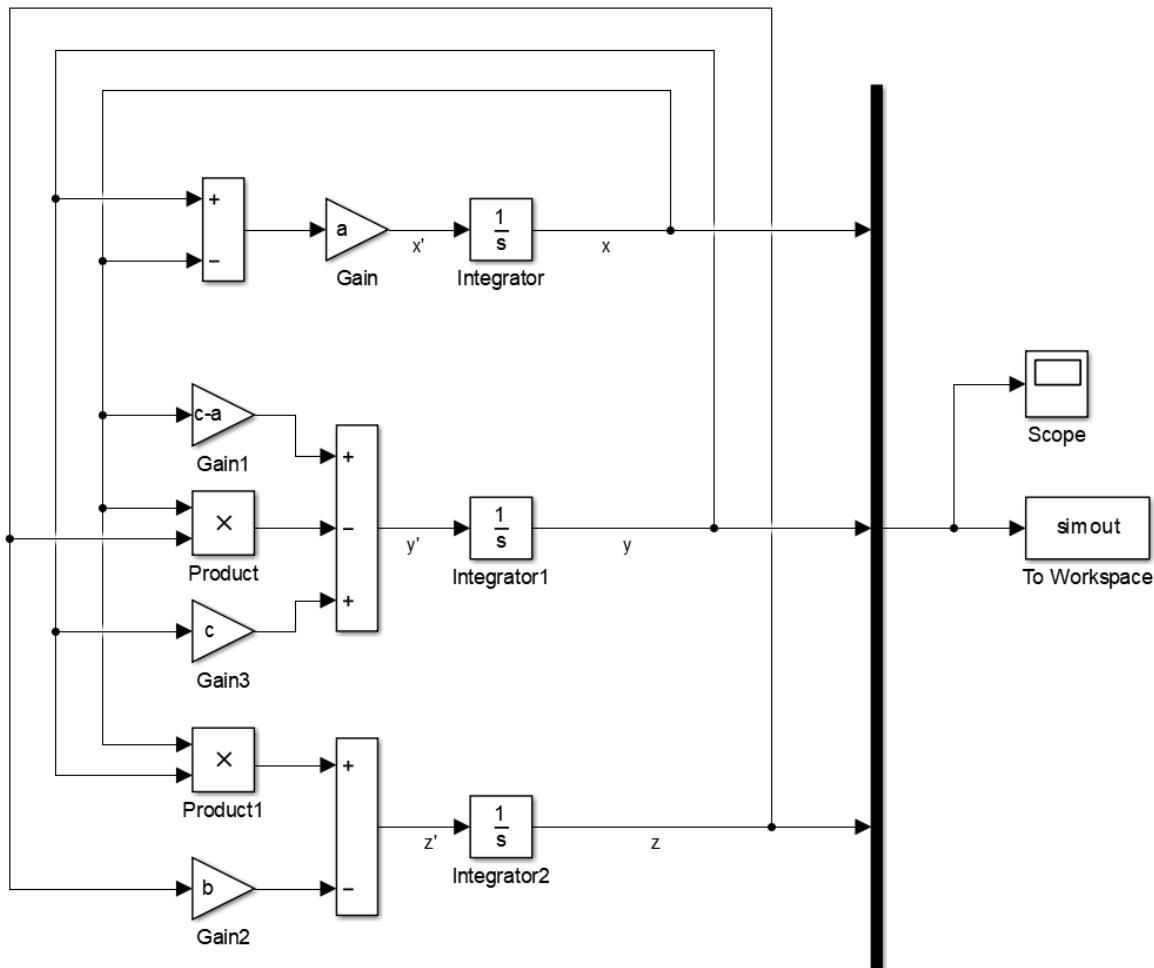


Figure 7: Chen Simulink model

3.4 Result

Chen
 $a=33.0, b=3.0, c=25.0$

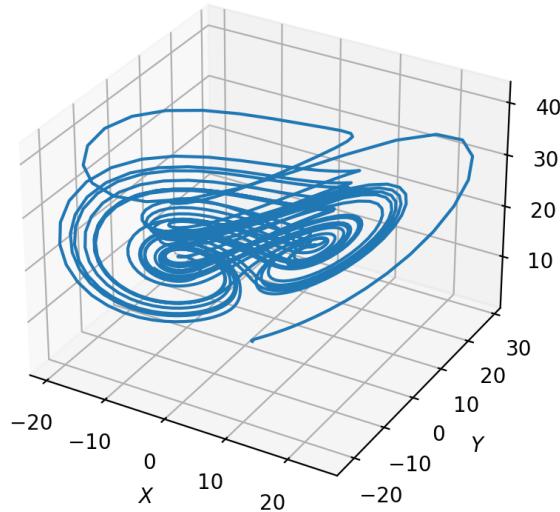


Figure 8: Chen system simulation results

Chen
 $a=33.0, b=3.0, c=25.0$

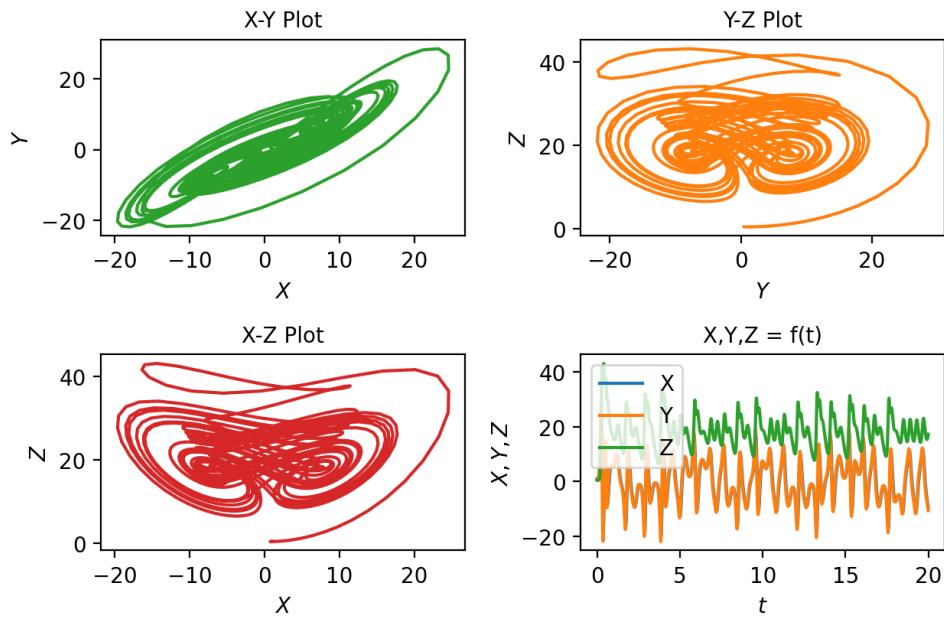


Figure 9: Chen system simulation results

4 Chua

4.1 Equation

$$\begin{cases} \dot{x} = a \cdot (y - x - \text{diode}) \\ \dot{y} = x - y + z \\ \dot{z} = -y \cdot b \end{cases}$$
$$\text{diode} = (m_1 \cdot x) + \left((m_0 - m_1) \cdot \frac{1}{2} (|x + 1| - |x - 1|) \right)$$

4.2 Python model

```
1 def chua(state, __time__):
2     x, y, z = state
3     diode = (m[1] * x) + ((m[0] - m[1]) * (abs(x + 1) - abs(x
4         - 1)) / 2)
5     return a * (y - x - diode), x - y + z, -y * b
```

