Metody numeryczne – Projekt 3.

Autor: Aleksandra Jamróz

Zadanie 3.20

Treść:

Ruch punktu na płaszczyźnie (y_1, y_2) *jest opisany równaniami:*

$$\frac{dy_1}{dx} = y_2 + y_1 (0.3 - y_1^2 - y_2^2)$$
$$\frac{dy_2}{dx} = -y_1 + y_2 (0.3 - y_1^2 - y_2^2)$$

Należy obliczyć przebieg trajektorii ruchu tego punktu w przedziale [0, 20] dla warunków początkowych: $y_1(0) = 0.002, y_2(0) = 0.01$
Rozwiązanie proszę znaleźć korzystając z zaimplementowanej przez siebie w języku Matlaba w formie funkcji (możliwie uniwersalnej, czyli solwera, o odpowiednich parametrach wejścia i wyjścia) ***metody Dormand-Prince'a czwartego rzędu przy zmiennym kroku z szacowaniem błędu techniką pary metod włożonych (DorPri45)****.*

Rozwiązanie:
Rozwiązanie rozpoczęłam od zaimplementowania równań podanych w treści zadania oraz połączeniu ich w funkcję zwracającą wektor obu pochodnych:

.

dy1dx.m

```
function ret = dy1dx(t, y)
    ret = y(2) + y(1) * (0.3 - y(1)^2 - y(2)^2);
end
```

dy2dx.m

```
function ret = dy2dx(t, y)
    ret = -y(1) + y(2) * (0.3 - y(1)^2 - y(2)^2);
end

```

dydx.m

```
function ret = dydx(t, y)
    ret_1 = dy1dx(t, y);
    ret_2 = dy2dx(t, y);
    ret = [ret_1; ret_2];
end
```

Obraz 1: Implementacja funkcji z treści zadania

Metoda wymieniona w treści zadania wymaga zastosowania metod RK (Rungego-Kutty): metody RK rzędu p oraz rzędu p+1. y_{n+1} to rozwiązanie w następnym kroku, x_n to zmienna niezależna, h to długość kroku, a parametry w, w*, a i c pochodzą z tabeli Butchera zamieszczonej poniżej.

Metoda RK rzędu p:

$$y_{n+1}=y_n+h\cdot\sum_{i=1}^{m} w_i^* k_i \quad \text{, gdzie}$$
$$k_1=f(x_n, y_n)$$
$$k_i=f\left(x_n+c_i h, y_n+h\cdot\sum_{j=1}^{i-1} a_{ij} k_j\right)$$

Metoda RK rzędu p+1:

$$y_{n+1}=y_n+h\cdot\sum_{i=1}^{m+1} w_i k_i \quad \text{, gdzie}$$
$$k_1=f(x_n, y_n)$$
$$k_i=f\left(x_n+c_i h, y_n+h\cdot\sum_{j=1}^{i-1} a_{ij} k_j\right)$$

Będziemy korzystać z tabeli parametrów pary metod włożonych dla metody Dormand-Princa dla rzędów 4 oraz 5. Znajdują się w niej wartości odpowiednich współczynników.

c_i	a_{ij}						w_i^*	w_i
0							$\frac{35}{384}$	$\frac{5179}{57600}$
$\frac{1}{5}$	$\frac{1}{5}$						0	0
$\frac{3}{10}$	$\frac{3}{40}$	$\frac{9}{40}$					$\frac{500}{1113}$	$\frac{7571}{16695}$
$\frac{4}{5}$	$\frac{44}{45}$	$-\frac{56}{15}$	$\frac{32}{9}$				$\frac{125}{192}$	$\frac{393}{640}$
$\frac{8}{9}$	$\frac{19372}{6561}$	$-\frac{25360}{2187}$	$\frac{64448}{6561}$	$-\frac{212}{729}$			$-\frac{2187}{6784}$	$-\frac{92097}{339200}$
1	$\frac{9017}{3168}$	$-\frac{355}{33}$	$\frac{46732}{5247}$	$\frac{49}{176}$	$-\frac{5103}{18656}$		$\frac{11}{84}$	$\frac{187}{2100}$
1	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$-\frac{2187}{6784}$	$\frac{11}{84}$		$\frac{1}{40}$

Algorytm liczenia składa się z następujących kroków:
Metoda DorPri45 ma następujący schemat:

