Dominik Szot

Laboratorium 09
Równania różniczkowe zwyczajne

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import scipy.linalg as scp
import scipy.integrate as integrate
import matplotlib.ticker
import sympy
from scipy.optimize import fsolve
import collections
```

Zadanie 1
Przedstaw każde z poniższych równań różniczkowych zwyczajnych jako równoważny układ równań pierwszego rzędu (ang. first-order system of ODEs):

- **równanie Van der Pol'a:**

$y'' = y'(1 - y^2) - y$

$$\begin{cases} y_1 = y' \\ y_1' = y_1(1 - y^2) - y \end{cases}$$

- **równanie Blasiusa:**

$y''' = -yy''$

$$\begin{cases} y_1 = y \\ y_2 = y_1' \\ y_3' = -y_1 y_2' \end{cases}$$

- **II zasada dynamiki Newtona dla problemu dwóch ciał:**

$y_1'' = -GM_{y1}/(y_1^2 + y_2^2)^{3/2}$
$y_2'' = -GM_{y2}/(y_1^2 + y_2^2)^{3/2}$

$$\begin{cases} y_3 = y_1' \\ y_4 = y_2' \\ y_3' = -GMy_1/(y_1^2 + y_2^2)^{3/2} \\ y_4' = -GMy_2/(y_1^2 + y_2^2)^{3/2} \end{cases}$$

Zadanie 2. Dane jest równanie różniczkowe zwyczajne $y_1 = -5y$ z warunkiem początkowym $y(0) = 1$. Równanie rozwiązujemy numerycznie z krokiem $h = 0.5$

- **Czy rozwiązania powyższego równania są stabilne?**

Stabilność w sensie Lapundowa

> Rozwiązanie $y(t)$ jest stabilne w sensie Lapunowa, jeśli dla dowolnego $\epsilon > 0$ istnieje $\delta > 0$, że każde rozwiązanie $x(t)$ tego równania, gdy warunki początkowe spełniają nierówność
>
> $$||x(t_0) - y(t_0)|| < \delta$$
>
> to
>
> $$||x(t) - y(t)|| < \epsilon, t \geq t_0$$

Równanie o zmiennych rozdzielonych, którego rozwiązaniem jest

$$y(t) = e^{-5t}$$

Dla dowolnego $\epsilon > 0$ szukamy $\delta > 0$, że prawdziwa będzie implikacja

$$|1 - 0| < \delta => |e^{-5t} - 0| < \epsilon$$

$$1 < \delta => |e^{-5t} - 0| < \epsilon$$

Ponieważ,

$$1 \geq e^{-5t}, t \geq 0$$

więc dla $\epsilon = \delta$ implikacja jest prawdziwa => rozwiązanie jest stabilne w sensie Lapundowa.

- **Czy metoda Euler'a jest stabilna dla tego równania z użytym krokiem h?**

$$Algorytm:$$
$$u^{n+1} = u^n - f(u^n, t^n) \cdot \Delta t$$

  Rejon bezwzględnej stabilności: $$
|{1 - \Delta t \cdot \lambda } | \le 1 $$

$$|1 - 0.5 \cdot (-5)| \not< 1$$

Warunek nie jest spełniony, więc metoda nie jest stabilna dla tego równania z użytym krokiem h

- **Oblicz numerycznie wartości przybliżonego rozwiązania dla t = 0.5 metodą**

Euler'a.

In [ ]:
```
f_01 = lambda y, t : -5*y
f_01_actual = lambda t : np.e**(-5*t)

def euler_method(y_0, x_0, h, t, f):
    y = y_0
    for _ in range(int(t//h)):
        y = y + h * f(y, t)

    return y

print(f"Numeryczna wartość obliczona metodą Euler'a: {euler_method(1,0, 0.5, 0.5, f_01)}")
print(f"Wartość prawidłowa: {f_01_actual(0.5)}")
```

```
Numeryczna wartość obliczona metodą Euler'a: -1.5
Wartość prawidłowa: 0.0820849986238988
```

- **Wyjaśnij, czy niejawna metoda Euler'a jest stabilna dla tego równania z**

użytym krokiem h?

$$Algorytm:$$
$$u^n = u^{n-1} + f(u^n, t^n) \cdot \Delta t$$

Rejon bezwzględnej stabilności:

$$|\frac{1}{1 - \Delta t \cdot \lambda}| \leq 1$$

$$|\frac{1}{1 - 0.5 \cdot (-5)}| < 1$$

Warunek jest spełniony, więc wyniki pozostaną skończone dla $n \mapsto \infty$

In [ ]:
```python
f_01 = lambda y, t : -5*y
f_01_actual = lambda t : np.e**(-5*t)

def euler_method_implicit(y_0, x_0, h, t, f):
    y = y_0
    for _ in range(int(t//h)):
        y = y/(1 - f(y, t)*h)

    return y

print(f"Numeryczna wartość obliczona metodą Euler'a: {euler_method_implicit(1, 0, 0.5, 0.5, f_0
print(f"Wartość prawidłowa: {f_01_actual(0.5)}")
```

