ŁUKASZ STANISZEWSKI, NR INDEKSU: 304098

METODY NUMERYCZNE

PROJEKT 4 – ZADANIE 4.8

1 WPROWADZENIE

Jednym z głównych zastosowań równań różniczkowych jest modelowanie matematyczne układów dynamicznych. Układy takich równań, które w gruncie rzeczy opisują układy dynamiczne ze świata rzeczywistego, są w zwyczaju nieliniowe i jedynym sposobem ich rozwiązywania jest rozwiązywanie metodami numerycznymi.

W projekcie tym rozważane jest zagadnienie numerycznego obliczania przebiegu trajektorii punktu (w konkretnym przedziale), którego ruch jest opisany właśnie za pomocą równań różniczkowych. Przy definiowaniu takiej trajektorii istotne jest określenie warunków początkowych – będą one punktem startowym dla konkretnych metod.

W przypadku tego zadania, ruch punktu opisany jest następującymi równaniami:

$$x_1' = x_2 + x_1(0,9 - x_1^2 - x_2^2)$$
$$x_2' = -x_1 + x_2(0,9 - x_1^2 - x_2^2)$$

Natomiast przebieg trajektorii ruchu powinien zostać obliczony na przedziale $[0, 20]$ dla następujących warunków początkowych:

a) $x_1(0) = 10,\ x_2(0) = 8$;
b) $x_1(0) = 0,\ x_2(0) = 9$;
c) $x_1(0) = 8,\ x_2(0) = 0$;
d) $x_1(0) = 0,001,\ x_2(0) = 0,001$;

2 METODA RK4 ZE STAŁYM KROKIEM

Pierwszym podejściem do tego problemu jest zastosowanie **metod jednokrokowych o stałej długości**. Ich ogólny wzór określający pojedynczy krok definiujemy jako:

$$y_{n+1} = y_n + h \cdot \phi_f(x_n, y_n; h)$$

gdzie $y(x_0) = y_0$, $x_n = x_0 + nh$, $n = 0, 1, \dots$; zaś $\phi_f(x_n, y_n; h)$ to funkcja definiująca metodę.

Jednymi z najpopularniejszych tego typu metod są **metody Rungego-Kutty (RK)**, które można zdefiniować następującym wzorem:

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

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

Do wykonania jednego kroku tej metody należy obliczyć wartości prawych stron równań różniczkowych m razy.

Metoda Rungego-Kutty czwartego rzędu (RK4) ze stałym krokiem polega na wyznaczaniu kolejnych wartości funkcji za pomocą wartości pochodnych w czterech kolejnych, pobliskich punktach i wykorzystania ich (z użyciem odpowiednich wag) do zmiany wartości dla nowego punktu. Ogólny wzór na metodę RK4 wygląda następująco:

$$\boldsymbol{y_{n+1} = y_n + \frac{1}{6} \cdot h \cdot (k_1 + 2k_2 + 2k_3 + k_4)}, \text{ gdzie:}$$

$\boldsymbol{k_1 = f(x_n, y_n)}$,

$\boldsymbol{k_2 = f\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_1\right)}$,

$\boldsymbol{k_3 = f\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_1\right)}$,

$\boldsymbol{k_4 = f(x_n + h, y_n + hk_3)}$.

W tym przypadku, współczynnik $\boldsymbol{k_1}$ jest pochodną rozwiązania w punkcie $\boldsymbol{(x_n, y_n)}$, natomiast $\boldsymbol{k_2}$ to pochodna rozwiązania wyznaczanego zwykłą metodą Eulera w punkcie $\boldsymbol{\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_1\right)}$. Wartość $\boldsymbol{k_3}$ wyznaczana jest podobnie jak $\boldsymbol{k_2}$, ale w punkcie $\boldsymbol{\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_2\right)}$, na końcu wartość $\boldsymbol{k_4}$ reprezentuje pochodną w punkcie $\boldsymbol{(x_n + h, y_n + hk_3)}$. W ten sposób otrzymywane są cztery wartości pochodnej rozwiązania – po jednej na końcach przedziału i dwie w jego środku. Aproksymacja pochodnej teraz jest wyznaczana jako ważona średnia arytmetyczna z tych wszystkich wartości.

