Mariusz Paluch

Sprawozdanie projekt nr 3 MNUM

1 Opis teoretyczny

Celem projektu jest obliczenie przebiegu trajektorii ruchu tego punktu na określonym przedziale za pomocą metody Rungego-Kutty trzeciego rzędu przy zmiennym kroku z szacowaniem błędu metodą zdwajania kroku.

Metoda Rungego-Kutty trzeciego rzędu można zdefiniować następującym wzorem:

$$y_{n+1}=y_n+h\sum_{i=1}^{3} w_i k_i$$

gdzie

$$k_1=f(x_n,y_n)$$
$$k_i=f(x_n+c_i h, y_n+h\sum_{j=1}^{i-1} a_{ij}k_j), dla \;\; i=2,3$$

przy czym $\sum_{j=1}^{i-1} a_{ij}=c_i, \;\; i=2,3$

Wyjaśnienie oznaczeń:

- y_n - aproksymacja funkcji dla punktu n
- $f(x_n,y_n)$ - pochodna funkcji dana równaniem różniczkowym
- h - krok metody

Używane współczynniki:

$$a=\begin{pmatrix} 0 & 0 & 0 \\ 1/2 & 0 & 0 \\ -1 & 2 & 0 \end{pmatrix}$$

$$w=[1/6 \;\; 2/3 \;\; 1/6]$$

$$c=\begin{pmatrix} 0 \\ 1/2 \\ 1 \end{pmatrix}$$

Aby wprowadzać korekty długości kroku, należy dobrze oszacować wartość błędu. W tym celu korzystam z zasady podwójnego kroku.

Zakładając identyczny błąd aproksymacji w każdym z dwóch kroków pomocniczych oznaczając przez p rząd rozważanej metody, $y_n^{(1)}$ jako punkt uzyskany po jednym kroku o długości 2h, oraz $y_n^{(2)}$ jako punkt uzyskany po dwóch krokach o długości h mamy:

$y(x_n + 2h) = y_n^{(1)} + \gamma (2h)^{p+1} + O(h^{p+2})$ - po kroku pojedynczym
$y(x_n + 2h) \approx y_n^{(2)} + 2\gamma h^{p+1} + O(h^{p+2})$ - po kroku podwójnym
gdzie
$\gamma = \frac{r_n^{p+1}(0)}{(p+1)!}$

Po wyznaczeniu nieznanego współczynnika γ z pierwszego równania i wstawieniu go do drugiego, otrzymamy:

$$y(x_n + 2h) = y_n^{(2)} + \frac{y_n^{(2)} - y_n^{(1)}}{2^p - 1} + O(h^{p+2})$$

Z czego wynika oszacowanie błędu dwóch kolejnych kroków o długości h:

$$\delta_n(2h) = \frac{y_n^{(2)} - y_n^{(1)}}{2^p - 1}$$

Postać ogólna na część główną błędu to:

$$\delta(2h) = \gamma (2h)^{p+1}$$

Gdy zmieniamy krok z h na αh, otrzymamy:

$$\delta_n(\alpha h) = \gamma (2\alpha h)^{p+1}$$

Co można zapisać jako:

$$\delta_n(\alpha h) = \alpha^{p+1} \delta_n(2h)$$

Założenie dokładności obliczeń na wartości ϵ oznacza

$$\epsilon = |\delta_n(\alpha 2h)|$$

Skąd dostajemy współczynnik modyfikacji kroku α

$$\alpha = \left(\frac{\epsilon}{\delta_n(2h)}\right)^{\frac{1}{p+1}}$$

Otrzymujemy zatem wzór:

$h_{n+1}=s\alpha h_n$ gdzie s to współczynnik bezpieczeństwa, zawsze mniejszy od jedynki.

