SPRAWOZDANIE PAWEŁ MOZGOWIEC LAB1

ZAD.1

Exercises

1. Consider the constant-effort harvesting model

$$\frac{dx}{dt} = rx\left(1 - \frac{x}{K}\right) - Ex. \tag{1}$$

Solve (1) numerically by the Euler method, $\frac{dx}{dt} = f(t, x)$, $x(t_0 + \Delta t) = x(t_0) + \Delta t f(t_0, y_0)$, i.e.

$$x_n = x_{n-1} + \Delta t\left(rx_{n-1}\left(1 - \frac{x_{n-1}}{K}\right) - Ex_{n-1}\right). \tag{2}$$

Compare obtained numeric solution with analytic solution given by

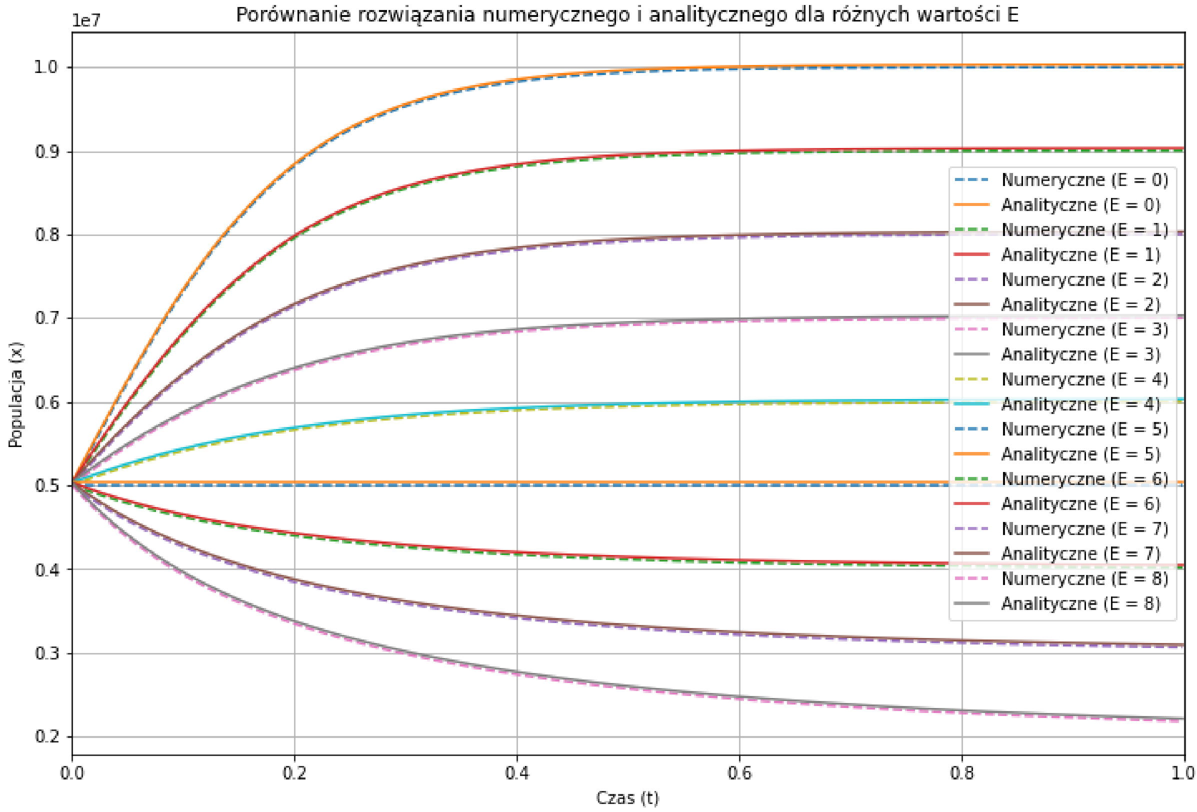
$$x(t) = \frac{Kx_0(r-E)}{rx_0 + (r(K-x_0) - EK)e^{t(E-r)}} \tag{3}$$

Note that for $E = 0$ (no harvesting) model (1) reduces to the logistic population model $\frac{dx}{dt} = rx\left(1 - \frac{x}{K}\right)$ with the solution $x(t) = \frac{Kx_0}{x_0 + (K-x_0)e^{-rt}}$.