1. Wyznacz y_{n+1} na podstawie x_n oraz h_n i współczynników.
2. Oblicz oszacowanie błędu techniką pary metod włożonych
3. Oblicz współczynnik korekty kroku α
4. Jeśli s * a >= 1 i następny krok osiągnie granicę przedziału, to zakończ działanie

5. Jeśli s * a >= 1, ale nie osiągnięto końca przedziału - oblicz nową długość kroku i przejdź do następnej iteracji
6. Jeśli s * a < 1, a potencjalny nowy krok jest mniejszy niż założony minimalny, zakończ niepowodzeniem
7. Ustaw obecny krok na wyliczony potencjalny krok i ponownie policz obecną iterację.

Parametr s to współczynnik bezpieczeństwa. Powyższe kroki opisałam na podstawie schematu blokowego zamieszczonego na slajdach wykładowych.

Wykonując krok metodami RK dla obu rzędów, otrzymujemy:
Dla metody rzędu p:

$$y(x_n+h)=y_n+h\cdot\sum_{i=1}^{m} w_i^* k_i(h)+\frac{r_n^{p+1}(0)}{(p+1)!}\cdot h^{p+1}+O(h^{p+3}) \quad \text{, gdzie}$$

$$y_{n+1}=y_n+h\cdot\sum_{i=1}^{m} w_i^* k_i(h)$$

$$\text{I główna część błędu} = \frac{r_n^{p+1}(0)}{(p+1)!}\cdot h^{p+1}$$

Dla metody rzędu p+1:

$$y(x_n+h)=y_n+h\cdot\sum_{i=1}^{m+1} w_i^* k_i(h)+\frac{r_n^{p+2}(0)}{(p+2)!}\cdot h^{p+2}+O(h^{p+4}) \quad \text{, gdzie}$$

$$y_{n+1}=y_n+h\cdot\sum_{i=1}^{m+1} w_i^* k_i(h)$$

$$\text{I główna część błędu} = \frac{r_n^{p+2}(0)}{(p+2)!}\cdot h^{p+2}$$

Po odjęciu stronami powyższych równań, otrzymujemy wzór na oszacowanie błędu metody rzędu p (niższego z dwóch wykorzystywanych do obliczeń):

Do obliczenia zmiennej długości kroku potrzebne są jeszcze 2 wzory. Pierwszy wykorzystywany jest do obliczenia wartości parametru dokładności obliczeń ϵ:

$$\epsilon=|y_n|\cdot\epsilon_w+\epsilon_b \quad \text{, gdzie} \quad \epsilon_w,\epsilon_b \quad \text{to parametry użytkownika.}$$

Drugi to wzór na modyfikację kroku α:

