Artur Gęsiarz,
Czerwiec 14, 2024

Laboratorium nr 9
MOwNiT – Równania różniczkowe zwyczajne

1. Treść zadania
   1.1. Zadanie pierwsze

   Przedstaw każde z ponizszych równan rózniczkowych zwyczajnych jako równoważny układ równań pierwszego rzędu (ang. first-order system of ODEs):

   (a) równanie Van der Pol'a:

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

   (b) równanie Blasiusa:

$$y''' = -yy''$$

   (c) II zasada dynamiki Newtona dla problemu dwóch ciał:

$$y_1'' = -\frac{GMy_1}{(y_1^2 + y_2^2)^{3/2}}$$
$$y_2'' = -\frac{GMy_2}{(y_1^2 + y_2^2)^{3/2}}$$

   1.2. Zadanie drugie

   Dane jest równanie rózniczkowe zwyczajne:

$$y' = -5y$$

   z warunkiem poczatkowym $y(0) = 1$. Równanie rozwiazujemy numerycznie z krokiem $h = (0{,}5)$.

   (a) Analityczna stabilnosc. Wyjasnij, czy rozwiazania powyzszego równania sa stabilne?

   (b) Numeryczna stabilnosc. Wyjasnij, czy metoda Euler'a jest stabilna dla tego równania z uzytym krokiem h?

   (c) Oblicz numerycznie wartosci przyblizonego rozwiazania dla t = 0:5 metoda Euler'a.

   (d) Wyjasnij, czy niejawna metoda Euler'a jest stabilna dla tego równania z uzytym krokiem h?

   (e) Oblicz numerycznie wartosci przyblizonego rozwiazania dla t = 0:5 niejawna metoda Euler'a.

1.3. Zadanie trzecie:

Model Kermack'a-McKendrick'a przebiegu epidemii w populacji
opisany jest układem równan rózniczkowych:

$$S' = -\frac{beta}{N} IS$$

$$I' = \frac{beta}{N} IS - gamma * I$$

$$R' = gamma * I$$

gdzie
S reprezentuje liczbe osób zdrowych, podatnych na zainfekowanie,
I reprezentuje liczbe osób zainfekowanych i roznoszacych infekcje,
R reprezentuje liczbe osób ozdrowiałych.

Liczba N to liczba osób w populacji. Parametr beta reprezentuje współczynnik zakaznosci (ang. transmission rate). Parametr gamma reprezentuje współczynnik wyzdrowień (ang. recovery rate). Wartosc 1/gamma reprezentuje sredni czas choroby.

Załozenia modelu:

- Przyrost liczby osób zakazonych jest proporcjonalny do liczby osób zakazonych oraz do liczby osób podatnych.
- Przyrost liczby osób odppornych lub zmarłych jest wprost proporcjonalny do liczby aktualnie chorych.
- Okres inkubacji choroby jest zaniedbywalnie krótki.
- Populacja jest wymieszana.

Jako wartosci poczatkowe ustal:

S(0) = 762; I(0) = 1; R(0) = 0.

Przyjmij tez N = S(0)+I(0)+R(0) = 763 oraz beta = 1. Zakładajac, ze sredni czas trwania grypy wynosi 1/gamma = 7 dni, przyjmij gamma = 1/7.

Całkujac od t = 0 do t = 14 z krokiem 0.2, rozwiaz powyzszy układ równan:

- jawna metoda Eulera

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

- niejawna metoda Eulera

$$y_{k+1} = y_k + h_k * f(t_{t+1}, y_{k+1})$$

- metoda Rungego-Kutty czwartego rzedu (RK4))

$$y_{k+1} = y_k + \frac{h_k}{6} * (k_1 + 2k_2 + 2k_3 + k_4),$$

$$gdzie,$$

$$k_1 = f(t_k, y_k)$$

$$k_2 = f\left(t_k + \frac{h_k}{2}, y_k + \frac{h_k k_1}{2}\right)$$

$$k_3 = f\left(t_k + \frac{h_k}{2}, y_k + \frac{h_k k_2}{2}\right)$$

$$k_4 = f(t_k + h_k, y_k + h_k h_3)$$

Wykonaj nastepujace wykresy:

- Dla kazdej metody przedstaw na wspólnym rysunku wykresy komponentów rozwiazania (S, I, R) jako funkcje t (3 wykresy).

- Na wspólnym rysunku przedstaw wykresy funkcji S(t)+I(t)+R(t) znalezione przez kazda metode (1 wykres). Czy niezmiennik S(t)+I(t)+R(t) N jest zachowany?