Błąd w tej metodzie szacuje się według zasady podwójnego kroku – obliczamy tu różnicę wartości następnego punktu obliczonego z normalnym krokiem i tego samego punktu obliczonego w dwóch o połowę krótszych krokach. Stąd też, wzór na oszacowanie błędu pojedynczego kroku n o długości h dla metody rzędu p wygląda następująco:

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

W przypadku tej metody, rząd $\boldsymbol{p = 4}$.

Gdy ma miejsce rozwiązywanie układu równań różniczkowych, istotny jest dobór parametru kroku. Dla tego zadania, koniecznym było wykonanie tylu prób, aby znaleźć taki krok, którego zmniejszenie wpływa znacząco na rozwiązanie (w porównaniu do rozwiązania dla metody ode45 z Matlaba), podczas gdy zwiększenie już nie wpływa. Fakt ten wynika z tego, że przy wyznaczaniu długości kroku występują dwie przeciwstawne tendencje:

1) Jeśli krok **h** maleje, to maleje też błąd metody (błąd aproksymacji).
2) Jeśli krok **h** maleje, to zwiększa się liczba iteracji (kroków) niezbędnych do wyznaczenia rozwiązania na zadanym odcinku $\boldsymbol{[a, b]}$, stąd też liczba obliczeń i związanych z nimi błędów numerycznych.

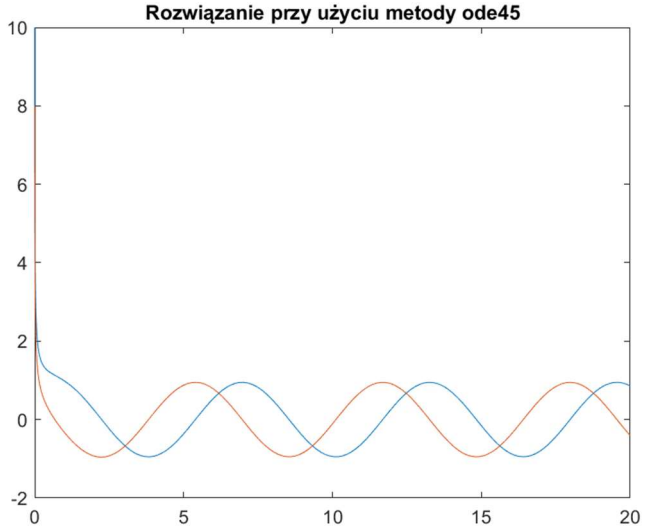
Stąd nie można nadmiernie zmniejszać długości kroku – powinien on wystarczyć do uzyskania założonej dokładności, ale nie powinien być znacznie mniejszy od wartości, przy której wymagana dokładność jest osiągana.

W celu rozwiązania tego zadania powstał następujący solver RK4ConstantSolver, zwracający kolejne pozycje ruchu punktu w kolejnych krokach wraz z czasem wykonania algorytmu i błędami w tych krokach, liczonymi zgodnie z zasadą dwóch kroków.

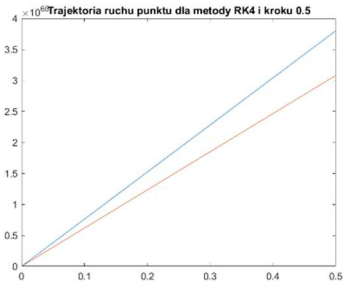
```
function [steps, x, time, errors] = RK4ConstantSolver(f, x_start, range, h)
%Solver oblicza przebieg trajektorii ruchu metodą RK4 ze stałym krokiem
    % f - funkcja opisująca ruch
    % range - zakres, w jakim jest wykonywany ruch
    % h - krok metody
    % x0 - warunek początkowy
    tic;
    h_error = 0.5*h;
    yn = x_start;
    steps_vec = range(1):h:range(2);
    steps = zeros(size(steps_vec,2),1);
    x = zeros(size(steps_vec, 2),2);
    errors = zeros(size(steps_vec, 2),2);
    iter = 1;
    x(1, :) = yn';
    for step=range(1):h:range(2)
        k1 = f(yn);
        k2 = f(yn + 0.5*h*k1);
        k3 = f(yn + 0.5*h*k2);
        k4 = f(yn + h*k3);
        yn_1 = yn + 1/6 * h * (k1 + 2*k2 + 2*k3 + k4);
        yn_error = yn;      % fragment odpowiedzialny za naliczanie błędów
        for iter_error=1:2
            k1 = f(yn_error);
            k2 = f(yn_error+0.5*h_error*k1);
            k3 = f(yn_error+0.5*h_error*k2);
            k4 = f(yn_error+h_error*k3);
            yn_error = yn_error + 1/6*h_error*(k1+2*k2+2*k3+k4);
        end
        iter = iter + 1;
        errors(iter, :) = (16/15) * abs(yn_error - yn_1);
        yn = yn_1;
        steps(iter,1) = step + h;
        x(iter, :) = yn;
    end
    time = toc;
end
```

