Aleksander Szpatowski, 04.02.2021r.

Numer indeksu: 314211, Prowadzący: dr Tomasz Pawłowski

II rok, ISSP

Metody numeryczne 1

Raport – zadanie 8

1. Treść zadania

Rozwiąż równanie różniczkowe w przedziale t=[0,1]. Wyznacz położenie ekstremum tego rozwiązania.

$$y'' + 11y' + 24y = 0, y(0) = 0, y'(0) = -7$$

2. Analiza problemu

Jest to liniowe równanie różniczkowe zwyczajne drugiego rzędu. Żeby je rozwiązać zastosuje funkcję **scipy.integrate.solve_ivp()**, ale żeby ową funkcję zastosować muszę najpierw przekształcić równanie do innej postaci. Do wyznaczenia ekstremum użyje flagi **„events”** w **solve_ivp()**, aby znaleźć miejsce zerowe y’(t), a tym samym ekstremum funkcji y(t).

3. Rozwiązanie problemu

Funkcja solve_ivp całkuje numerycznie układ równań różniczkowych zwyczajnych dla których podano wartość początkową:

$$\frac{dy}{dt} = f(t, y)$$

$$y(t_0) = y_0$$

Więc musimy nasze równanie przekształcić do takiej formy.

Możemy to zrobić za pomocą prostego podstawienia zmiennych.

Definiujemy:

$$x_2(t) = y'(t)$$

$$x_1(t) = y(t)$$

dla równania w postaci:

$$y'' = Ay' + By$$

Z tego wynika, że:

$$x_1' = y' = x_2$$

$$x_2' = y'' = Ax_1 + Bx_2$$

Upraszczając wychodzimy z takim ogólnym układem równań:

$$x_1' = x_2$$

$$x_2' = Ax_1 + Bx_2$$

Podstawiając wartości podane w pierwotnym równaniu, otrzymujemy:

$$x_1' = x_2, \qquad x_1(0) = 0,$$

$$x_2' = -24x_1 - 11x_2, \qquad x_2(0) = -7.$$

Taka postać możemy już użyć w solve_ivp().

I żeby to zrobić definiujemy funkcje:

```
f = lambda t, Y: (Y[1], - 24 * Y[0] - 11 * Y[1])
```

Tworzymy następnie macierz y0:

```
y0 = [0, -7]  # y(0) = 0, y'(0) = -7
```

Tworzymy przedział t=[0,1] i używamy solve_ivp()

```
# t = [0,1]
tp = 0
tk = 1
t = np.linspace(tp, tk, 51)  # 21 do wyników, 51 do wykresu
y = solve_ivp(f, [tp, tk], y0, t_eval=t, events=ekstremum)  # Domyślna metoda to RK45
```

t_eval – to ilość punktów w zakresie, dla których ma znaleźć rozwiązanie.

events – określenie zdarzenia, które ma być śledzone (dokładne wytłumaczenie działania w dalszej części raportu).

W zmiennej y, są teraz przechowywane wyniki.

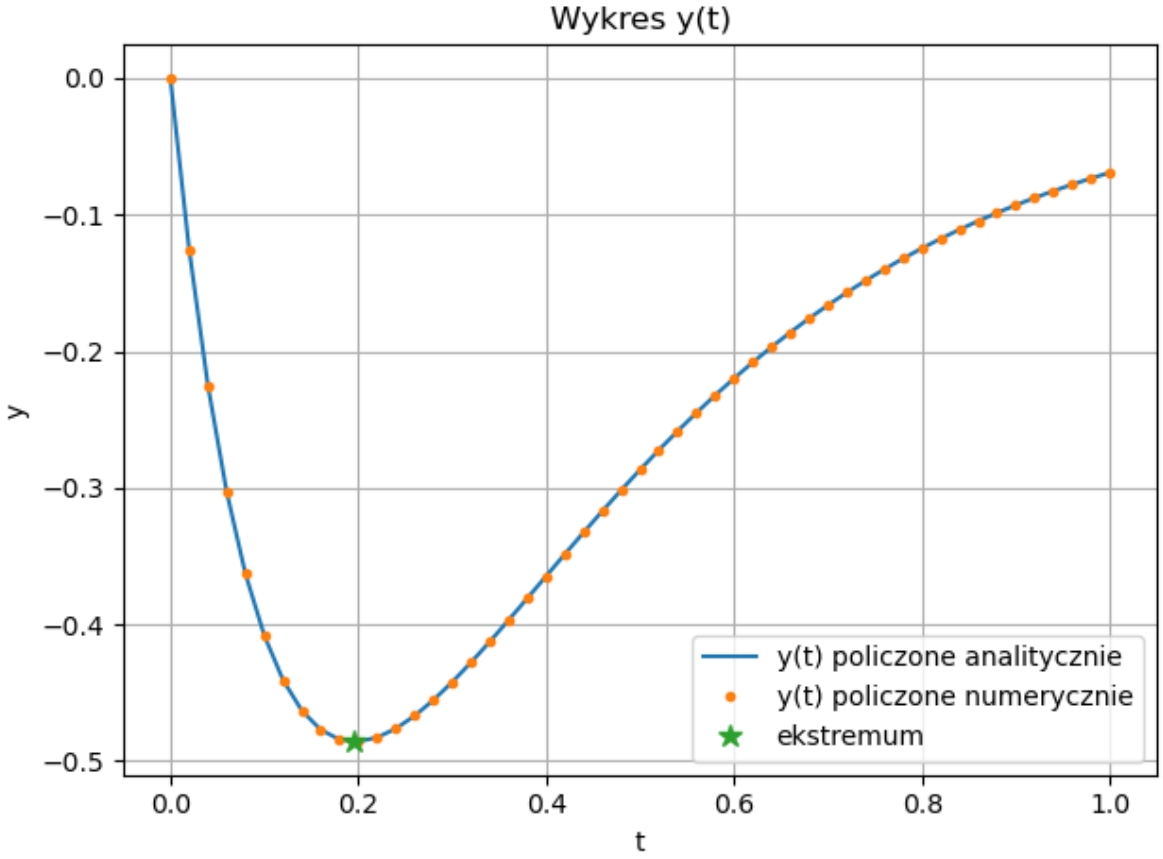
y.t – punkty t w przedziale [0,1] z krokiem 0,05 (lub 0,02, jest to zależne od ilości punktów zadanej w np.linspace())

y.y[0] – wartości y(t)

y.y[1] – wartości y’(t)