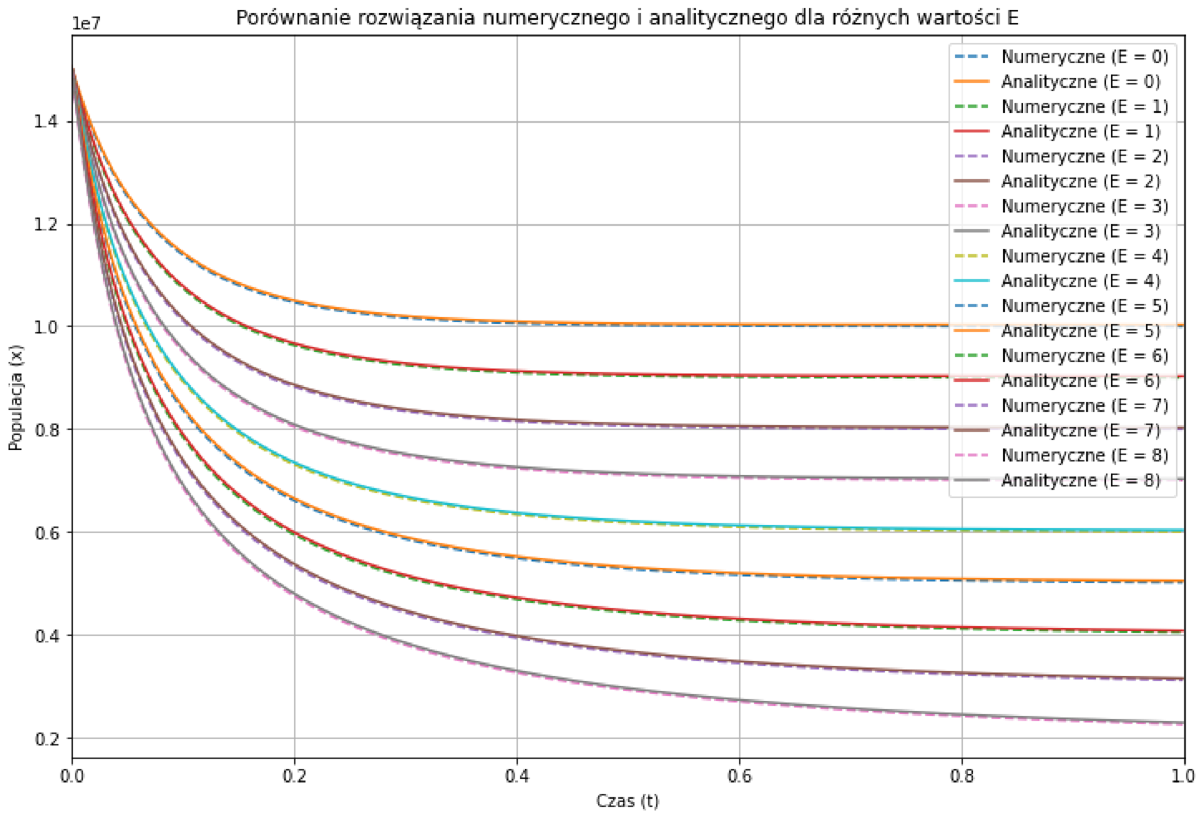
Plot fixed point asymptotes

$$\frac{dx}{dt} = rx\left(1 - \frac{x}{K}\right) - Ex = 0, \text{ i.e. } x = K\left(1 - \frac{E}{r}\right), E \leq r \tag{4}$$

Compute and plot bifurcation diagram of numeric solution (2).