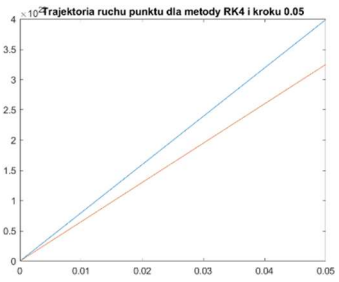
2.1 WARUNEK POCZĄTKOWY $x_1(0) = 10, x_2(0) = 8$

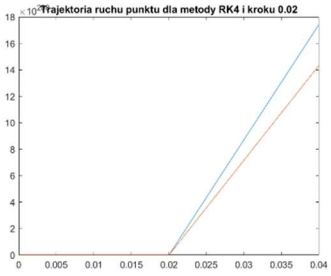
Rozwiązanie przy użyciu ode45 wygląda następująco:

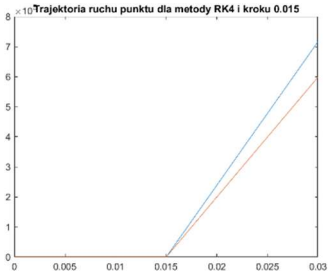


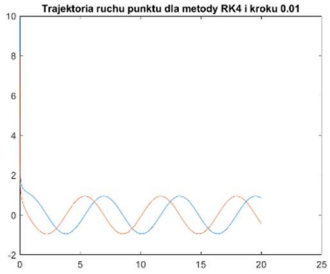
W celu otrzymania podobnego rozwiązania, wykonałem symulacje dla coraz mniejszych kroków czasowych, oto wykresy dla niektórych z nich:



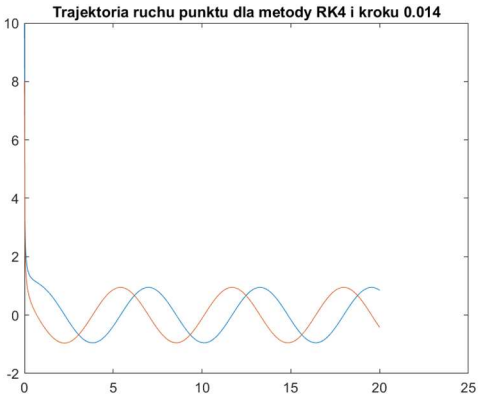








Analizując wyniki, uznałem, że najlepszym krokiem w tym przypadku będzie krok 0.014. Poniżej zamieszczona została trajektoria punktu dla tego kroku wraz z wykresem błędu liczonego zgodnie z zasadą podwójnego kroku:

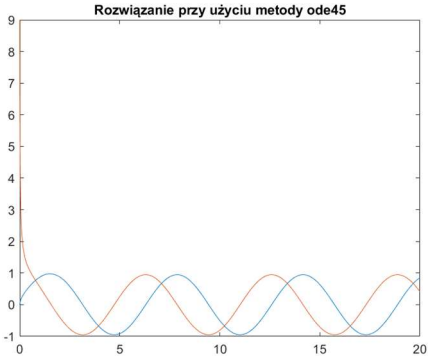


Dodatkowo, na podstawie wyników, można zauważyć tendencję wzrostową czasu wykonywania w stosunku do zmniejszania się długości kroku:

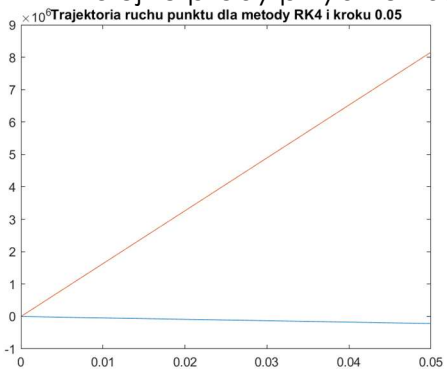
```
>> rk4vis(f, [10, 8], [0, 20], 0.5);
    "Rozwiązanie zajęło: "    "0.0008666"    " s"

>> rk4vis(f, [10, 8], [0, 20], 0.05);
    "Rozwiązanie zajęło: "    "0.0017182"    " s"

>> rk4vis(f, [10, 8], [0, 20], 0.02);
    "Rozwiązanie zajęło: "    "0.0027423"    " s"

>> rk4vis(f, [10, 8], [0, 20], 0.015);
    "Rozwiązanie zajęło: "    "0.0034641"    " s"

>> rk4vis(f, [10, 8], [0, 20], 0.014);
    "Rozwiązanie zajęło: "    "0.0046331"    " s"

>> rk4vis(f, [10, 8], [0, 20], 0.01);
    "Rozwiązanie zajęło: "    "0.0066625"    " s"
```

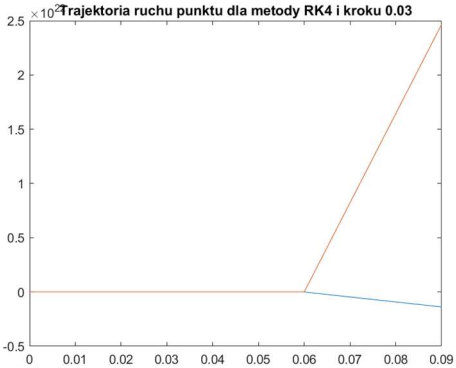
2.2 PUNKT POCZĄTKOWY $x_1(0) = 0, x_2(0) = 9$

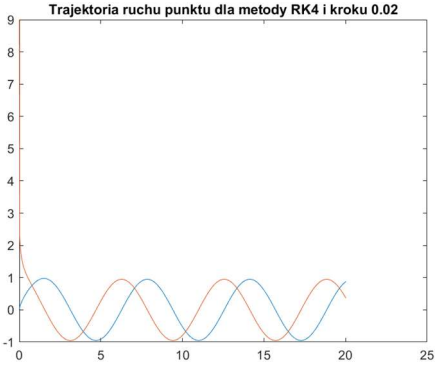
Rozwiązanie **ode45**:



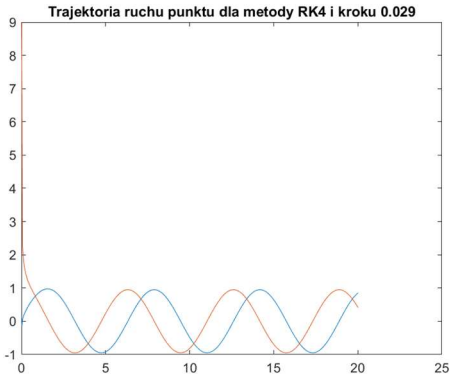
Kolejne próby przybliżenia:





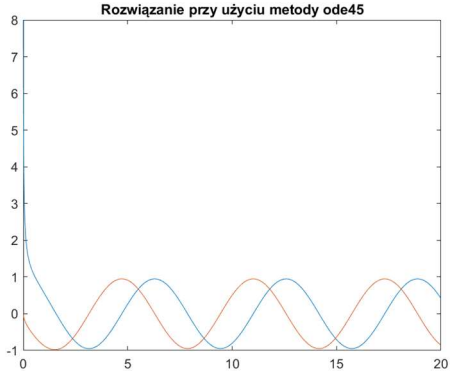


W tym przypadku wystarczający okazał się krok 0.029:

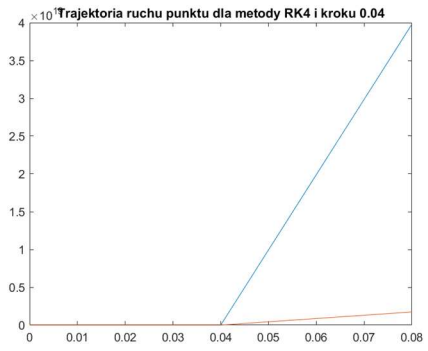


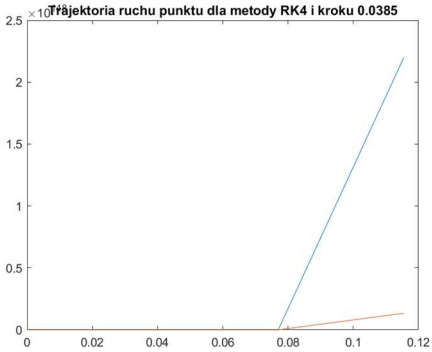
2.3 PUNKT POCZĄTKOWY $x_1(0) = 8, x_2(0) = 0$