Wiemy, ze liczba osób zakazonych w pewnej szkole kształtowała sie nastepujaco:

Dzien, t 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
Zakazeni, I 1 3 6 25 73 222 294 258 237 191 125 69 27 11 4

Wybierz jedna z powyzszych metod numerycznych i oszacuj prawdziwe wartosci współczynników $\theta = [\beta, \gamma]$. W tym celu wykonaj minimalizacje funkcji kosztu. Jako funkcje kosztu wykorzystaj sume kwadratów reszt (ang. residual sum of squares):

$$L(\theta) = \sum_{i=0}^{T} \left(I_i - \widehat{I_\iota}\right)^2$$

gdzie I_i oznacza prawdziwa liczbe zakazonych, a $\widehat{I_\iota}$ oznacza liczbe zakazonych wyznaczonych metoda numeryczna. Poniewaz nie znamy gradientu $\nabla_\theta L(\theta)$, do minimalizacji wykorzystaj metode Neldera-Meada, która nie wymaga informacji o gradiencie. Powtórz obliczenia, tym razem jako funkcje kosztu wykorzystujac:

$$L(\theta) = -\sum_{i=0}^{T} I_i * ln\widehat{I_\iota} + \sum_{i=0}^{T} \widehat{I_\iota}$$

Ile wynosił współczynnik reprodukcji kazdym przypadku?

2. Rozwiązanie zadań

2.1. Implementacja zadania pierwszego

2.1.1. Równanie Van der Pol'a

```
def van_der_pol(t, y):
    y1, y2 = y
    dy1_dt = y2
    dy2_dt = dy1_dt * (1 - y1 ** 2) - y1
    return [dy1_dt, dy2_dt]
```

2.1.2. Warunki początkowe

```
y0_van_der_pol = [2, 0]
```

2.1.3. Czas symulacji

```
t_span = (0, 20)
t_eval = np.linspace(0, 20, 1000)
```

2.1.4. Rozwiązanie równania

```
sol_van_der_pol = solve_ivp(van_der_pol, t_span, y0_van_der_pol, t_eval=t_eval)
```

2.1.5. Wykres Van der Pol'a

```
plt.plot(sol_van_der_pol.t, sol_van_der_pol.y[0], label='y(t)')
plt.plot(sol_van_der_pol.t, sol_van_der_pol.y[1], label="y'(t)")
plt.title("Równanie Van der Pol'a")
plt.xlabel('Czas t')
plt.ylabel('y')
plt.legend()
plt.show()
```

2.1.6. Równanie Blasiusa

```
def blasius(t, y):
    y1, y2, y3 = y
    dy1_dt = y2
    dy2_dt = y3
    dy3_dt = - y1 * dy2_dt
    return [dy1_dt, dy2_dt, dy3_dt]
```

2.1.7. Warunki początkowe

```
y0_blasius = [0, 0, 0.332]
```

2.1.8. Czas symulacji

```
t_span = (0, 10)
t_eval = np.linspace(0, 10, 1000)
```

2.1.9. Rozwiązanie równania

```
sol_blasius = solve_ivp(blasius, t_span, y0_blasius, t_eval=t_eval)
```

2.1.10. Wykres Blasiusa

```
plt.plot(sol_blasius.t, sol_blasius.y[0], label='y(t)')
plt.plot(sol_blasius.t, sol_blasius.y[1], label="y'(t)")
plt.plot(sol_blasius.t, sol_blasius.y[2], label="y''(t)")
plt.title("Równanie Blasiusa")
plt.xlabel('Czas t')
plt.ylabel('y')
plt.legend()
plt.show()
```

2.1.11. II zasada dynamiki Newtona dla problemu dwóch ciał

```
def two_body_problem(t, y, G, M):
    y1, y2, y3, y4 = y
    r = (y1 ** 2 + y2 ** 2) ** (3/2)
    dy1_dt = y3
    dy2_dt = y4
    dy3_dt = -G * M * y1 / r
    dy4_dt = -G * M * y2 / r
    return [dy1_dt, dy2_dt, dy3_dt, dy4_dt]
```

2.1.12. Stałe

```
G = 6.67430e-11  # Stała grawitacyjna
M = 5.972e24     # Masa Ziemi (kg)
```

2.1.13. Warunki początkowe

```
y0_two_body = [7.0e6, 0.0, 0.0, 7.12e3]
```

2.1.14. Czas symulacji

```
t_span = (0, 6000)
t_eval = np.linspace(0, 6000, 1000)
```

2.1.15. Rozwiązanie równania

```
sol_two_body = solve_ivp(two_body_problem, t_span, y0_two_body, t_eval=t_eval, args=(G, M))
```

2.1.16. Wykres II zasady dynamiki Newtona dla problemu dwóch ciał

```
plt.plot(sol_two_body.y[0], sol_two_body.y[1])
plt.title("II zasada dynamiki Newtona dla problemu dwóch ciał")
plt.xlabel('y1')
plt.ylabel('y2')
plt.axis('equal')
plt.show()
```

2.2. Implementacja zadania drugiego

2.2.1. Analityczna stabilność – wyjaśnienie, dlaczego rozwiązania równania w zadaniu drugim jest stabilne

Rozwiązanie analityczne równania różniczkowego można znaleźć, rozwiązując je jako równanie różniczkowe pierwszego rzędu.
Dla równania:

$$y' = -5y$$

Rozwiązaniem jest:

$$y(t) = y(0) * e^{-5t}$$

Z warunku początkowego $y(0) = 1$

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

Analityczna stabilność oznacza, że jeśli zaburzymy początkowy warunek $y(0)$ to rozwiązanie nie powinno zmieniać się gwałtownie. Ponieważ wykładnicza funkcja e^{-5t} zawsze zmierza do zera, rozwiązanie jest stabilne dla każdej wartości t. Rozwiązania są stabilne, ponieważ dla $t \longrightarrow \infty, y(t) \longrightarrow 0$