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

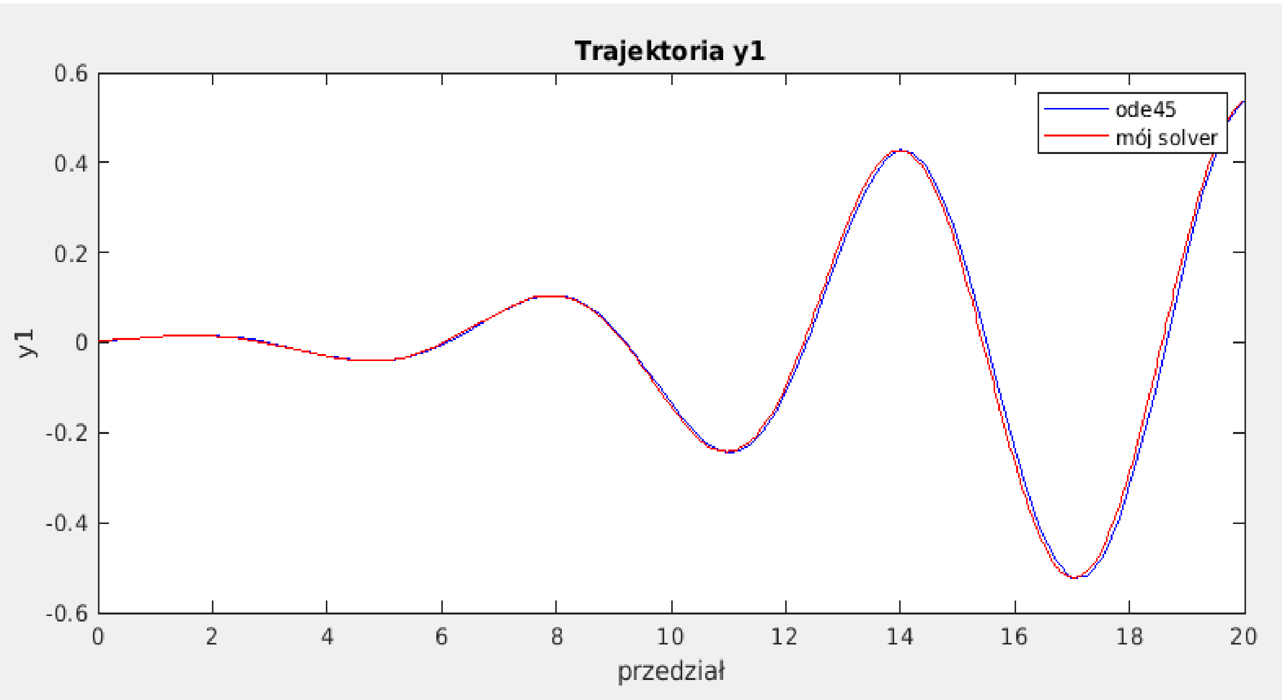
Implemenacja w matlabie:

```
function [t, y, hs, es] = dormand_prince(func, tspan, y0, h_start, h_min, wb, beta, ew, eb)

    y(:, 1) = y0;         % ustalanie wartosci poczatkowych
    t = tspan(1);         % poczatek przedzialu
    h = h_start;          % krok poczatkowy
    s = wb;               % wspolczynnik bezpieczenstwa
    i = 2;                % iteracja
    es(1) = 0;
    hs(1) = 0;

    % tablica butchera
    c = [0, 1/5, 3/10, 4/5, 8/9, 1, 1];

    a = [0          0           0           0        0           0;
        1/5         0           0           0        0           0;
        3/40        9/40        0           0        0           0;
        44/45      -56/15       32/9        0        0           0;
        19372/6561 -25360/2187  64448/6561 -212/729  0           0;
        9017/3168  -355/33      46732/5247  49/176  -5103/18656  0;
        35/384      0           500/1113    125/192 -2187/6784   11/84];

    w_star = [35/384     0 500/1113   125/192 -2187/6784    11/84    0   ];
    w      = [5179/57600 0 7571/16695 393/640 -92097/339200 197/2100 1/40];


    while 1
        % aproksymacja wartości funkcji w poprzednim punkcie
        yn = y(:, i - 1);
        % czas
        tn = t(i-1);
        k = zeros(2, 7);
        for j = 1:7
            sec_arg = yn + [dot(a(j, :) , k(1,1:6)); dot(a(j,:), k(2,1:6))];
            k(:, j) = h * func(tn+c(j), sec_arg);
        end

        % obliczanie wartosci funkcji w kolejnym punkcie dla rzedu p
        p = yn + [dot(w_star , k(1,:)); dot(w_star, k(2,:))];
        % obliczanie wartosci funkcji w kolejnym punkcie dla rzedu p+1
        p1 = yn + [dot(w , k(1,:)); dot(w, k(2,:))];

        delta = abs(p - p1);            % oszacowanie bledu
        e = abs(yn) * ew + eb;          % parametr dokladnosci obliczen
        alfa = (e./delta).^(1/5);       % wspolczynnik modyfikacji kroku
        alfa = min([alfa(1) alfa(2)]);
        h_new = s * alfa * h;           % nowa wartosc kroku
        y(:,i) = p1;                    % zapisanie nowej wartosci funkcji

        if (s * alfa >= 1)
            if (tn + h >= tspan(2))
                % koniec
                y = y(:,1:end-1);
                hs = hs(1:end-1);
                es = es(1:end-1);
                break
            else
```