4.3 Simulink model

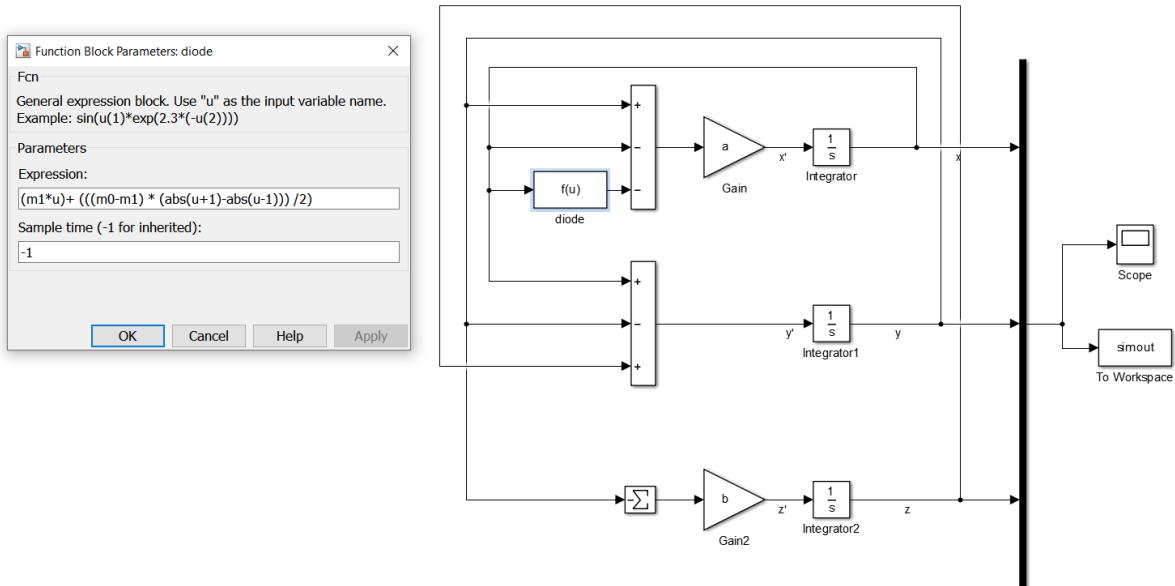


Figure 10: Chua Simulink model

4.4 Result

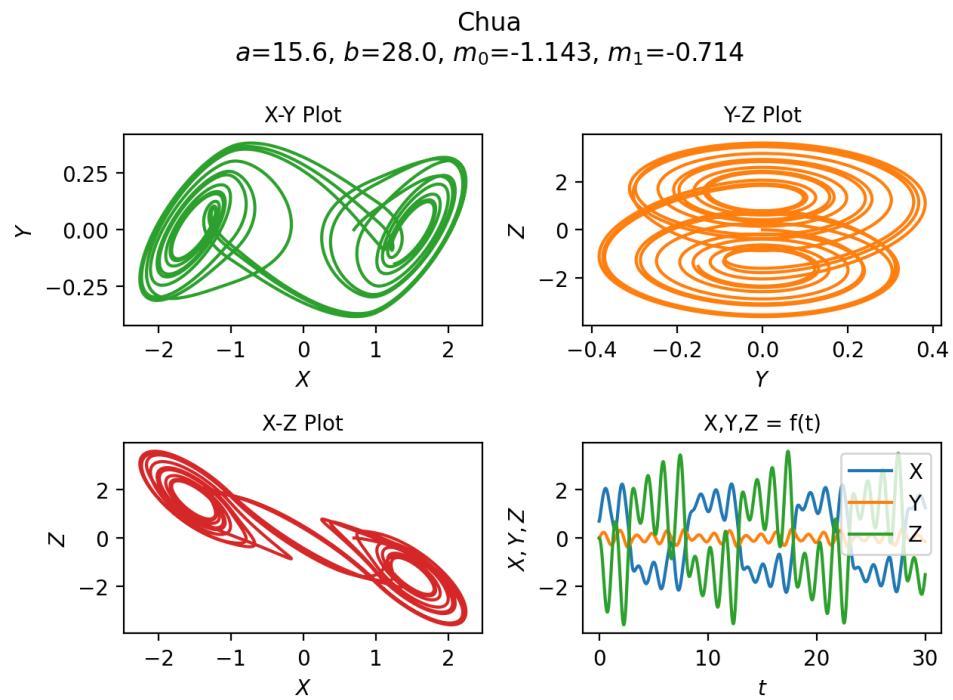


Figure 11: Chua system simulation results

4.5 Electronics Circuit

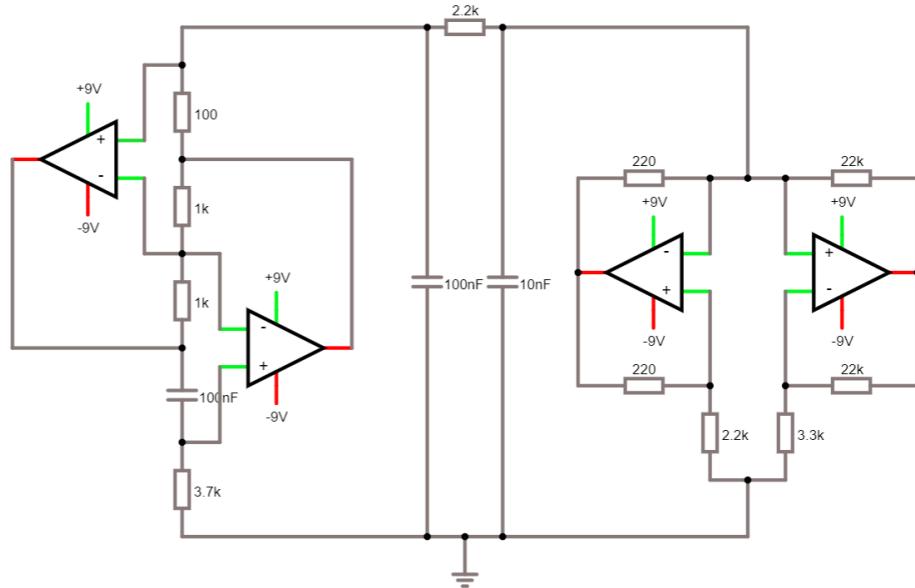


Figure 12: Chua system electronics circuit

5 Chua (Hyperchaotic)

5.1 Equation

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

5.2 Python model

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

5.3 Simulink model

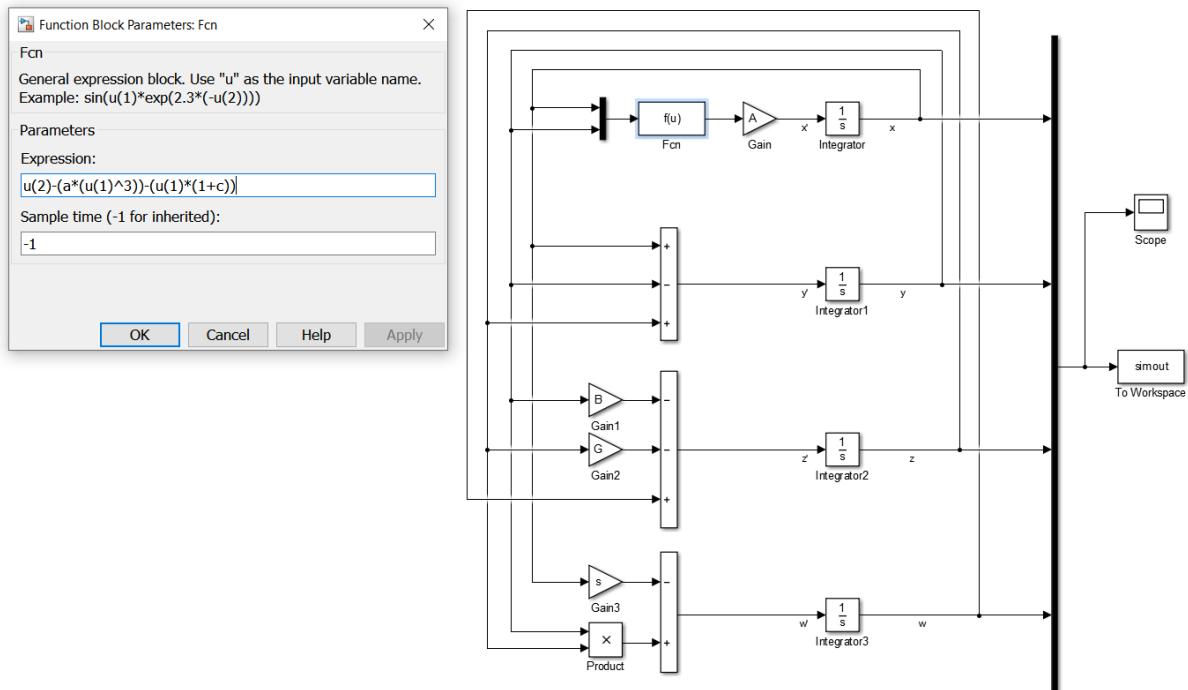


Figure 13: Hyperchaotic Chua Simulink model

5.4 Result

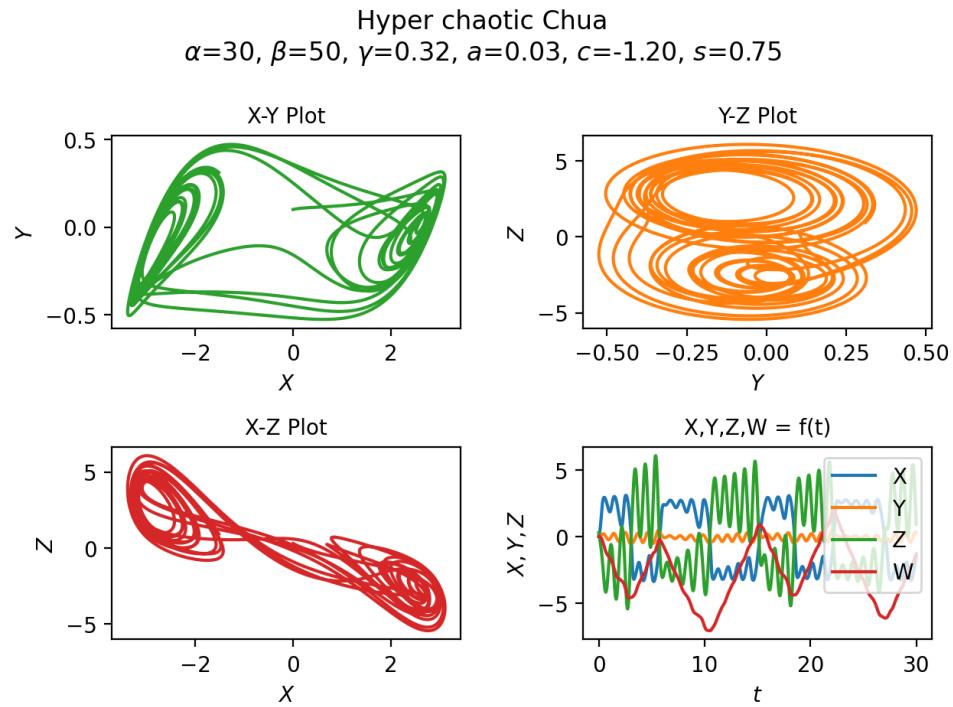


Figure 14: Hyperchaotic Chua system simulation results

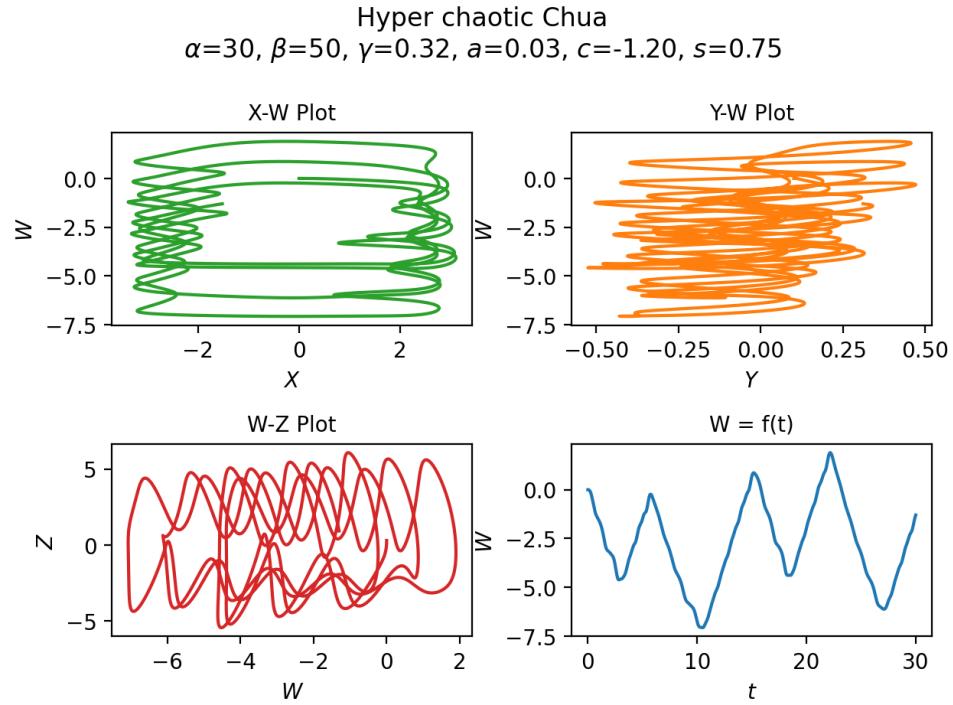


Figure 15: Hyperchaotic Chua system simulation results

6 Dadras

6.1 Equation

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

6.2 Python model

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

6.3 Simulink model