```
Numeryczna wartość obliczona metodą Euler'a: 0.2857142857142857
Wartość prawidłowa: 0.0820849986238988
```

Zadanie 3. Rozwiąż układ równań

$$x'' = -GMx/r^3$$

$$y'' = -GMy/r^3$$

dla $GM = 1$, $r = (x^2 + y^2)^{1/2}$

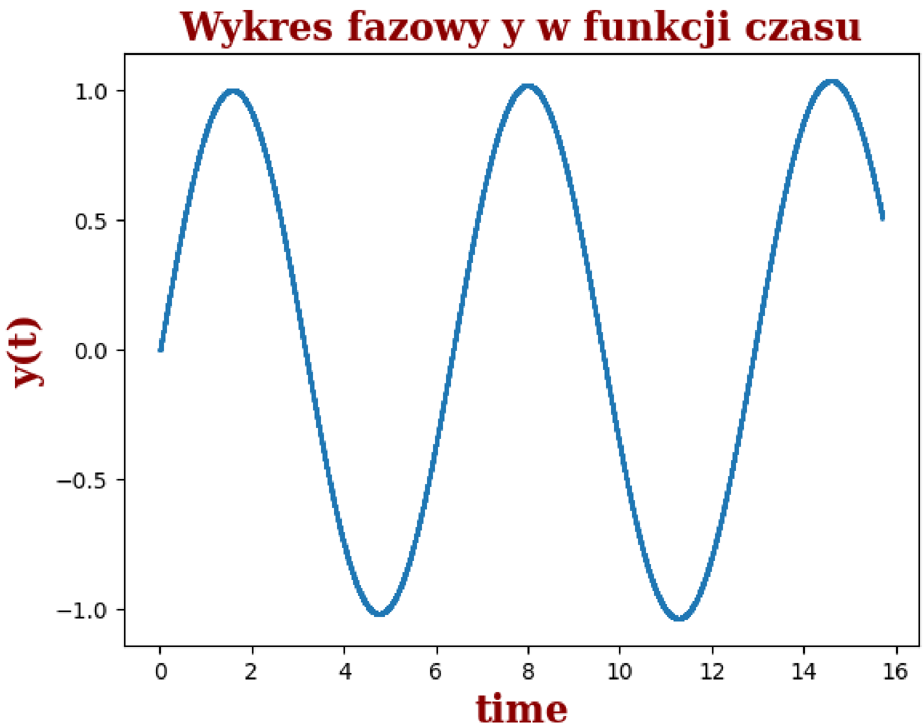
- **używając jawnej metody Eulera**

$$y_{k+1} = y_k + h_k f'(t_k, y_k)$$

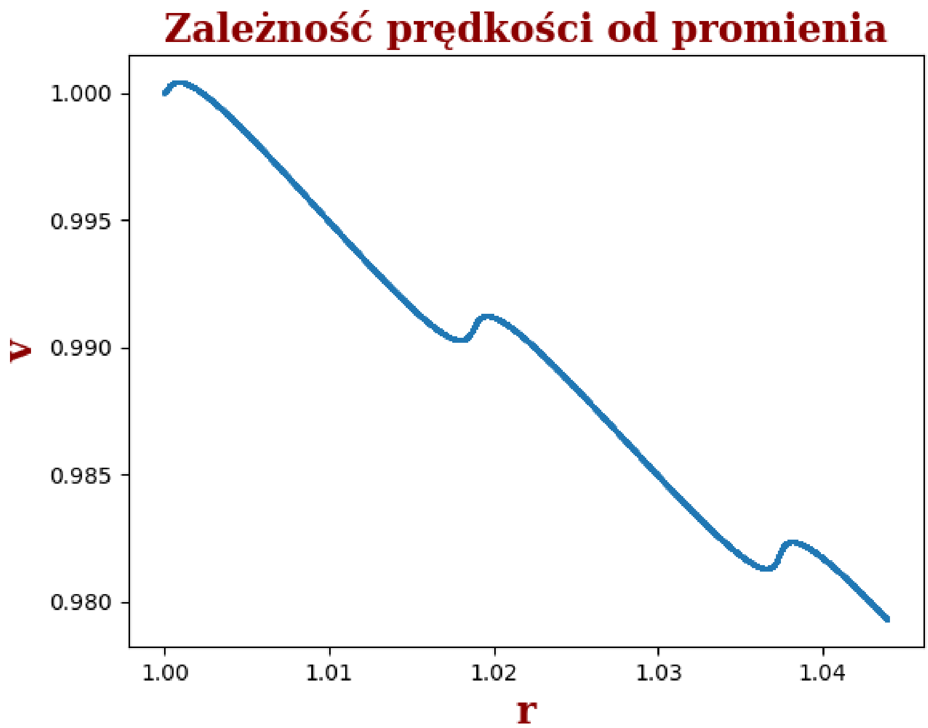
In [ ]:
```python
# Simulation settings

r_0 = [1, 0] # Position vector
v_0 = [0, 1] # Velocity vector

simulation_time = (0,5*np.pi)

dt = 0.0015  # Time step
initial_values = [r_0[0], r_0[1], v_0[0], v_0[1]]
```

In [ ]:
```python
# Forward Euler Method
def euler_method(initial_values, dt, steps):
    # y_{k+1} = y{k} + h_{k}*f'(t_{k}, y_{k})

    t0, t1 = steps
    uvals = []
    tvals = []
    u = initial_values

    def calculate_step(u):
        #   Update position based on velocity
        #   Update velocity from given equations

        norm =  np.linalg.norm(u[0:2]) ** 3
        return np.array([u[2], u[3], -u[0]/norm, -u[1]/norm])

    while t0 < t1:
        u += calculate_step(u) * dt
        uvals.append(u.copy())

        t0 += dt
        tvals.append(t0)

    return np.array(uvals), tvals

u, t = euler_method(initial_values, dt, simulation_time)
x, y, x_velocity, y_velocity = np.array_split(u, 4, axis=1)

r = np.sqrt(x**2 + y**2) # Radius
vel =  np.sqrt(x_velocity**2 + y_velocity**2)  # Velocity
energy = vel/2 - 1/r     # Energy
momentum = x*y_velocity - y*x_velocity  # Momentum
```

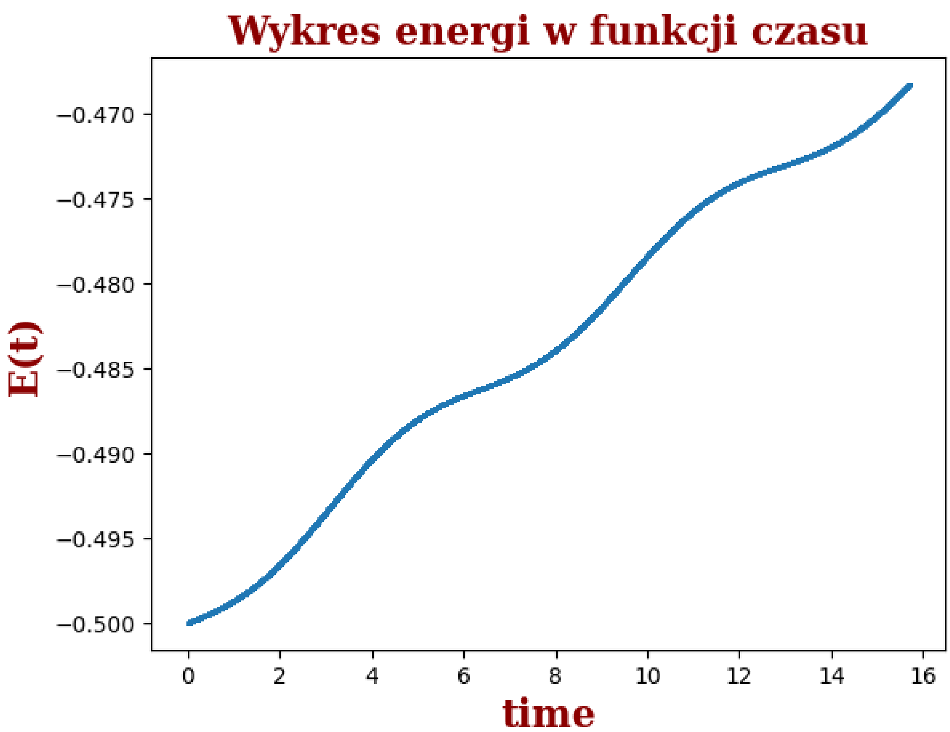
```python
font = {'family': 'serif',
        'color':  'darkred',
        'weight': 'bold',
        'size': 17,
        }

plt.scatter(x,y, s=1)
plt.title("Zależność współrzędnych y od x", fontdict=font)
plt.xlabel("x", fontdict=font)
plt.ylabel("y", fontdict=font)
plt.show()