Wyniki:

t	y(t)
0.00	0.0000000000000000000
0.05	-0.26659285266095616995
0.10	-0.40808522259314500280
0.15	-0.47105096260793166696
0.20	-0.48565494444085005599
0.25	-0.47183248344330164237
0.30	-0.44215494794994969485
0.35	-0.40474732367742422001
0.40	-0.36459758281300530536
0.45	-0.32466943693658700809
0.50	-0.28672480341650230518
0.55	-0.25168286622605962055
0.60	-0.21987245997409260445
0.65	-0.19146017111990459858
0.70	-0.16625967322487228417
0.75	-0.14406839893487941029
0.80	-0.12467555115446274372
0.85	-0.10775460866440372820
0.90	-0.09302873835475526654
0.95	-0.08027024777752229945
1.00	-0.06922498021574501059



Pokusiłem się o obliczenie równania jeszcze w sposób analityczny dla sprawdzenia poprawności wyników (obliczenia na https://github.com/aszpatowski/metody_numeryczne1_zadanie_koncowe/blob/main/rozwiazanie_analitycznie.png)

```
y_analitic = lambda t: 1.4 * np.exp(-8 * t) - 1.4 * np.exp(-3 * t)
```

Drugą rzeczą, jaką należało wyliczyć było ekstremum już obliczonej funkcji y(t), w tym celu napisałem prostą funkcję, którą stosuje we fladze **events** w **solve_ivp()**.

```
ekstremum = lambda t, Y: Y[1]  # Znajdowanie miejsca zerowego y'(t)
```

Wskazuje to solve_ivp() „zdarzenie do śledzenia”. Tym zdarzeniem jest zerowanie się y’(t), funkcja znajdzie dokładną wartość t dla y’(t) = 0 za pomocą algorytmu do znajdowania pierwiastków.

Możemy się do tych wyników dobrać za pomocą:

y.t_events,

y.y_events (tutaj są zwracane dwie wartości y(t) oraz y’(t) dla znalezionego t)

```
# liczenie ekstremum
t_ekstr = y.t_events[0][0]
y_ekstr = y.y_events[0][0][0]
```

Wyniki:

```
Położenie ekstremum
y(0.19619037613537366) = -0.48572991396205717
Ekstremum policzone analitycznie
y(0.19616585060234523) = -0.48576566388938874
```

Znowu pokusiłem się o sprawdzenie tego w sposób analityczny (obliczenia na https://github.com/aszpatowski/metody_numeryczne1_zadanie_koncowe/blob/main/ekstremum_analitycznie.png)

```
t_extreme_analitic = -np.log(3 / 8) / 5  # Obliczenia na githubie
```

Skrypt:

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

f = lambda t, Y: (Y[1], - 24 * Y[0] - 11 * Y[1])  # Przekształcone równanie

ekstremum = lambda t, Y: Y[1]  # Znajdowanie miejsca zerowego y'(t)

y_analitic = lambda t: 1.4 * np.exp(-8 * t) - 1.4 * np.exp(-3 * t)  # Obliczenia na githubie

t_extreme_analitic = -np.log(3 / 8) / 5  # Obliczenia na githubie

y0 = [0, -7]  # y(0) = 0, y'(0) = -7
# t = [0,1]
tp = 0
tk = 1
t = np.linspace(tp, tk, 51)  # 21 do wyników, 51 do wykresu
y = solve_ivp(f, [tp, tk], y0, t_eval=t, events=ekstremum)  # Domyślna metoda to RK45
y_analitic_points = [y_analitic(t_point) for t_point in t]

print(f"{'t':10s}{'y(t)':22s}")
for i in range(len(y.t)):
    print(f"{y.t[i]:1.2f} | {y.y[0][i]:6.20f}")

print("\n")

plt.plot(t, y_analitic_points)
plt.plot(y.t, y.y[0], '.')

# liczenie ekstremum
t_ekstr = y.t_events[0][0]
y_ekstr = y.y_events[0][0][0]
print("Położenie ekstremum")
print(f"y({t_ekstr}) = {y_ekstr}")
print("Ekstremum policzone analitycznie")
print(f"y({t_extreme_analitic}) = {y_analitic(t_extreme_analitic)}")
plt.plot(t_ekstr, y_ekstr, "*", markersize=9)
plt.legend(["y(t) policzone analitycznie", "y(t) policzone numerycznie", "ekstremum"])
plt.title("Wykres y(t)")
plt.xlabel("t")
plt.ylabel("y")
plt.grid()
plt.show()
```

4. Podsumowanie

Wyniki uzyskane za pomocą **solve_ivp()** z domyślną metodą RK45 (metoda Rungego-Kutty rzędu 5) są bardzo dokładne i nie różnią się znacząco od analitycznego rozwiązania (różnice w wynikach wahają się w granicach 1e-07 a 1e-05). Materiały oraz skrypt na: https://github.com/aszpatowski/metody_numeryczne1_zadanie_koncowe.