2.2.2. Numeryczna stabilność – wyjaśnienie, dlaczego metoda jawna Euler'a nie jest stabilna

Stabilność numeryczna oznacza, że metoda numeryczna nie generuje rosnących błędów przy obliczeniach. Dla metody Eulera zastosowanej do równania $y' = -5y$ z krokiem $h = 0.5$

Metoda Eulera:

$$y_{n+1} = y_n + h * f(t_n, y_n)$$

Podstawiając $f(t, y) = -5y$

$$y_{n+1} = y_n + h(-5y_n) = y_n * (1 - 5h)$$

Dla stabilności, wartość $|1 - 5h|$ musi byc mniejsza niz 1:

$$|1 - 5 * 0.5| = |1 - 2.5| = |-1.5| = 1.5$$

Ponieważ $1.5 > 1$, metoda Eulera nie jest stabilna dla tego równania przy kroku $h = 0.5$

2.2.3. Obliczenie numeryczne stabliności dla metody jawnej Euler'a

2.2.3.1. Metoda Eulera

```python
def euler_step(y, t, h, f):
    return y + h * f(t, y)
```

```python
def f(t, y):
    return -5 * y
```

2.2.3.2. Warunki początkowe

```
y0 = 1
t0 = 0
h = 0.5
```

2.2.3.3. Obliczenie wartości przybliżonej dla $t = 0.5$

```
y_approx_euler = euler_step(y0, t0, h, f)
print(f"Metoda Eulera: y(0.5) = {y_approx_euler}")
```

$$Metoda\ Eulera: y(0.5) = -1.5$$

2.2.4. Numeryczna stabilność – wyjaśnienie, dlaczego metoda niejawna Eulera' jest stabilna

Niejawna metoda Eulera:

$$y_{n+1} = y_n + h * f(t_{n+1}, y_{n+1}).$$

Podstawiając $f(t, y) = -5y$

$$y_{n+1} = y_n + h(-5y_{n+1})$$

$$y_{n+1}(1 + 5h) = y_n$$

$$y_{n+1} = \frac{y_n}{1 + 5h}$$

Dla stabilności, wartość $|\frac{1}{1+5h}|$ musi byc mniejsze niz 1:

$$|\frac{1}{1 + 5 * 0.5}| = 0.286$$

Ponieważ $0.286 < 1$, niejawna metoda Eulera jest stabilna dla tego równania przy kroku $h = 0.5$

2.2.5. Obliczenie numeryczne stabilności dla metody niejawnej Euler'a

2.2.5.1. Niejawna metoda Eulera

```
def implicit_euler_step(y, t, h, f):
    return y / (1 + 5 * h)
```

2.2.5.2. Warunki początkowe

```
y0 = 1
t0 = 0
h = 0.5
```

2.2.5.3. Obliczenie wartości przybliżonej dla $t = 0.5$

```
y_approx_implicit_euler = implicit_euler_step(y0, t0, h, f)
print(f"Niejawna metoda Eulera: y(0.5) = {y_approx_implicit_euler}")
```

$$Niejawna\ metoda\ Eulera: y(0.5) = 0.285$$

2.2.6. Wnioski do zadania drugiego:

Wykazaliśmy matematycznie, że metoda jawna Eulera jest nie stabilna, a metoda niejawna jest stabilna. Dzięki obliczeniom można łatwo zauważyć rozbieżność wyników co potwierdza wcześniejsze obliczenia.

2.3. Implementacja zadania trzeciego

2.3.1. Paramtery modelu

```
beta = 1.0
gamma = 1.0 / 7.0
```

2.3.2. Warunki początkowe

```
S0 = 762
I0 = 1
R0 = 0
N = S0 + I0 + R0
```

2.3.3. Czas

```
t_span = (0, 14)
t_eval = np.arange(0, 14.2, 0.2)
```

2.3.4. Funkcja opisująca model SIR

```
def sir_model(t, y, beta, gamma):
    S, I, R = y
    dSdt = np.array(-beta / N * I * S)
    dIdt = np.array(beta / N * I * S - gamma * I)
    dRdt = np.array(gamma * I)
    return np.array([dSdt, dIdt, dRdt])
```

2.3.5. Metoda jawna Eulera

```
def euler_method(f, t_span, y0, h, beta, gamma):
    t0, tf = t_span
    t = np.arange(t0, tf + h, h)
    n = len(t)
    y = np.zeros((n, len(y0)))
    y[0] = y0

    for i in range(n - 1):
        y[i + 1] = y[i] + h * np.array(f(t[i], y[i], beta, gamma))

    return t, y
```

