Rozwiązywanie układów równań różniczkowych zwyczajnych

Sofiia Kuzmenko

Oświadczam, że niniejsza praca, stanowiąca podstawę do uznania osiągnięcia efektów uczenia się z przedmiotu Modelowanie matematyczne została wykonana przeze mnie samodzielnie.

Modelowanie matematyczne

Prowadzący: J.Wagner

Politechnika Warszawska
Wydział Matematyki i Nauk Informacyjnych
Warszawa
30 XI 2023

Spis treści

1 Listy symboli matematycznych i akronimów

t, h	–	zmienne skalarne
x, y_1, y_2	–	wektory zmiennej skalarnej t
$\dot{y_1}, \dot{y_2}, \hat{y_2}, \hat{y_1}$	–	wektory liczb rzeczywistych
y, f	–	macierze zmiennej skalarnej t
A	–	macierz liczb rzeczywistych
$\frac{\partial y_1}{\partial t}, \frac{\partial y_2}{\partial t}$	–	pochodne funkcji y_1 oraz y_2 po t
$\delta_1(h), \delta_2(h)$	–	błędy względne agregowane przy kroku h dla wektorów y_1 i y_2 odpowiednio

Lista akronimów

URRZ – układ równań różniczkowych zwyczajny

2 Wprowadzenie

W tym raporcie będziemy rozwiązywali **URRZ**(1), gdzie $t \in [0, 8]$ oraz $x(t) = exp(-t)sin(t)$

$$\begin{cases} \frac{\partial y_1}{\partial t} = \frac{-26}{3} y_1(t) - \frac{10}{3} y_2(t) + x(t) \\ \frac{\partial y_2}{\partial t} = \frac{10}{3} y_1(t) - \frac{1}{3} y_2(t) + x(t) \end{cases} \tag{1}$$

za pomocą wbudowanych metod Matlaba ***dsolve*** i ***ode45***, oraz modyfikowanych method Eulera, zdefiniowanych wzorami (b), (c) i (d), które w dalszej częsci raportu będą się pojawiały pod nazwami *metoda (b), metoda (c)* oraz *metoda (d)* odpowiednio. Przyjmujemy, że $\boldsymbol{y}_n = [y_1(t_n), y_2(t_n)]^T$, a $\boldsymbol{f}(t_n, \boldsymbol{y}_n)$ jest określona jako $\frac{\partial y(t)}{\partial t}|_{t=t_n} = \boldsymbol{f}(t_n, \boldsymbol{y}_n)$

$$\boldsymbol{y}_n = \boldsymbol{y}_{n-1} + \frac{h}{2}[3\boldsymbol{f}(t_{n-1}, \boldsymbol{y}_{n-1}) - \boldsymbol{f}(t_{n-2}, \boldsymbol{y}_{n-2})] \tag{b}$$

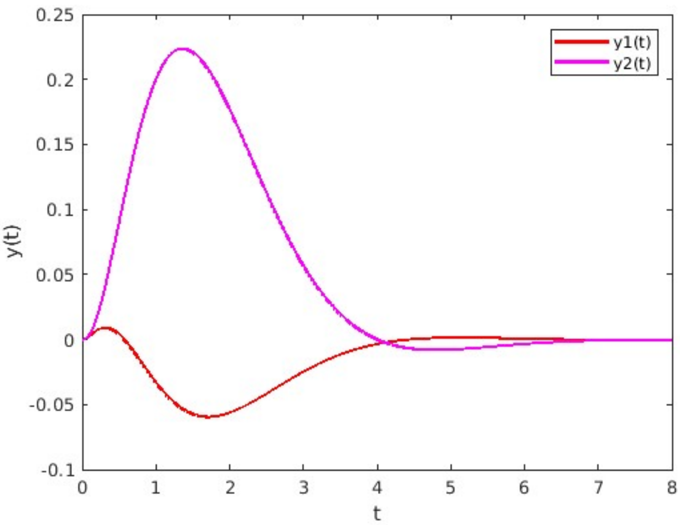
$$\boldsymbol{y}_n = \boldsymbol{y}_{n-1} + \frac{h}{12}[5\boldsymbol{f}(t_n, \boldsymbol{y}_n) + 8\boldsymbol{f}(t_{n-1}, \boldsymbol{y}_{n-1}) - \boldsymbol{f}(t_{n-2}, \boldsymbol{y}_{n-2})] \tag{c}$$

$$\boldsymbol{y}_n = \boldsymbol{y}_{n-1} + h \sum_{k=1}^{3} w_k \boldsymbol{f}_k \tag{d}$$

W *metodzie (d)* $\boldsymbol{f}_k = \boldsymbol{f}(t_{n-1} + c_k h, \boldsymbol{y}_{n-1} + h \sum_{K=1}^{3} a_{K,k} \boldsymbol{f}_k)$, a współczynniki $\alpha_{i,i}, \omega_i, c_i$ przyjmują wartości podane w tabeli Butchera na rysunku 1.

$$
\begin{array}{c|ccc}
c_1 & a_{1,1} & a_{1,2} & a_{1,3} \\
c_2 & a_{2,1} & a_{2,2} & a_{2,3} \\
c_3 & a_{3,1} & a_{3,2} & a_{3,3} \\
\hline
 & w_1 & w_2 & w_3
\end{array}
=
\begin{array}{c|ccc}
0 & \frac{1}{6} & -\frac{1}{6} & 0 \\
\frac{1}{2} & \frac{1}{6} & \frac{1}{3} & 0 \\
1 & \frac{1}{6} & \frac{5}{6} & 0 \\
\hline
 & \frac{1}{6} & \frac{2}{3} & \frac{1}{6}
\end{array}
$$