plt.scatter(t,y, s=1)
plt.title("Wykres fazowy y w funkcji czasu", fontdict=font)
plt.xlabel("time", fontdict=font)
plt.ylabel("y(t)", fontdict=font)
plt.show()

plt.scatter(t,x, s=1)
plt.title("Wykres fazowy x w funkcji czasu", fontdict=font)
plt.xlabel("y", fontdict=font)
plt.ylabel("x(t)", fontdict=font)
plt.show()

plt.scatter(r,vel, s=1)
plt.title("Zależność prędkości od promienia", fontdict=font)
plt.xlabel("r", fontdict=font)
plt.ylabel("v", fontdict=font)
plt.show()

plt.scatter(t,energy, s=1)
plt.title("Wykres energi w funkcji czasu", fontdict=font)
plt.xlabel("time", fontdict=font)
plt.ylabel("E(t)", fontdict=font)
plt.show()

plt.scatter(t,momentum, s=1)
plt.title("Wykres pędu w funkcji czasu", fontdict=font)
plt.xlabel("time", fontdict=font)
plt.ylabel("L(t)", fontdict=font)
plt.show()
```


Wykres fazowy y w funkcji czasu
y(t)
1.0
0.5
0.0
−0.5
−1.0
0
2
4
6
8
10
12
14
16
time

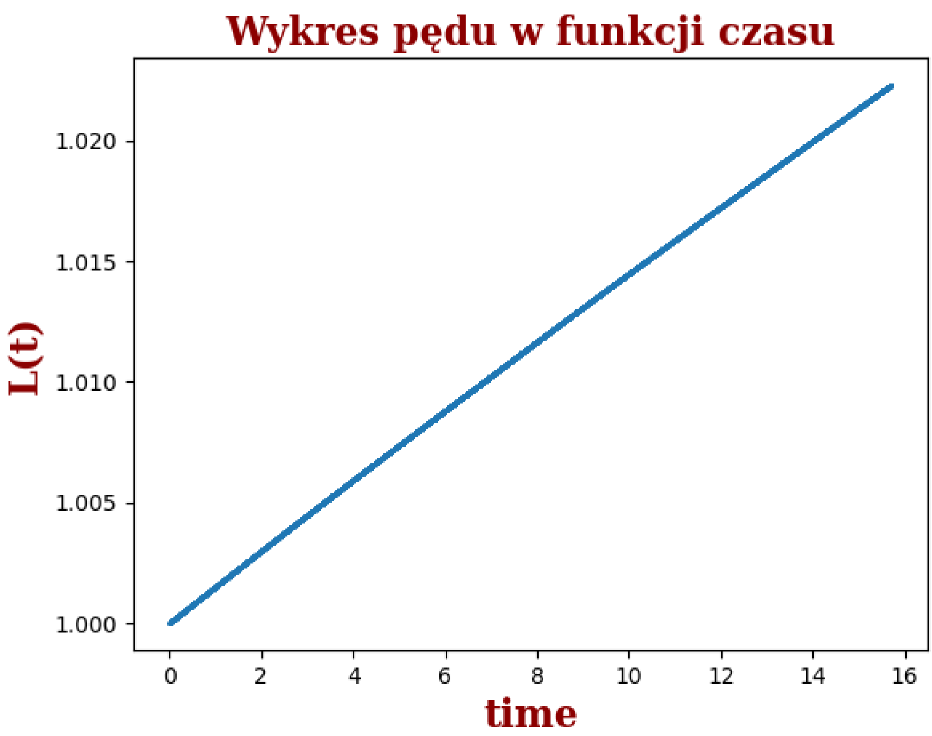

Wykres fazowy x w funkcji czasu
x(t)
1.0
0.5
0.0
−0.5
−1.0
0
2
4
6
8
10
12
14
16
y



Zależność prędkości od promienia
v
1.000
0.995
0.990
0.985
0.980
1.00
1.01
1.02
1.03
1.04
r

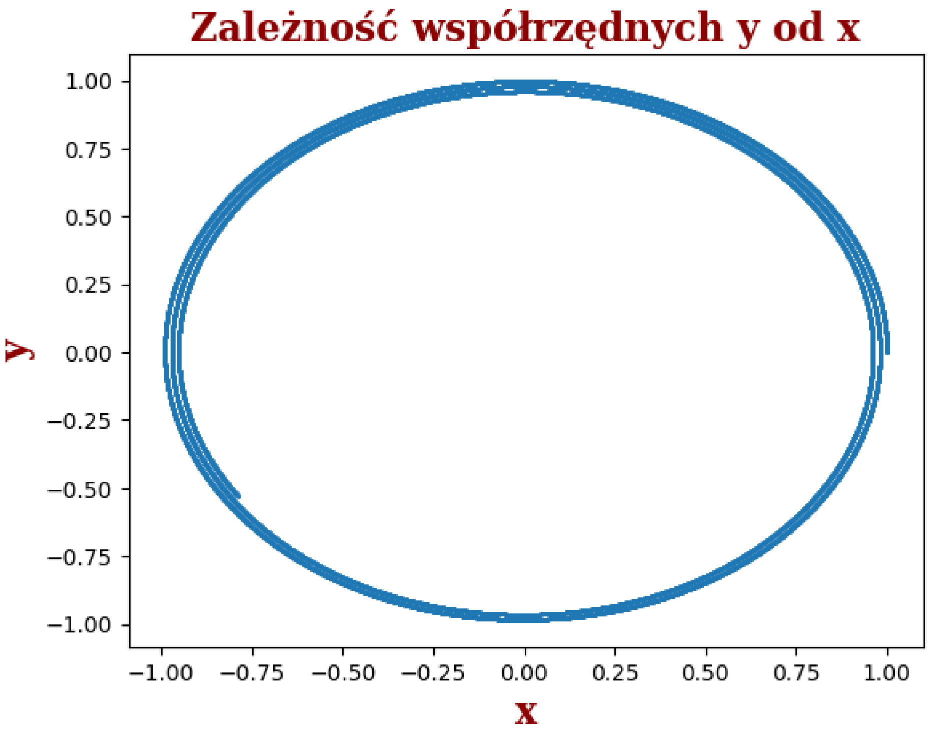


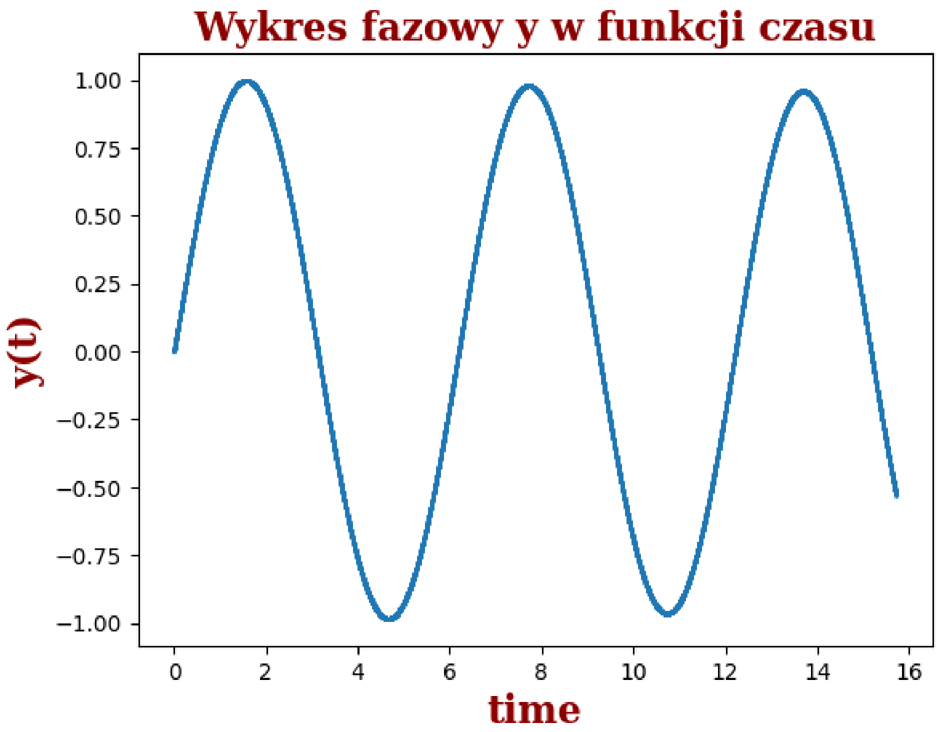
Wykres energi w funkcji czasu
E(t)
−0.470
−0.475
−0.480
−0.485
−0.490
−0.495
−0.500
0
2
4
6
8
10
12
14
16
time



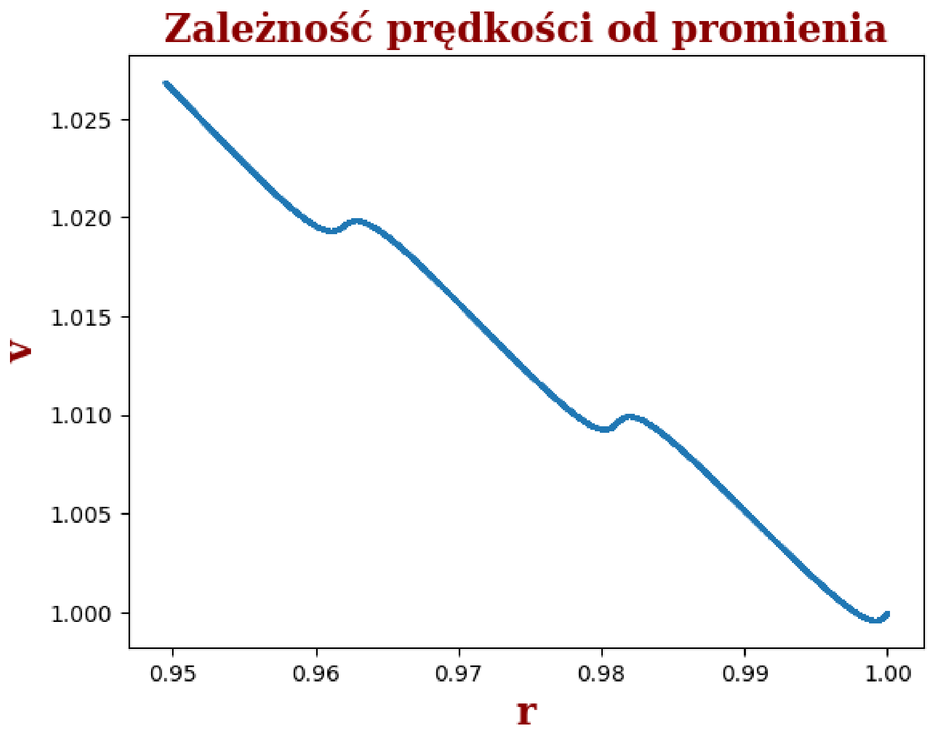
In [ ]:
```
# Backward Euler Method

