# Sprawozdanie MSD - Lista III

Konrad Borkowicz

4 maja 2022

## 1 Wstęp

W następującym sprawozdaniu badane są dwa modele - Lotki-Volterry (opracowywany już w poprzednim sprawozdaniu) i Lorenza.

Układ równań Lotki-Volterry modeluje interakcje między dwoma grupami - drapieżnikami i ofiarami. By uprościć opisywaną sytuację przyjmuje kilka przypuszczeń - na przykład nieskończony dostęp do jedzenia przez ofiary.

Układ Lorenza jest zaś układem trzech nieliniowych równań różniczkowych służący do uproszczonego modelowania zjawiska konwekcji termicznej w atmosferze. Charakteryzuje się chaotycznym zachowaniem dla pewnego zbioru parametrów, tj. rozwiązania reagują bardzo gwałtownie na bardzo małe zaburzenia parametrów.

## 2 Model

### 2.1 Układ Lotki-Volterry

Model przyjmuje następującą postać:

$$\begin{aligned} \frac{dx}{dt} &= (a - by)x \\ \frac{dy}{dt} &= (cx - d)y \end{aligned} \tag{1}$$

Gdzie $\frac{dx}{dt}$ to zmiana populacji ofiar w czasie, $\frac{dy}{dt}$ to zmiana populacji drapieżników w czasie, $a, c$ to częstość narodzin odpowiednio ofiar i drapieżników, $b, d$ to częstość umierania odpowiednio ofiar i drapieżników. Model zaimplementowałem na dwa sposoby - za pomocą metody Eulera i `scipy.integrate.odeint`. Obie implementacje napisane są w języku Python.

### 2.1.1 Metoda Eulera

```python
import numpy as np
from nsteps import nsteps


def calc_euler(
    interval: int,
    dt: float,
    x0: int,
    y0: int,
    params: tuple[float, float, float, float],
):

    a, b, c, d = params
    steps = nsteps(interval, dt)
    t = np.linspace(0, interval, steps)
    xy = np.zeros((steps, 2))
    xy[0, 0] = x0
    xy[0, 1] = y0

    for i in range(np.shape(xy)[0] - 1):
        x, y = xy[i, 0], xy[i, 1]
        dx = (a - b * y) * x * dt
        dy = (c * x - d) * y * dt
        xy[i + 1, 0] = x + dx
        xy[i + 1, 1] = y + dy
    return xy, t
```

`nsteps` jest funkcją pomocniczą wyliczającą ilość kroków między zerem a pożądanym punktem końcowym.

### 2.1.2 scipy.integrate.odeint

```python
from matplotlib import pyplot as plt
import numpy as np
from scipy.integrate import odeint
from nsteps import nsteps


def lotka_volterra(xy, t, a, b, c, d):
    prey, pred = xy
    dydt = [(a - b * pred) * prey, (c * prey - d) * pred]
    return dydt

```

```
12
13 def calc_odeint(interval, dt, x0, y0, params):
14     a, b, c, d = params
15     steps = nsteps(interval, dt)
16     t = np.linspace(0, interval, steps)
17     xy0 = [x0, y0]
18     xy = odeint(lotka_volterra, xy0, t, args=(a, b, c, d))
19     return xy, t
```

Funkcja `odeint` sama z siebie zwraca wyniki w całkiem dobrej formie, jednak ja manipuluję je, by ułatwić porównanie wyników (i przez to precyzji aproksymacji).

## 2.2 Układ Lorenza

Model przyjmuje następującą postać:

$$\begin{aligned} \frac{dx}{dt} &= \sigma(y - x) \\ \frac{dy}{dt} &= x(\rho - z) - y \\ \frac{dz}{dt} &= xy - \beta z \end{aligned} \tag{2}$$

Gdzie $\sigma$ to liczba Prandtla, charakteryzująca lepkość ośrodka, $\rho$ to liczba Rayleigha, przewodnictwo cieplne ośrodka, a $\beta$ to stała charakteryzująca rozmiar obszaru przepływu konwekcyjnego. Przyjmujemy, że $\sigma, \beta, \rho > 0$. Najczęściej korzysta się z $\sigma = 10, \beta = \frac{8}{3}$ i zmiennego $\rho$. Model zaimplementowałem na dwa sposoby - za pomocą metody Eulera i `scipy.integrate.odeint`. Obie implementacje napisane są w języku Python.