2.3.6. Metoda niejawna Eulera

```
def implicit_euler_method(f, t_span, y0, h, beta, gamma):
    t0, tf = t_span
    t = np.arange(t0, tf + h, h)
    n = len(t)
    y = np.zeros((n, len(y0)))
    y[0] = y0

    for i in range(n - 1):
        y_pred = y[i] + h * np.array(f(t[i], y[i], beta, gamma))
        y[i + 1] = y[i] + h * np.array(f(t[i + 1], y_pred, beta, gamma))

    return t, y
```

2.3.7. Metoda Rungego-Kutty czwartego rzędu

```
def rk4_method(f, t_span, y0, h, beta, gamma):
    t0, tf = t_span
    t = np.arange(t0, tf + h, h)
    n = len(t)
    y = np.zeros((n, len(y0)))
    y[0] = y0

    for i in range(n - 1):
        k1 = np.array(f(t[i], y[i], beta, gamma))
        k2 = np.array(f(t[i] + h/2, y[i] + h*k1/2, beta, gamma))
        k3 = np.array(f(t[i] + h/2, y[i] + h*k2/2, beta, gamma))
        k4 = np.array(f(t[i] + h, y[i] + h*k3, beta, gamma))
        y[i + 1] = y[i] + (h / 6) * (k1 + 2*k2 + 2*k3 + k4)

    return t, y
```

2.3.8. Początkowe wartości

```
y0 = [S0, I0, R0]
```

2.3.9. Rozwiązanie układu metodą jawną Eulera

```
t_euler, y_euler = euler_method(sir_model, t_span, y0, 0.2, beta, gamma)
```

2.3.10. Rozwiązanie układu metodą niejawną Eulera

```
t_implicit_euler, y_implicit_euler = implicit_euler_method(sir_model, t_span, y0, 0.2, beta, gamma)
```

2.3.11. Rozwiązanie układu metodą Rungego-Kutty czwartego rzędu

```
t_rk4, y_rk4 = rk4_method(sir_model, t_span, y0, 0.2, beta, gamma)
```

2.3.12. Wykresy komponentów rozwiązania (S,I,R) dla każdej metody

```
plt.title("Wykres komponentu S dla kazdej metody")
plt.plot(t_euler, y_euler[:, 0], 'b', label='S(t) - Euler')
plt.plot(t_implicit_euler, y_implicit_euler[:, 0], 'g', label='S(t) - Implicit Euler')
plt.plot(t_rk4, y_rk4[:, 0], 'r', label='S(t) - RK4')
plt.xlabel('t')
plt.ylabel('S(t)')
plt.yscale('log')
plt.legend()
plt.show()
```

```
plt.title("Wykres komponentu I dla kazdej metody")
plt.plot(t_euler, y_euler[:, 1], 'b', label='I(t) - Euler')
plt.plot(t_implicit_euler, y_implicit_euler[:, 1], 'g', label='I(t) - Implicit Euler')
plt.plot(t_rk4, y_rk4[:, 1], 'r', label='I(t) - RK4')
plt.xlabel('t')
plt.ylabel('I(t)')
plt.yscale('log')
plt.legend()
plt.show()
```

```
plt.title("Wykres komponentu R dla kazdej metody")
plt.plot(t_euler, y_euler[:, 2], 'b', label='R(t) - Euler')
plt.plot(t_implicit_euler, y_implicit_euler[:, 2], 'g', label='R(t) - Implicit Euler')
plt.plot(t_rk4, y_rk4[:, 2], 'r', label='R(t) - RK4')
plt.xlabel('t')
plt.ylabel('R(t)')
plt.yscale('log')
plt.legend()
plt.show()
```

2.3.13. Wykresy S(t) + I(t) + R(t) dla każdej metody

```
plt.figure(figsize=(10, 6))
plt.plot(t_euler, y_euler[:, 0] + y_euler[:, 1] + y_euler[:, 2], 'b', label='Euler')
plt.plot(t_implicit_euler, y_implicit_euler[:, 0] + y_implicit_euler[:, 1] + y_implicit_euler[:, 2], 'g',
 label='Implicit Euler')
plt.plot(t_rk4, y_rk4[:, 0] + y_rk4[:, 1] + y_rk4[:, 2], 'r', label='RK4')
plt.xlabel('t')
plt.ylabel('S(t) + I(t) + R(t)')
plt.legend()
plt.title('S(t) + I(t) + R(t) dla każdej metody')
plt.show()
```

2.3.14. Prawdziwe dane zakażonych

```
true_infected = np.array([1, 3, 6, 25, 73, 222, 294, 258, 237, 191, 125, 69, 27, 11, 4])
days = np.arange(15)
```

2.3.15. Funkcja kosztu – suma kwadratów reszt

```
def cost_function(params, t, true_infected):
    beta, gamma = params
    t_span = (0, 14)
    h = 1
    t_rk4, y_rk4 = rk4_method(sir_model, t_span, y0, h, beta, gamma)
    I_pred = y_rk4[:,1]
    solution = np.sum((np.array(true_infected) - I_pred) ** 2)
    return solution
```