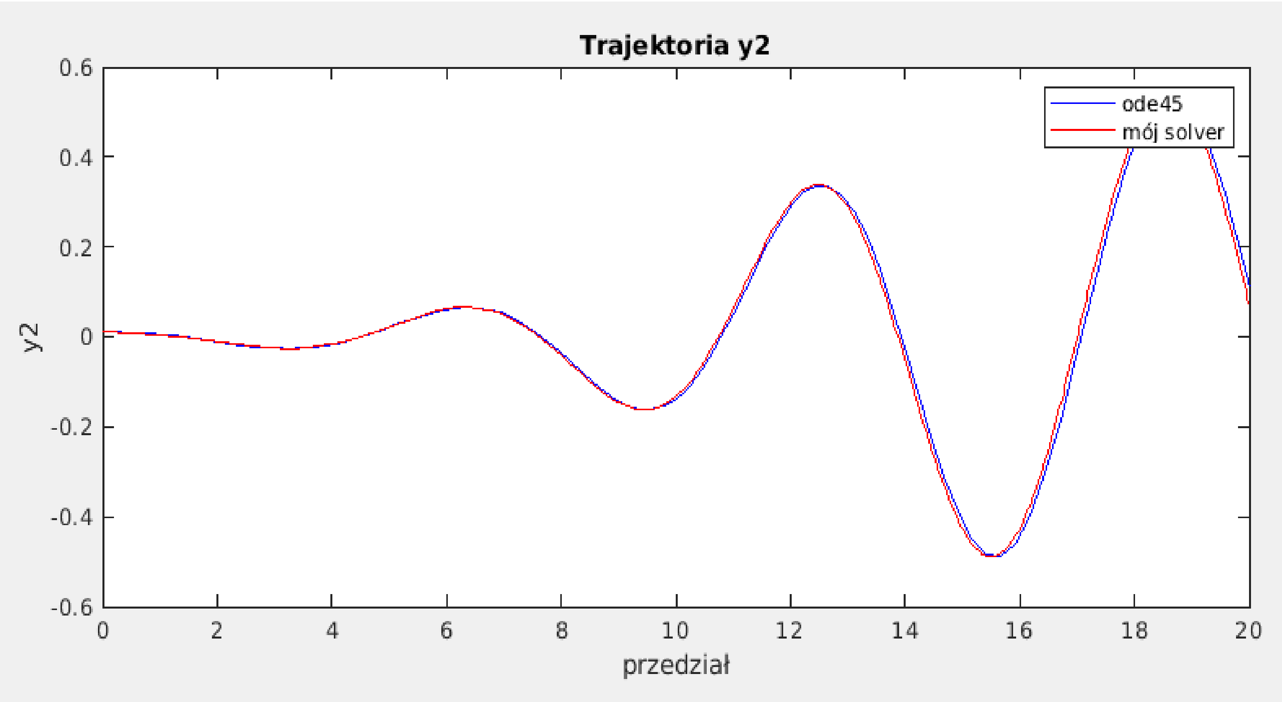
```
                t(i) = t(i-1) + h;       % przesuniecie w czasie
                % wyznaczenie nowej wartosci kroku
                h_new = min([h_new beta*h tspan(2)-t(i-1)]);
                i = i+1;
                h = h_new;
                hs(i) = h;
                es(i) = e(1);
            end
        else
            if (h_new < h_min)
                error('Niemozliwe rozwiazanie z zadana dokladnoscia');
            else
                % powtorzenie iteracji z nowa wartoscia kroku
                h = h_new;
            end
        end
    end
end
```
Obraz 2: Implementacja solvera w Matlabie