def euler_implicit(initial_values, dt, steps):

    t0, t1 = steps
    uvals = []
    tvals = []
    u = initial_values

    def step_function(values,next_values,dt):
        x,y,x_velocity,y_velocity = values
        x_next, y_next, x_velocity_next, y_velocity_next = next_values

        norm = np.linalg.norm(next_values[0:2])

        return [
            x - x_next - dt * x_velocity,
            y - y_next - dt * y_velocity,
            x_velocity - x_velocity_next + dt * x/(norm**3),
            y_velocity - y_velocity_next + dt * y/(norm**3),
        ]

    while t0 < t1:
        u_tmp = u.copy()
        u = fsolve(step_function, u, args=(u_tmp, dt))
        uvals.append(u.copy())
        t0 += dt
        tvals.append(t0)

    return np.array(uvals), tvals


u, t = euler_implicit(initial_values, dt, simulation_time)
x, y, x_velocity, y_velocity = np.array_split(u, 4, axis=1)

r = np.sqrt(x**2 + y**2) # Radius
vel =  np.sqrt(x_velocity**2 + y_velocity**2)  # Velocity
energy = vel/2 - 1/r    # Energy
momentum = x*y_velocity - y*x_velocity  # Momentum

font = {'family': 'serif',
        'color':  'darkred',
        'weight': 'bold',
        'size': 17,
        }
```

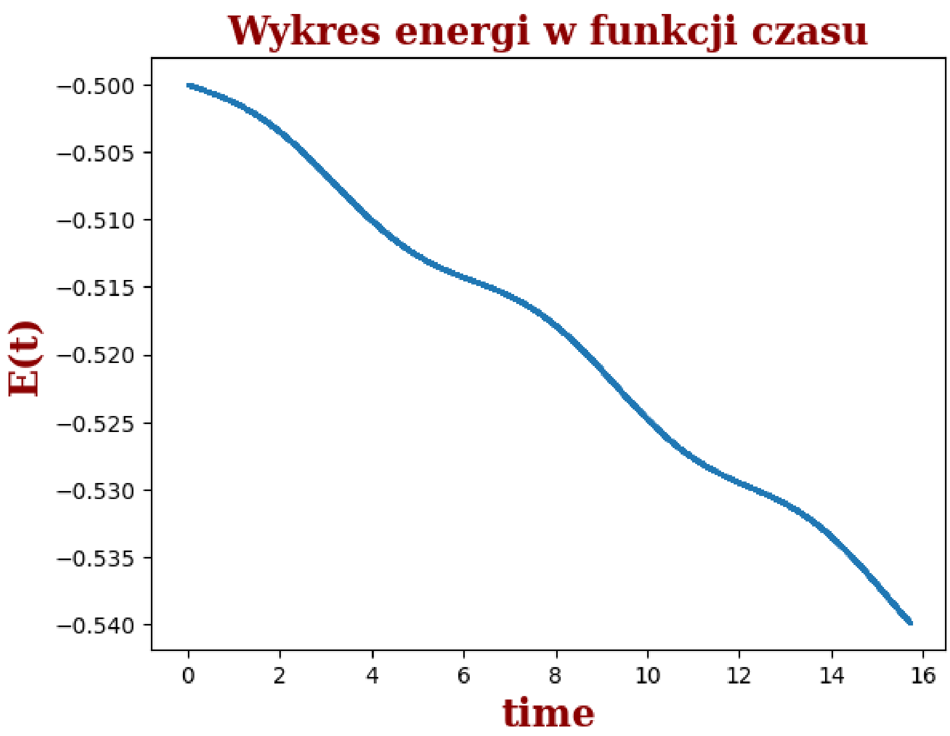
```
plt.scatter(x,y, s=1)
plt.title("Zależność współrzędnych y od x", fontdict=font)
plt.xlabel("x", fontdict=font)
plt.ylabel("y", fontdict=font)
plt.show()