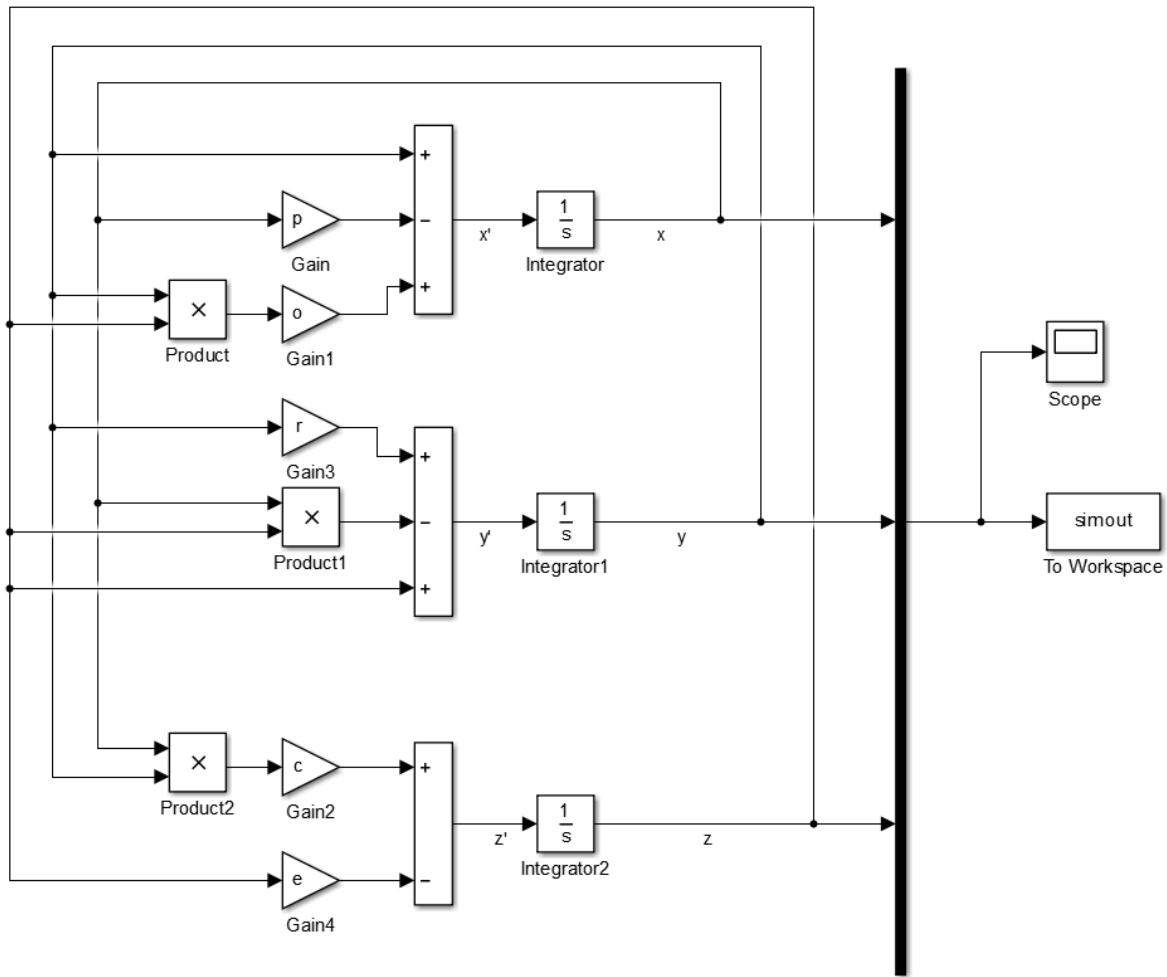


Figure 16: Dadras Simulink model

6.4 Result

Dadras
 $p=3.0, o=2.7, r=1.7, c=2.0, e=9.0$

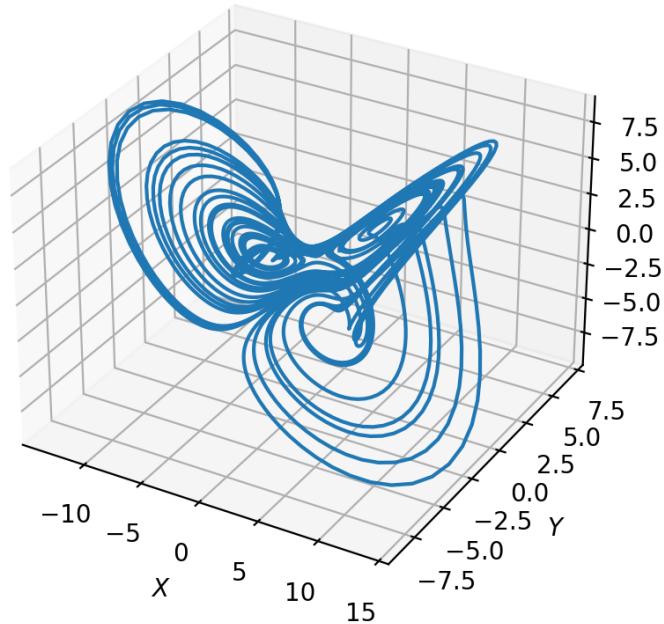


Figure 17: Dadras system simulation results

Dadras
 $p=3.0, o=2.7, r=1.7, c=2.0, e=9.0$

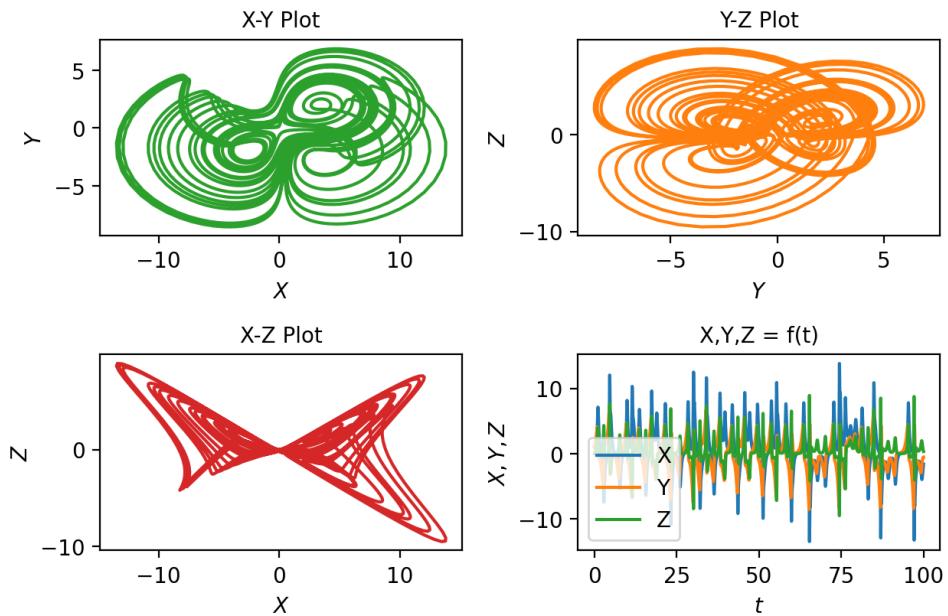


Figure 18: Dadras system simulation results

7 Duffing

7.1 Equation

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

7.2 Python model

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

7.3 Simulink model

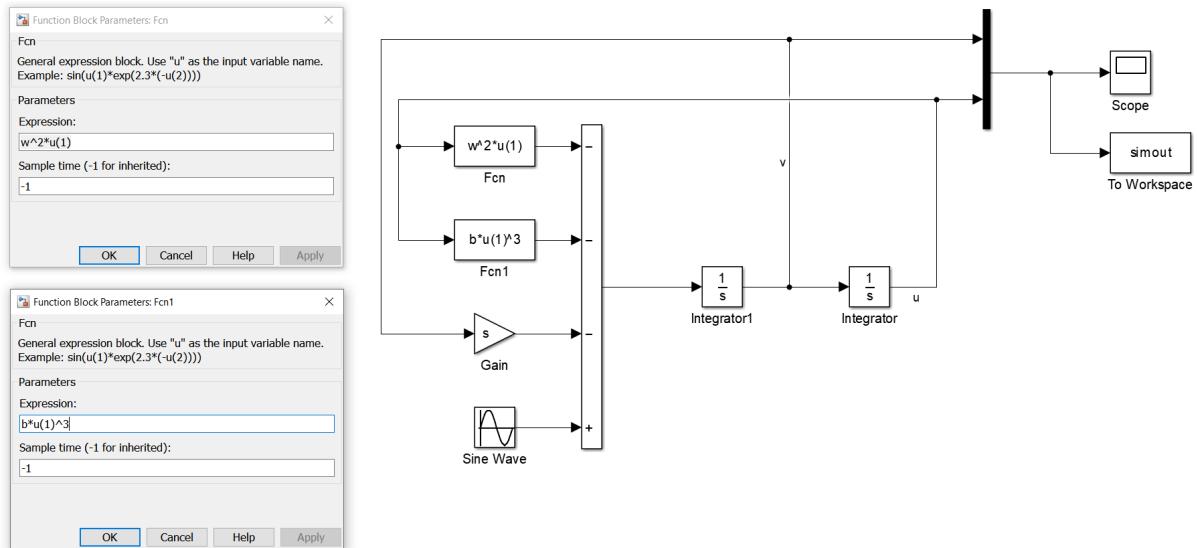


Figure 19: Duffing Simulink model

7.4 Result

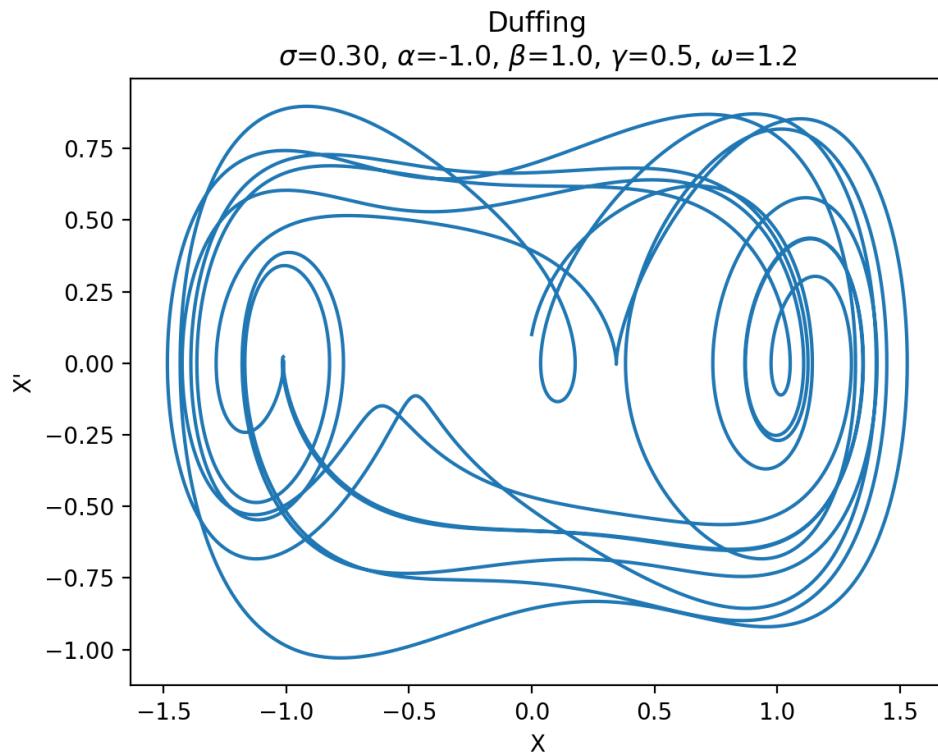


Figure 20: Duffing system simulation results

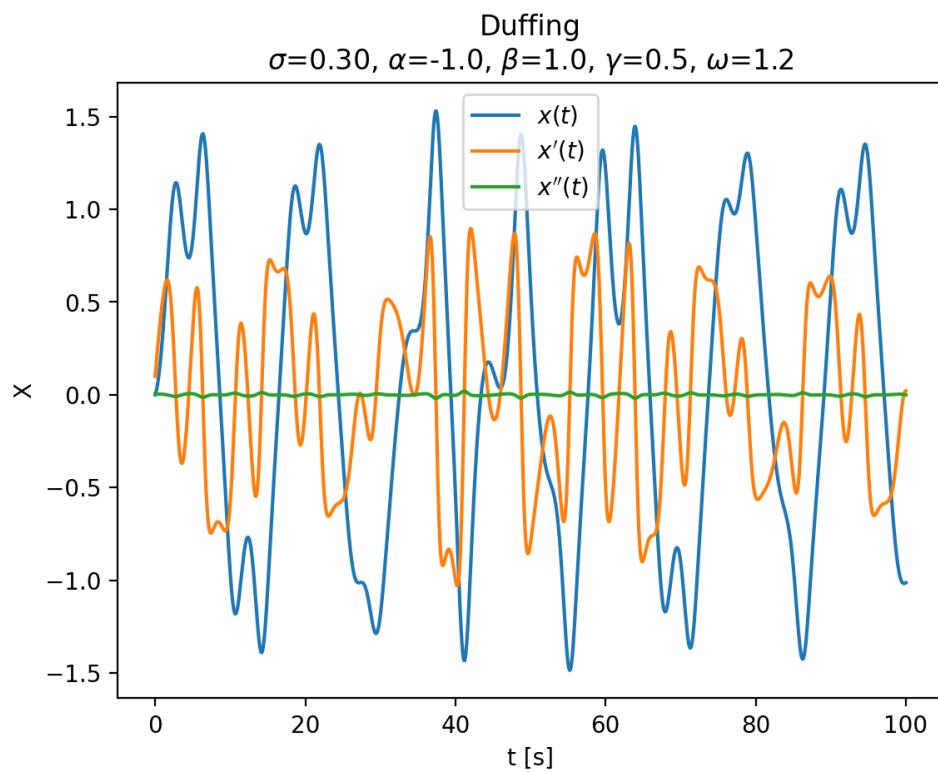


Figure 21: Duffing system simulation results

8 Halvorsen

8.1 Equation

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

8.2 Python model

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