Rysunek 1: Tabela współczynników Butchera

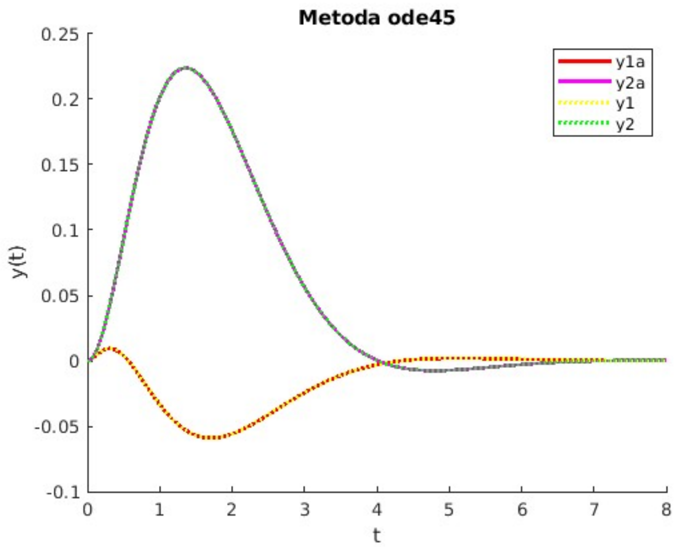


Rysunek 2: Wykres wyników procedury ***dsolve***

3 Metodyka i wyniki doświadczeń

3.1 Obliczanie dokładnego rozwiązania

W funkcji *zad1*, przedstawionej na urywku kodu (1), obliczamy wartości $\dot{y}_1, \dot{y}_2$ za pomocą wbudowanej procedury ***dsolve*** środowiska MATLAB. W dalszej częsci tego raportu będziemy przyjmowali je za dokładne rozwiązanie **URRZ**(1). Posłużą nam do badania dokładności rozwiązań numerycznych uzyskanych za pomocą *metod (b), (c), (d)*. Warunki początkowe z założenia są zerowe. Wykres wektorów uzyskanych za pomocą procedury ***dsolve*** $y_1(t), y_2(t)$, gdzie $y_1(t) = \dot{y}_1, y_2(t) = \dot{y}_2$, jest przedstawiony na rysunku 2



Rysunek 3: Wykres wyników procedury ***ode45***

3.2 Procedura *ode45*

Procedura `ode45` akceptuje jako pierwszy argument funkcję dwóch zmiennych, a dla uzyskania wyniku w naszym przypadku trzeba przekazać wektor składający się z dwóch funkcji dwóch zmiennych. Posługując się dokumentacją MATLABa tworzymy zatem dodatkową funkcję, *rown*, podaną w urywku kodu (1), która będzie przyjmowała dwa wektory i zwracała odpowiednio wektor, zawierający dwie funkcji dwóch zmiennych, i przekazujemy do ***ode45*** uchwyt do niej. W wektor wyjsciowy funkcji *rown* wstawiamy zamiast $x(t)$ bezpośrednio funkcję $exp(-t)sin(t)$. Wykres wyników, gdzie $y_1 = \dot{y_1}, y_2 = \dot{y_2}$ oraz $y1a, y2a$ - przybliżone wartosci rozwiązań **URRZ** (1), uzyskane za pomocą procedury ***ode45***, jest przedstawiony na rysunku 3

(1)

```
function dydt = rown(t,y)
dydt = [
(-26/3)*y(1)+ (-10/3)*y(2) + exp(-t)*sin(t);
(10/3)*y(1) + (-1/3)*y(2) + exp(-t)*sin(t)];
end % function
```

3.3 *Metoda (b)*

Dla zaimplemetowania metody (b) przekształcamy jej wzór, podany w (3.1.1), gdzie (3.1.2)

$$\boldsymbol{y}_n = \boldsymbol{y}_{n-1} + \frac{h}{2}[3\boldsymbol{f}(t_{n-1}, \boldsymbol{y}_{n-1}) - \boldsymbol{f}(t_{n-2}, \boldsymbol{y}_{n-2})] \tag{3.1.1}$$

$$\frac{\partial y(t)}{\partial t}|_{t=t_n} = \boldsymbol{f}(t_n, \boldsymbol{y}_n) \tag{3.1.2}$$

Najpierw *URRZ* (1) przekształcamy do postaci (3.2.1), gdzie (3.2.2)

$$\frac{\partial \boldsymbol{y}(t)}{\partial t} = A\boldsymbol{y}(t) + \boldsymbol{b}x(t) \tag{3.2.1}$$

$$A = \begin{bmatrix} -\frac{26}{3} & -\frac{10}{3} \\ \frac{10}{3} & -\frac{1}{3} \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad \boldsymbol{y}_n = \begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix} \tag{3.2.2}$$

Podstawiając $\boldsymbol{f}(t_n, \boldsymbol{y}_n)$ do wzorów (3.1.2) oraz (3.2.1), dostajemy wzór (3.3.1)

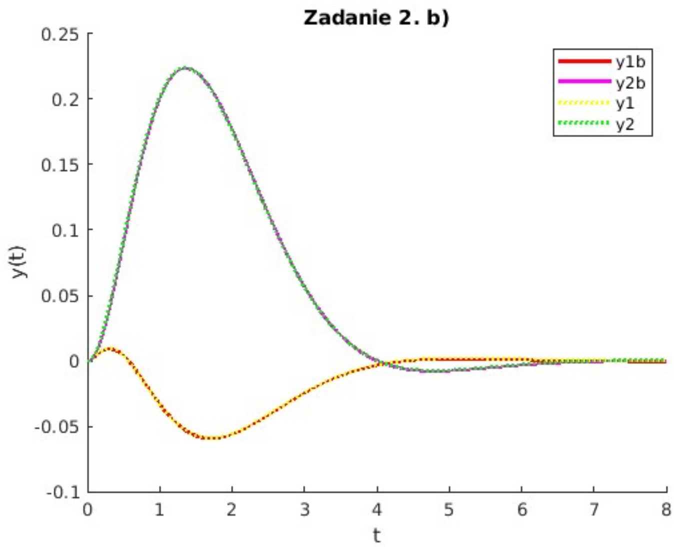
$$\boldsymbol{f}(t_n, \boldsymbol{y}_n) = A\boldsymbol{y}_n + \boldsymbol{b}x(t_n) \tag{3.3.1}$$

Podstawiając otrzymaną postać $\boldsymbol{f}(t_n, \boldsymbol{y}_n)$ we wzór (3.1.1) dostajemy wzór (3.4.3). Proces przekształcania podany we wzorach (3.4.1)-(3.4.3)

$$\boldsymbol{y}_n = \boldsymbol{y}_{n-1} + \frac{h}{2}[\ 3(A\boldsymbol{y}_{n-1} + \boldsymbol{b}x(t_{n-1})) - (A\boldsymbol{y}_{n-2} + \boldsymbol{b}x(t_{n-2}))\ ] \tag{3.4.1}$$

$$\boldsymbol{y}_n = \boldsymbol{y}_{n-1} + \frac{3h}{2}A\boldsymbol{y}_{n-1} + \frac{3h}{2}\boldsymbol{b}x(t_{n-1}) - \frac{h}{2}[A\boldsymbol{y}_{n-2} + \boldsymbol{b}x(t_{n-2})] \tag{3.4.2}$$

$$\boldsymbol{y}_n = (I + \frac{3h}{2}A)\boldsymbol{y}_{n-1} + \frac{3h}{2}\boldsymbol{b}x(t_{n-1}) - \frac{h}{2}[A\boldsymbol{y}_{n-2} + \boldsymbol{b}x(t_{n-2})] \tag{3.4.3}$$