Ponadto, parametr dokładności obliczeń ϵ (który jest parametrem użytkownika) określa się na ogół następująco:

$$\epsilon=|y_n|\epsilon_w+\epsilon_b$$

gdzie
ϵ_w - dokładność względna
ϵ_b - dokładność bezwzględna

Dla układu k równań przyjmuje się współczynnik wyliczony dla najbardziej krytycznego równania, czyli:

$$\alpha=\min_{1<i<k}\left(\frac{\epsilon_i}{|(\delta_n(2h_i))|}\right)^{\frac{1}{p+1}}$$

Limit zmniejszania kroku znajduje się w parametrze h_{min}
Program sprawdza też czy krok nie zwiększa się bardziej niż β - krotnie, oraz sprawdza też, czy krok nie przejdzie przez koniec przedziału.

2. Parametry programu

$X(0)=(0,0.3)$ - punkt początkowy w chwili $t=0$
$t\in[0,20]$ - rozpatrywany przedział czasowy
$h_{start}=10^{-4}$ - początkowy parametr kroku
$h_{min}=10^{-9}$ - minimalna wartość parametru kroku. Zabezpiecza algorytm przed stagnacją i utknięciem w jednym punkcie.
$\varepsilon_w=10^{-3}$ - dokładność względna – zbyt wysoka wartość sprawia, że algorytm jest szybszy, ale trajektoria staje się mniej dokładna, zbyt niska sprawia, że algorytm działa dłużej, niekoniecznie wpływając na lepsze działanie
$\varepsilon_b=10^{-3}$ - dokładność bezwzględna – zbyt niska wartość sprawia, że parametr kroku zaczyna znacząco oscylować
$s=0.9$ - współczynnik bezpieczeństwa
$\beta=5$ - współczynnik ograniczający maksymalny nowy krok do beta-krotności

3. Kod programu

3.1 Funkcja solwera

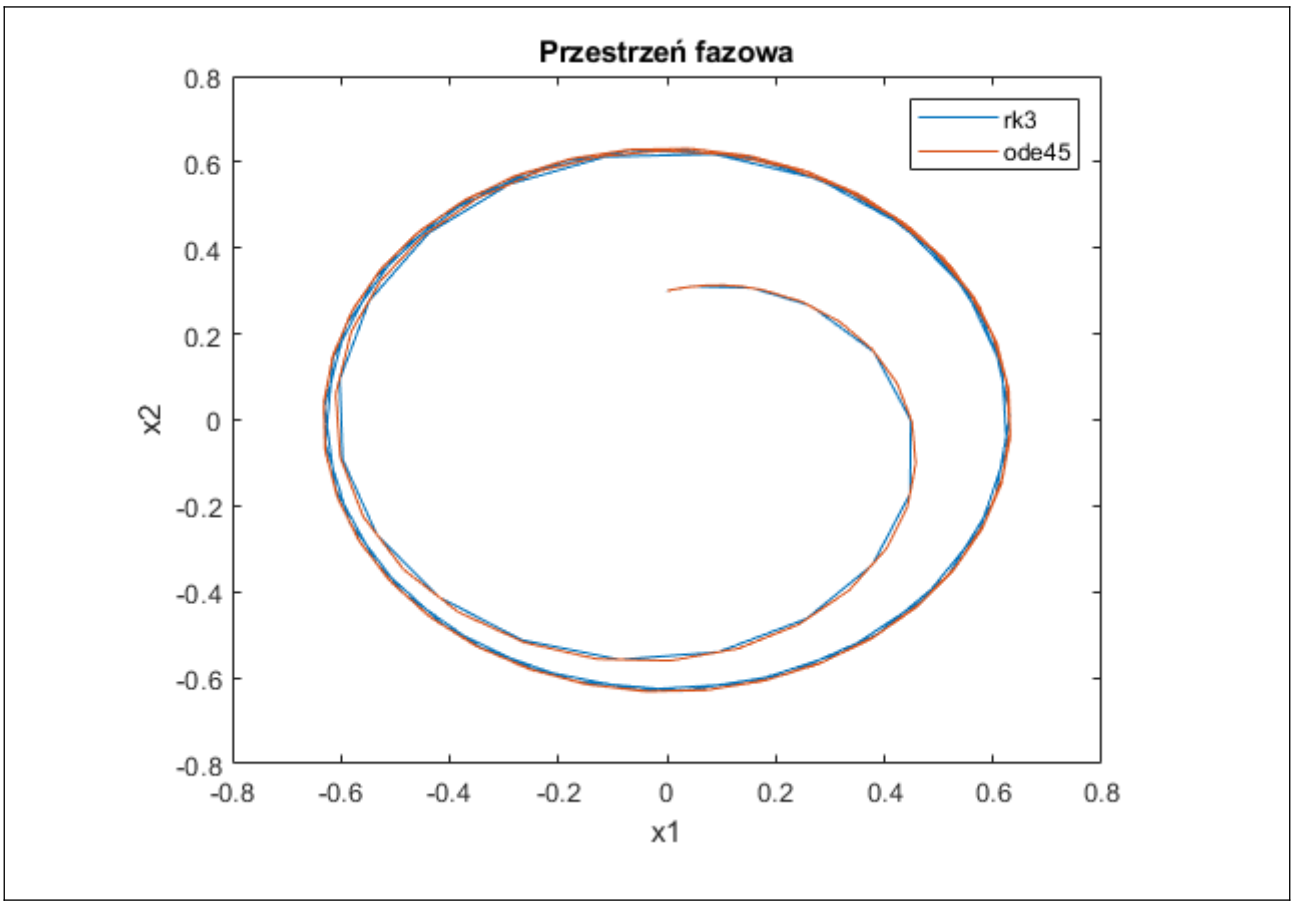
```
function [x, t, h, delta] = rk3(x_start, h_start, t_start, t_end, f, w_epsilon, b_epsilon, h_min)
```