8.3 Simulink model

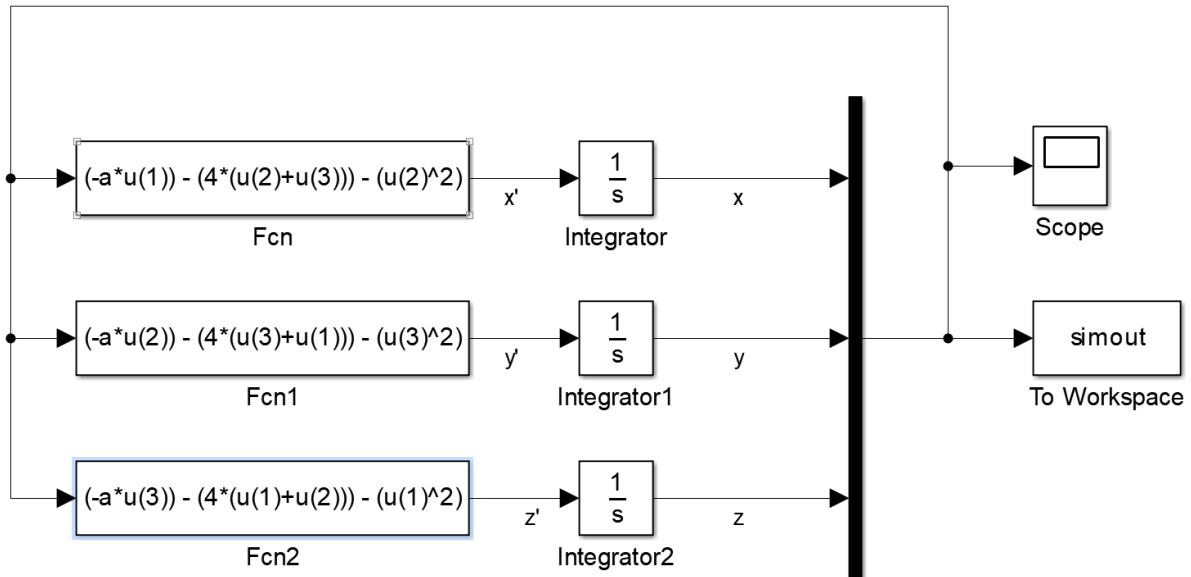


Figure 22: Halvorsen Simulink model

8.4 Result

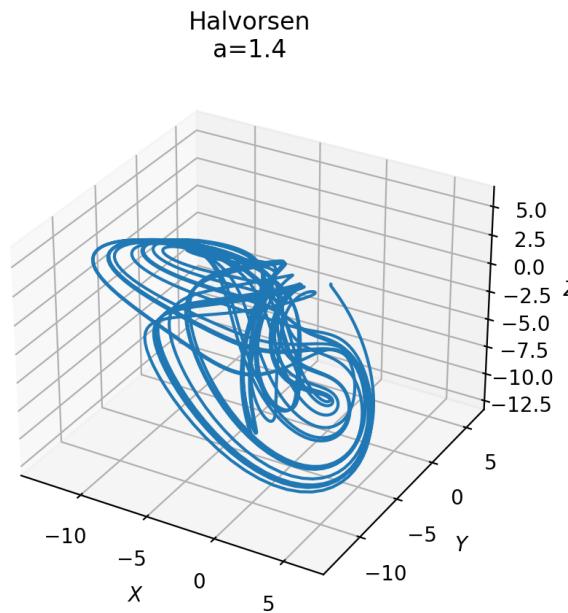


Figure 23: Halvorsen system simulation results

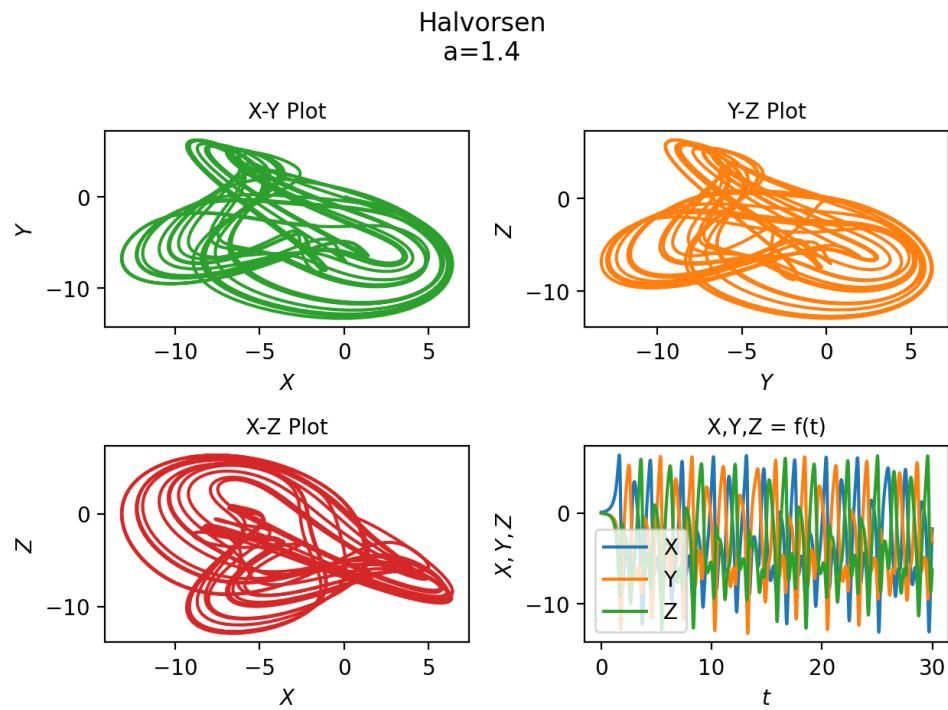


Figure 24: Halvorsen system simulation results

9 Henon-Heiles

9.1 Equation

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

9.2 Python model

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

9.3 Simulink model

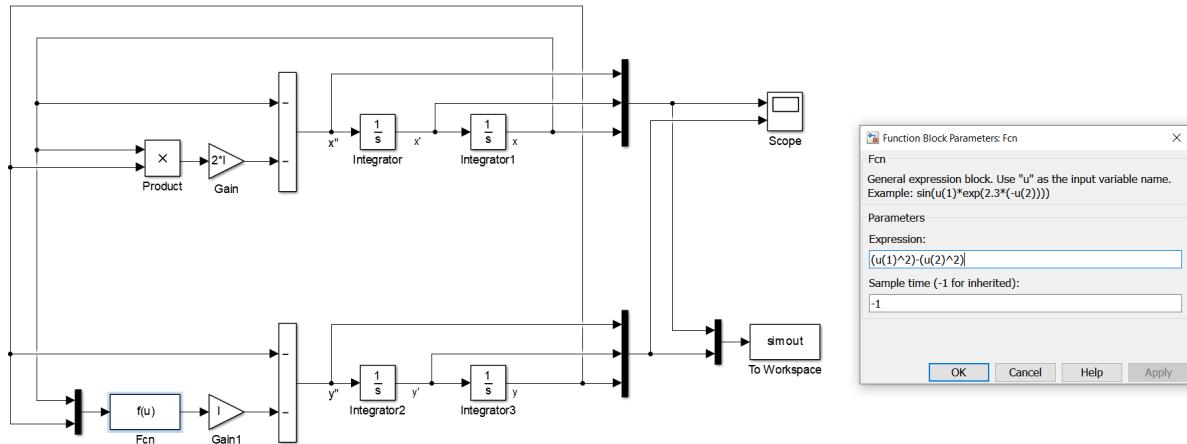


Figure 25: Henon-Heiles Simulink model

9.4 Result

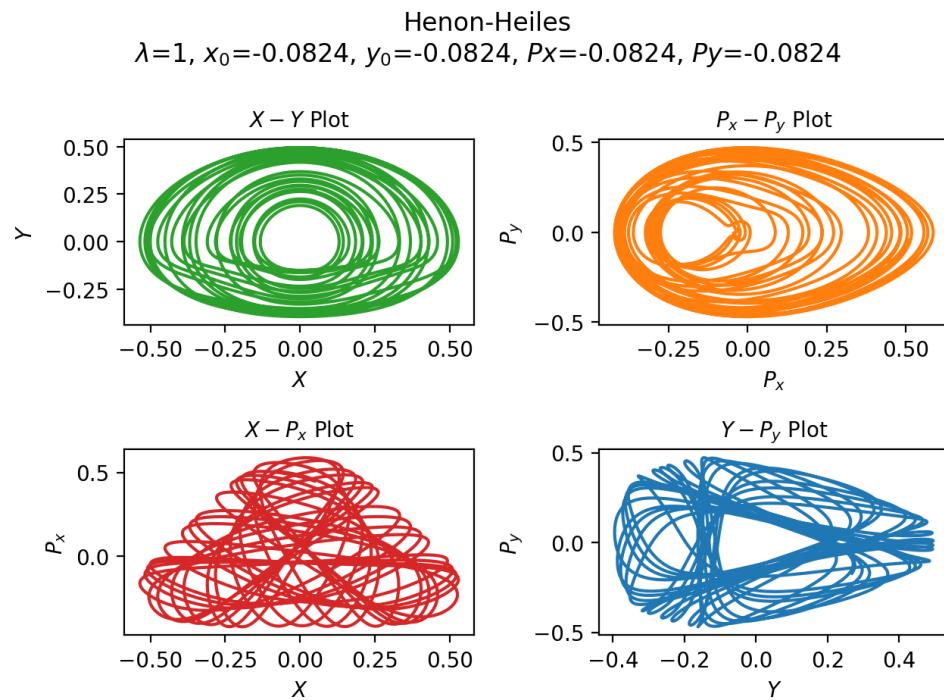


Figure 26: Henon-Heiles system simulation results

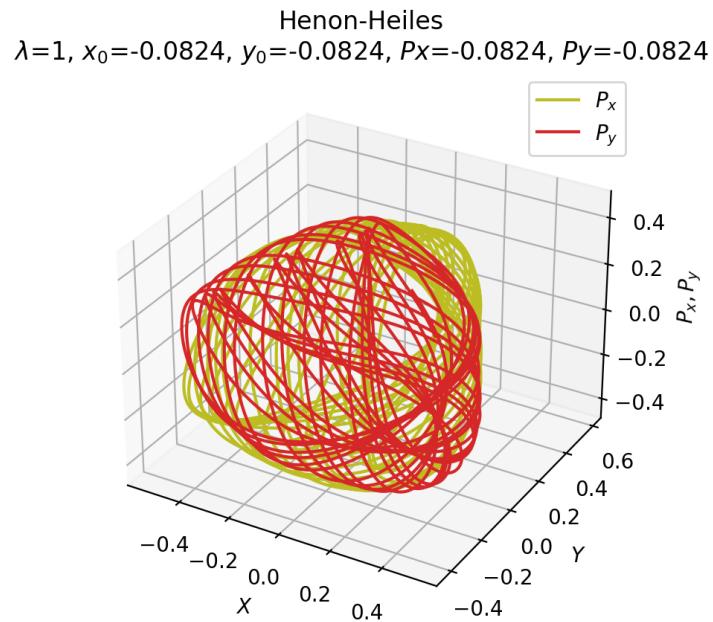


Figure 27: Henon-Heiles system simulation results

10 Hindmarsh-Rose

10.1 Equation

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

$$\begin{aligned} \Phi(x) &= -a \cdot x^3 + b \cdot x^2 \\ \Psi(x) &= c - d \cdot x^2 \end{aligned}$$