Rysunek 4: Wykres wyników funkcji implemetującej *metodę (b)*

Ponieważ *metoda (b)* jest 2-krokowa, obliczamy $\boldsymbol{y}_2$ za pomocą niejawnej metody Eulera (3.5.1), otrzymując (3.5.2)

$$\boldsymbol{y}_2 = \boldsymbol{y}_1 + h\boldsymbol{f}(t_2, y_2) \tag{3.5.1}$$

$$\boldsymbol{y}_2 = (I - \frac{h}{2}A)\backslash(\boldsymbol{y}_1 + \frac{h}{2}\boldsymbol{b}x(t_2)) \tag{3.5.2}$$

Wykres wyników dla kroku h=0.005, gdzie $y_1 = \dot{y_1}, y_2 = \dot{y_2}$ oraz $y1b, y2b$ - przybliżone wartosci rozwiązań **URRZ** (1), uzyskane za pomocą funkcji *zad2bc* implementującej *metodę (b)*, jest przedstawiony na rysunku 4

3.4 *Metoda (c)*

Przekształcamy wzór *metody (c)* (4.1.1) analogicznie jak w przypadku *metody (b)*, rozwijając $\boldsymbol{f}(t_n, \boldsymbol{y}_n)$. Przekształcenia można prześledzić na wzorach (4.2.1)-(4.2.4)

$$\boldsymbol{y}_n = \boldsymbol{y}_{n-1} + \frac{h}{12}[5\boldsymbol{f}(t_n, \boldsymbol{y}_n) + 8\boldsymbol{f}(t_{n-1}, \boldsymbol{y}_{n-1}) - \boldsymbol{f}(t_{n-2}, \boldsymbol{y}_{n-2})] \tag{4.1.1}$$

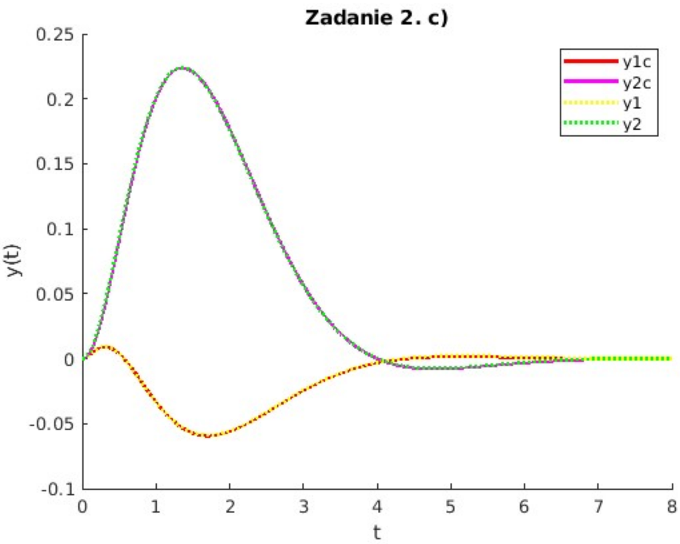
$$\boldsymbol{y}_n = \boldsymbol{y}_{n-1} + \frac{h}{12}[\ 5(A\boldsymbol{y}_n + \boldsymbol{b}x(t_n)) + 8(A\boldsymbol{y}_{n-1} + \boldsymbol{b}x(t_{n-1})) \\ -(A\boldsymbol{y}_{n-2} + \boldsymbol{b}x(t_{n-2}))\ ] \quad (4.2.1)$$

$$\boldsymbol{y}_n - \frac{5h}{12}A\boldsymbol{y}_n = (\boldsymbol{y}_{n-1} + \frac{8h}{12}A\boldsymbol{y}_{n-1}) + \frac{5h}{12}\boldsymbol{b}x(t_n)) + \frac{8h}{12}\boldsymbol{b}x(t_{n-1}) \\ -\frac{h}{12}(A\boldsymbol{y}_{n-2} + \boldsymbol{b}x(t_{n-2})) \quad (4.2.2)$$

$$(I - \frac{5h}{12}A)\boldsymbol{y}_n = (I + \frac{8h}{12}A)\boldsymbol{y}_{n-1} + \frac{5h}{12}\boldsymbol{b}x(t_n)) + \frac{8h}{12}\boldsymbol{b}x(t_{n-1}) \\ -\frac{h}{12}(A\boldsymbol{y}_{n-2} + \boldsymbol{b}x(t_{n-2})) \quad (4.2.3)$$

$$\boldsymbol{y}_n = (I - \frac{5h}{12}A)\backslash[(I + \frac{8h}{12}A)\boldsymbol{y}_{n-1} + \frac{5h}{12}\boldsymbol{b}x(t_n)) + \frac{8h}{12}\boldsymbol{b}x(t_{n-1}) \\ -\frac{h}{12}(A\boldsymbol{y}_{n-2} + \boldsymbol{b}x(t_{n-2}))] \quad (4.2.4)$$

Analogicznie jak w przypadku z *metodą (b)* obliczamy y_2 według wzoru (3.5.2).
Wykres wyników, gdzie $y_1 = \dot{y}_1, y_2 = \dot{y}_2$ oraz $y1c, y2c$ - przybliżone wartosci rozwiązań **URRZ** (1), uzyskane za pomocą funkcji *zad2bc* implementującej *metodę (c)*, jest przedstawiony na rysunku 5



Rysunek 5: Wykres wyników funkcji implemetującej *metodę (c)*

3.5 *Metoda (d)*

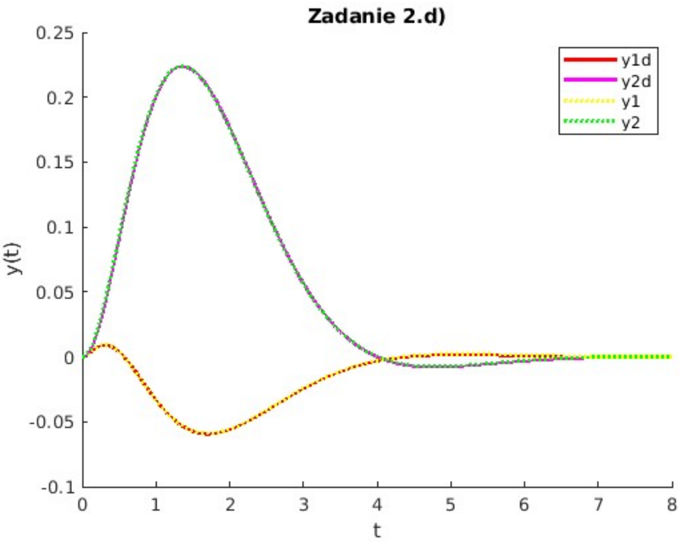
Przekształcamy wzór *metody (d)* (5.1.1). Najpierw rozwijamy $\boldsymbol{f}_k$ w (5.2.1)-(5.2.2)

$$\boldsymbol{y}_n = \boldsymbol{y}_{n-1} + h\sum_{k=1}^{3} w_k \boldsymbol{f}_k \tag{5.1.1}$$

gdzie $\boldsymbol{f}_k = \boldsymbol{f}(t_{n-1} + c_k h, \boldsymbol{y}_{n-1} + h\sum_{K=1}^{3} a_{K,k}\boldsymbol{f}_k)$, a współczynniki $\alpha_{i,i}, \omega_i, c_i$ przyjmują wartości podane w tabeli Butchera na rysunku 1.