plt.scatter(t,y, s=1)
plt.title("Wykres fazowy y w funkcji czasu", fontdict=font)
plt.xlabel("time", fontdict=font)
plt.ylabel("y(t)", fontdict=font)
plt.show()

plt.scatter(t,x, s=1)
plt.title("Wykres fazowy x w funkcji czasu", fontdict=font)
plt.xlabel("y", fontdict=font)
plt.ylabel("x(t)", fontdict=font)
plt.show()

plt.scatter(r,vel, s=1)
plt.title("Zależność prędkości od promienia", fontdict=font)
plt.xlabel("r", fontdict=font)
plt.ylabel("v", fontdict=font)
plt.show()

plt.scatter(t,energy, s=1)
plt.title("Wykres energi w funkcji czasu", fontdict=font)
plt.xlabel("time", fontdict=font)
plt.ylabel("E(t)", fontdict=font)
plt.show()

plt.scatter(t,momentum, s=1)
plt.title("Wykres pędu w funkcji czasu", fontdict=font)
plt.xlabel("time", fontdict=font)
plt.ylabel("L(t)", fontdict=font)
plt.show()
```

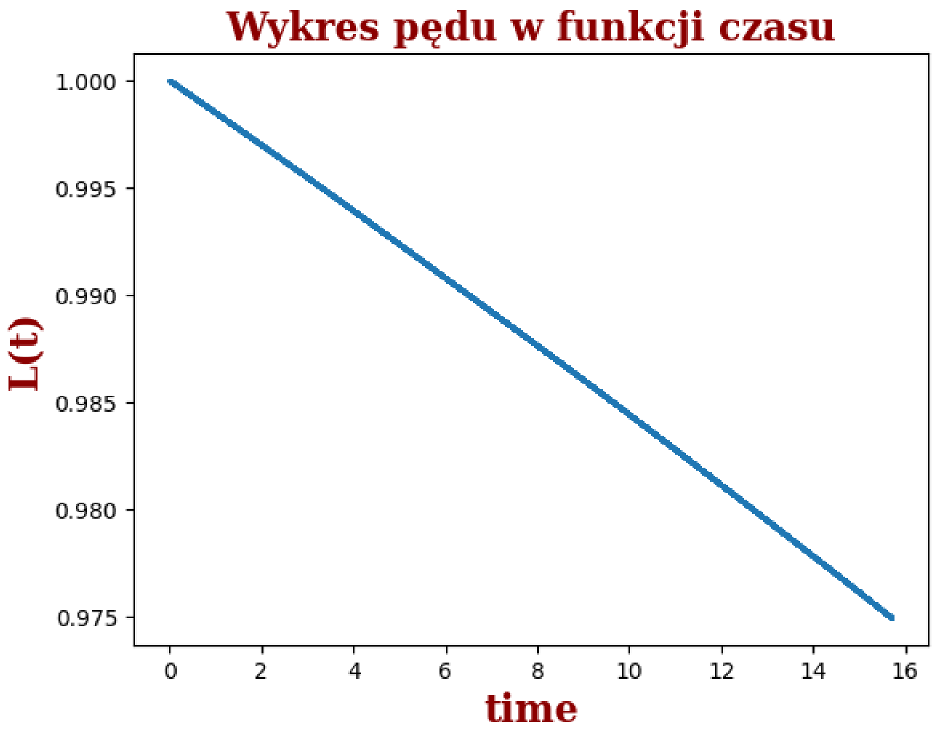



Wykres fazowy y w funkcji czasu
y(t)
time
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
2
4
6
8
10
12
14
16


Wykres fazowy x w funkcji czasu
x(t)
y
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
2
4
6
8
10
12
14
16

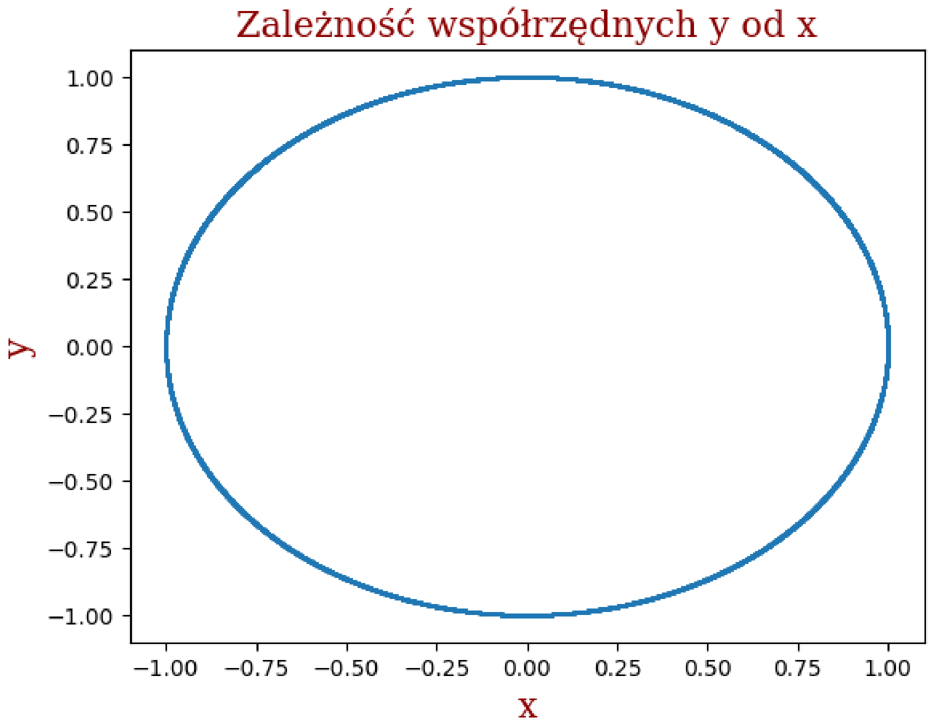

Zależność prędkości od promienia
v
1.025
1.020
1.015
1.010
1.005
1.000
0.95
0.96
0.97
0.98
0.99
1.00
r


Wykres energi w funkcji czasu
E(t)
−0.500
−0.505
−0.510
−0.515
−0.520
−0.525
−0.530
−0.535
−0.540
0
2
4
6
8
10
12
14
16
time



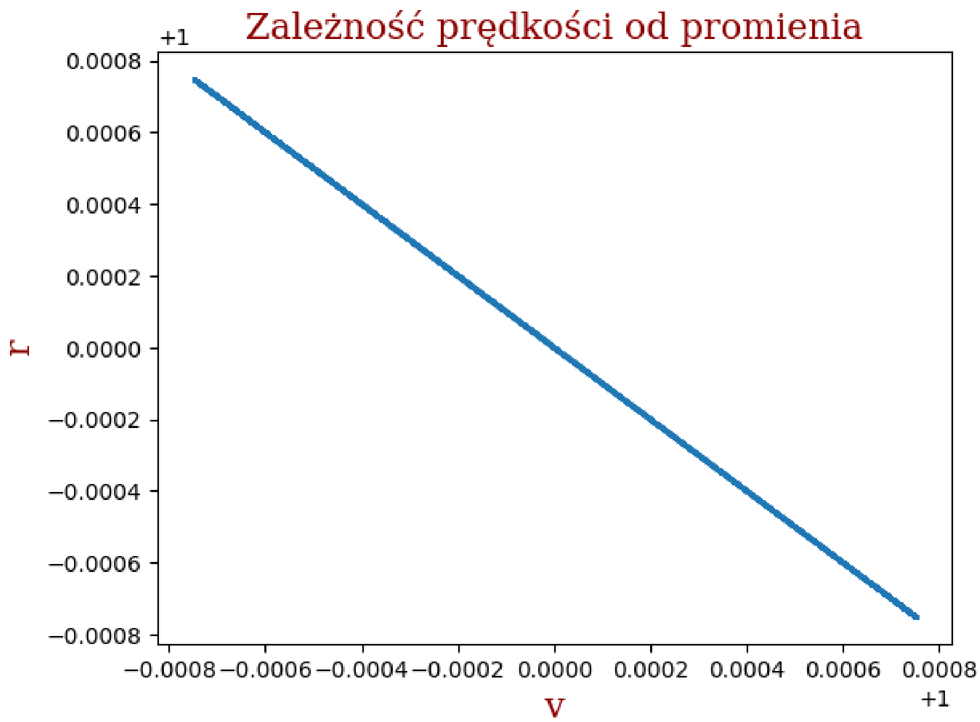
In [ ]:
```
def euler_semi_implicit(initial_values, dt, steps):