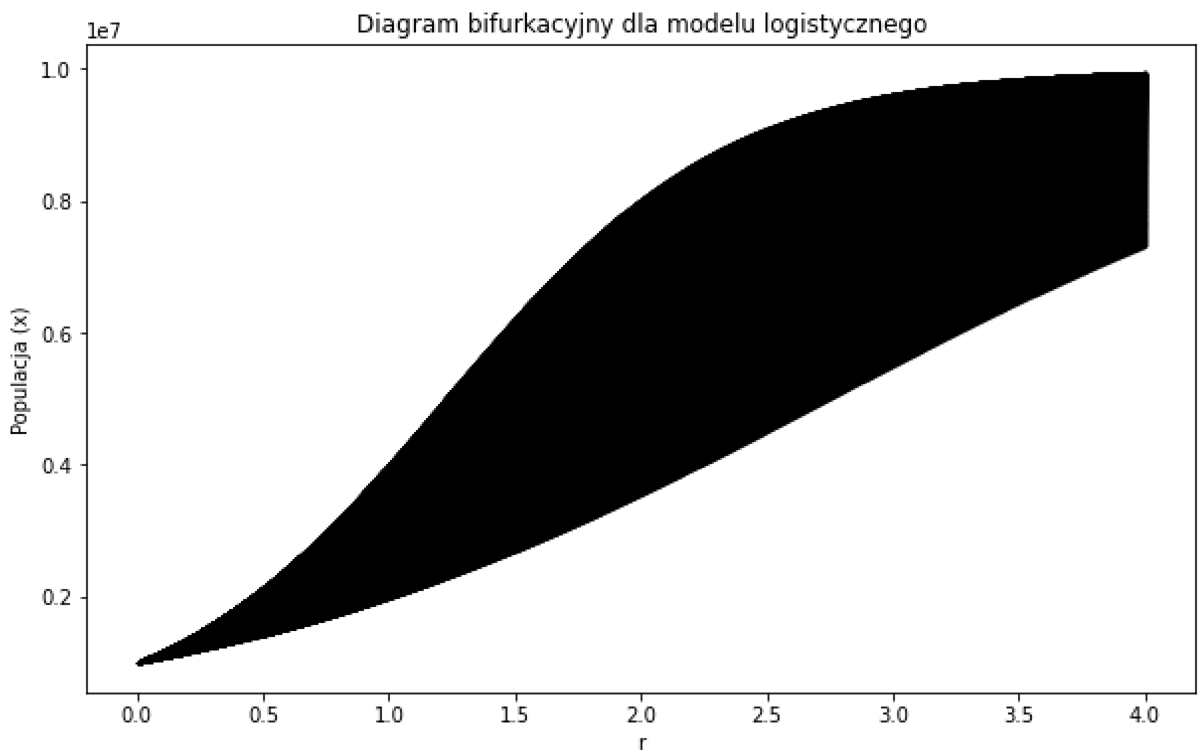
Porównanie rozwiązania numerycznego i analitycznego dla różnych wartości E
1e7
Populacja (x)
Czas (t)
Numeryczne (E = 0)
Analityczne (E = 0)
Numeryczne (E = 1)
Analityczne (E = 1)
Numeryczne (E = 2)
Analityczne (E = 2)
Numeryczne (E = 3)
Analityczne (E = 3)
Numeryczne (E = 4)
Analityczne (E = 4)
Numeryczne (E = 5)
Analityczne (E = 5)
Numeryczne (E = 6)
Analityczne (E = 6)
Numeryczne (E = 7)
Analityczne (E = 7)
Numeryczne (E = 8)
Analityczne (E = 8)



Porównanie rozwiązania numerycznego i analitycznego dla różnych wartości E
1e7
Populacja (x)
Czas (t)
Numeryczne (E = 0)
Analityczne (E = 0)
Numeryczne (E = 1)
Analityczne (E = 1)
Numeryczne (E = 2)
Analityczne (E = 2)
Numeryczne (E = 3)
Analityczne (E = 3)
Numeryczne (E = 4)
Analityczne (E = 4)
Numeryczne (E = 5)
Analityczne (E = 5)
Numeryczne (E = 6)
Analityczne (E = 6)
Numeryczne (E = 7)
Analityczne (E = 7)
Numeryczne (E = 8)
Analityczne (E = 8)