$$\begin{cases} \boldsymbol{f}_1 = A(\boldsymbol{y}_{n-1} + h\alpha_{1,1}\boldsymbol{f}_1 + h\alpha_{1,2}\boldsymbol{f}_2 + h\alpha_{1,3}\boldsymbol{f}_3) + \boldsymbol{b}x(t_{n-1} + c_1 h) \\ \boldsymbol{f}_2 = A(\boldsymbol{y}_{n-1} + h\alpha_{2,1}\boldsymbol{f}_1 + h\alpha_{2,2}\boldsymbol{f}_2 + h\alpha_{2,3}\boldsymbol{f}_3) + \boldsymbol{b}x(t_{n-1} + c_2 h) \\ \boldsymbol{f}_3 = A(\boldsymbol{y}_{n-1} + h\alpha_{3,1}\boldsymbol{f}_1 + h\alpha_{3,2}\boldsymbol{f}_2 + h\alpha_{3,3}\boldsymbol{f}_3) + \boldsymbol{b}x(t_{n-1} + c_3 h) \end{cases} \tag{5.2.1}$$

$$\begin{cases} (I - h\alpha_{1,1}A)\boldsymbol{f}_1 + (-h\alpha_{1,2}A)\boldsymbol{f}_2 + (-h\alpha_{1,3}A)\boldsymbol{f}_3 = A\boldsymbol{y}_{n-1} + \boldsymbol{b}x(t_{n-1} + c_1 h) \\ (-h\alpha_{2,1}A)\boldsymbol{f}_1 + (I - h\alpha_{2,2}A)\boldsymbol{f}_2 + (-h\alpha_{2,3})\boldsymbol{f}_3 = A\boldsymbol{y}_{n-1} + \boldsymbol{b}x(t_{n-1} + c_2 h) \\ (-h\alpha_{3,1})\boldsymbol{f}_1 + (-h\alpha_{3,2})\boldsymbol{f}_2 + (I - h\alpha_{3,3}A)\boldsymbol{f}_3 = A\boldsymbol{y}_{n-1} + \boldsymbol{b}x(t_{n-1} + c_3 h) \end{cases} \tag{5.2.2}$$



Rysunek 6: Wykres wyników funkcji implementującej *metodę (d)*

Przekształćmy to do postaci (5.3.1), gdzie (5.3.2) oraz (5.3.3)

$$\boldsymbol{g} = L \backslash \boldsymbol{p} \tag{5.3.1}$$

$$\boldsymbol{g} = \begin{bmatrix} \boldsymbol{f}_1 \\ \boldsymbol{f}_2 \\ \boldsymbol{f}_3 \end{bmatrix}, \quad \boldsymbol{p} = \begin{bmatrix} A\boldsymbol{y}_{n-1} + \boldsymbol{b}x(t_{n-1} + c_1 h) \\ A\boldsymbol{y}_{n-1} + \boldsymbol{b}x(t_{n-1} + c_2 h) \\ A\boldsymbol{y}_{n-1} + \boldsymbol{b}x(t_{n-1} + c_3 h) \end{bmatrix} \tag{5.3.2}$$

$$L = \begin{bmatrix} (I - h\alpha_{1,1}A) & (-h\alpha_{1,2}A) & (-h\alpha_{1,3}A) \\ (-h\alpha_{2,1}A) & (I - h\alpha_{2,2}A) & (-h\alpha_{2,3}) \\ (-h\alpha_{3,1}) & -h\alpha_{3,2}) & (I - h\alpha_{3,3}A) \end{bmatrix} \tag{5.3.3}$$

Wyliczamy $\boldsymbol{g}$ dla zmiennych symbolicznych $x1$, $x2$, $x3$, $yn1$, $yn2$, gdzie $x1 = x(t_{n-1} + c_1 h)$, $x2 = x(t_{n-1} + c_2 h)$, $x3 = x(t_{n-1} + c_3 h)$ oraz
$\boldsymbol{y_{n-1}} = [yn1, yn2]^T$

Następnie przekształcamy nasz wektor **g** za pomocą procedury ***matlabFunction*** Otrzymaliśmy wektor funkcji, które możemy podstawić do wzoru (5.1.1)
Wykres wyników dla h=0.005, gdzie $y_1 = \dot{y}_1, y_2 = \dot{y}_2$ oraz $y1d, y2d$ - przybliżone wartosci rozwiązań **URRZ** (1),uzyskane za pomocą funkcji *zad2d* implementującej *metodę (d)*, jest przedstawiony na rysunku 6

3.6 Badanie zależności dokładności rozwiązań numerycznych uzyskanych za pomocą *metod (b), (c), (d)*

Jako kryterium dokładności rozwiązań przyjmujemy zagregowane błędy względne $\delta_1(h), \delta_2(h)$ zdefiniowane wzorami (6.1)