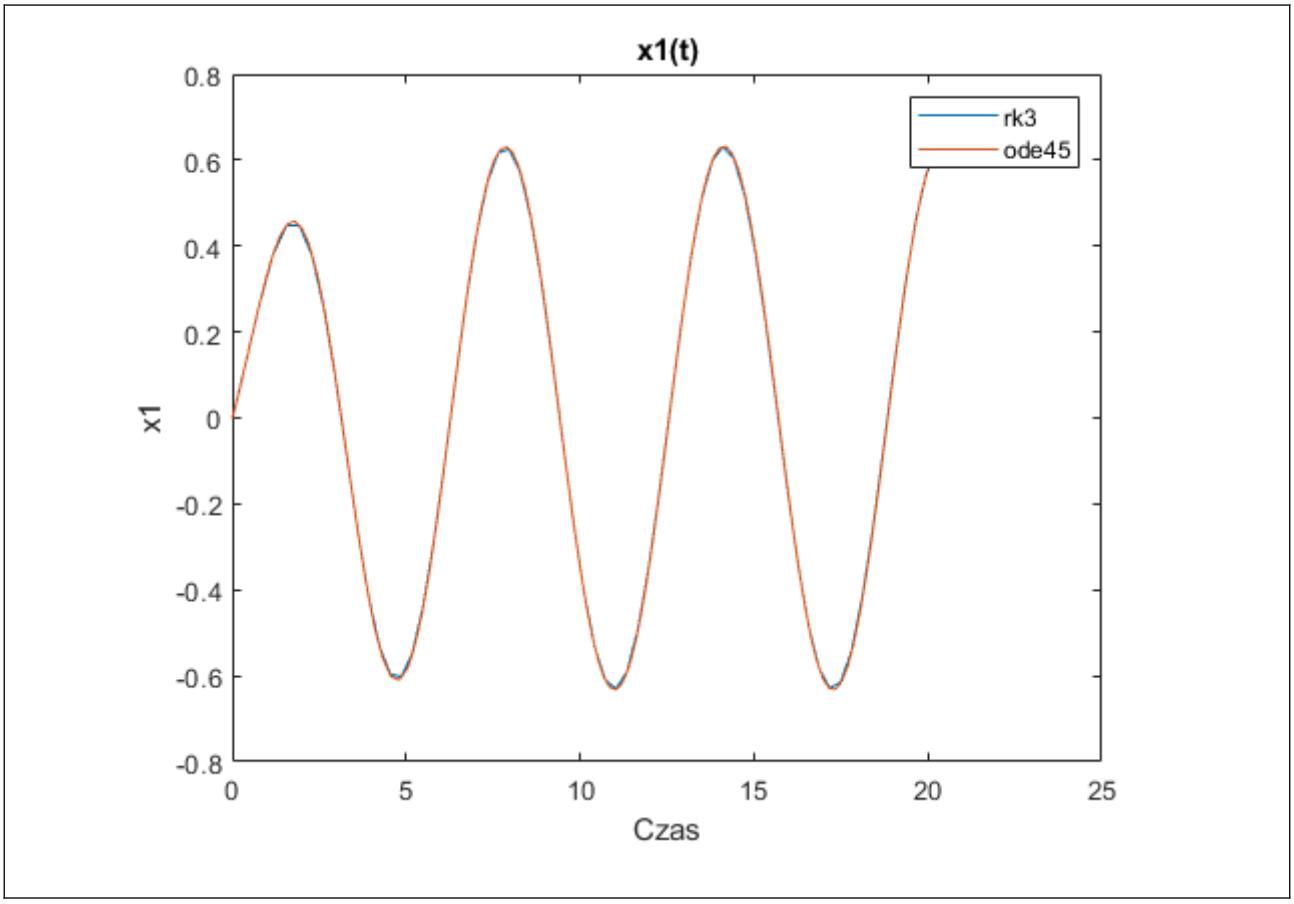
```
w = [1/6 2/3 1/6];

x = zeros(1, length(x_start));
delta = zeros(1, length(x_start));
x(1, : ) = x_start;
h(1) = h_start;
t(1) = t_start;
s = 0.9;
beta = 5;
i = 1;

while t(i) < t_end
   i = i + 1;
   t(i) = t(i-1) + h(i-1);
   x(i, :) =  x(i-1, :) + h(i-1)*w*calc_k(x(i-1,:), t(i-1), h(i-1), f);
   if i > 2 && mod(i, 2)
      x2 = x(i-2, :) + 2*h(i-2)*w*calc_k(x(i-2,:), t(i-2), 2*h(i-2), f);
      delta(i, :) = (x(i, :) - x2) / 7;
      eps = abs(x(i, :)) * w_epsilon + b_epsilon;
      alfa = min((eps./abs(delta(i, :))).^(1/4));
      h_next = s*h(i-1)*alfa;
      if h_next < h_min
         fprintf("Niemożliwe rozwiązanie z zadaną dokładnością")
         return
      end
      h(i) =  min( [h_next, beta*h(i - 1), t_end - t(i - 1)] );
   else
      h(i) = h(i-1);
      delta(i, :) = delta(i - 1, :);
   end
end

end
```