    t0, t1 = steps
    uvals = []
    tvals = []
    u = initial_values

    while t0 < t1:
        r = np.linalg.norm([u[0:2]])

        u[2] -= dt*u[0]/r**3
        u[3] -= dt*u[1]/r**3
        u[0] += dt*u[2]
        u[1] += dt*u[3]

        uvals.append(u.copy())
        t0 += dt
        tvals.append(t0)

    return np.array(uvals), tvals

u, t = euler_semi_implicit(initial_values, dt, simulation_time)
x, y, u, v = np.array_split(u, 4, axis=1)

vel = np.sqrt(u**2 + v**2)
r = np.sqrt(x**2 + y**2)

energy = vel/2 - 1/r
momentum = x*v - y*u


font = {'family': 'serif',
        'color':  'darkred',
        'weight': 'normal',
        'size': 16,
        }

plt.scatter(x,y, s=1)
plt.title("Zależność współrzędnych y od x", fontdict=font)
plt.xlabel("x", fontdict=font)
plt.ylabel("y", fontdict=font)
plt.show()

plt.scatter(t,y, s=1)
plt.title("Wykres fazowy y w funkcji czasu", fontdict=font)
plt.xlabel("t", fontdict=font)
```

```
plt.ylabel("y", fontdict=font)
plt.show()

plt.scatter(t,x, s=1)
plt.title("Wykres fazowy x w funkcji czasu", fontdict=font)
plt.xlabel("y", fontdict=font)
plt.ylabel("x", fontdict=font)
plt.show()

plt.scatter(r,vel, s=1)
plt.title("Zależność prędkości od promienia", fontdict=font)
plt.xlabel("v", fontdict=font)
plt.ylabel("r", fontdict=font)
plt.show()

plt.scatter(t,energy, s=1)
plt.title("Wykres energi w funkcji czasu", fontdict=font)
plt.xlabel("t", fontdict=font)
plt.ylabel("energy", fontdict=font)
plt.show()