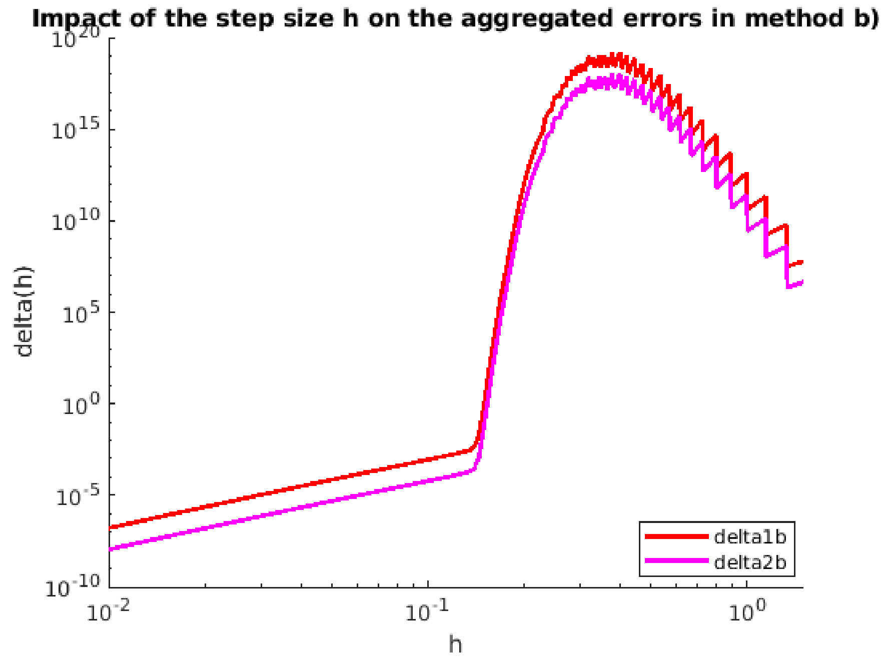
$$\delta_1(h) = \frac{\sum_{n=1}^{N(h)} (\hat{y_1}(t_n, h) - \dot{y_1}(t_n))^2}{\sum_{n=1}^{N(h)} (\dot{y_1}(t_n))^2}, \quad \delta_2(h) = \frac{\sum_{n=1}^{N(h)} (\hat{y_2}(t_n, h) - \dot{y_2}(t_n))^2}{\sum_{n=1}^{N(h)} (\dot{y_2}(t_n))^2} \tag{6.1}$$

Stworzymy też dwie funkcje pomocnicze, *procescurrenth* oraz *countdeltas1*. Funkcja *processcurrenth* przyjmuje bieżącą wartość h oraz $\dot{y_1}, \dot{y_2}$ - wartości dokładne rozwiązania **URRZ** (1) od funkcji głównej, zad3, liczy dla otrzymanego h przybliżone wartości $y_1(t), y_2(t)$ za pomocą *metod (b), (c), i (d)* i przekazuje ich parami do funkcji *countdeltas1*, która z kolei liczy dla przekazanej pary wartości $\delta_1(h), \delta_2(h)$

Wykres wyników dokładności rozwiązań uzyskanych *metodą (b)* dla $h \in [0.1,\ 1.5]$, gdzie $delta1 = \delta_1(h), delta2 = \delta_2(h)$ jest przedstawiony na rysunku 7

Wykres wyników dokładności rozwiązań uzyskanych *metodą (c)* dla $h \in [0.01,\ 1.5]$, gdzie $delta1 = \delta_1(h), delta2 = \delta_2(h)$ jest przedstawiony na rysunku 8

Wykres wyników dokładności rozwiązań uzyskanych *metodą (d)* dla $h \in [0.1,\ 1.5]$, gdzie $delta1 = \delta_1(h), delta2 = \delta_2(h)$ jest przedstawiony na rysunku 9



Rysunek 7: Wykres dokładności rozwiązań numerycznych
dla *metody (b)* w zależności od kroku h

4 Dyskusja wyników eksperymentów numerycznych

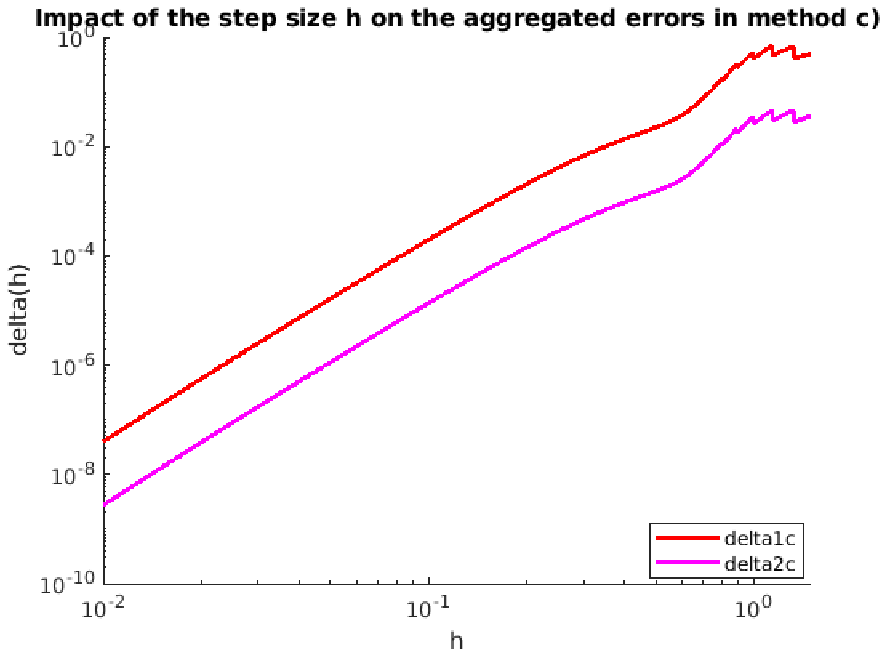
Porównajmy dokładność rozwiązań numerycznych dla małego kroku, jak np h=0.01 w tabelach (4) i (5)

(4)

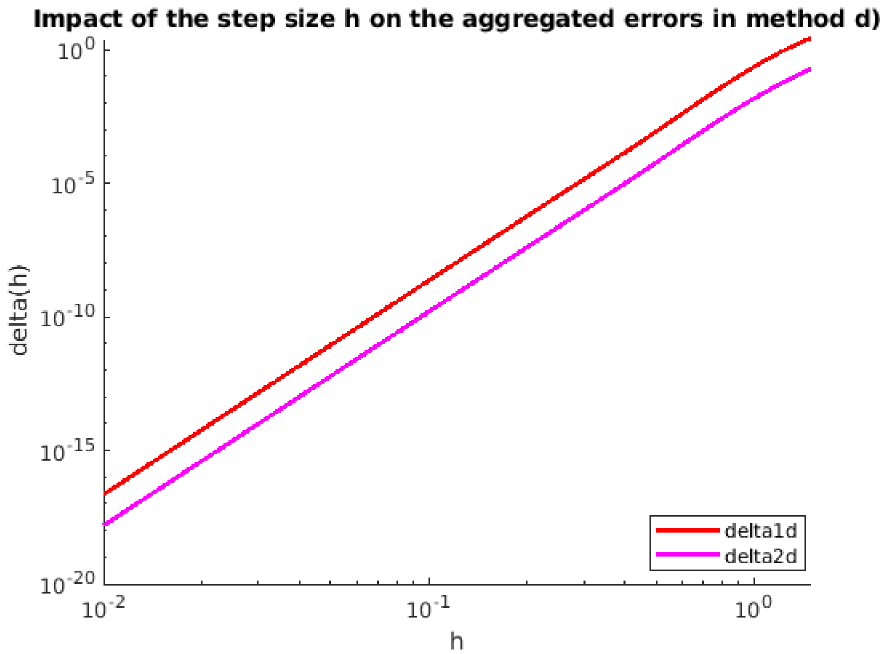
—	Metoda (b)	Metoda (c)	Metoda(d)
$\delta_1(0.01)$	0.0000002	0.00000004	0.000000000000000002
$\delta_1(0.15)$	4.5194916	0.0008	0.000000004
$\delta_1(0.2)$	1085064626320	0.002	0.00000055