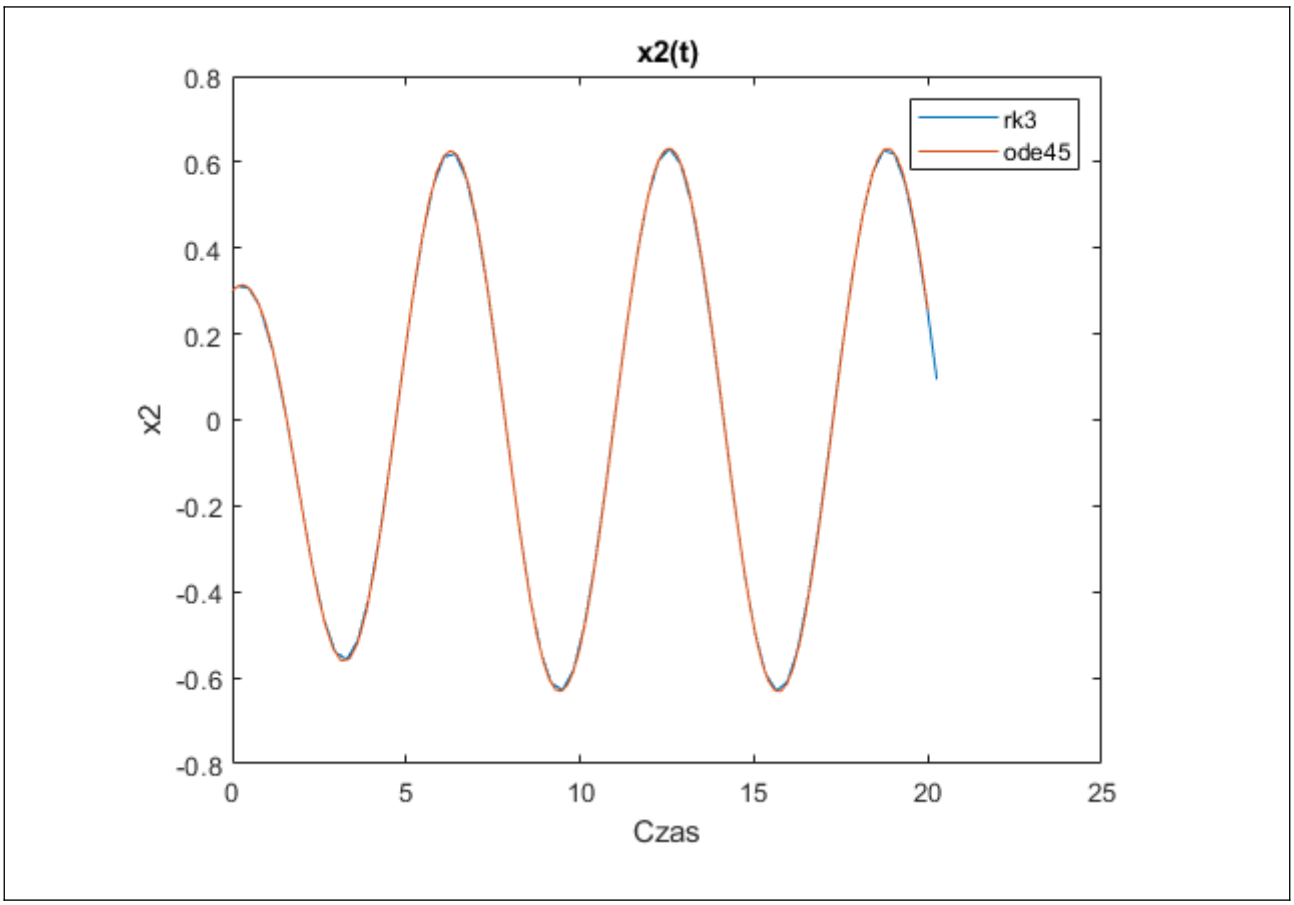
3.2 Kod funkcji obliczającej zadane k

```
function [k] = calc_k(x, t, h, f)
   a = [0 0 0;
        1/2 0 0;
        -1 2 0];

   c = [0 1/2 1]';

   k = zeros(3, length(x));

   k(1, : ) = f(t, x);
   k(2, : ) = f(t*c(2)*h, x + h*a(2, 1)*k(1, :));
   k(3, : ) = f(t*c(3)*h, x + h*a(3, 1:2)*k(1:2, :));
end
```