2.3.16. Funkcja kosztu – log-likelihood

```
def log_likelihood_function(params, t, true_infected):
    beta, gamma = params
    y0 = [S0, I0, R0]
    t_span = (0, 14)
    h = 1
    t_rk4, y_rk4 = rk4_method(sir_model, t_span, y0, h, beta, gamma)
    I_pred = y_rk4[:,1]
    solution = np.sum(-np.array(true_infected * np.log(I_pred)) + I_pred)
    return solution
```

2.3.17. Minimalizacja funkcji kosztu

```
options = {'maxiter': 200, 'maxfun': 300, 'disp': True}
initial_guess = [beta, gamma]
```

```
result = minimize(cost_function, initial_guess, args=(days, true_infected), method='Nelder-Mead',
 options=options)
beta_est, gamma_est = result.x
R0_est = beta_est / gamma_est
print(f"Estymowane wartości - suma kwadratów reszt: beta = {beta_est}, gamma = {gamma_est}, R0 = {R0_est}")
```

$$Estymowane\ wartości\ -\\ beta\ =\ 1.6697415516194567,\\ gamma\ =\ 0.44564002141982473,\\ R0\ =\ 3.7468393128148625$$

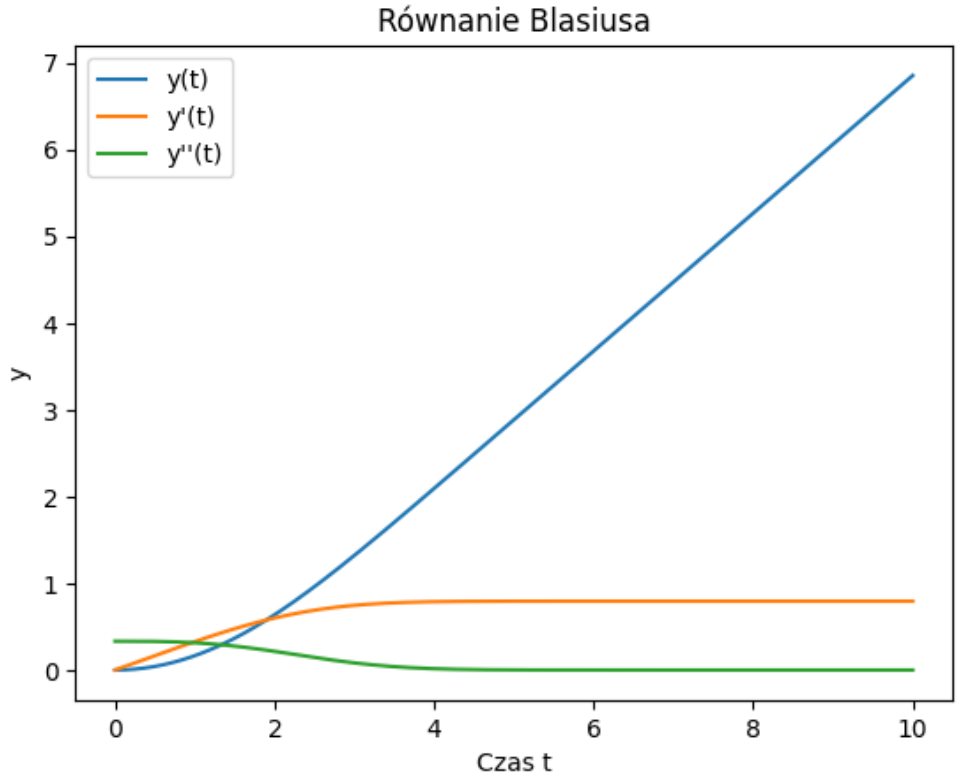
```
result_log_likelihood = minimize(log_likelihood_function, initial_guess, args=(days, true_infected),
 method='Nelder-Mead', options=options)
beta_est_log, gamma_est_log = result_log_likelihood.x
R0_est_log = beta_est_log / gamma_est_log
print(f"Estymowane wartości - log-likelihood: beta = {beta_est_log}, gamma = {gamma_est_log}, R0 =
 {R0_est_log}")
```

$$Estymowane\ wartości\ -\\ beta\ =\ 1.69255947736147,\\ gamma\ =\ 0.480862879543448,\\ R0\ =\ 3.5198380855857687$$