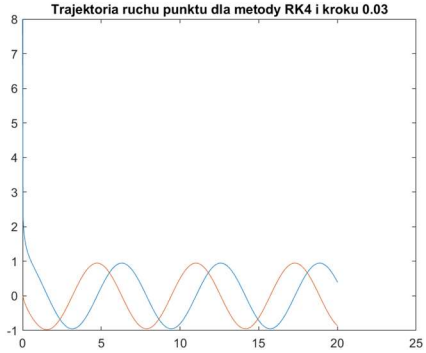
Rozwiązanie ode45:



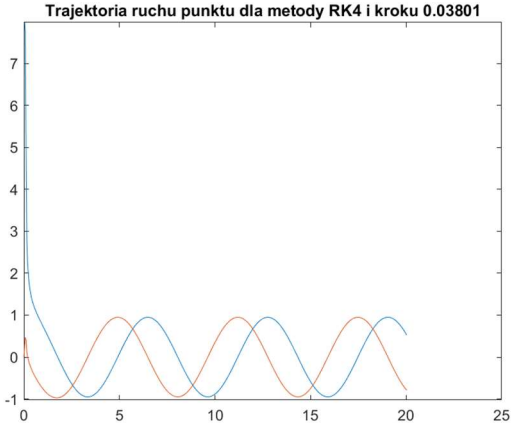
Kolejne przybliżenia:





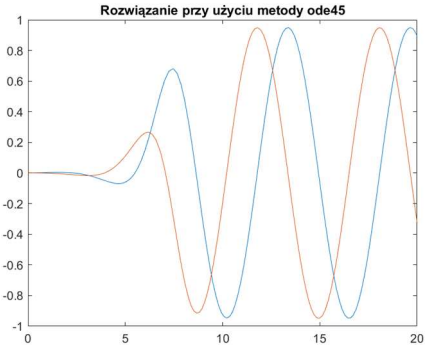


Tutaj za dobry krok uznałem krok $\mathbf{0.03801}$:

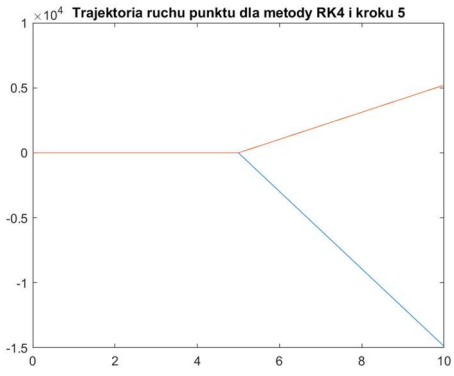


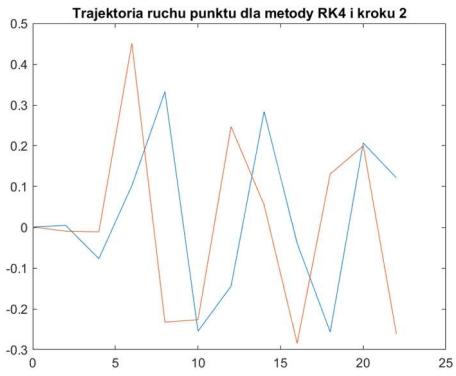
2.4 PUNKT POCZĄTKOWY $x_1(0) = 0.001, x_2(0) = 0.001$

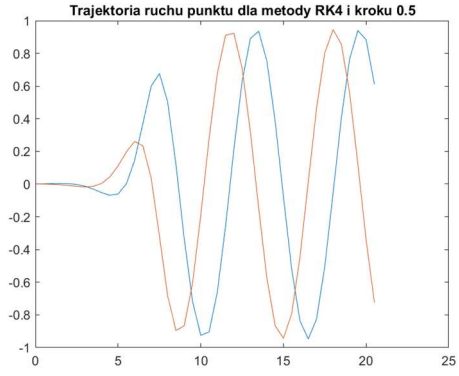
Rozwiązanie **ode45**:

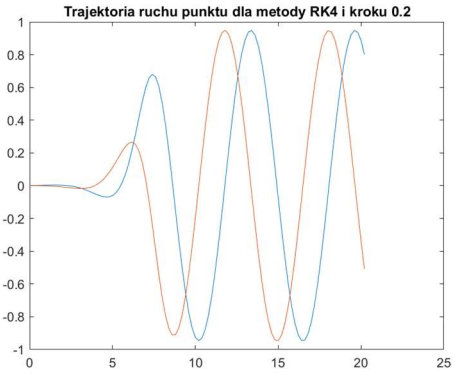


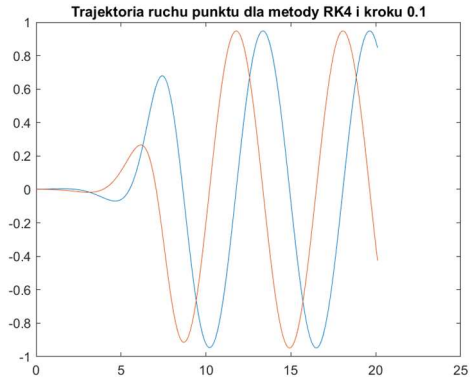
Kolejne przybliżenia:



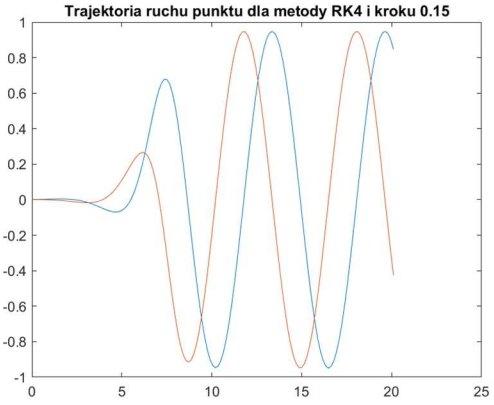


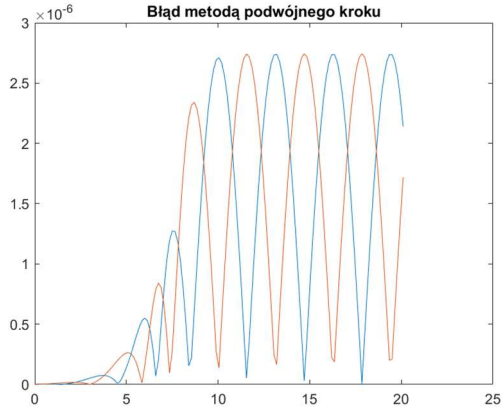






W przypadku tej metody najlepszym krokiem okazał krok o długości $\mathbf{0.15}$:





Tak samo zestawienie czasów:

```
>> rk4vis(f, [0.001, 0.001], [0, 20], 5);
    "Rozwiązanie zajęło: "    "0.0001106"    " s"
>> rk4vis(f, [0.001, 0.001], [0, 20], 2);
    "Rozwiązanie zajęło: "    "0.0002301"    " s"
>> rk4vis(f, [0.001, 0.001], [0, 20], 1);
    "Rozwiązanie zajęło: "    "0.0005162"    " s"
>> rk4vis(f, [0.001, 0.001], [0, 20], 0.2);
    "Rozwiązanie zajęło: "    "0.0005755"    " s"
>> rk4vis(f, [0.001, 0.001], [0, 20], 0.15);
    "Rozwiązanie zajęło: "    "0.0005826"    " s"
>> rk4vis(f, [0.001, 0.001], [0, 20], 0.1);
    "Rozwiązanie zajęło: "    "0.0009184"    " s"
```

Możemy na podstawie powyższych wyników zauważyć, że wraz ze zmniejszeniem kroku, czas tworzenia trajektorii punktu rośnie, ale otrzymujemy coraz lepszą dokładność względem ode45.

Powodem tego, że metoda nie działa przy dużych krokach, jest to, że pochodne są brane w punktach zbyt bardzo oddalonych od aktualnego punktu, więc trudno, żeby opisywały oryginalną pochodną w tym punkcie.