plt.scatter(t,momentum, s=1)
plt.title("Wykres pędu w funkcji czasu", fontdict=font)
plt.xlabel("t", fontdict=font)
plt.ylabel("moment", fontdict=font)
plt.show()
```

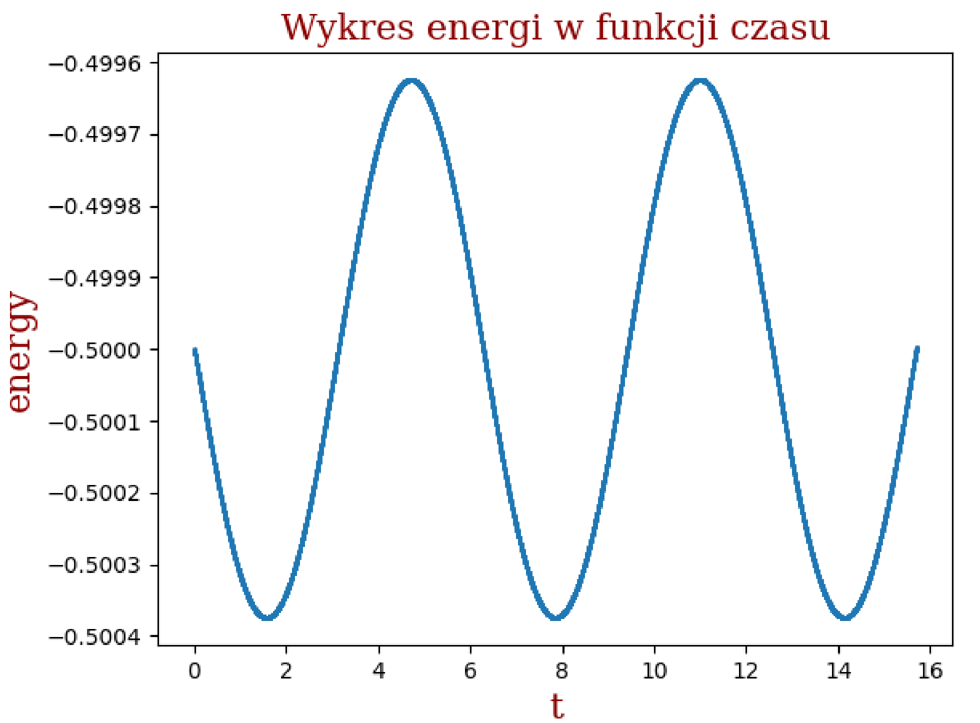



Wykres fazowy y w funkcji czasu
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
y
0
2
4
6
8
10
12
14
16
t

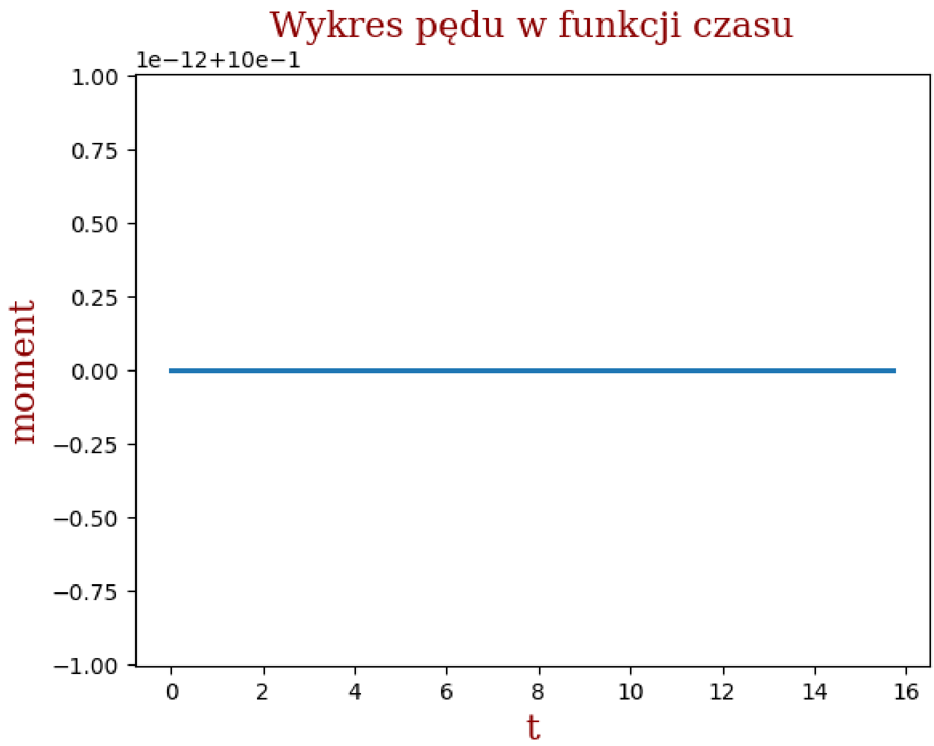

Wykres fazowy x w funkcji czasu
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
x
0
2
4
6
8
10
12
14
16
y



Zależność prędkości od promienia
+1
0.0008
0.0006
0.0004
0.0002
0.0000
−0.0002
−0.0004
−0.0006
−0.0008
r
−0.0008
−0.0006
−0.0004
−0.0002
0.0000
0.0002
0.0004
0.0006
0.0008
+1
v



Wykres energi w funkcji czasu
−0.4996
−0.4997
−0.4998
−0.4999
−0.5000
−0.5001
−0.5002
−0.5003
−0.5004
energy
0
2
4
6
8
10
12
14
16
t

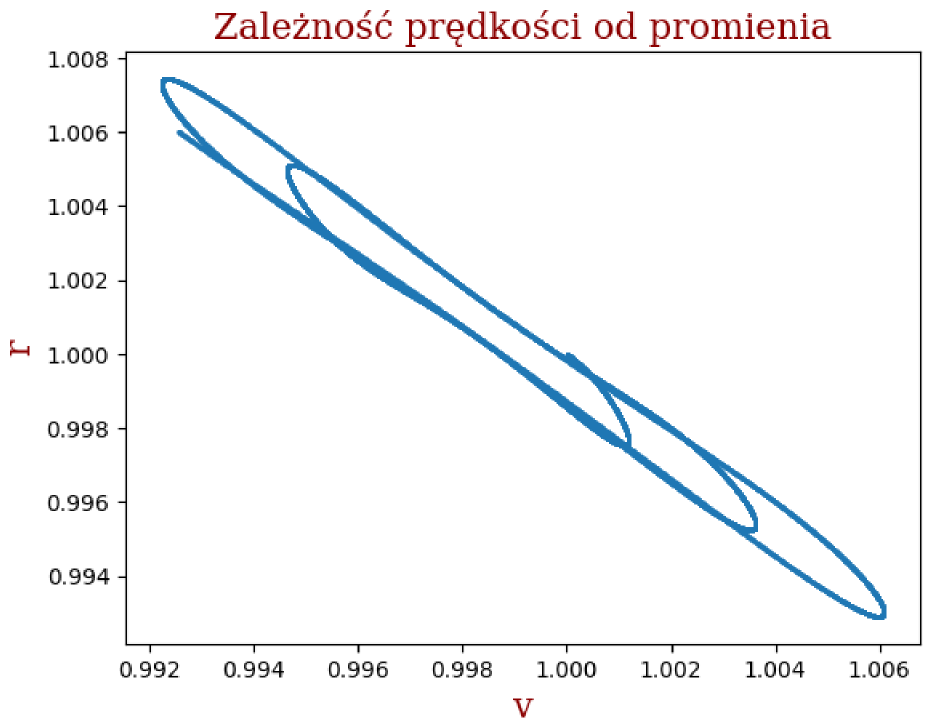


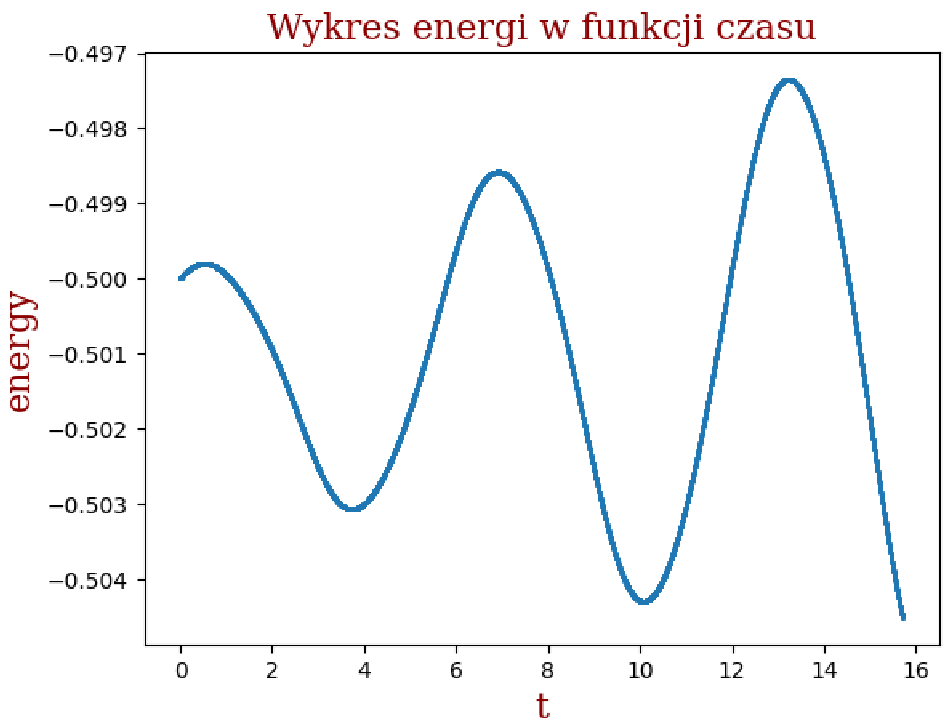
In [ ]:
```
def runge(u_0, dt, steps):
    def helper(u, wsp, k):
        r = np.linalg.norm(u[0:2])