Testy – porównanie mojego solvera do solvera ode45 Matlaba

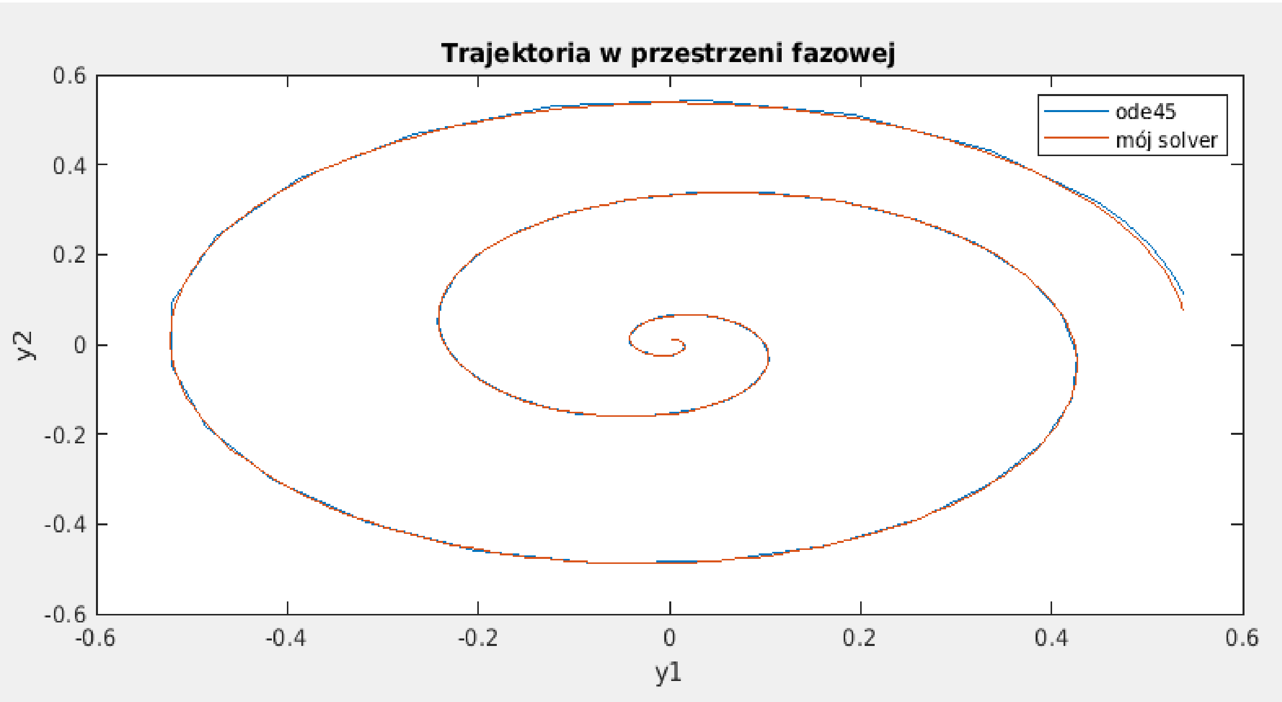
Otrzymane przeze mnie wyniki nie różnią się od tych otrzymanych przez solver Matlabowy. Świadczy to o udanej samodzielnej implementacji solvera. Za minimalny krok do narysowania wykresów przyjęłam 0.01, jednak podczas przeprowadzania eksperymentów, nie było zauważalnej różnicy podczas zmieniania tego parametru w przedziale 1e-4 – 1. Za błąd względny i bezwzględny przyjęłam 1e-4. Jego zmniejszenie powodowało znaczne wydłużenie wykonywania się programu, a wyniki uzyskane z błędem 1e-4 uznałam za wystarczająco dobre. Jego zwiększenie natomiast przyspieszało program, ale nie przynosiło oczekiwanych rezultatów – uzyskane wtedy wykresy trajektorii były "kanciaste".



