SPRAWOZDANIE 6

Modelowanie Systemów Dynamicznych

Temat ćwiczenia: Rozwiązywanie ODE w Matlabie

Michał Midor, gr. 5, 26.11.2025 r. - Środa 13.15

Cel ćwiczenia:

Celem laboratorium jest praktyczne zrozumienie numerycznych metod rozwiązywania równań różniczkowych zwyczajnych (ODE). Ćwiczenie ma na celu:

1. Zbadanie wpływu wielkości kroku całkowania (h) na dokładność i stabilność najprostszej metody numerycznej – metody Eulera.
2. Zrozumienie kompromisu pomiędzy kosztem obliczeniowym a precyzją wyniku.
3. Porównanie metody Eulera z zaawansowanym solverem ode45 (wykorzystującym metodę Rungego-Kutty rzędu 4/5).
4. Implementację układu dynamicznego w przestrzeni stanów na przykładzie modelu lądującego samolotu na lotniskowcu oraz wizualizację zmiennych, które nie są bezpośrednio zmiennymi stanu (przyspieszenie).

Rozwiązanie zadań:

Równania różniczkowe opisują większość zjawisk fizycznych, jednak rzadko posiadają proste rozwiązania analityczne. W inżynierii standardem jest wykorzystywanie metod numerycznych. W pierwszej części zadania analizowana jest metoda Eulera, która przybliża pochodną ilorazem różnicowym. Jest ona intuicyjna, lecz generuje błędy kumulujące się w czasie, co wymusza stosowanie bardzo małego kroku czasowego. W drugiej części wykorzystany jest wbudowany w środowisko MATLAB solver `ode45`. Jest to algorytm jednokrokowy, który dynamicznie dobiera wielkość kroku całkowania, zagęszczając punkty tam, gdzie funkcja zmienia się gwałtownie, a rozrzedzając tam, gdzie jest "gładka". Pozwala to na uzyskanie wysokiej dokładności przy zminimalizowanym czasie obliczeń. Finalnie przeprowadzona jest symulacja systemu hamowania samolotu, sprowadzając równania ruchu drugiego rzędu do układu równań pierwszego rzędu w postaci wektorowej.

Zadanie 1:

W pierwszym etapie ćwiczenia skupiamy się na numerycznym rozwiązaniu równania różniczkowego:

$$\frac{dy}{dt} = f(t, y) = 2t$$

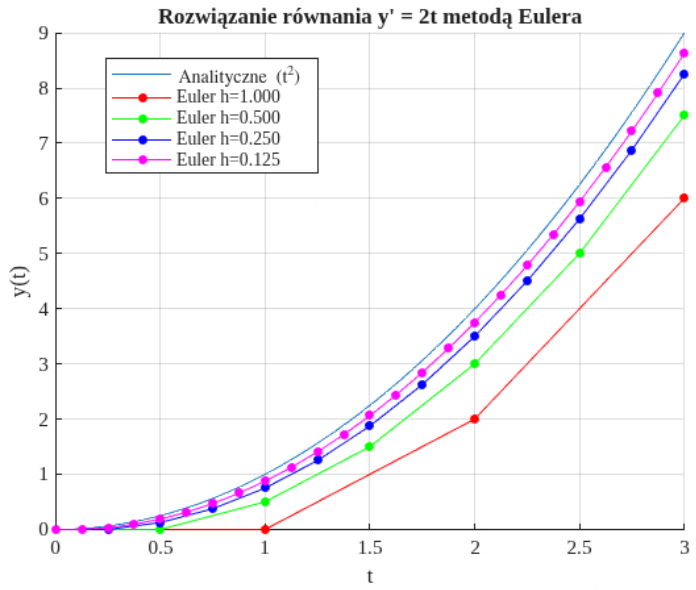
na przedziale t [0, 3], z warunkiem początkowym y0 = 0. Wykorzystujemy metodę Eulera, która jest metodą iteracyjną opartą na schemacie:

$$y_{n+1} = y_n + h \cdot f(t_n, y_n),$$
$$t_{n+1} = t_n + h$$

Kluczowym parametrem jest tutaj krok 'h', który determinuje gęstość próbkowania czasu. Celem jest porównanie wyników numerycznych z rozwiązaniem analitycznym y(t)=t^2 dla różnych wartości kroku 'h'.

```
t_start = 0;
```

```
t_end = 3;
y0 = 0; % Warunek początkowy

h_values = [1, 0.5, 0.25, 0.125];

figure;
hold on;
grid on;
title('Rozwiązanie równania y'' = 2t metodą Eulera');
xlabel('t');
ylabel('y(t)');

t_anal = t_start:0.01:t_end;
y_anal = t_anal.^2;
plot(t_anal, y_anal, 'DisplayName', 'Analityczne (t^2)');

dy = @(t, y) 2*t; % uchwyt do funkcji

colors = {'r', 'g', 'b', 'm'};

for i = 1:length(h_values)
    h = h_values(i);
    t = t_start:h:t_end;
    y = zeros(size(t)); % dla wydajności

    y(1) = y0; % Warunek początkowy

    for n = 1:length(t)-1 % implementacja metody Eulera
        tn = t(n);
        yn = y(n);

        current_dy = dy(tn, yn);

        y(n+1) = yn + h*current_dy;
    end
    plot(t, y, 'o-', 'Color', colors{i}, 'MarkerFaceColor', colors{i},
'MarkerSize', 4, ...
         'DisplayName', sprintf('Euler h=%.3f', h));
end

legend('Location', 'best');
hold off;
```



Wraz ze zmniejszaniem parametru 'h' otrzymywane rozwiązania są coraz bardziej zbliżone do analitycznego. Niestety, nawet przy najmniejszym zastosowanym kroku (h=0.125), metoda Eulera nadal generuje zauważalny błąd i odchylenie od krzywej wzorcowej, co wynika z jej prostej konstrukcji i kumulowania się błędów w kolejnych krokach.

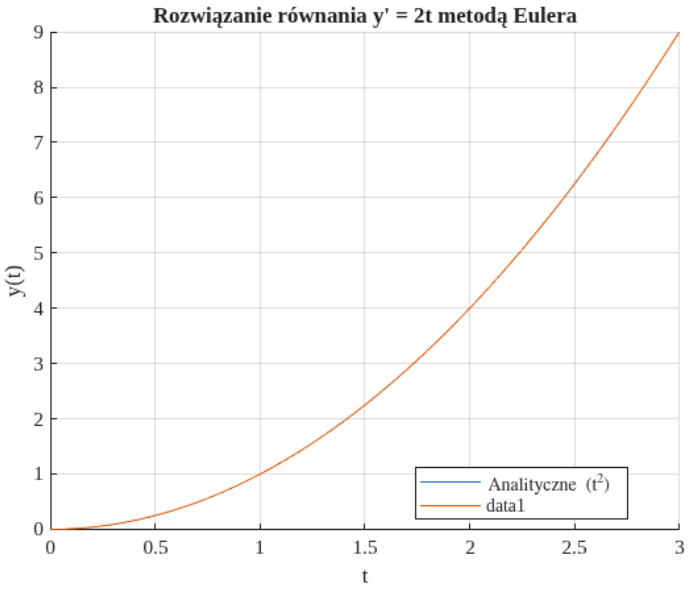
Zadanie 2:

Środowisko MATLAB oferuje zaawansowane narzędzia do rozwiązywania ODE, tzw. solvery, które automatycznie dobierają optymalny krok całkowania w każdej iteracji, kontrolując błąd. Najpopularniejszym z nich jest ode45, oparty na metodzie Rungego-Kutty rzędu 4 i 5. W tym zadaniu rozwiązujemy to samo równanie y'=2t przy użyciu ode45, aby porównać jego skuteczność z metodą Eulera. Używany jest tu uchwy do funkcji pod nazwą 'ode_fun' który pozwala przekazać solverowi funkcję do rozwiązania.

```
figure;
hold on;
grid on;
title('Rozwiązanie równania y'' = 2t metodą Eulera');
xlabel('t');
ylabel('y(t)');

plot(t_anal, y_anal, 'DisplayName', 'Analityczne (t^2)');

ode_fun = @(t,y) 2*t;

legend('Location', 'best');

opts = odeset('stats','on');
tspan = [t_start t_end];
y0 = 0;
[t,y] = ode45(ode_fun, tspan, y0, opts); % dopasowuje krok - jeśli funkcja
skręca to zmniejsza krok, by
```

```
10 successful steps
0 failed attempts
61 function evaluations
```

```
% zachować dokładność, a jeżeli jest płaska, to zwiększa, by oszczędzić moc obliczeniową
plot(t,y(:,1));
hold off;
```



Użycie solvera ode45 pozwoliło na osiągnięcie praktycznie idealnego odtworzenia wyniku analitycznego y=t^2. Solver dynamicznie dostosował punkty obliczeniowe, zapewniając wysoką dokładność przy zoptymalizowanym koszcie obliczeniowym, co stanowi znaczącą przewagę nad stałokrokową metodą Eulera. Rozwiązanie za pomocą ode45 pokrywa się idealnie z analitycznym (data1 to rozwiązanie ode45).

Przykład - Wahadło z tłumieniem:

Aby przetestować solver na bardziej złożonym obiekcie, modelujemy ruch wahadła z tłumieniem opisanego równaniem nieliniowym drugiego rzędu:

$$\ddot{\theta} = -\frac{b}{m}\dot{\theta} - \frac{mg}{l(m-2b)}sin\theta$$

Równanie to sprowadzamy do układu dwóch równań pierwszego rzędu w przestrzeni stanów:

$$\dot{y_1} = y_2$$

$$\dot{y_2} = -\frac{b}{m}y_2 - \frac{mg}{l(m-2b)}siny_1$$

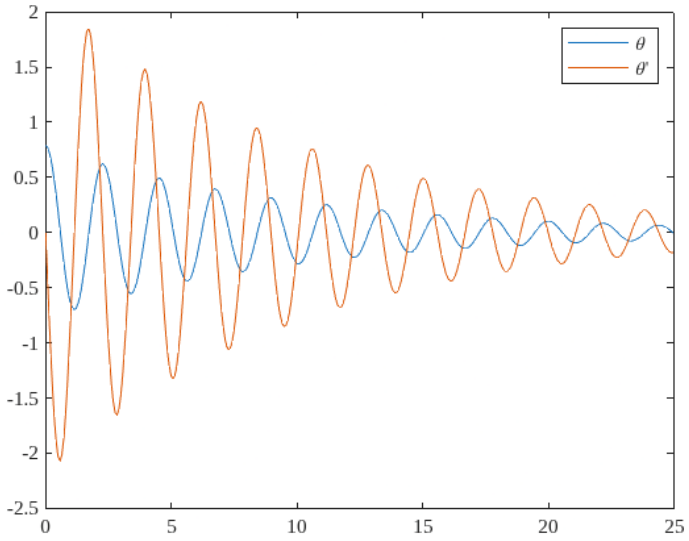
Symulacja startuje z wychylenia 45 stopni (pi/4) i zerowej prędkości początkowej.

```
opts = odeset('stats','on');
```

```
tspan = [0 25];
y0 = [pi/4, 0];
[t,y] = ode45(@wahadlo, tspan, y0, opts);
```

```
96 successful steps
0 failed attempts
577 function evaluations
```

```
plot(t,y(:,1),t,y(:,2))
legend('\theta','\theta''')
```