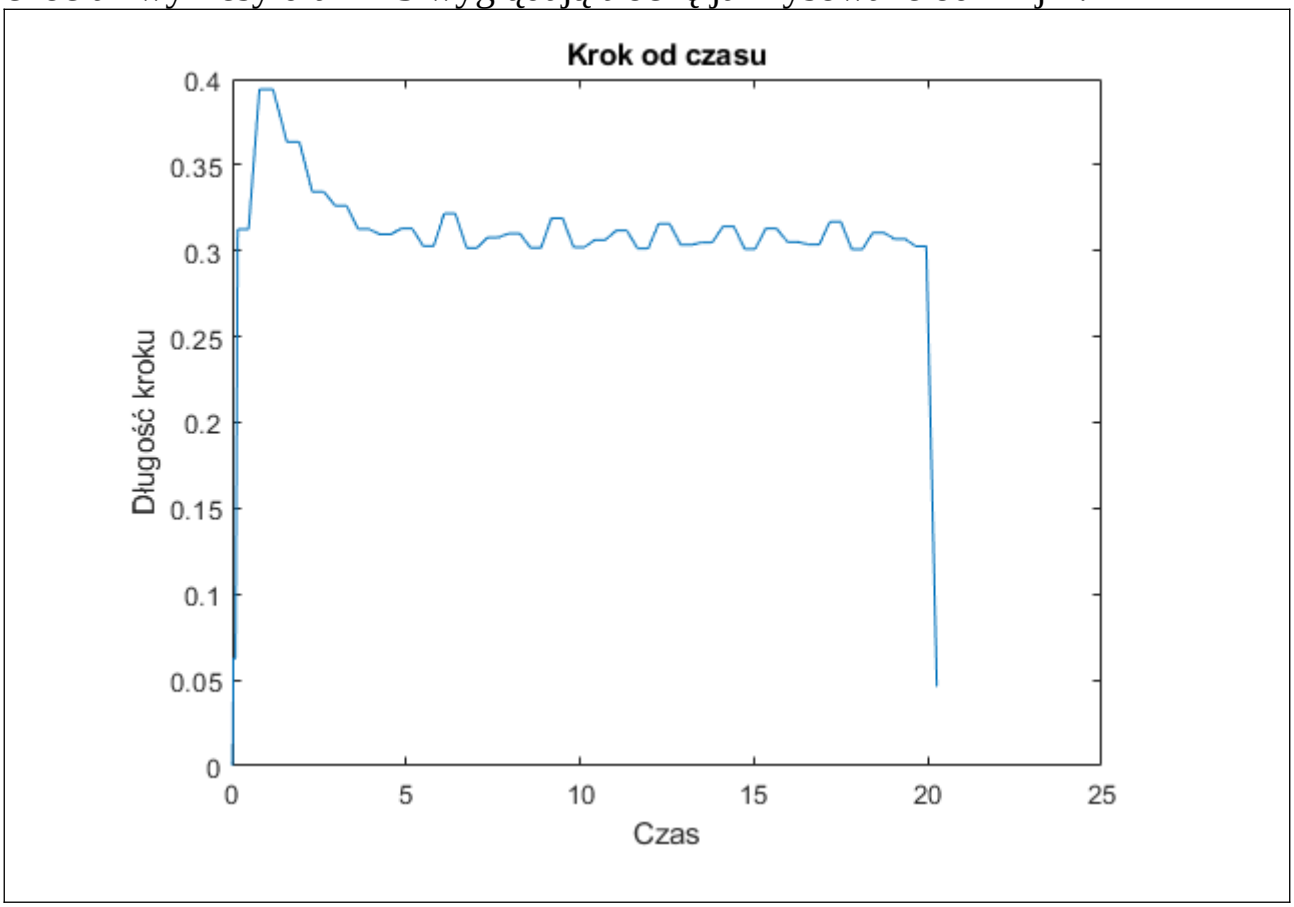
4. Wyniki


Przestrzeń fazowa
rk3
ode45
x2
x1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8