### 2.2.1 Metoda Eulera

```
1  import numpy as np
2  from nsteps import nsteps
3
4
5  def calc_euler(
6      interval: int,
7      dt: float,
8      x0: int,
9      y0: int,
10     z0: int,
11     params: tuple[float, float, float],
12 ):
13     sigma, beta, rho = params
14     steps = nsteps(interval, dt)
```

```
    t = np.linspace(0, interval, steps)
    xyz = np.zeros((steps, 3))
    xyz[0, 0] = x0
    xyz[0, 1] = y0
    xyz[0, 2] = z0

    for i in range(np.shape(xyz)[0] - 1):
        x, y, z = xyz[i, 0], xyz[i, 1], xyz[i, 2]
        dx = sigma * (y - x) * dt
        dy = (x * (rho - z) - y) * dt
        dz = ((x * y) - (beta * z)) * dt
        xyz[i + 1, 0] = x + dx
        xyz[i + 1, 1] = y + dy
        xyz[i + 1, 2] = z + dz
    return xyz, t
```

### 2.2.2 scipy.integrate.odeint

```
import numpy as np
from scipy.integrate import odeint
from nsteps import nsteps


def lorenz(xyz, t, sigma, beta, rho):
    x, y, z = xyz
    delta = [sigma * (y - x), (x * (rho - z) - y), ((x * y) - (beta * z))]
    return delta


def calc_odeint(interval, dt, x0, y0, z0, params):
    sigma, beta, rho = params
    steps = nsteps(interval, dt)
    t = np.linspace(0, interval, steps)
    xyz0 = [x0, y0, z0]
    xyz = odeint(lorenz, xyz0, t, args=(sigma, beta, rho))
    return xyz, t
```

## 2.3 Porównania między metodami

By porównać obie metody (i jednocześnie przedstawić układy na wykresach) skorzystałem z właściwości macierzy w bibliotece `numpy` pozwalającej mi na przeprowadzanie działań między elementami.

### 2.3.1 Układ L-V

```
import euler_lv
import odeint_lv
import matplotlib.pyplot as plt
import numpy as np

PARAMS = (1.2, 0.6, 0.3, 0.8)
X0 = 2
Y0 = 1
TESTED_STEPS = (0.3, 0.1, 0.01, 0.001, 0.0001)
INTERVAL = 25

print(f"{'step':>6} | {'x':>9} | {'y':>9} | {'(x+y)/2':>9}")
fig, axs = plt.subplots(nrows=len(TESTED_STEPS))
for i, stepsize in enumerate(TESTED_STEPS):
    eul, t = euler_lv.calc_euler(INTERVAL, stepsize, X0, Y0, PARAMS)
    ode, t = odeint_lv.calc_odeint(INTERVAL, stepsize, X0, Y0, PARAMS)
    # print(t.shape, eul[:, 0].shape)
    avg_deviation_x = np.average(np.abs(eul[:, 0] - ode[:, 0]))
    avg_deviation_y = np.average(np.abs(eul[:, 1] - ode[:, 1]))
    print(
        f"{stepsize:6} | {avg_deviation_x:9.6f} "
        f"| {avg_deviation_y:9.6f} | {(avg_deviation_x+avg_deviation_y)/2:9.6f}"
    )
    axs[i].plot(t, eul[:, 0], "g")
    axs[i].plot(t, ode[:, 0], "r")
    axs[i].set_title(f"dt = {stepsize}")

plt.suptitle("Porównanie między metodą eulera i scipy.odeint (porównywane wartości x)
plt.show()
```

### 2.3.2 Układ Lorenza

```
from euler_lorenz import calc_euler
from odeint_lorenz import calc_odeint
import matplotlib.pyplot as plt
import numpy as np

X0 = Y0 = Z0 = 1
SIGMA = 10
BETA = 8 / 3
RHO = 28
TESTED_STEPS = (0.02, 0.01, 0.005, 0.001, 0.0001)
```

```
INTERVAL = 25


fig, axs = plt.subplots(nrows=len(TESTED_STEPS), ncols=3)
print(f"{'step':>6} | {'x':>10} | {'y':>10} | {'z':>10} | {'(x+y+z)/3':>10}")
for i, stepsize in enumerate(TESTED_STEPS):
    eul, t = calc_euler(INTERVAL, stepsize, X0, Y0, Z0, (SIGMA, BETA, RHO))
    ode, _ = calc_odeint(INTERVAL, stepsize, X0, Y0, Z0, (SIGMA, BETA, RHO))
    avg_deviation_x = np.average(np.abs(eul[:, 0] - ode[:, 0]))
    avg_deviation_y = np.average(np.abs(eul[:, 1] - ode[:, 1]))
    avg_deviation_z = np.average(np.abs(eul[:, 1] - ode[:, 1]))

    print(
        f"{stepsize:6} | {avg_deviation_x:10.6f} | {avg_deviation_y:10.6f} | {avg_dev
    )

    axs[i, 0].plot(eul[:, 0], eul[:, 1])  # x(y)
    axs[i, 1].plot(eul[:, 1], eul[:, 2])  # y(z)
    axs[i, 2].plot(eul[:, 0], eul[:, 2])  # x(z)

plt.show()
```