10.2 Python model

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

10.3 Simulink model

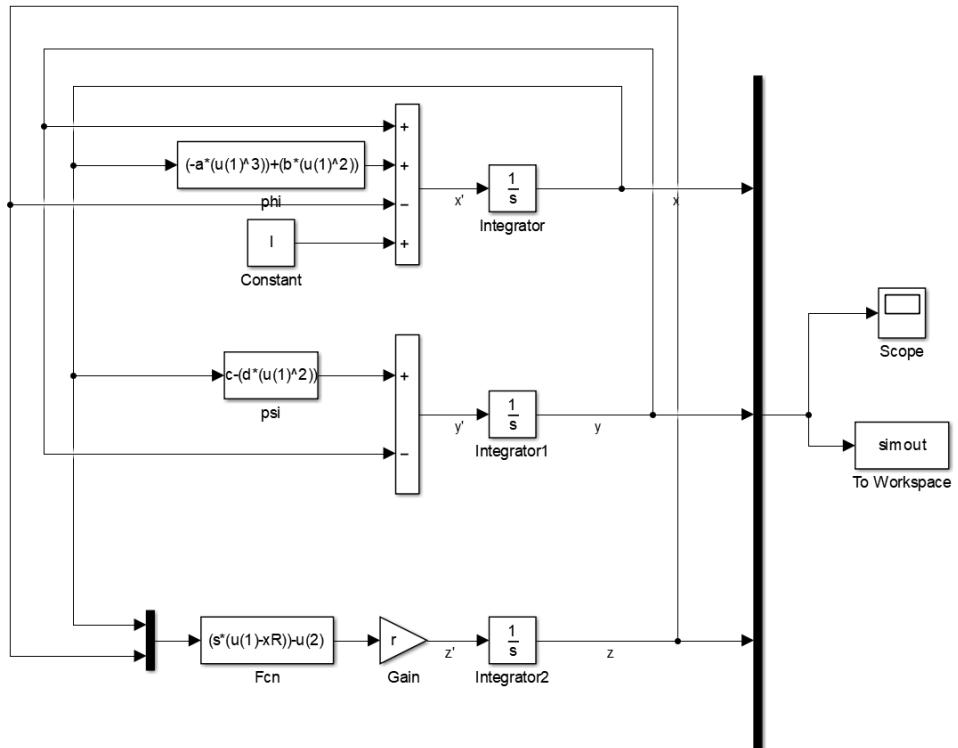


Figure 28: Hindmarsh-Rose Simulink model

10.4 Result

Hindmarsh-Rose
 $b=2.6, l=3.0, r=0.01, s=4.0$

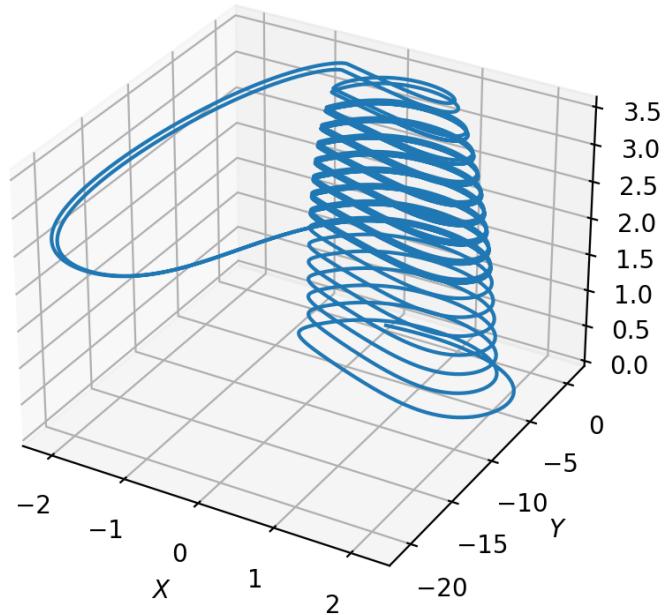


Figure 29: Hindmarsh-Rose system simulation results

Hindmarsh-Rose
 $b=2.6, l=3.0, r=0.01, s=4.0$

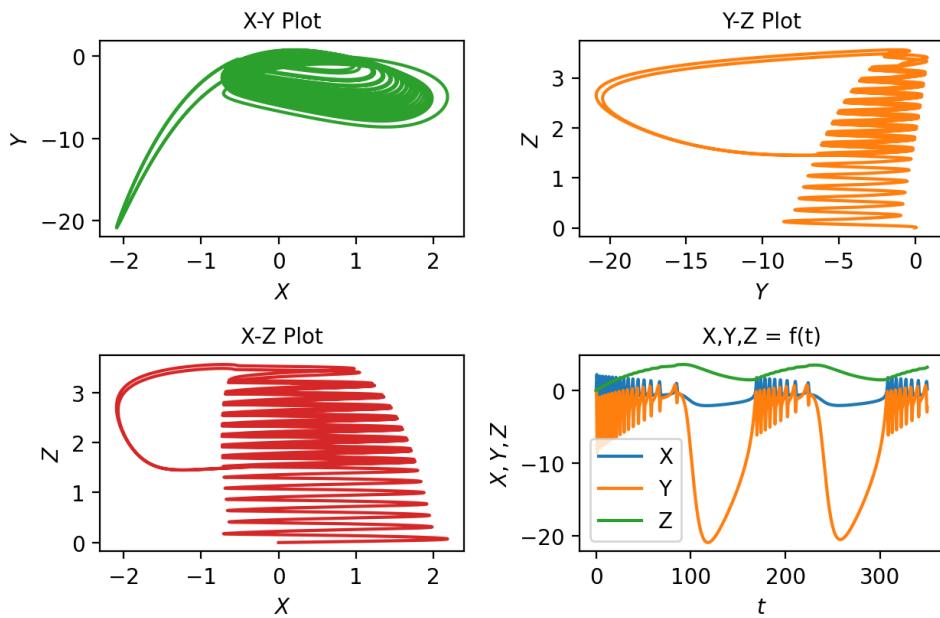


Figure 30: Hindmarsh-Rose system simulation results

11 Lorenz

11.1 Equation

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

11.2 Python model

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

11.3 Simulink model

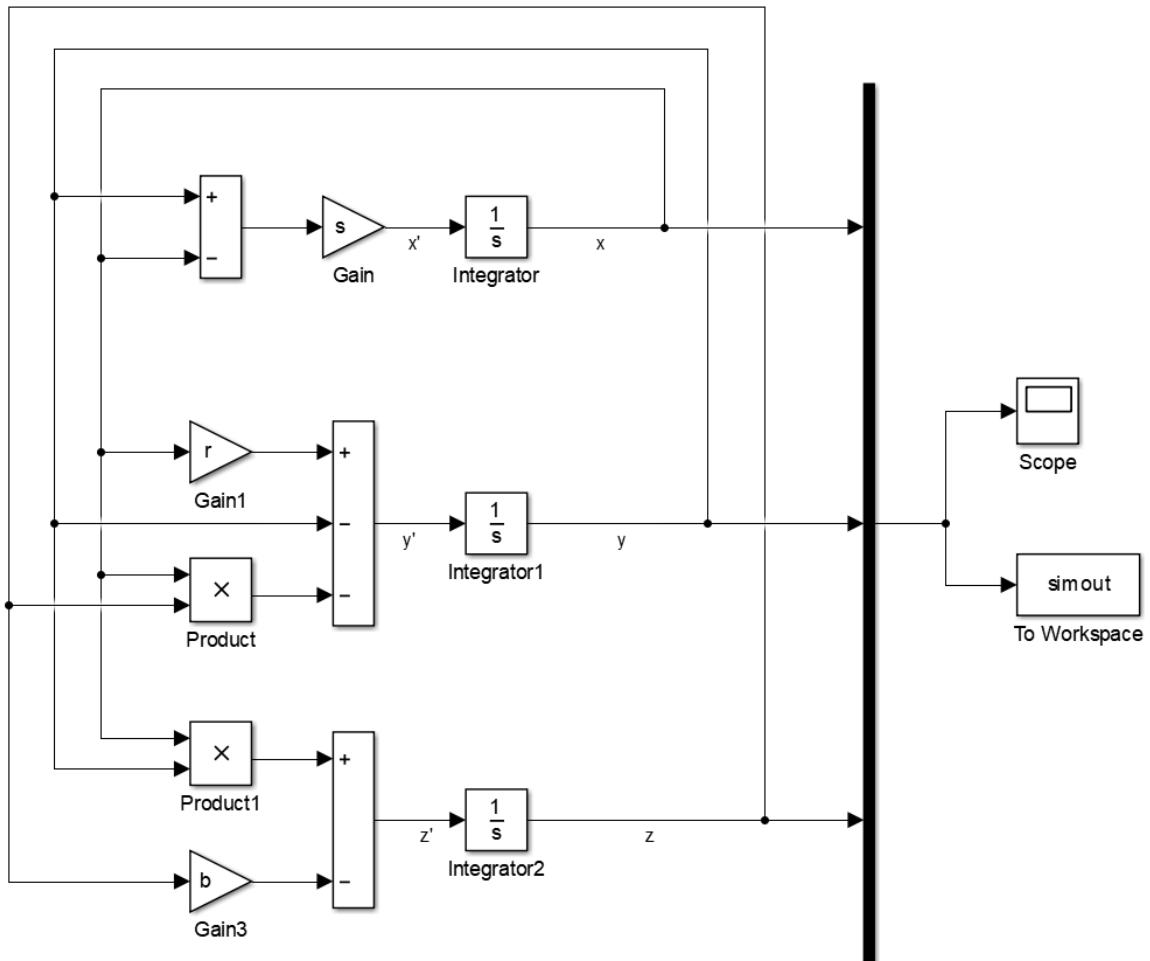


Figure 31: Lorenz Simulink model

11.4 Result

Lorenz
 $\sigma=10.0, \rho = 26.0, \beta = 2.67$

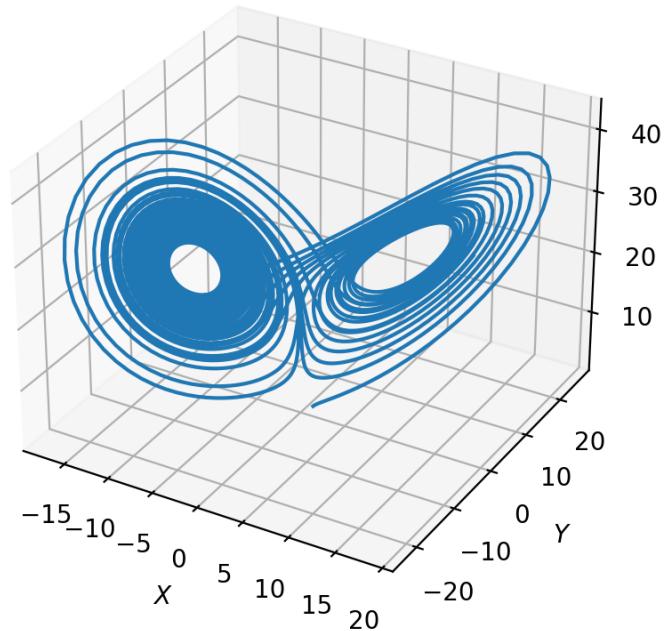


Figure 32: Lorenz system simulation results

Lorenz
 $\sigma=10.0, \rho = 26.0, \beta = 2.67$

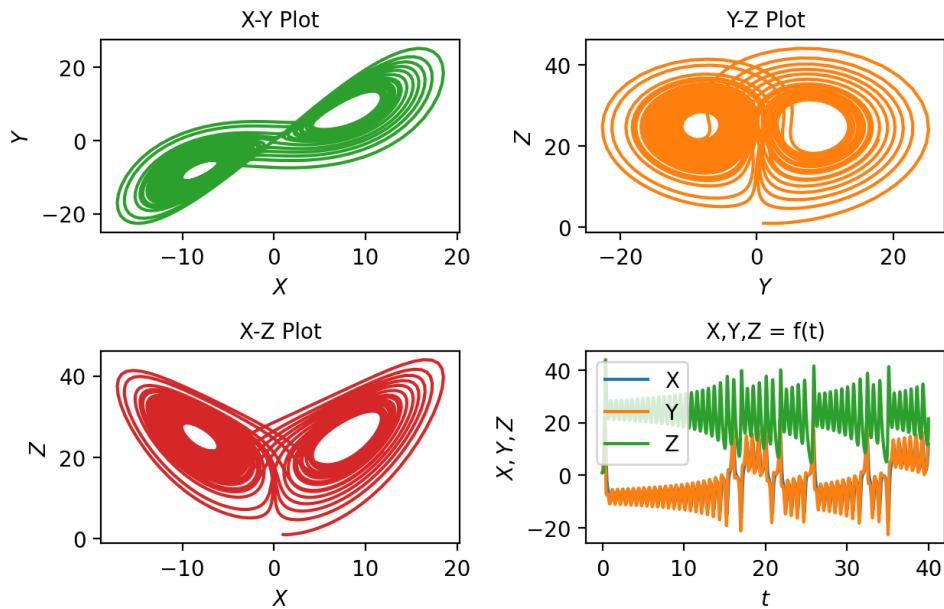


Figure 33: Lorenz system simulation results

12 Mackey-Glass

12.1 Equation

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

12.2 Python model

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