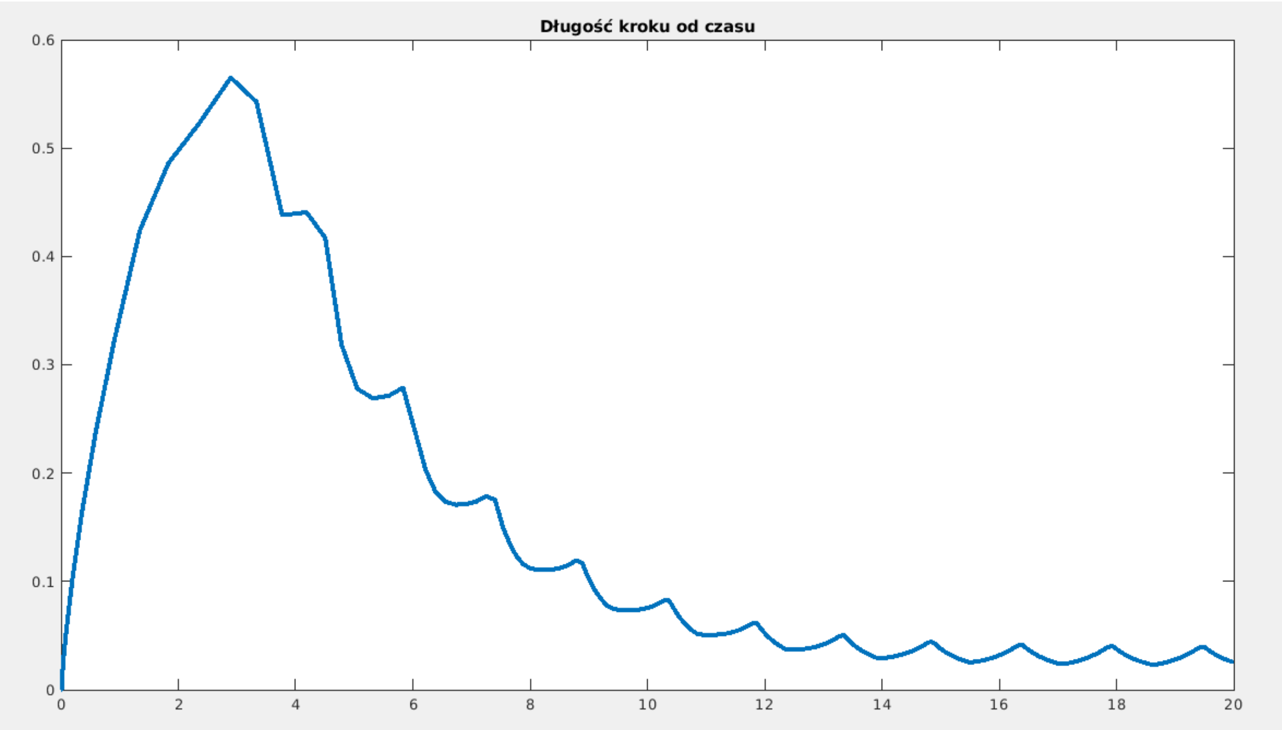
Obraz 3: Trajektoria y1



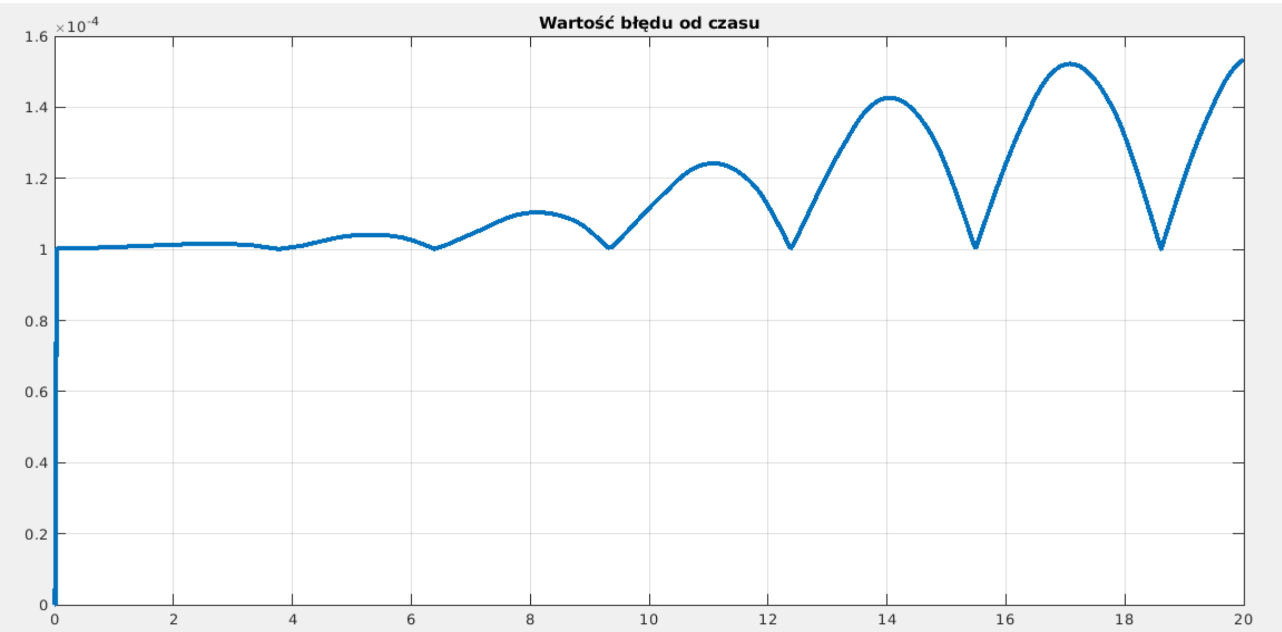
Obraz 4: Trajektoria y2



Obraz 5: Trajektoria w przestrzeni fazowej



Obraz 6: Wykres długości kroku od czasu



Obraz 7: Wykres uzyskanego błędu do czasu

Obraz 8: "Kanciasty" wykres trajektorii uzyskany przy ustawieniu błędów względnego i bezwzględnego na zbyt duży