3 METODA PREDYKTOR-KOREKTOR ADAMSA

Innym podejściem do rozwiązywania tego typu równań jest stosowanie **metod wielokrokowych**. Realizują one następujące równanie:

$$y_n = \sum_{j=1}^{k} \alpha_j y_{n-j} + h \sum_{j=0}^{k} \beta_j \cdot f(x_{n-j}, y_{n-j}), y_0 = y_a$$

Metody wielokrokowe dzielimy na **jawne** (gdy parametr $\beta_0 = 0$) oraz **niejawne** (gdy $\beta_0 \neq 0$). Metody jawne do obliczenia następującej wartości posługują się tylko poprzednimi wartościami, natomiast niejawne używają poprzednich wartości oraz wartości w punkcie bieżącym.

Metody Adamsa natomiast opierają się na przekształceniu równań różniczkowych

$$y'(x) = f(x, y(x)), \qquad y(a) = y_a, \qquad x \in [a, b]$$

w równoważne im równania całkowite:

$$y(x) = y(a) + \int_a^x f(t, y(t))dt$$

Metody jawne (Adamsa-Bashfortha) polegają na tym, że przybliżamy $f(x, y(x))$ wielomianem interpolacyjnym $W(x)$ opartym na **k** węzłach. Realizuje równanie:

$$y_n = y_{n-1} + h \sum_{j=1}^{k} \beta_j f(x_{n-j}, y_{n-j})$$

Metoda ta korzysta ze stablicowanych parametrów β – w przypadku naszym będzie nas interesował ten fragment:

k	β_1	β_2	β_3	β_4
4	$55/24$	$-59/24$	$37/24$	$-9/24$

Metody jawne charakteryzują się małą ilością obliczeń na iterację.

Metody niejawne (Adamsa-Moultona) polegają na tym, że $f(x, y(x))$ przybliżamy wielomianem interpolacyjnym $W^*(x)$ opartym na węzłach. Otrzymujemy tu ostatecznie:

$$y_n = y_{n-1} + h \cdot \beta_0^* \cdot f(x_n, y_n) + h \sum_{j=1}^{k} \beta_j^* \cdot f(x_{n-j}, y_{n-j})$$

Gdzie parametry β^* są stablicowane – w naszym przypadku interesuje nas:

k	β_0^*	β_1^*	β_2^*	β_3^*	β_4^*
4	$251/720$	$646/720$	$-264/720$	$106/720$	$-19/720$

Metody niejawne charakteryzują się tym, że są to metody o wysokim rzędzie i małej stałej błędu, a także o dużym obszarze absolutnej stabilności.

Metody **predyktor-korektor** łączą ze sobą podejście metod wielokrokowych niejawnych i metod wielokrokowych jawnych. Stanowią one praktyczne realizacje metod wielokrokowych. Algorytm dla metody **predyktor-korektor Adamsa** wygląda następująco:

$$\boldsymbol{P}: y_n^{[0]} = y_{n-1} + h\sum_{j=1}^{k} \beta_j f_{n-j}$$

$$\boldsymbol{E}: f_n^{[0]} = f\left(x_n, y_n^{[0]}\right)$$

$$\boldsymbol{K}: y_n = y_{n-1} + h\sum_{j=1}^{k} \beta_j^* \cdot f_{n-j} + h \cdot \beta_0^* \cdot f_n^{[0]}$$

$$\boldsymbol{E}: f_n = f(x_n, y_n)$$

Natomiast oszacowanie błędu liczy się w sposób następujący:

$$\delta_n(h) = \frac{c_{p+1}^*}{c_{p+1} - c_{p+1}^*}\left(y_n^{[0]} - y_n\right) \overset{\textit{na podstawie tablic}}{\Longrightarrow} \delta_n(h_n) = -\frac{19}{232}\left(y_n^{[0]} - y_n\right)$$

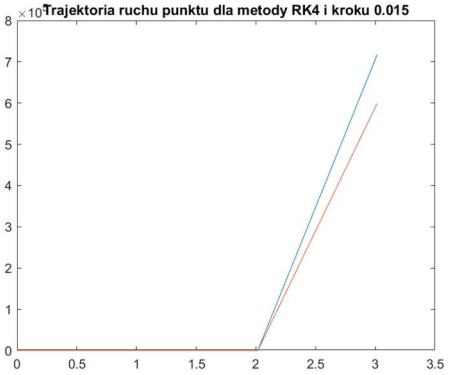
W wyniku pracy, stworzyłem następujący solver zwracający kolejne pozycje ruchu punktu w kolejnych krokach metodą predyktor-korektor Adamsa wraz z czasem wykonania algorytmu i błędami w tych krokach.