# 3 Wyniki

Badałem wpływ zwiększania kroku czasu przy aproksymacji układów Lotki-Volterry i Lorenza. Korzystając ze wzoru na średni błąd bezwzględny w dyskretnej formie obliczyłem błędy w zależności od długości kroku, porównując nietrafną metodę Eulera do wartości wyliczonych przez metodę `odeint`.

## 3.1 Średni błąd bezwzględny

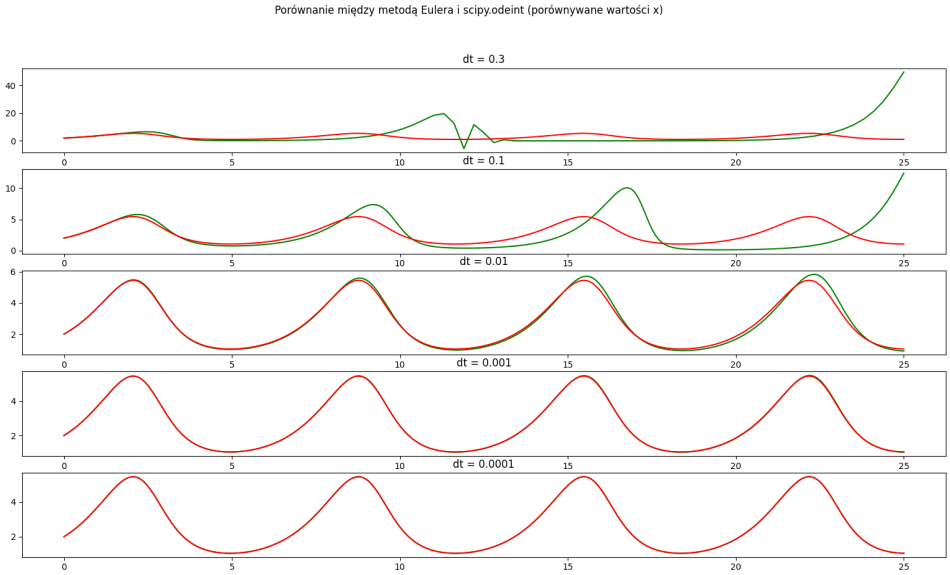
Wyniki są następujące:
*Układ Lotki-Volterry*

| step | x | y | (x+y)/2 |
|---|---|---|---|
| 0.3 | 4.628571 | 2.063101 | 3.345836 |
| 0.1 | 1.662668 | 0.960586 | 1.311627 |
| 0.01 | 0.108122 | 0.070028 | 0.089075 |
| 0.001 | 0.010168 | 0.006588 | 0.008378 |
| 0.0001 | 0.001010 | 0.000655 | 0.000833 |