3. Wykresy

3.1. Wykresy dla zadania pierwszego

3.1.1. Wykres Blasiusa

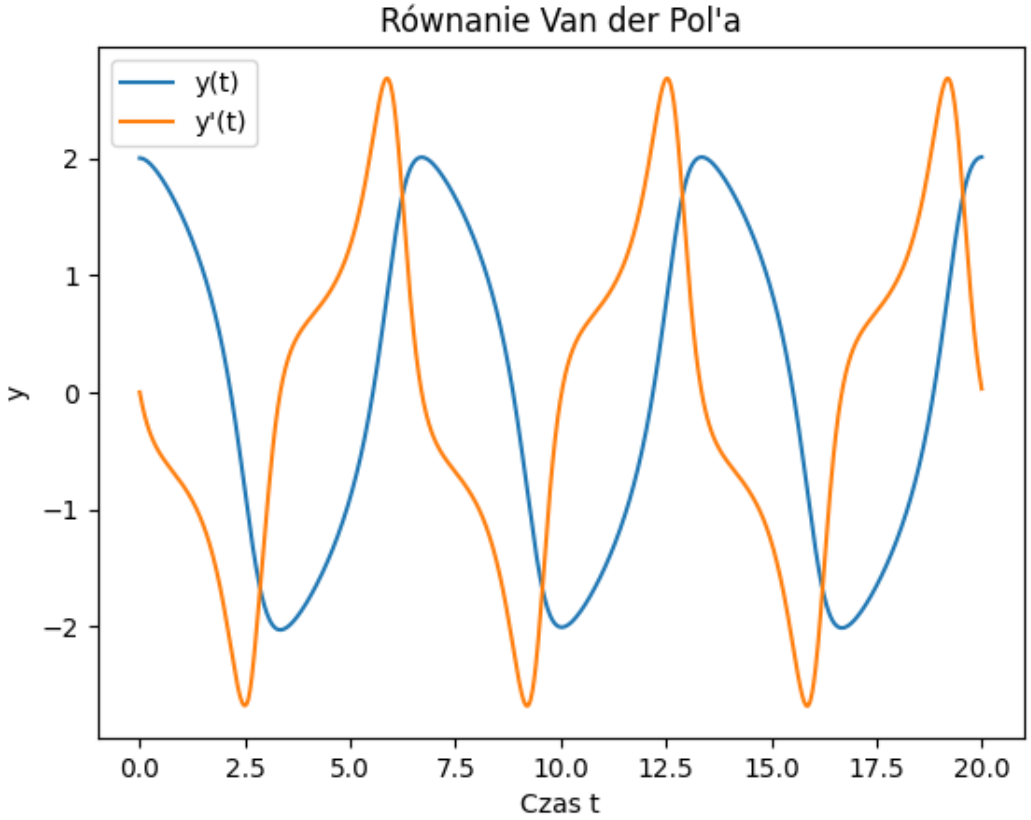


Wykres 1. Równanie Blasiusa

3.1.2. Wykres Newtona

Wykres 2. Równanie Newtona

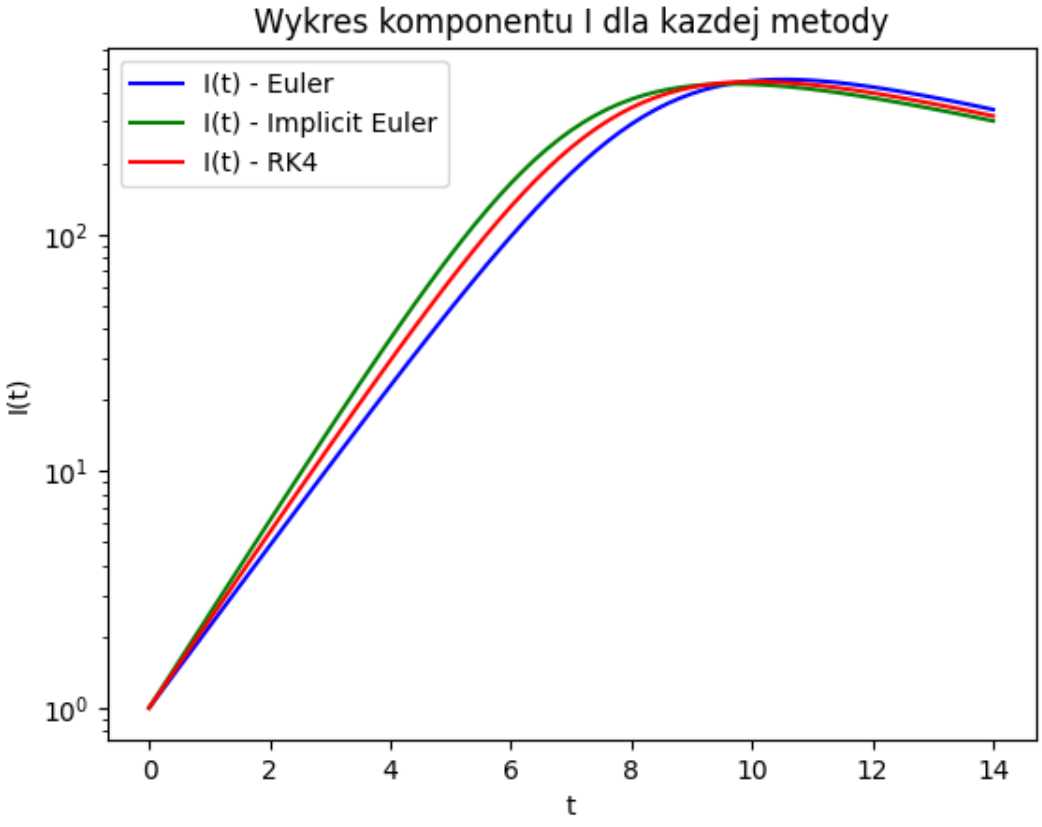