12.3 Simulink model

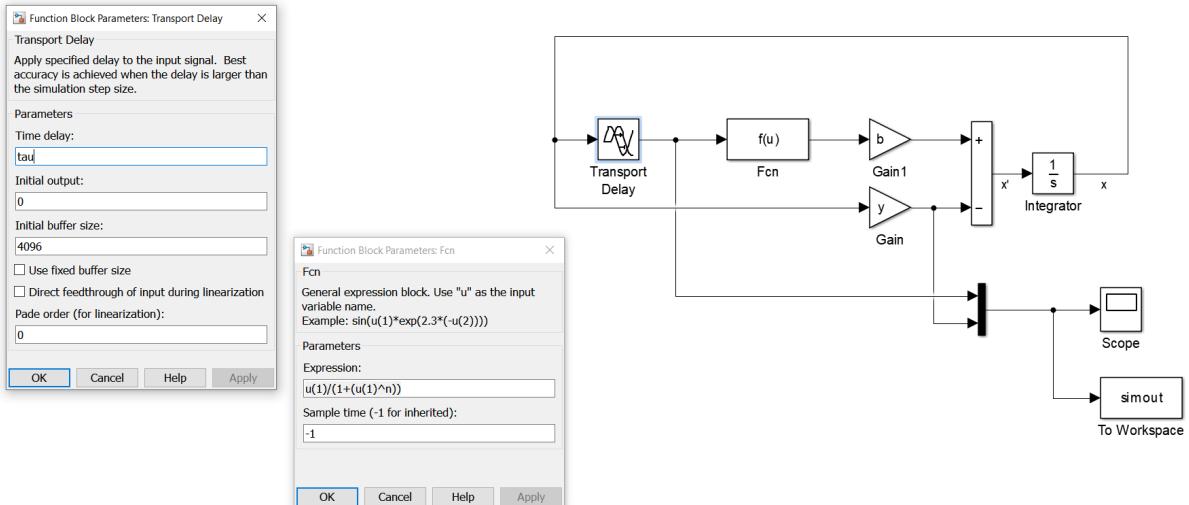


Figure 34: Mackey-Glass Simulink model

12.4 Result

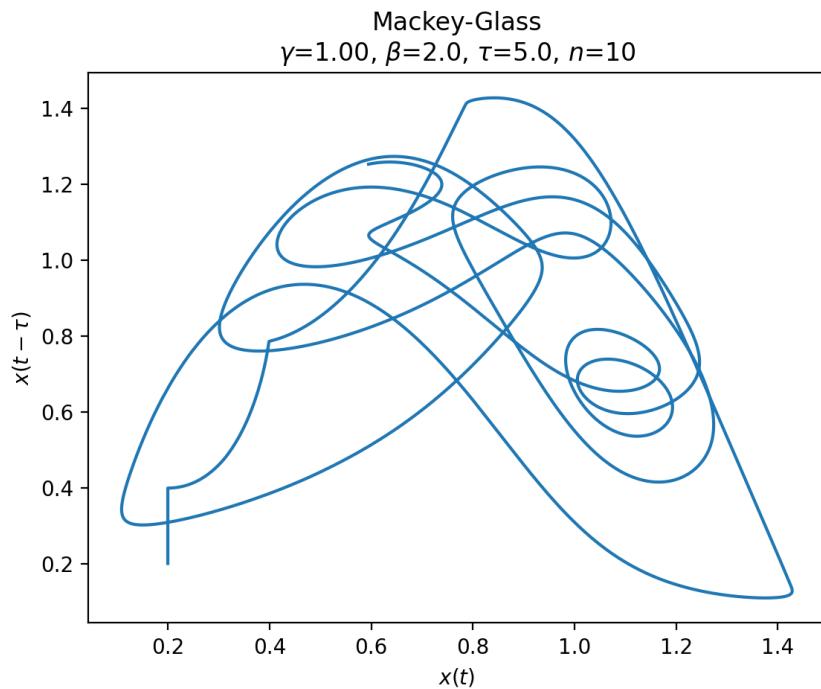


Figure 35: Mackey-Glass system simulation results

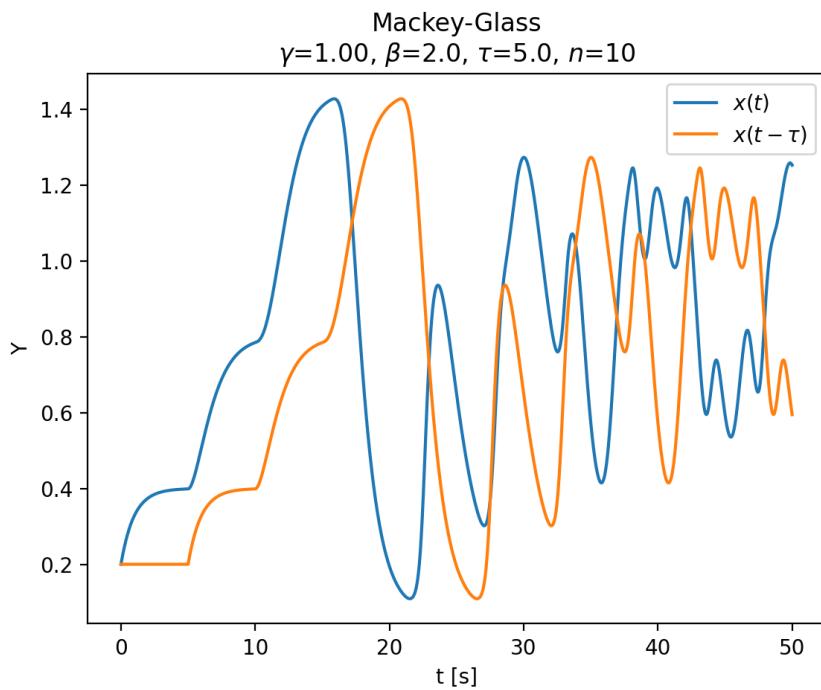


Figure 36: Mackey-Glass system simulation results

13 Nose-Hoover

13.1 Equation

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

13.2 Python model

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

13.3 Simulink model

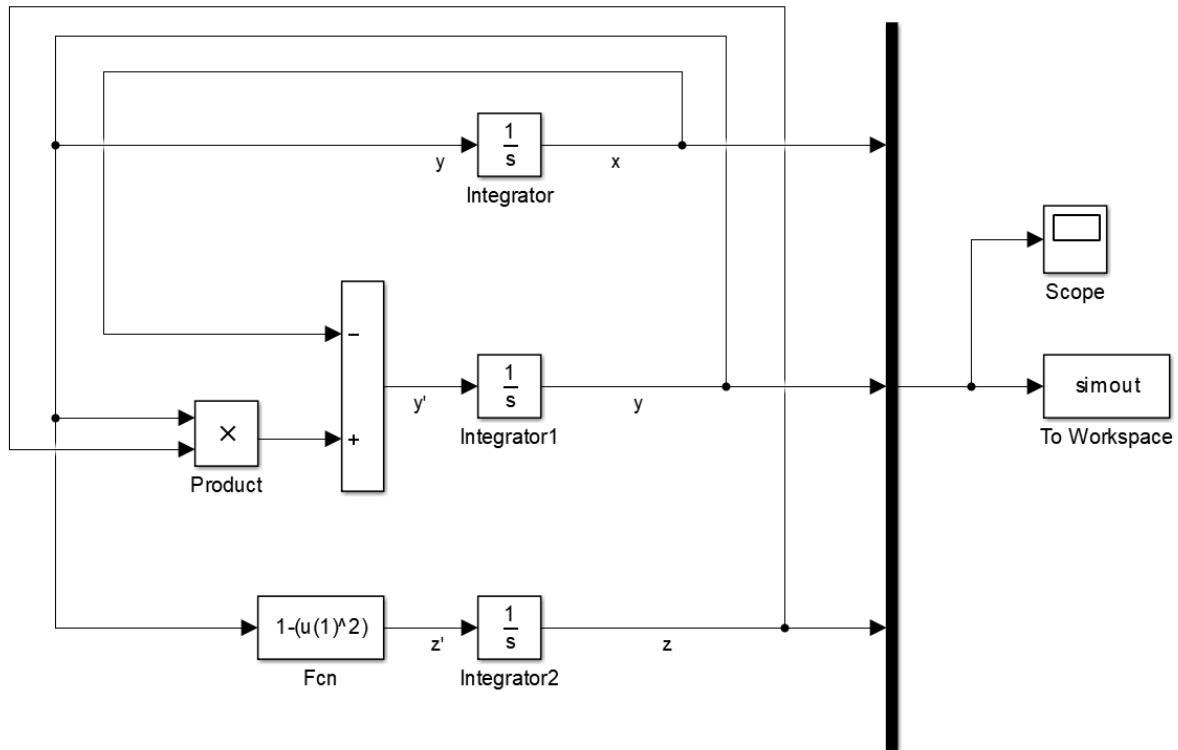


Figure 37: Nose-Hoover Simulink model

13.4 Result

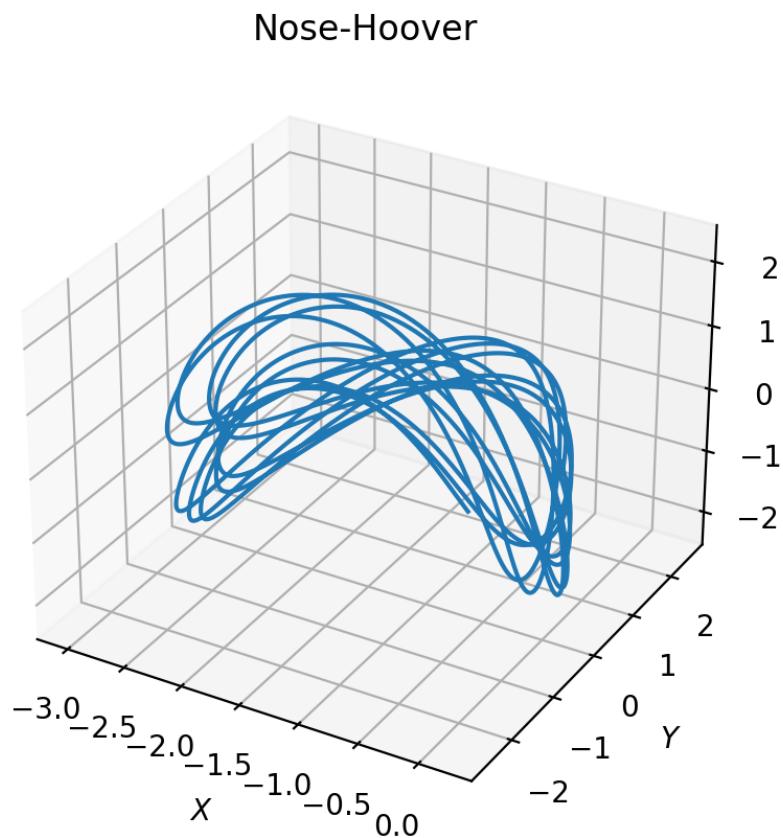


Figure 38: Nose-Hoover system simulation results

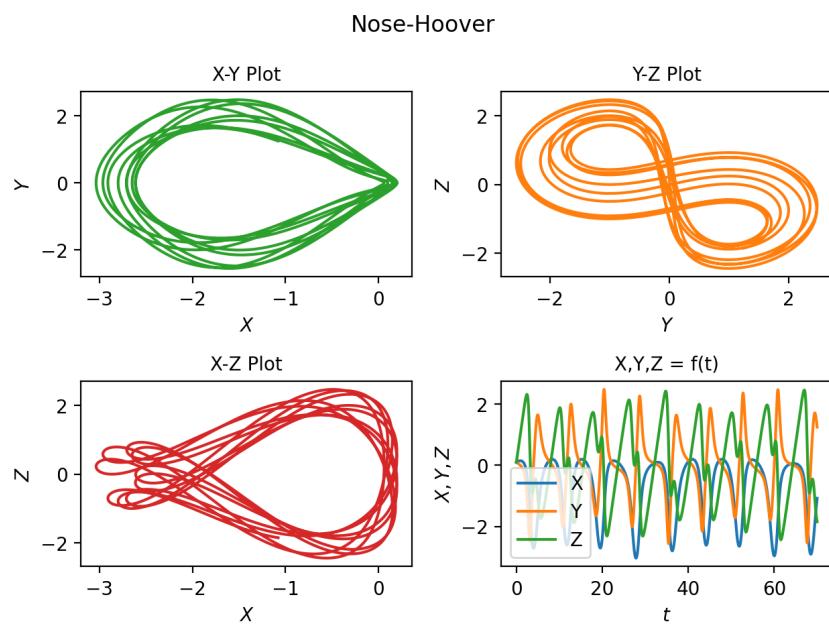


Figure 39: Nose-Hoover system simulation results

14 Rabinovich-Fabrikant

14.1 Equation

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

14.2 Python model

```
1 def rabinovich_fabrikant(state, __time__):
2     x, y, z = state
3     return y * (z - 1 + x ** 2) + InputParams.gamma * x, \
4            x * (3 * z + 1 - x ** 2) + InputParams.gamma * y,
4            \
5            -2 * z * (InputParams.alpha + x * y)
```