(5)

—	Metoda (b)	Metoda (c)	Metoda(d)
$\delta_2(0.01)$	0.000000011	0.000000003	0.000000000000000001
$\delta_2(0.15)$	0.31	0.000057	0.000000062590
$\delta_2(0.2)$	74412125424	0.00013	.00 00000038



Rysunek 8: Wykres dokładności rozwiązań numerycznych
dla *metody (c)* w zależności od kroku h



Rysunek 9: Wykres dokładności rozwiązań numerycznych
dla *metody (d)* w zależności od kroku h

Możemy zaobserwować, że *metoda (b)* traci stabilność w okolicy h=0.15 ale jest stabilna dla wszystkich mniejszych h. *Metoda (c)* traci stabilność w okolicy h=0.6, lecz przy wykorzystaniu *metody (d)* obserwujemy zachowanie stabilności dla kroku tak dużego jak h=1 i nawet h = 1.5, co jest naprawdę imponujące.

5 Wnioski

Z względu na obserwację wykonane w poprzednim rozdziale, możemy wywnioskować, że *metoda (b)* nie jest dobrą metodą na przybliżanie **URRZ** (1) z krokiem większym niż 0.005. Bład rośnie bardzo szybko tak dla *metody (b)*, jak i dla *metody (c)* , chociaż dla *metody(c)* odrobinę wolniej. *Metoda (d)* daje o wiele dokładniejsze wyniki, i też pozwala na mniejszą ilość obliczeń potrzebnych do uzyskania zadowalającego wyniku (dostaniemy wynik o dobrej dokładności nawet jeśli weźmiemy mniejszą liczbę przedziałów) w porównaniu z *metodami (b)* oraz *(c)*.

6 Listing programów

6.1 zad1.m

```
function [y1sol, y2sol] = zad1()
% Solving a system of differential equations provided below with built-in
% matlab function dsolve
% dy_1(t)/dt = ( -26/3)y_1(t) - ( 10/3)y_2(t) + x(t)
% dy_2(t)/dt = ( 10/3) y_1(t) - ( 1/3)y_2(t) + x(t)
% over the interval [0,8], where x(t) = exp(-t)sin(t),
% for zero initial conditions
% INPUT: NONE
% OUTPUT:
%   y1sol, y2sol - vectors containing solutions y_1 and y_2 found by dsolve
%                  over the interval [0,8]

syms t y1(t) y2(t);
x = exp(-t)*sin(t);
eqns = [ diff(y1, t) == -26/3*y1 - 10/3*y2 + x,...
    diff(y2, t) == 10/3*y1 - 1/3*y2 + x];
conds = [y1(0) == 0, y2(0) == 0];
[y1sol(t), y2sol(t)] = dsolve(eqns, conds);

end % function
```

6.2 zad2a.m

```
function [t,y] = zad2a()
% Solving a system of differential equations provided below with built-in
% matlab function ode45
% dy_1(t)/dt = ( -26/3)y_1(t) - ( 10/3)y_2(t) + x(t)
% dy_2(t)/dt = ( 10/3) y_1(t) - ( 1/3)y_2(t) + x(t)
% over the interval [0,8], where x(t) = exp(-t)sin(t),
% for zero initial conditions
```

```
% INPUT: NONE
% OUTPUT:
%   y1sol, y2sol - vectors containing solutions y_1
   and y_2 found by dsolve
%                  over the interval [0,8]

tspan = [0 8];
[t,y] = ode45(@rown,tspan,zeros(2,1));

end % function
```

6.3 rown.m

```
function dydt = rown(t,y)
% Packing a system of differential equations
   provided below
% dy_1(t)/dt = ( -26/3)y_1(t) - ( 10/3)y_2(t) + x(t)
% dy_2(t)/dt = ( 10/3) y_1(t) - ( 1/3)y_2(t) + x(t)
% into a single vector
% INPUT: NONE
% OUTPUT:
%   dydt - vector that contains the system of
   differential equations,
%           provided in the desciption of the
   function

dydt = [
    (-26/3)*y(1)+ (-10/3)*y(2) +  exp(-t)*sin(t);
    (10/3)*y(1) + (-1/3)*y(2) +  exp(-t)*sin(t)];

end % function
```

6.4 zad2bc.m

```
function[y1,y2] = zad2bc(h)
% Solving a system of differential equations
   provided below
% dy_1(t)/dt = ( -26/3)y_1(t) - ( 10/3)y_2(t) + x(t)
```

```
% dy_2(t)/dt = ( 10/3) y_1(t) - ( 1/3)y_2(t) + x(t)
% over the interval [0,8], where x(t) = exp(-t)sin(t),
% for zero initial conditions
% using methods defined by formula b) and c)
% b) Y_n = Y_{n-1} + (h/2)[ 3f(t_{n-1},Y_{n-1}) - f(t_{n-2},Y_{n-2}) ]
% c) Y_n = Y_{n-1} + (h/12)[ 5f(t_n,Y_n) + 8f(t_{n-1},Y_{n-1})
%                                                 - f(t_{n-2},Y_{n-2}) ]
% where Y_n = [ y_1(t_n), y_2(t_n)]'
% and f is defined by formula  (t=t_n)=> dY(t)/dt = f(t_n, Y_n);
% INPUT :
%   h        - size of the step
% OUTPUT:
%   y1, y2   - are vectors [y_1, y_2]', containg y_1 and y_2 calculated
%                using method b) and c) accordingly

tspan = [0 8];
t = tspan(1):h:tspan(2); % time vector t
x = exp(-t).*sin(t);
n = length(t);
A = [-26/3, -10/3;
     10/3, -1/3];
b = ones(2,1);
y1 = zeros(2,n);
y2 = zeros(2,n);
y1(:,2) = (eye(2) -h*A)\(y1(:,1) + h*b*x(2)); % b)
y2(:,2) = (eye(2) -h*A)\(y2(:,1)+h*b*x(2)); % c)
for i = 3:n
    % b)
    y1(:,i) = (eye(2) + (3*h/2)*A)*y1(:,i-1) + b*x(i-1)*(3*h/2) - ...
        (h/2)*( A*y1(:,i-2) + b*x(i-2));
    % c)
    y2(:,i) = (eye(2) -(5*h/12)*A)\...
        ( (eye(2) + (8*h/12)*A)*y2(:,i-1) + (5*h/12)*b*x(i) + ...
```

```
        (8*h/12)*b*x(i-1) - (h/12)*(A*y2(:,i-2) + b*
          x(i-2)) );
end

end % function
```