3.1.3. Wykres Van der Pole’a



Wykres 3. Równanie Van der Pol’a

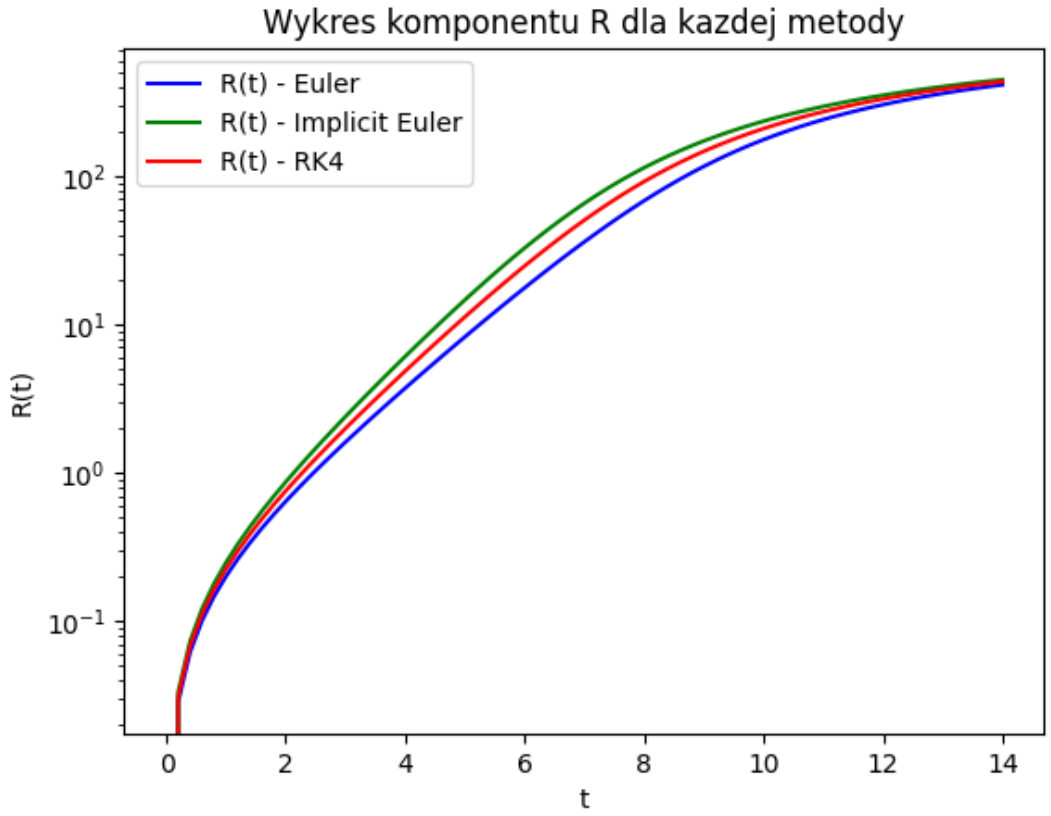
3.2. Wykresy dla zadania trzeciego

3.2.1. Wykres komponentu I(t)



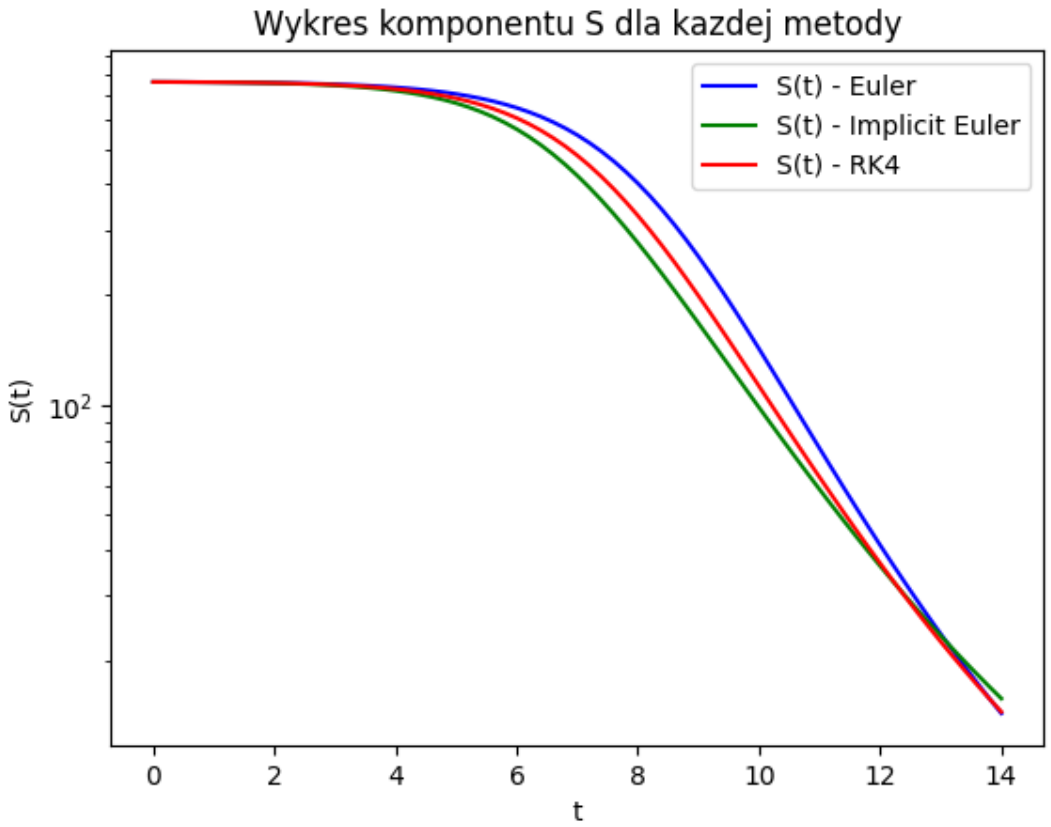
Wykres 4. Równanie komponentu I dla każdej metody