14.3 Simulink model

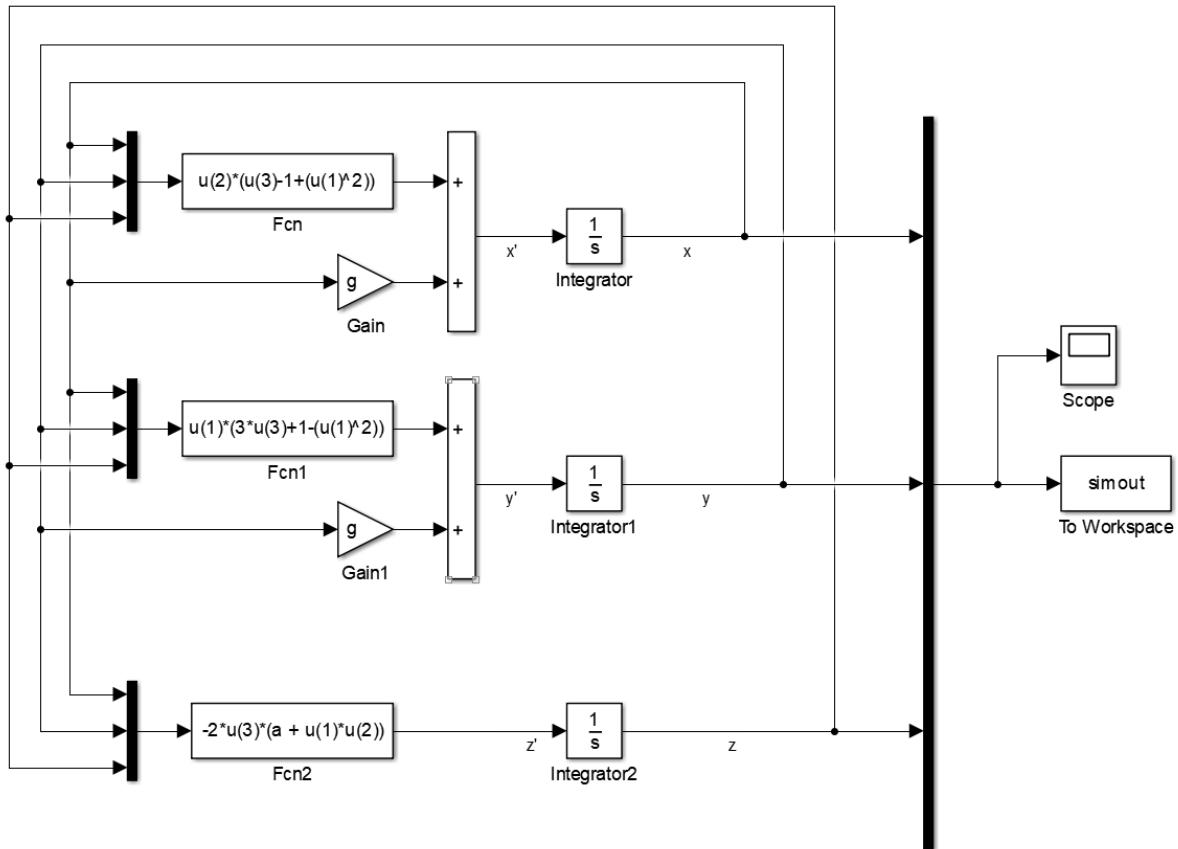


Figure 40: Rabinovich-Fabrikant Simulink model

14.4 Result

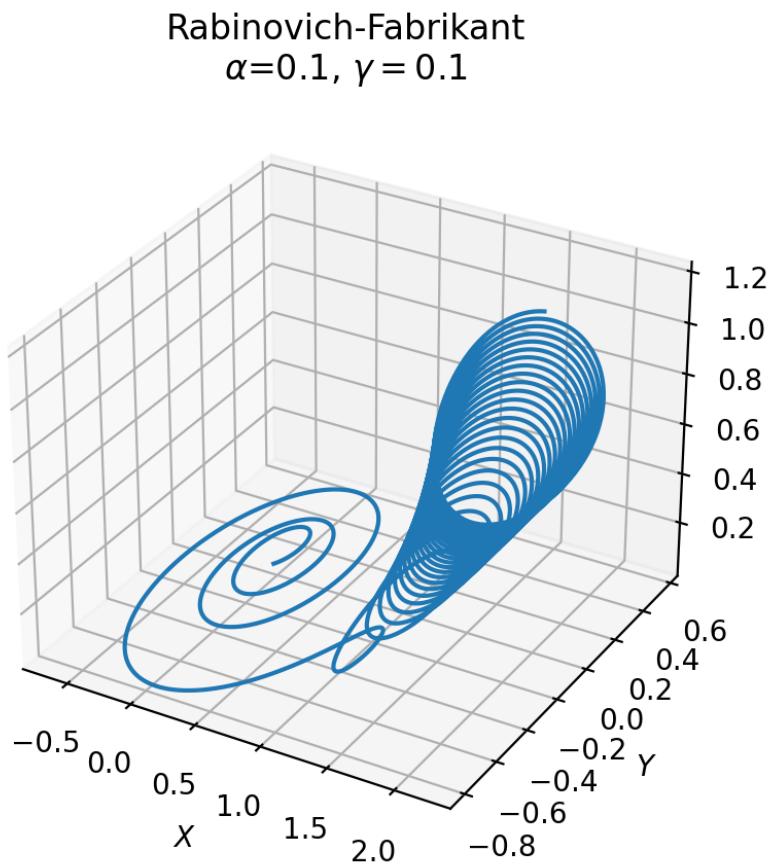


Figure 41: Rabinovich-Fabrikant system simulation results

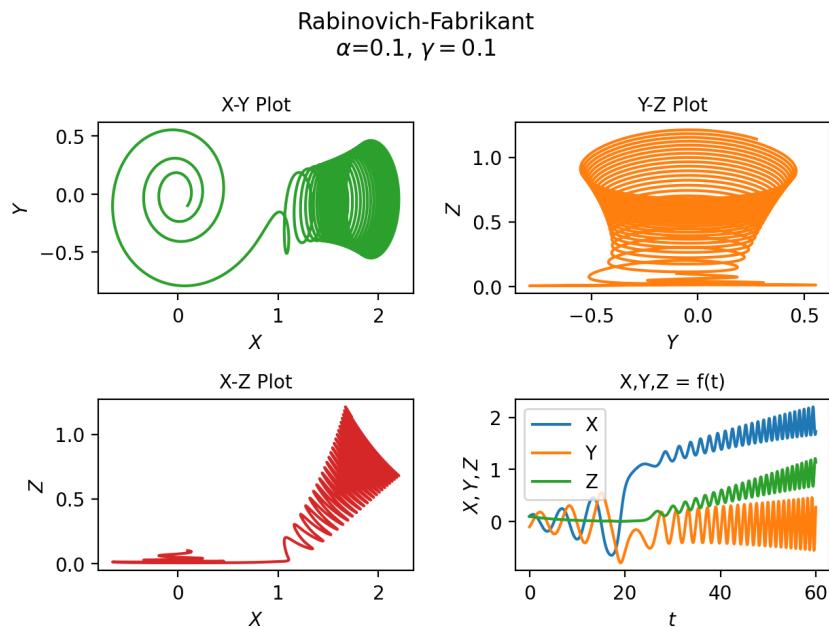


Figure 42: Rabinovich-Fabrikant system simulation results

15 Rössler

15.1 Equation

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

15.2 Python model

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

15.3 Simulink model

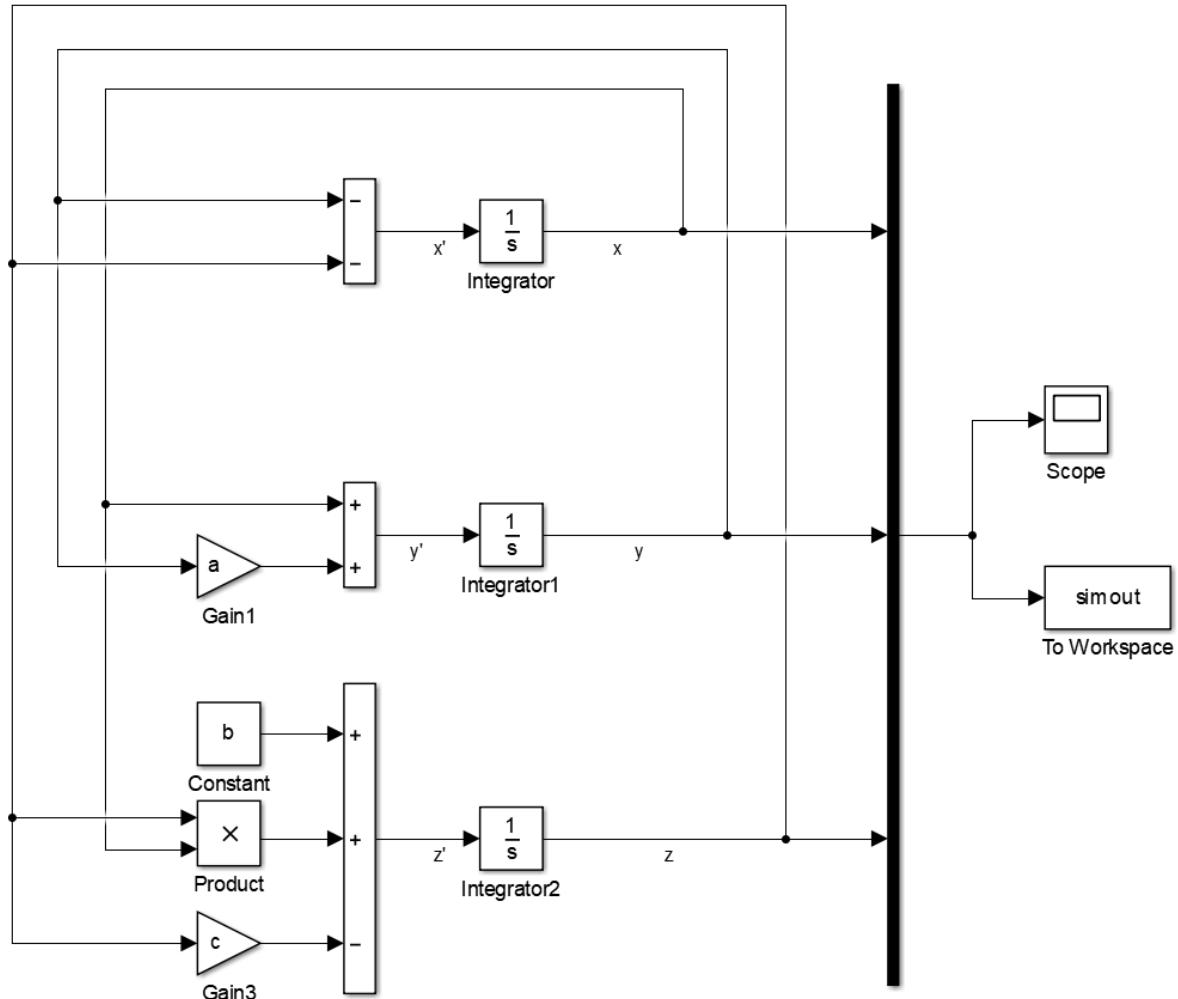


Figure 43: Rössler Simulink model

15.4 Result

Roessler
 $a=0.38, b=0.20, c=5.70$

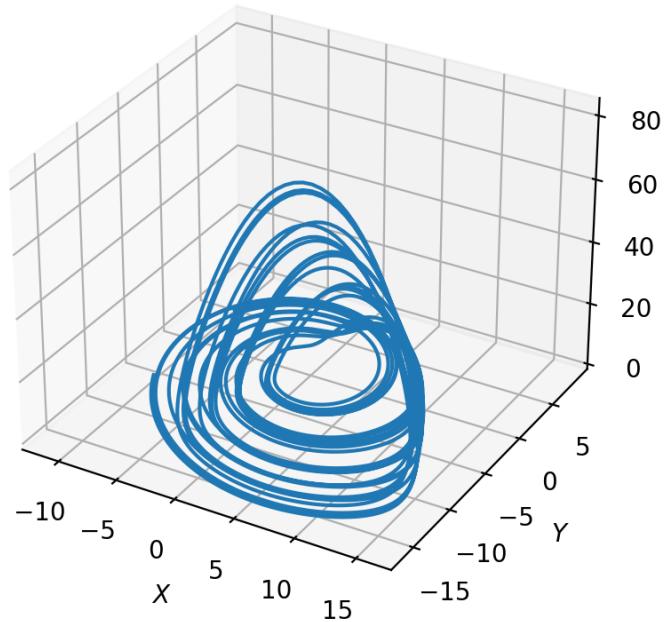


Figure 44: Rössler system simulation results

Roessler
 $a=0.38, b=0.20, c=5.70$

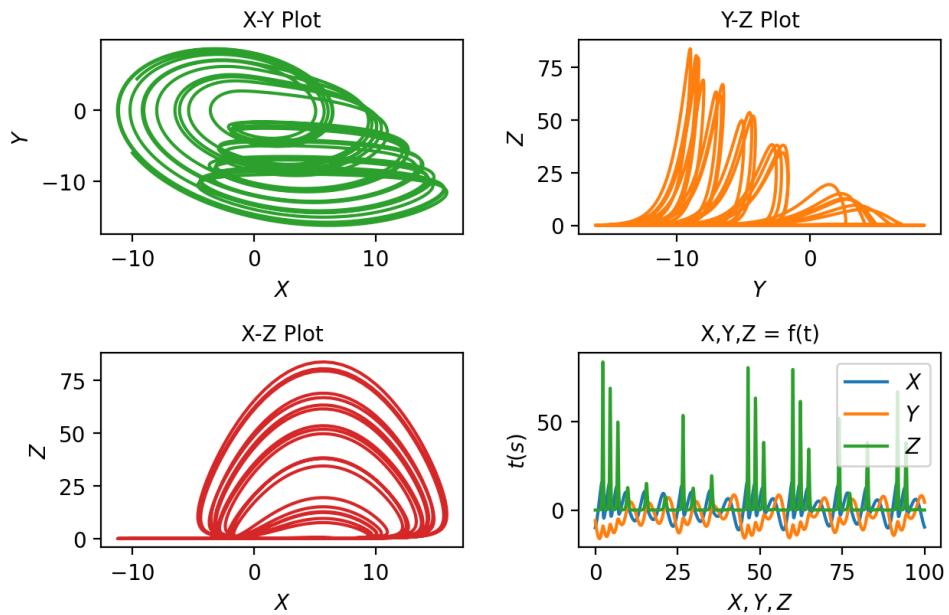


Figure 45: Rössler system simulation results

16 Rössler (Hyperchaotic)

16.1 Equation

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

16.2 Python model

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

16.3 Simulink model

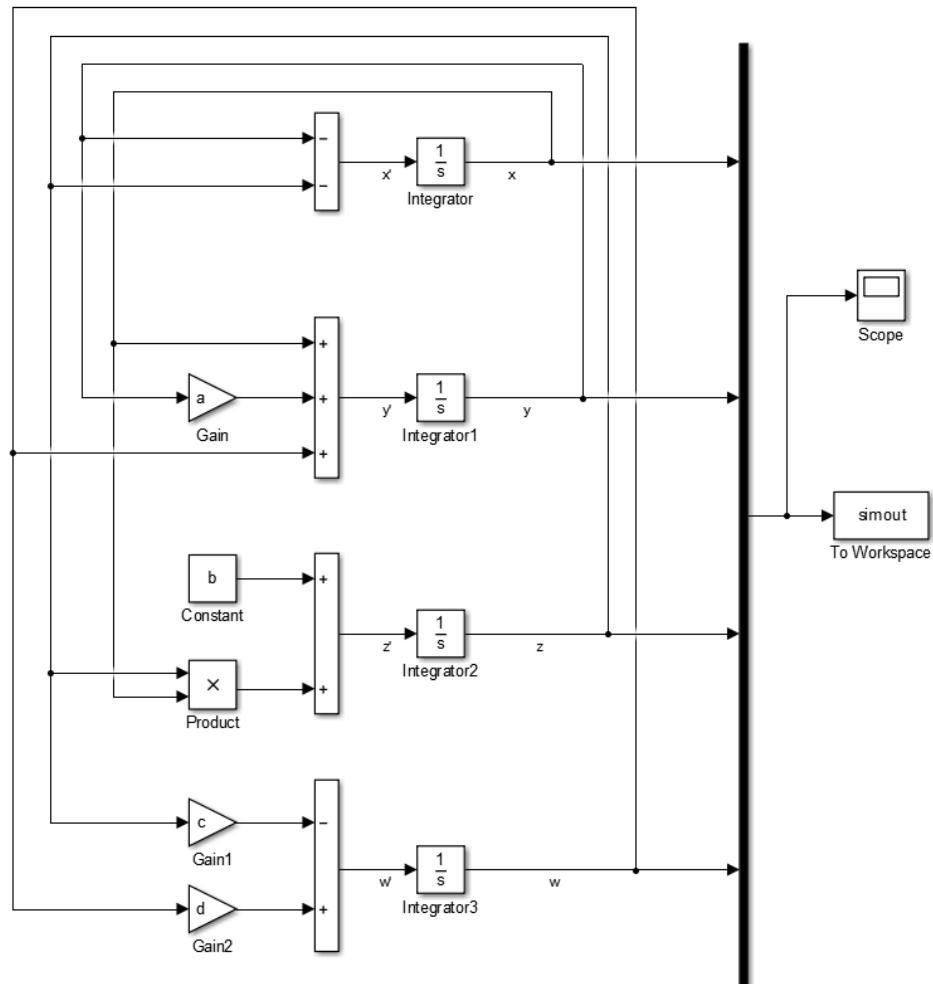


Figure 46: Hyperchaotic Rössler Simulink model

16.4 Result

Hyper chaotic Roessler
 $a=0.25, b=2.00, c=0.50, d=0.05$

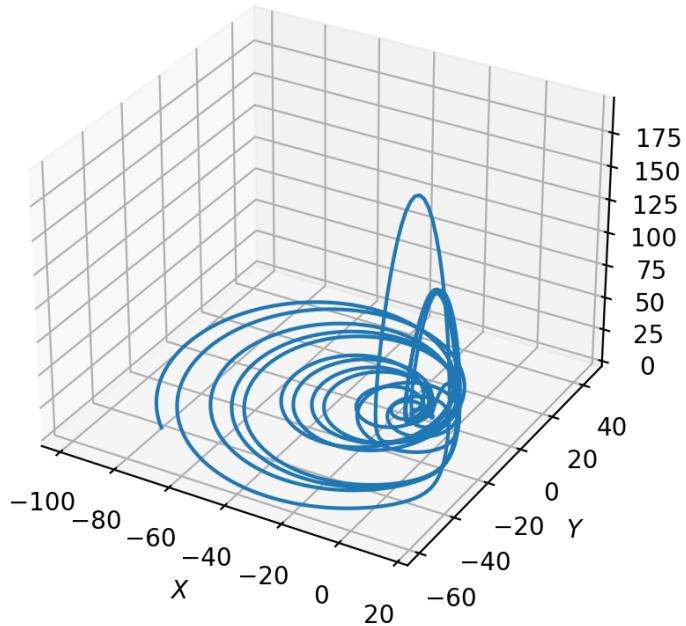


Figure 47: Hyperchaotic Rössler system simulation results

Hyper chaotic Roessler
 $a=0.25, b=2.00, c=0.50, d=0.05$

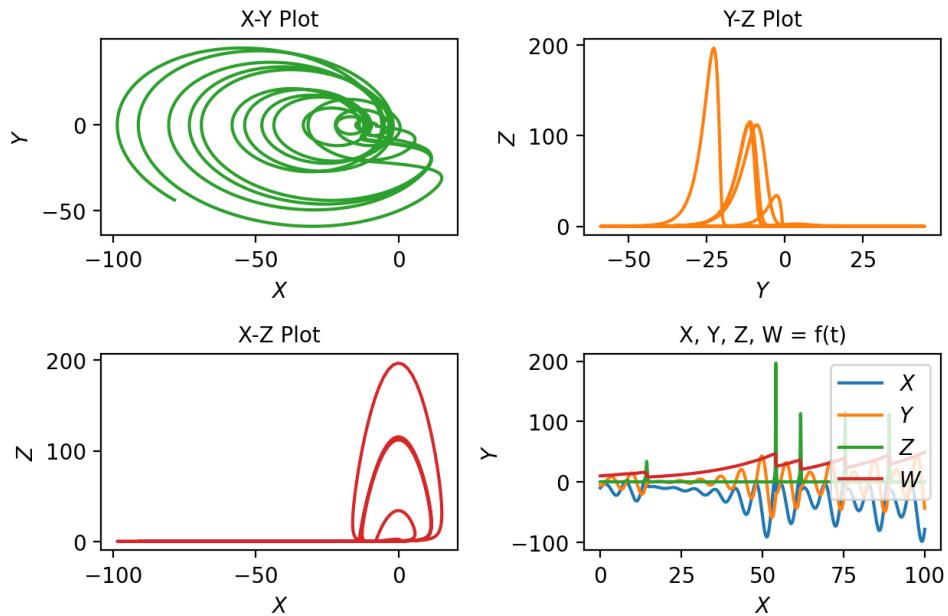


Figure 48: Hyperchaotic Rössler system simulation results