6.5 zad2d.m

```
function y = zad2d(h)
% Solving a system of differential equations
  provided below
% dy_1(t)/dt = ( -26/3)y_1(t) - ( 10/3)y_2(t) + x(t)
% dy_2(t)/dt = ( 10/3) y_1(t) - ( 1/3)y_2(t) + x(t)
% over the interval [0,8], where x(t) = exp(-t)sin(t
  ),
% for zero initial conditions
% using the method defined by formula d)
% d) Y_n = Y_{n-1} + h    {k=1...3} w_k*f_k,
% where Y_n = [ y_1(t_n), y_2(t_n)]' ,
% f_k = f(t_{n-1} + c_k*h, y_{n-1} + h*   {K=1...3}
  a_{k,K}f_K ,
% coefficients w_*, a_{*,*} and c_* are taken from
  the Butcher's table
% and f is defined by formula  (t=t_n)=> dY(t)/dt =
  f(t_n, Y_n)
%
% Butcher's table
%  c_1 | a_{1,1} | a_{1,2} | a{1,3}
%  c_1 | a_{2,1} | a_{2,2} | a{2,3}
%  c_1 | a_{3,1} | a_{3,2} | a{3,3}
%  --------------------------------
%      |   w_1   |   w_2   |   w_3
%
%              |    |
%              v    v
%
%   0   | 1/6 | -1/6 |  0
%  1/2  | 1/6 |  1/3 |  0
%   1   | 1/6 |  5/6 |  0
```

```
%   ------------------------
%      | 1/6 |   2/3 |  1/6
%
% INPUT :
%    h   - size of the step
% OUTPUT:
%    y1 - horizontal vector [y_1, y_2]', containg y_1
    and y_2
%          calculated  using method d)

% Creating a matrix of coefficients
al = [ 1/6, -1/6, 0;
    1/6, 1/3, 0;
    1/6, 5/6, 0];
A = [ -26/3, -10/3;
    10/3, -1/3 ];
I =  eye(2); % identity matrix
b = ones(2,1);
syms x1 x2 x3 yn1 yn2;
L = [ (I-h*al(1,1)*A), -h*al(1,2)*A, -h*al(1,3)*A;
    -h*al(2,1)*A, (I-h*al(2,2)*A), -h*al(2,3)*A;
    -h*al(3,1)*A, -h*al(3,2)*A, (I-h*al(3,3)*A) ];
p = [ (A*[yn1;yn2]) + b*x1;
    (A*[yn1;yn2]) + b*x2;
    (A*[yn1;yn2]) + b*x3;];
g = L\p;
% Main part of the task
tspan = [0,8];
t = tspan(1):h:tspan(2);
x = @(t) exp(-t).*sin(t);
N = length(t);
y = zeros(2,N);
w = [1/6, 2/3, 1/6];
gn = matlabFunction(g);

for i = 2:N
    gi = gn(x(t(i-1)), x(t(i-1)+1/2*h), x(t(i)), y
        (1,i-1), y(2,i-1));
    y(:,i) = y(:,i-1) + h*(w(1)*gi(1:2) +w(2)*gi
        (3:4) + w(3) * gi(5:6));
end
```

```

end % function
```

6.6 plotter.m

```
function plotter()
% Comparing the solutions of the system of differential equations
% provided below
% dy_1(t)/dt = ( -26/3)y_1(t) - ( 10/3)y_2(t) + x(t)
% dy_2(t)/dt = ( 10/3) y_1(t) - ( 1/3)y_2(t) + x(t)
% where x(t) = exp(-t)sin(t), over the interval [0,8]
% for zero initial conditions.
% The solutions are calculated by functions:
% zad1 (being referred to as dsolve in the code below),
% zad2a (being referred to as ode45 in the code below)
% zad2bc ( being referred to as 'method b)' and 'method c)' in code below)
% for more details visit function's specification
% zad2d (being referred to as 'method d)' in the code below),
% INPUT: NONE
% OUTPUT: NONE

% Calculating solutions
h = 0.005; % step
[y1sol,y2sol] = zad1(); % dsolve, considered an 'exact' solution
y_d = zad2d(h); % method d)
[t_a,y_a] = zad2a(); % ode45
[y_b,y_c] = zad2bc(h); % methods b) and c)
% creating a time vector
tspan = [0 8];
t = tspan(1):h:tspan(2);
% plotting
% ode45
```

```
help_plotter(1, t_a, t, y1sol, y2sol, y_a', "ode45",
    ["y_1ode", "y_2ode"]);
% method b)
help_plotter(2, t, t, y1sol, y2sol, y_b, "Method b)
   ", ["y_1b","y_2b"]);
% method c)
help_plotter(3, t, t, y1sol, y2sol, y_c, "Method c)
   ", ["y_1c","y_2c"]);
% method d)
help_plotter(4, t, t, y1sol, y2sol, y_d, "Method d)
   ", ["y_1d","y_2d"]);
% dsolve
figure(5)
plot(t, y1sol(t),"y", t, y2sol(t),"g",'LineWidth',2
   );
legend('y1(t)', 'y2(t)');
xlabel('t');
ylabel('y(t)');

end % function
```

6.7 help_plotter.m

```
function help_plotter(f_ind, tmethod, t, y1, y2,
   ymethod, mtitle, slegend)
% Plotting an approximated solution of the system of
    differential equations
% provided below
% dy_1(t)/dt = ( -26/3)y_1(t) - ( 10/3)y_2(t) + x(t)
% dy_2(t)/dt = ( 10/3) y_1(t) - ( 1/3)y_2(t) + x(t)
% where x(t) = exp(-t)sin(t), over the interval
   [0,8]
% for zero initial conditions,
% in order to compare with exact solution
% INPUT:
%   f_ind    - figure index
%   tmethod - time vector for appproximated
   solutions
%   t        - time vector for exact solutions
```

```
%   y1, y2  - horizontal vectors containing exact solutions
%   ymethod - array consisting of approximated solutions of y1 and y2
%             accordingly
%   mtitle  - title for the plot, for example name of the method used
%             to calculate approximated solutions
%   slegend - vector consisting of legend for approximated solutions
%             of y1 and y2 accordingly
% OUTPUT: NONE

figure(f_ind)
hold on;
plot(tmethod, ymethod(1,:),"r", tmethod ,ymethod(2,:),"m", "LineWidth",2);
plot(t, y1(t),":y", t, y2(t),":g",'LineWidth',2 );
legend(slegend(1), slegend(2), 'y1', 'y2');
xlabel('t');
ylabel('y(t)');
title(mtitle);

end % function
```