        return np.array([
            u[2] + wsp*k[2],
            u[3] + wsp*k[3],
            -(u[0] + wsp*k[0]) / (r + wsp*k[0])**3,
            -(u[1] + wsp*k[1]) / (r + wsp*k[1])**3,
        ])

    def calculate_step(u, dt):
        k1 = helper(u, 0, [1,1,1,1])
        k2 = helper(u, dt*0.5, k1)
        k3 = helper(u, dt*0.5, k2)
        k4 = helper(u, dt, k3)

        return np.multiply(dt, np.divide((k1 + 2*k2 + 2*k3 + k4), 6))


    t0, t1 = steps
    uvals = []
    tvals = []

    u = u_0

    while t0 < t1:
        u += calculate_step(u,dt)
        uvals.append(u.copy())
        t0 += dt
        tvals.append(t0)

    return np.array(uvals), tvals

u, t = runge(initial_values, dt, simulation_time)
x, y, u, v = np.array_split(u, 4, axis=1)

vel = np.sqrt(u**2 + v**2)
r = np.sqrt(x**2 + y**2)

energy = vel/2 - 1/r
momentum = x*v - y*u


font = {'family': 'serif',
```

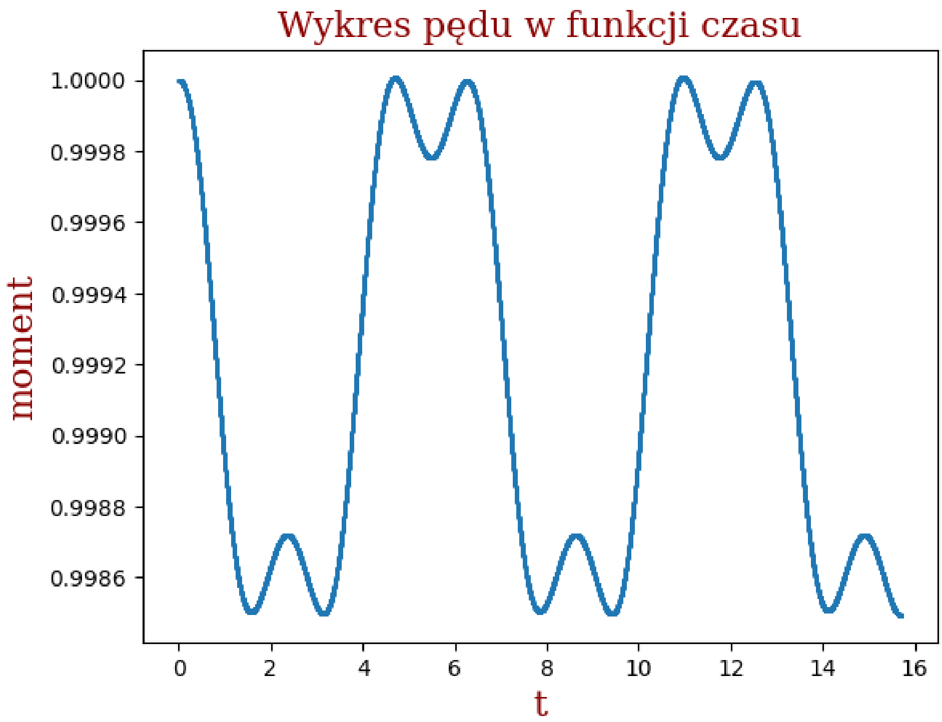
```
        'color':  'darkred',
        'weight': 'normal',
        'size': 16,
        }

plt.scatter(x,y, s=1)
plt.title("Zależność współrzędnych y od x", fontdict=font)
plt.xlabel("x", fontdict=font)
plt.ylabel("y", fontdict=font)
plt.show()

plt.scatter(t,y, s=1)
plt.title("Wykres fazowy y w funkcji czasu", fontdict=font)
plt.xlabel("t", fontdict=font)
plt.ylabel("y", fontdict=font)
plt.show()

plt.scatter(t,x, s=1)
plt.title("Wykres fazowy x w funkcji czasu", fontdict=font)
plt.xlabel("y", fontdict=font)
plt.ylabel("x", fontdict=font)
plt.show()
d = {}

for i in zip(r, vel):
    d[i[0][0]] = i[1][0]

od = collections.OrderedDict(sorted(d.items()))

plt.scatter(od.keys(),od.values(), s = 1)
plt.title("Zależność prędkości od promienia", fontdict=font)
plt.xlabel("v", fontdict=font)
plt.ylabel("r", fontdict=font)
plt.show()

plt.scatter(t,energy, s=1)
plt.title("Wykres energi w funkcji czasu", fontdict=font)
plt.xlabel("t", fontdict=font)
plt.ylabel("energy", fontdict=font)
plt.show()

plt.scatter(t,momentum, s=1)
plt.title("Wykres pędu w funkcji czasu", fontdict=font)
plt.xlabel("t", fontdict=font)
plt.ylabel("moment", fontdict=font)
plt.show()
```


Wykres fazowy y w funkcji czasu
y
t
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
2
4
6
8
10
12
14
16


Wykres fazowy x w funkcji czasu
x
y
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
2
4
6
8
10
12
14
16


Zależność prędkości od promienia
r
1.008
1.006
1.004
1.002
1.000
0.998
0.996
0.994
0.992
0.994
0.996
0.998
1.000
1.002
1.004
1.006
v


Wykres energi w funkcji czasu
energy
−0.497
−0.498
−0.499
−0.500
−0.501
−0.502
−0.503
−0.504
0
2
4
6
8
10
12
14
16
t



Bibliografia

- Katarzyna Rycerz: Wykład z przedmiotu Metody Obliczeniowe w Nauce i Technice
- Materiały do zajęć
- Julian Janus: Stabilność rozwiązań równań różniczkowych zwyczajnych - open.agh.edu.pl