17 Sprott B

17.1 Equation

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

17.2 Python model

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

17.3 Simulink model

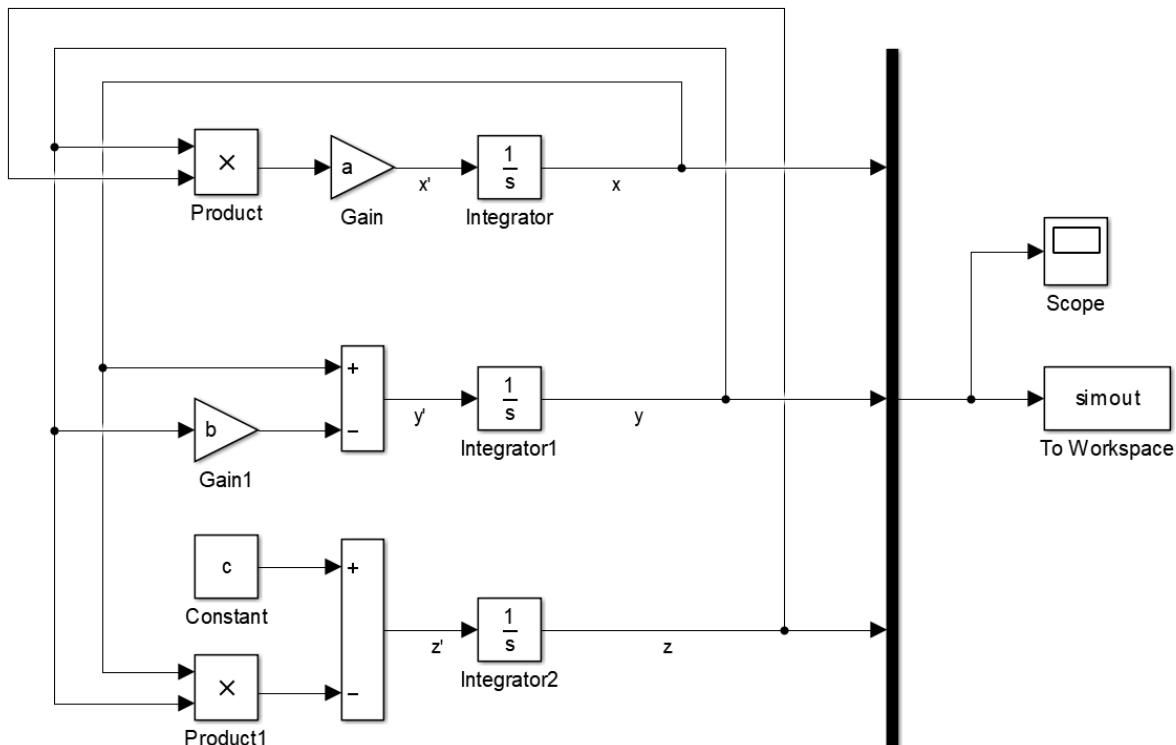


Figure 49: Sprott B Simulink model

17.4 Result

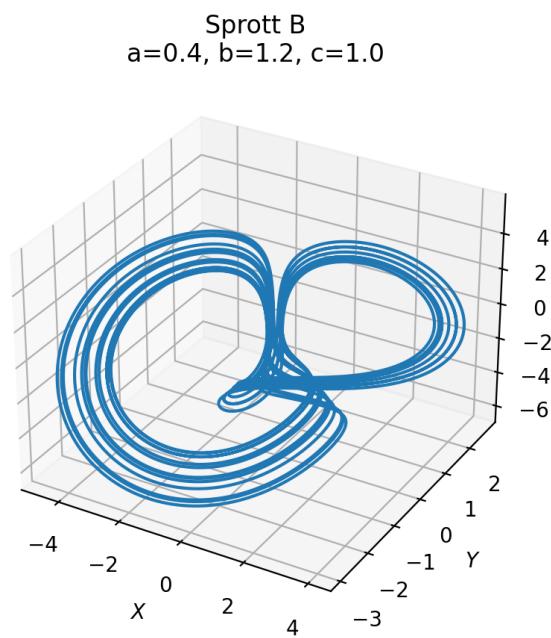


Figure 50: Sprott B system simulation results

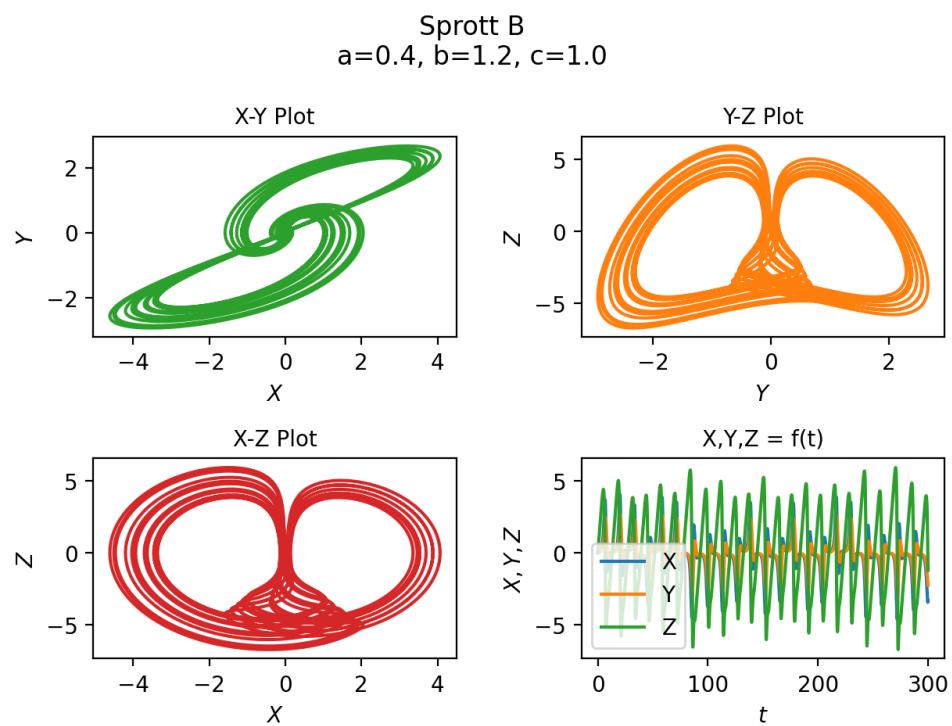


Figure 51: Sprott B system simulation results

18 Thomas' Cyclically Symmetric Attractor

18.1 Equation

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

18.2 Python model

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

18.3 Simulink model

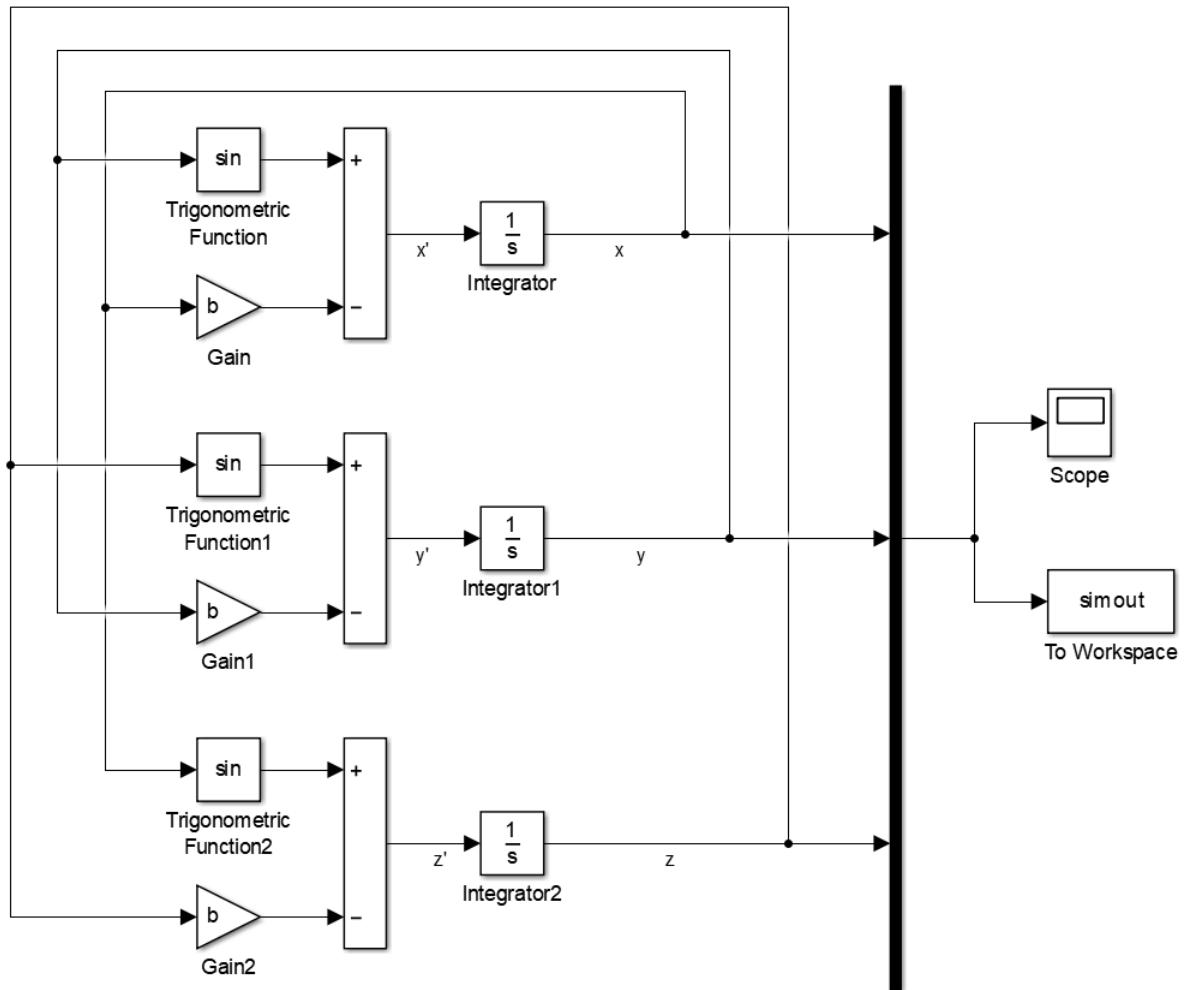


Figure 52: Thomas' attractor Simulink model

18.4 Result

Thomas' Attractor
 $b=0.19$

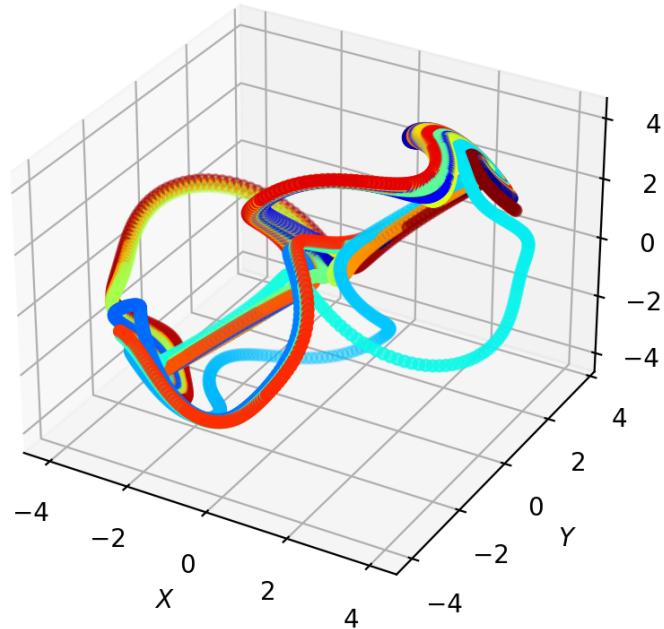


Figure 53: Thomas' cyclically symmetric attractor simulation results

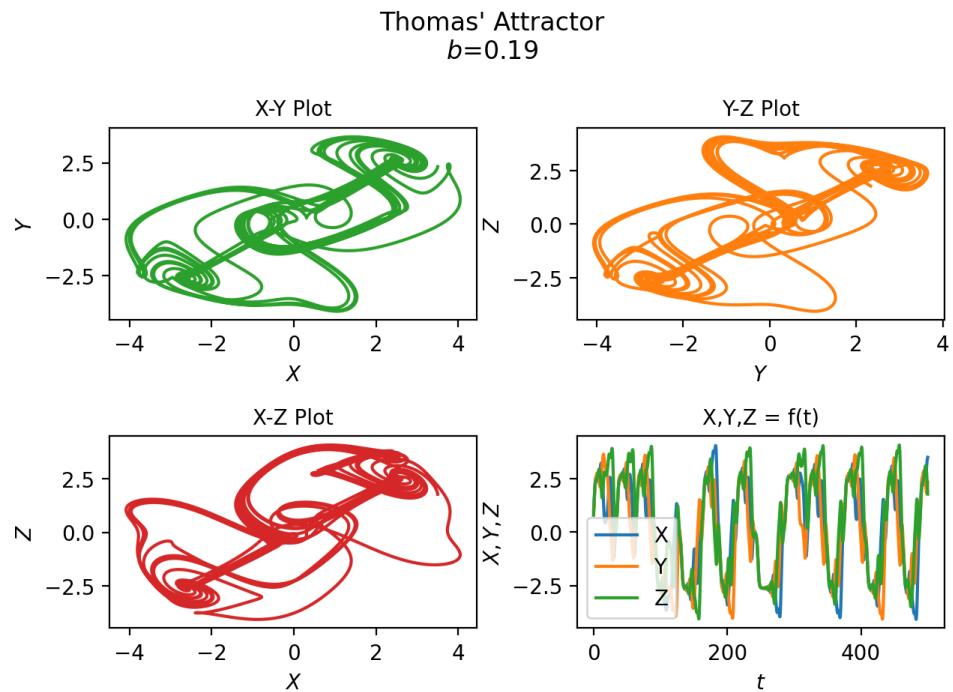


Figure 54: Thomas' cyclically symmetric attractor simulation results

19 Van der Pol

19.1 Equation

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

19.2 Python model

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

19.3 Simulink model

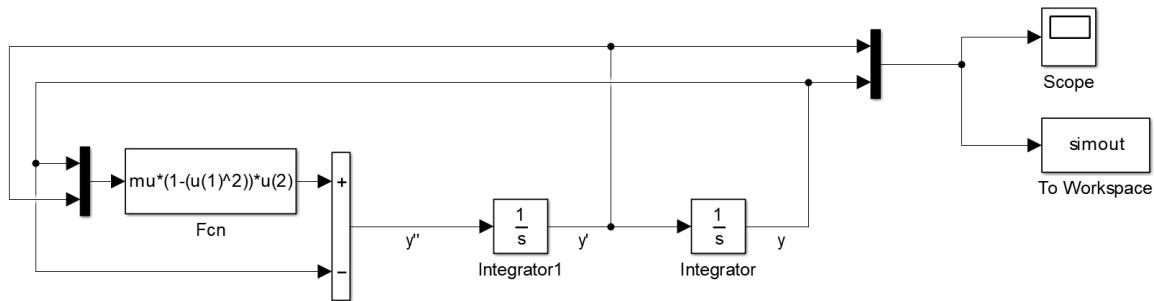


Figure 55: Van der Pol system simulation results

19.4 Result

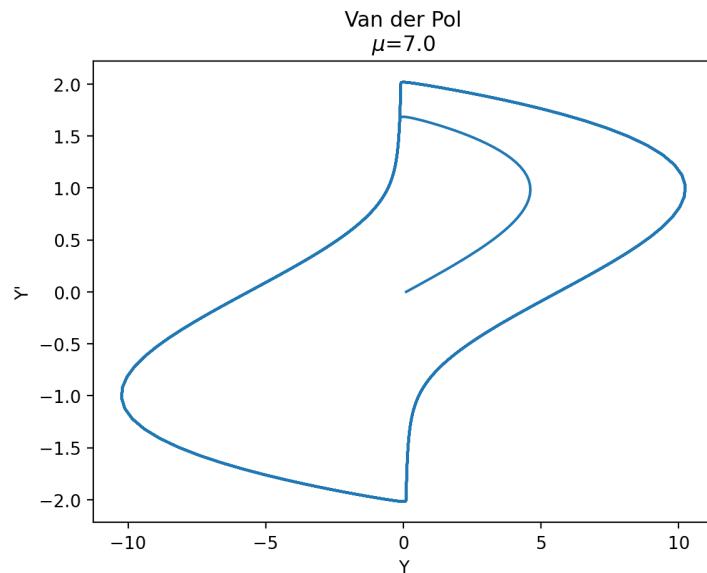


Figure 56: Van der Pol Simulink model

Conclusion

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