6.8 zad3.m

```
function zad3()
% Examining the impact of the step size h on the aggregated errors delta_1
% and delta_2 defined by the formulas:
% delta_1(h) = [    {n=1...N(h)} (yhat1(t_n,h) - ydot1(t_n))^2 ] /
%                                        [    {n=1..N(h)} ( ydot1(t_n)^2) ]
%
% delta_2(h) = [    {n=1...N(h)} (yhat2(t_n,h) - ydot2(t_n))^2 ] /
```

```
%                                                     [     {n=1..N(h)} ( ydot2(t_n)^2) ]
%
% where ydot1 and ydot2 are exact solutions of the system of differential
% equations provided below, solved using built-in matlab function dsolve
% dy_1(t)/dt = ( -26/3)y_1(t) - ( 10/3)y_2(t) + x(t)
% dy_2(t)/dt = ( 10/3) y_1(t) - ( 1/3)y_2(t) + x(t)
% over the interval [0,8], where x(t) = exp(-t)sin(t),
% for zero initial conditions
%
% and yhat1 and yhat2 are their approximations accordingly
%
% INPUT: NONE
% OUTPUT: NONE

% setting the limits for step step sizes to be examined
hmin = 0.01;
hmax = 1.5;
% creating a vector of the step sizes
h = linspace(hmin, hmax, 300);
n = length(h);
% extracting 'exact' solution, ydot1 and ydot2
[ydot1,ydot2] = zad1;
% preallocating space
deltasb = zeros(2,n);
deltasc = zeros(2,n);
deltasd = zeros(2,n);
for i = 1:n
    currdeltas = procescurrenth(h(i), ydot1, ydot2);
    deltasb(:,i) = currdeltas(1,:);
    deltasc(:,i) = currdeltas(2,:);
    deltasd(:,i) = currdeltas(3,:);
end
% plotting
% method b)
help_zad3(1,h,deltasb, "b");
```

```
% method c)
help_zad3(2, h, deltasc, "c");
% method d)
help_zad3(3, h, deltasd, "d");

end % function
```

6.9 procescurrenth.m

```
function deltas = procescurrenth(h, ydot1, ydot2)
% calculating approximated solutions of the system of differential
% equations provided below:
% dy_1(t)/dt = ( -26/3)y_1(t) - ( 10/3)y_2(t) + x(t)
% dy_2(t)/dt = ( 10/3) y_1(t) - ( 1/3)y_2(t) + x(t)
% over the interval [0,8], where x(t) = exp(-t)sin(t),
% for zero initial conditions
% using ode45 and other methods ( b), c) and d),
% for more details see function descriptions of zad2bc and zad2d)
% and calculating their aggregated errors delta_1 and delta_2
% (for formulas see zad3)
% INPUT:
%   h              - current step size
%  ydot1, ydot2 - vectors of 'exact' solutions of the above's system of
%                  differential equations
% OUTPUT:
%   deltas       - vertical vector consisting of three sets of
%                  [delta_1, delta_2] calculated for methods b), c)
%                  and d) accordingly

tspan = [0,8];
t = tspan(1):h:tspan(2);
y1 = double(ydot1(t));
```

```
y2 = double(ydot2(t));
y1sum = sum(y1.*y1);
y2sum = sum(y2.*y2);
[ybhat, ychat] = zad2bc(h); % method b) and method c)
ydhat = zad2d(h); % method d)
deltas = zeros(3,2);
deltas(1,:) = countdeltas1(ybhat,y1,y2);
deltas(2,:) = countdeltas1(ychat,y1,y2);
deltas(3,:) = countdeltas1(ydhat,y1,y2);
deltas(:,1) = deltas(:,1)/y1sum;
deltas(:,2) = deltas(:,2)/y2sum;

end % function
```

6.10 countdeltas.m

```
function [delta1,delta2] = countdeltas1(yhat,y1,y2)
% Partly calculating the values of aggregated errors delta_1
% and delta_2 defined by the formulas:
% delta_1(h) = [    {n=1...N(h)} (yhat1(t_n,h) - ydot1(t_n))^2 ] /
%                                          [    {n=1..N(h)} ( ydot1(t_n)^2) ]
%
% delta_2(h) = [    {n=1...N(h)} (yhat2(t_n,h) - ydot2(t_n))^2 ] /
%                                          [    {n=1..N(h)} ( ydot2(t_n)^2) ]
%
% where ydot1 and ydot2 are exact solutions of a certain system
% of differential equations
% and yhat1 and yhat2 are their approximations accordingly
% INPUT:
%  yhat           - array: [yhat1;yhat2] consisting of the approximated
```

```
%                              solutions of a certain system of differential equations
%  y1, y2          - vectors of exact solution
% OUTPUT:
%  delta1, delta2 - partly calculated vectorsof delta_1 and delta_2
%                              accordingly

s11 = (yhat(1,:) - y1);
s11 = sum(s11.*s11);
delta1 = s11;
s21 = yhat(2,:) - y2;
s21 = sum(s21.*s21);
delta2 = s21;

end % function
```

6.11 help_zad3.m

```
function help_zad3(f_ind, H, deltasm, mn)
% Plotting the impact of the step size h on the aggregated errors delta_1
% and delta_2 defined by the formulas:
% delta_1(h) = [    {n=1...N(h)} (yhat1(t_n,h) - ydot1(t_n))^2 ] /
%                                                   [    {n=1..N(h)} ( ydot1(t_n)^2) ]
%
% delta_2(h) = [    {n=1...N(h)} (yhat2(t_n,h) - ydot2(t_n))^2 ] /
%                                                   [    {n=1..N(h)} ( ydot2(t_n)^2) ]
% where ydot1 and ydot2 are exact solutions and yhat1 and yhat2 are their
% approximations accordingly
% INPUT:
%   f_ind    - figure index
%   H        - vector of step sizes
```

```
%   deltasm - array consisting of two vectors of aggregated error values,
%               delta1 and delta2, where
%               deltai(j) corresponds to the step size h(j)
%               for i = 1,2 and j = 1,...,length(h)
%   mn      - string consisting of a name of the method used to calculate
%               yhat1 and yhat2
% OUTPUT: NONE

figure(f_ind)
clf;
hold on;
plot(H,deltasm(1,:),"r",H,deltasm(2,:), "m","LineWidth",2);
title("Impact of the step size h on the aggregated errors in method " +...
    mn +")");
legend("delta1" + mn, "delta2"+ mn);
xlabel("h");
ylabel("delta(h)");
xscale('log')
yscale('log')
```