Przesunąłem wartości analityczne o stałe 30000 jednostek aby widzieć je na wykresie (dokładność oby metod jest tak duża że ich wykresy powielają się na siebie)

```python
# Funkcja do rozwiązania numerycznego metodą Eulera
def euler_method(r, K, E, x0, t_max, dt):
    t_values = np.arange(0, t_max, dt)
    x_values = np.zeros(len(t_values))
    x_values[0] = x0

    for i in range(1, len(t_values)):
        dx_dt = r * x_values[i-1] * (1 - x_values[i-1] / K) - E * x_values[i-1]
        x_values[i] = x_values[i-1] + dx_dt * dt

        # Warunki ograniczające:
        # Unikamy ujemnych wartości populacji
        if x_values[i] < 0:
            x_values[i] = 0
        # Jeśli populacja przekroczyła pojemność środowiska, ustawiamy ją na K
        elif x_values[i] > K:
            x_values[i] = K

    return t_values, x_values

# Funkcja do rozwiązania analitycznego
def analytic_solution(r, K, E, x0, t_values):
    return (K * x0 * (r - E)) / (r * x0 + (r * (K - x0) - E * K) * np.exp(t_values * (E - r)))

# Rysowanie wykresów dla różnych wartości E
plt.figure(figsize=(12, 8))

for E in E_values:
    t_values, x_values_num = euler_method(r, K, E, x0, t_max, dt)
    x_values_analytic = analytic_solution(r, K, E, x0, t_values)

    # Porównanie wykresów: numeryczne i analityczne
    plt.plot(t_values, x_values_num, label=f'Numeryczne (E = {E})', linestyle='--')
    plt.plot(t_values, x_values_analytic + 30000, label=f'Analityczne (E = {E})', linestyle='-')

plt.title('Porównanie rozwiązania numerycznego i analitycznego dla różnych wartości E')
plt.xlabel('Czas (t)')
plt.ylabel('Populacja (x)')
plt.legend()
plt.grid(True)
plt.xlim([0, 1])
plt.show()
```

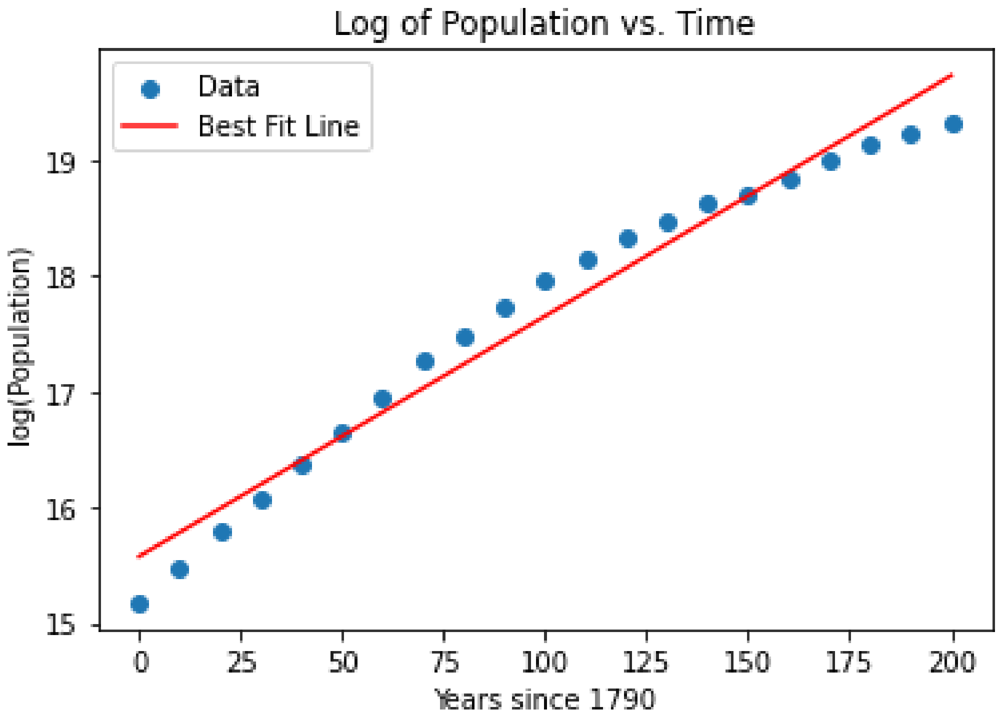



```python
import numpy as np
import matplotlib.pyplot as plt

# Parametry modelu
K = 1.0e+07  # Pojemność środowiska
E = 0  # Brak zbiorów
dt = 0.001  # Krok czasowy
t_max = 1  # Czas symulacji

def bifurcation_diagram(seed, n_skip, n_iter, step=0.001, r_min=0, r_max=20):
    print(f"Starting with x0 = {seed}, skipping first {n_skip} iterations, then plotting {n_iter} iterations.")

    R = []  # Oś X - wartości r
    X = []  # Oś Y - wartości populacji

    r_range = np.linspace(r_min, r_max, int((r_max - r_min) / step))

    for r in r_range:
        x = np.zeros(n_iter + n_skip + 1)
        x[0] = seed

        for i in range(1, n_iter + n_skip + 1):
            dx_dt = r * x[i-1] * (1 - x[i-1] / K) - E * x[i-1]
            x[i] = x[i-1] + dx_dt * dt

            if i >= n_skip:
                R.append(r)
                X.append(x[i])

    # Rysowanie wykresu
    plt.figure(figsize=(10, 6))
    plt.scatter(R, X, s=0.1, color="black", alpha=0.5)
    plt.xlabel("r")
    plt.ylabel("Populacja (x)")
    plt.title("Diagram bifurkacyjny dla modelu logistycznego")
    plt.show()

# Wywołanie funkcji
bifurcation_diagram(1.0e+06, 800, int(t_max / dt), step=0.001, r_min=0, r_max=4)
```