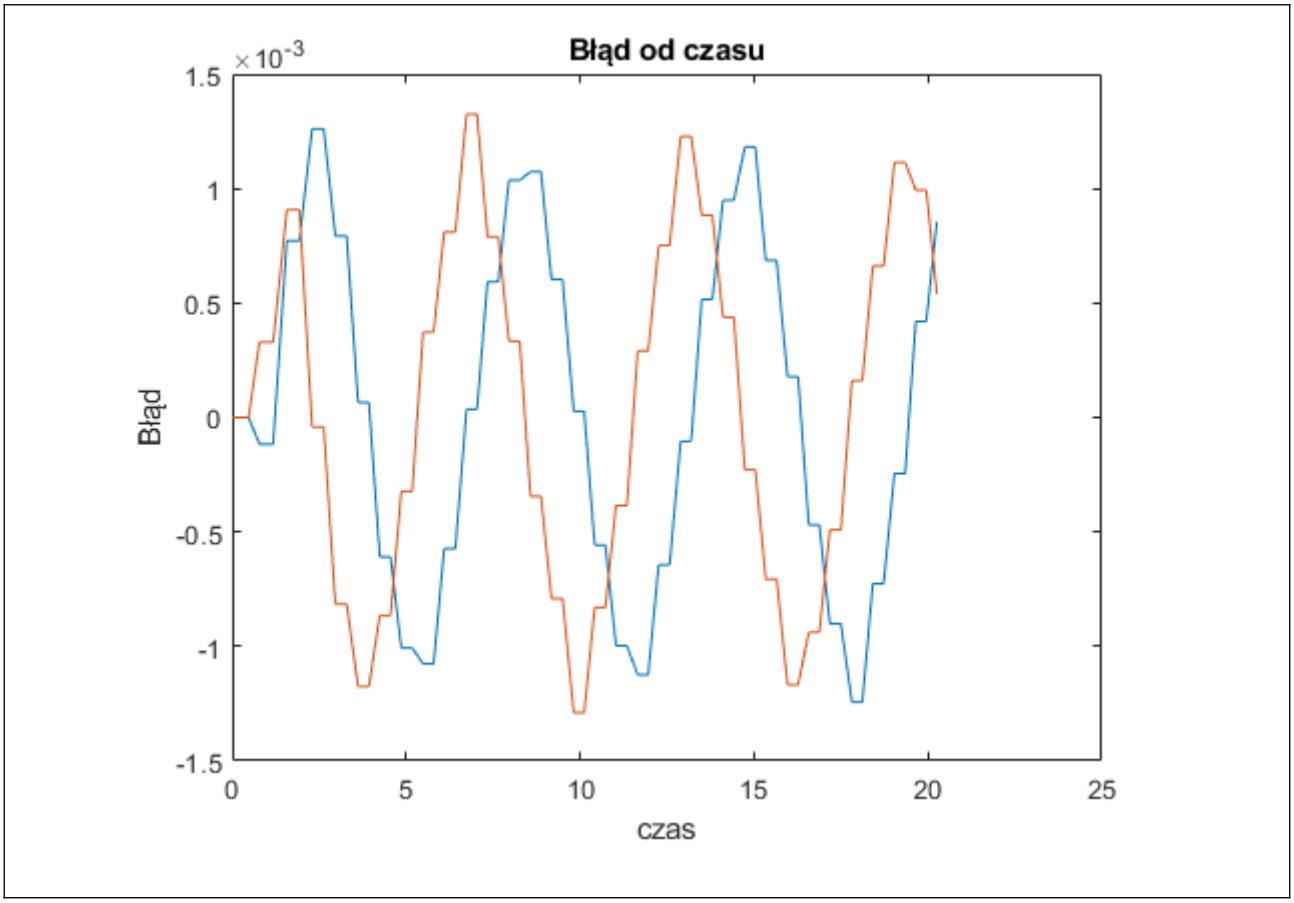

x1(t)
rk3
ode45
x1
Czas
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
0
5
10
15
20
25



Jak widać własna implementacja RK3 dotrzymuje kroku solverowi ODE45, chociaż wykresy dla RK3 wyglądają trochę jak rysowane od linijki.



Na wykresie długości kroku od czasu widać na początku gwałtowny wzrost, zatrzymany przez współczynnik β. Następnie algorytm wpada w pewne oscylacje, jednakże nie są na tyle znaczące, aby zaburzyć działanie algorytmu. Na samym końcu krok jest gwałtownie zmniejszany, ponieważ umożliwia to nie przekroczenie badanego zakresu.



Wykresy błędów dla współrzędnych wyraźnie oscylują. Wynika to z tego, że gdy długość kroku zaczyna za bardzo rosnąć/spadać, algorytm musi zacząć reagować dostosowywaniem kroku, przez co zmienia się też wyliczany błąd.

Liczba kroków dla poszczególnych metod

RK3	ODE45
75	125

Jak widać własna implemntacja RK3 jest o 40% szybsza niż solver ODE45. Jakość rozwiązania również znacząco nie odstawała od wbudowanego rozwiązania.

5. Podsumowanie

Dla badanego przykładu własna implementacja RK3 okazała się być lepszą alternatywą dla wbudowanego solvera. Jest ona znacząco szybsza, i posiada podobną precyzję jak ODE45.