3.2.2. Wykres komponentu R(t)



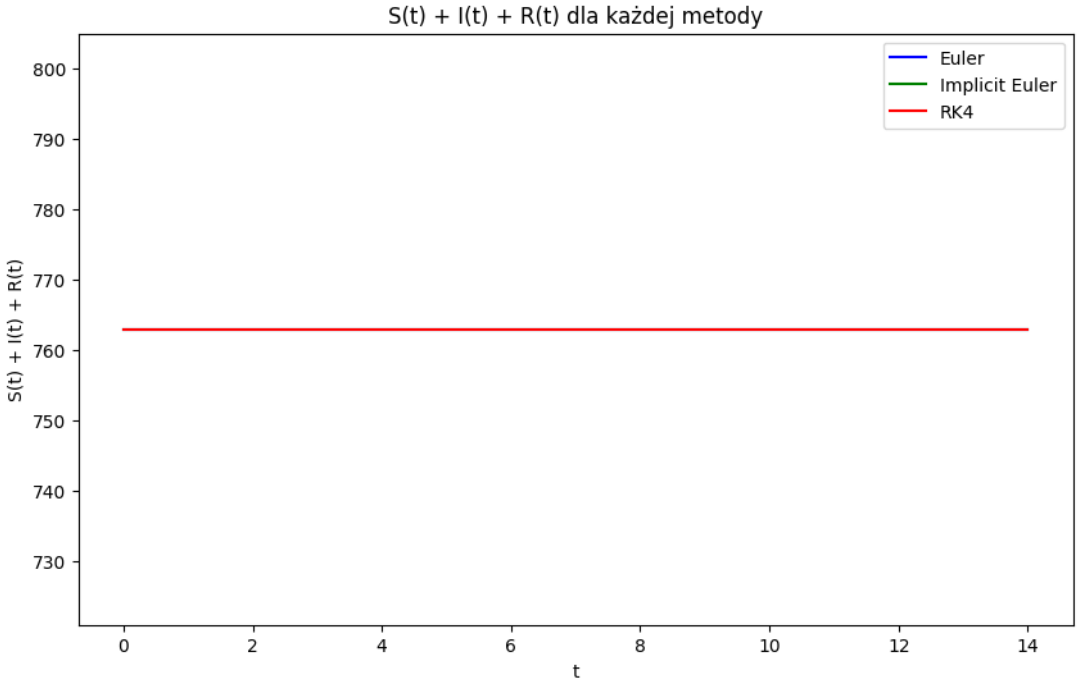
Wykres 4. Równanie komponentu R dla każdej metody

3.2.3. Wykres komponentu S(t)



Wykres 4. Równanie komponentu S dla każdej metody

3.2.4. Wykres I(t) + R(t) + S(t)



Wykres 5. Suma S(t) + I(t) + R(t) dla każdej metody

4. Wnioski

   W zadaniu drugim kiedy sprawdzaliśmy matematycznie czy faktycznie metoda jawna Eulera jest stabilna dla tego przypadku wyszło nam, że nie jest podobnie jak w przypadku niejawnej ale tam wyszło że jest stabilna ta metoda, porównując ich wartości można od razu zobaczyć ze wyniki znacznie się różnią od siebie.

   W zadaniu trzecim można zauważyć ze niezmiennik **sumy S(t) + I(t) + R(t)** zostaje zachowany z wykresu nr 5.

   Stosując minimalizacje korzystając z funkcji kosztu $L(\theta) = -\sum_{i=0}^{T} I_i * ln\widehat{I_i} + \sum_{i=0}^{T} \widehat{I_i}$, otrzymujemy mniejszy współczynnik reprodukcji wynoszący $R0 = 3.51$, za to $beta, gamma$ wynoszą mniej więcej tyle samo dla obu funkcji kosztu

5. Bibliografia

   Wykład MOwNiT - prowadzony przez dr. Inż. K. Rycerz
   Prezentacje – dr. Inż. M. Kuta

6. Dodatkowe informacje

   Rozwiązanie wszystkich zadan znajduje się odpowiednio w plikach ex1.ipynb, ex2.ipynb, ex3.ipynb.