Zad2

```
Liczba lat potrzebnych do osiągnięcia 1000000: 316
```

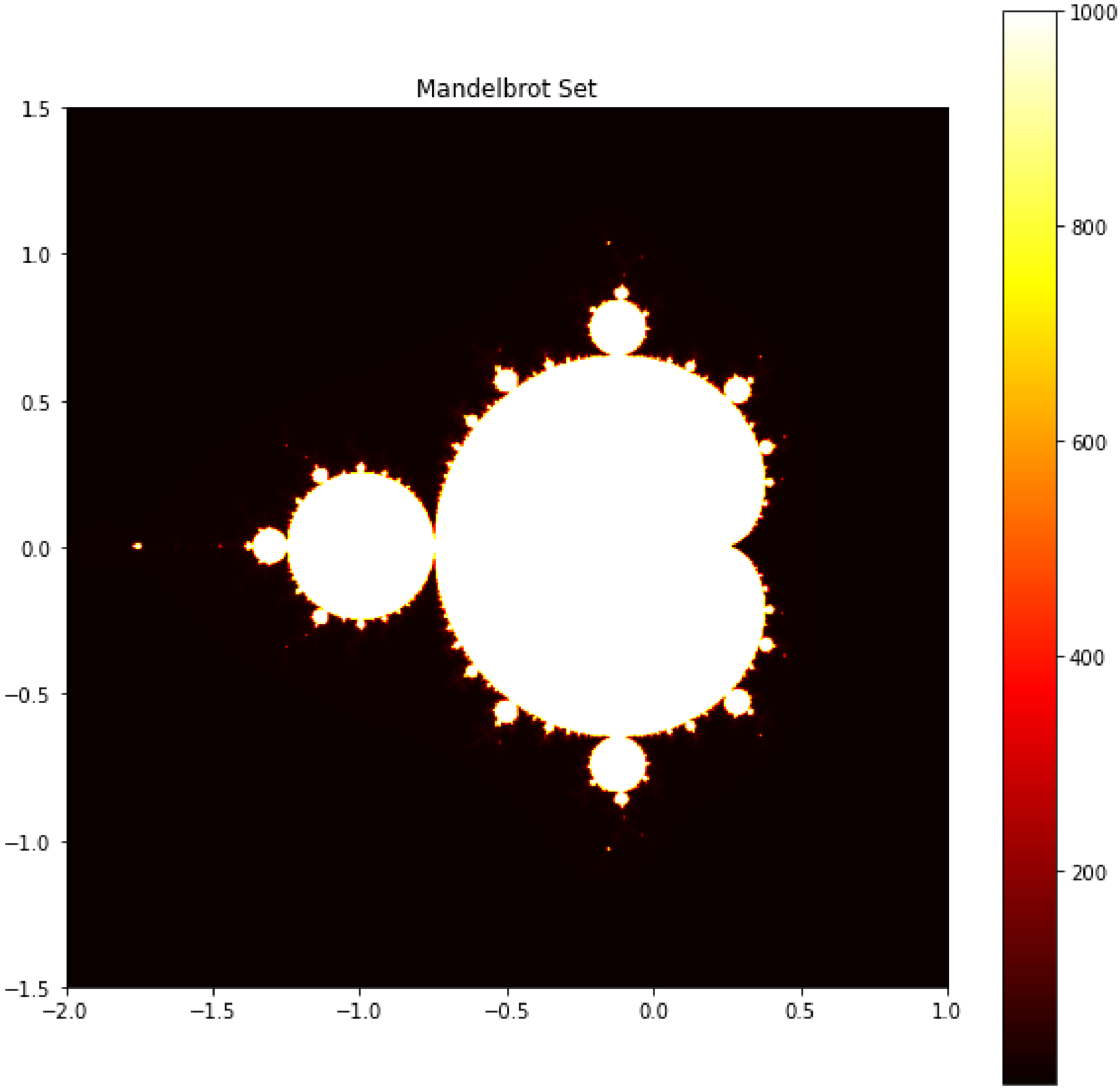
```python
target = 1_000_000  # Cel oszczędności
annual_deposit = 0.01  # Roczna wpłata
interest_rate = 0.05  # Oprocentowanie (5%)

years = 0
balance = 0

while balance < target:
    balance += annual_deposit  # Dodanie rocznej wpłaty
    balance *= (1 + interest_rate)  # Kapitalizacja odsetek
    years += 1

print(f"Liczba lat potrzebnych do osiągnięcia {target}: {years}")
```