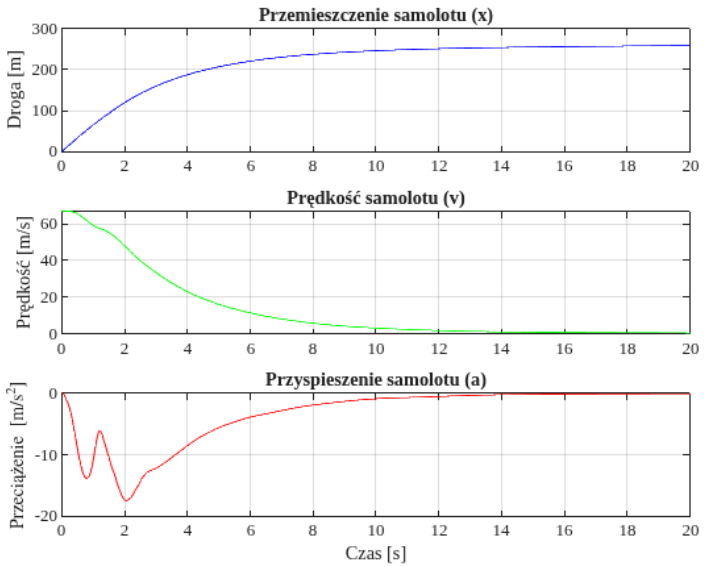
Solver `ode45` poprawnie rozwiązał układ równań. Na wykresach (kąt i prędkość kątowa) wyraźnie widać charakterystykę tłumionych oscylacji harmonicznych. Amplituda wychyleń maleje w czasie, co jest zgodne z fizyczną naturą zjawiska zanikania energii przez tłumienie.

Zadanie 3:

Ostatnim etapem jest symulacja lądowania samolotu na lotniskowcu poprzez zamodelowanie układu hamownika. Model uwzględnia nieliniowe siły sprężystości lin oraz siłę tłumienia hydraulicznego zależną od prędkości. Równania ruchu obejmują 6 zmiennych stanu. Ponieważ przyspieszenie samolotu nie jest zmienną stanu (jest pochodną prędkości), solver nie zwraca go bezpośrednio. Aby je wyznaczyć i zwizualizować, wykorzystujemy funkcję wyjścia (`OutputFcn`) oraz zmienną globalną `w3`, która gromadzi obliczone wartości przyspieszenia w każdym kroku solvera. Używamy opcji `Refine` ustawionej na 1, aby zsynchronizować kroki solvera z zapisem do zmiennej globalnej `w3`.

```
global w3

options = odeset('OutputFcn',@hamownik_out,'Refine',1);
[T,Y] = ode45(@hamownik,[0 20],[0 67 0 0 0 0],options);
Y; % solver zwraca tylko zmienne stanu, czyli całki | x, x', y2, y2', y3, y3'
% x (zmodyfikowane geometrycznie y1, bo nieliniowe) - samolot , y2 -
bloczek, y3 - tłumik wodny
```

```
w3; % macierz z przyspieszeniem samolotu, trzeba ją "wydobyć" od solvera

figure('Name', 'Analiza Lądowania', 'Color', 'white');

% 1. Położenie samolotu
subplot(3,1,1);
plot(T, Y(:,1), 'b'); % pierwsza kolumna zmiennych stanu
grid on;
ylabel('Droga [m]');
title('Przemieszczenie samolotu (x)');

% 2. Prędkość Samolotu
subplot(3,1,2);
plot(T, Y(:,2), 'g'); % druga kolumna zmiennych stanu
grid on;
ylabel('Prędkość [m/s]');
title('Prędkość samolotu (v)');

% 3. Przyspieszenie Samolotu
subplot(3,1,3);
len = min(length(T), length(w3)); % Zabezpieczenie przed różną długością
wektorów, co może się zdarzyć przy używaniu ode45 w połaczeniu z global
plot(T(1:len), w3(1:len), 'r');
grid on;
xlabel('Czas [s]');
ylabel('Przeciążenie [m/s^2]');
title('Przyspieszenie samolotu (a)');
```



Uzyskane przebiegi (droga, prędkość, przyspieszenie) pokrywają się z wynikami oczekiwanymi. Wykres przyspieszenia pokazuje skoki sił działających na pilota i samolot.