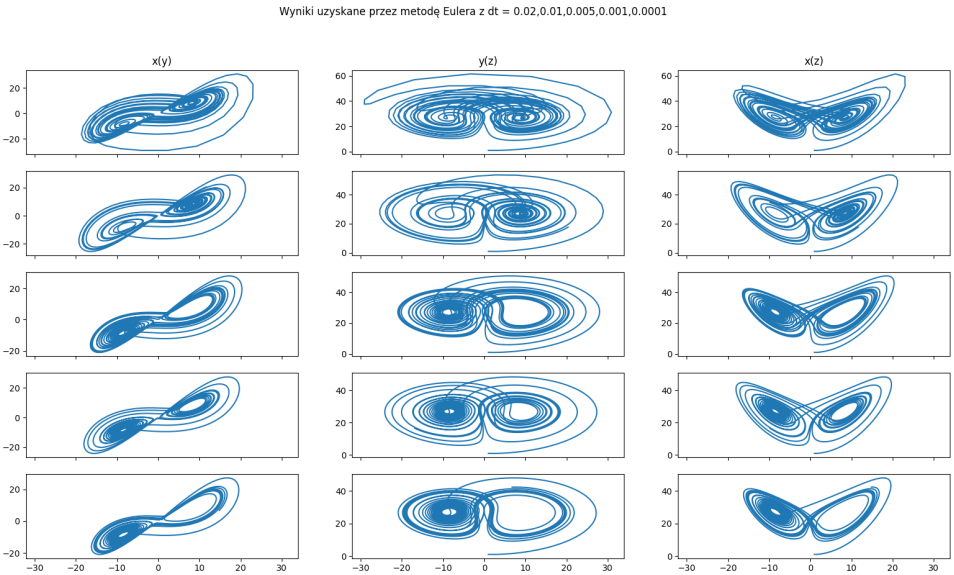
*Układ Lorenza*

| step | x | y | z | (x+y+z)/3 |
|---|---|---|---|---|
| 0.02 | 9.937296 | 10.805932 | 10.805932 | 10.516387 |
| 0.01 | 10.331855 | 11.173354 | 11.173354 | 10.892854 |
| 0.005 | 7.288995 | 8.385183 | 8.385183 | 8.019787 |
| 0.001 | 5.266733 | 5.896026 | 5.896026 | 5.686262 |
| 0.0001 | 3.397206 | 4.016689 | 4.016689 | 3.810195 |

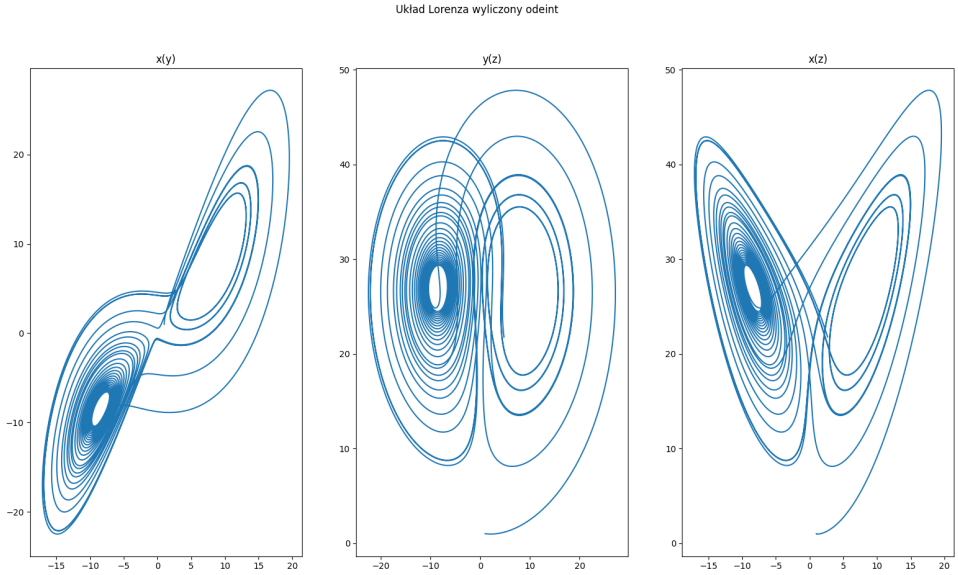
## 3.2 Wykresy



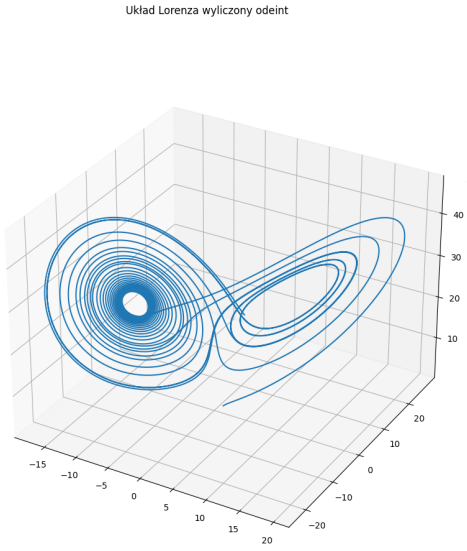

Rysunek 1: Porównanie obu metod (ukł. L-V)




Rysunek 2: Wykresy ukł. Lorenza uzyskane przez metodę Eulera




Rysunek 3: Wykresy ukł. Lorenza uzyskane przez odeint




Rysunek 4: Wykres trójwymiarowy z(x,y) ukł. Lorenza uzyskany przez odeint

# 4 Wnioski

## 4.1 Model Lotki-Volterry

Układ jest bardzo responsywny na zmniejszanie kroku - możemy osiągnąć bardzo dobrą aproksymację układu korzystając z metody Eulera - różnice w wartościach przy odpowiednio dużej precyzji spadają do dziesięciotysięcznych jednostki.

## 4.2 Model Lorenza

Układ przez to, że jest bardzo chaotyczny reaguje mocno na wszelkie zmiany w długości kroku. Nawet przy teoretycznie bardzo dużej precyzji (małym kroku) aproksymowany układ odbiega od prawdziwej wartości o kilka jednostek.