Zad3

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import linregress

# Given data
years = np.array([1790, 1800, 1810, 1820, 1830, 1840, 1850, 1860, 1870, 1880, 1890,
                  1900, 1910, 1920, 1930, 1940, 1950, 1960, 1970, 1980, 1990])
population = np.array([3.9e6, 5.3e6, 7.2e6, 9.6e6, 12.9e6, 17.1e6, 23.1e6, 31.4e6, 38.6e6,
                        50.2e6, 62.9e6, 76.0e6, 92.0e6, 105.7e6, 122.8e6, 131.7e6, 150.7e6,
                        179.0e6, 205.0e6, 226.5e6, 248.7e6])

# Convert to log scale
y_log = np.log(population)
x_years = years - 1790  # Shift years to start from 0

# Perform linear regression
slope, intercept, r_value, p_value, std_err = linregress(x_years, y_log)

# Extract estimated growth rate r
r_estimated = slope
x0_estimated = np.exp(intercept)

# Print results
print(f"Estimated growth rate (r): {r_estimated:.6f}")
print(f"Estimated initial population (x0): {x0_estimated:.2f}")

# Plot data and regression line
plt.scatter(x_years, y_log, label='Data')
plt.plot(x_years, intercept + slope * x_years, color='red', label='Best Fit Line')
plt.xlabel("Years since 1790")
plt.ylabel("Log(Population)")
plt.title("Log of Population vs. Time")
plt.legend()
plt.show()
```

Zad4


Mandelbrot Set
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1000
800
600
400
200

```python
import numpy as np
import matplotlib.pyplot as plt

def mandelbrot(c, max_iter):
    z = 0
    for n in range(max_iter):
        if abs(z) > 2:
            return n
        z = z*z + c
    return max_iter

def compute_mandelbrot(xmin, xmax, ymin, ymax, width, height, max_iter):
    x = np.linspace(xmin, xmax, width)
    y = np.linspace(ymin, ymax, height)
    fractal = np.zeros((height, width))

    for i in range(height):
        for j in range(width):
            fractal[i, j] = mandelbrot(complex(x[j], y[i]), max_iter)

    return fractal

def plot_mandelbrot(fractal):
    plt.figure(figsize=(10, 10))
    plt.imshow(fractal, extent=(-2, 1, -1.5, 1.5), cmap='hot', interpolation='bilinear')
    plt.colorbar()
    plt.title("Mandelbrot Set")
    plt.show()

# Ustawienia zakresu i rozdzielczości
xmin, xmax, ymin, ymax = -2, 1, -1.5, 1.5
width, height = 800, 600
max_iter = 1000

# Obliczanie i rysowanie fraktala Mandelbrota
fractal = compute_mandelbrot(xmin, xmax, ymin, ymax, width, height, max_iter)
plot_mandelbrot(fractal)
```