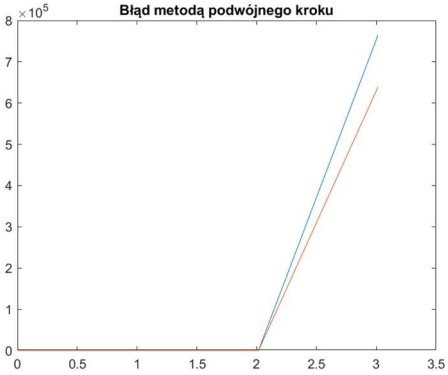
```
function [steps, x, time, errors] = PredictorCorectorSolver(f, x_start, range, h)
%Solver oblicza przebieg trajektorii ruchu metodą RK4 ze stałym krokiem
    % f - funkcja opisująca ruch
    % range - zakres, w jakim jest wykonywany ruch
    % h - krok metody
    % x0 - warunek początkowy
    number_steps = 4;
    tic;
    h_error = 0.5*h;
    yn = x_start;
    steps_vec = range(1):h:range(2);
    steps = zeros(size(steps_vec,2),1);
    x = zeros(size(steps_vec, 2),2);
    errors = zeros(size(steps_vec, 2),2);
    iter = 1;
    x(1, :) = yn';
    for step=2:number_steps    % pierwsze 4 sa obliczane jak poprzednio
        k1 = f(yn);
        k2 = f(yn + 0.5*h*k1);
        k3 = f(yn + 0.5*h*k2);
        k4 = f(yn + h*k3);
        yn_1 = yn + 1/6 * h * (k1 + 2*k2 + 2*k3 + k4);
        yn_error = yn;      % fragment odpowiedzialny za naliczanie błędów
        for iter_error=1:2
            k1 = f(yn_error);
            k2 = f(yn_error+0.5*h_error*k1);
            k3 = f(yn_error+0.5*h_error*k2);
            k4 = f(yn_error+h_error*k3);
            yn_error = yn_error + 1/6*h_error*(k1+2*k2+2*k3+k4);
        end
        iter = iter + 1;
        errors(iter, :) = (16/15) * abs(yn_error - yn_1);
        yn = yn_1;
        steps(iter,1) = step + h;
        x(iter, :) = yn;
    end
    beta = [55/24, -59/24, 37/24, -9/24];
    beta_star = [251/720, 646/720, -264/720, 106/720, -19/720];
    iter = iter + 1;
```

```
    for step=number_steps+1:size(steps_vec,2)
        sum_beta = [0, 0]; % P
        for j=1:number_steps
            sum_beta = sum_beta + beta(j)*f(x(iter-j, :));
        end
        yn0 = x(iter-1,:) + h*sum_beta;
        fn0 = f(yn0);   %E
        sum_beta_star = [0, 0]; % K
        for j=1:number_steps
            sum_beta_star = sum_beta_star + beta_star(j+1)*f(x(iter-j, :));
        end
        yn = x(iter-1,:) + h*sum_beta_star + h*beta_star(1)*fn0;
        x(iter,:) = yn'; %E
        errors(iter,:)= -19/232 * (yn0-yn);
        steps(iter,1) = step + h;
        iter = iter + 1;
    end
    time = toc;
end
```

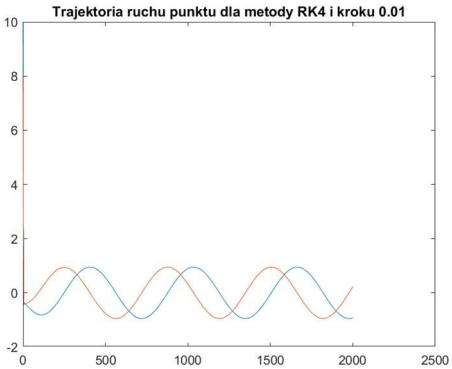
3.1 PUNKT $X_1(0) = 10$, $X_2(0) = 8$

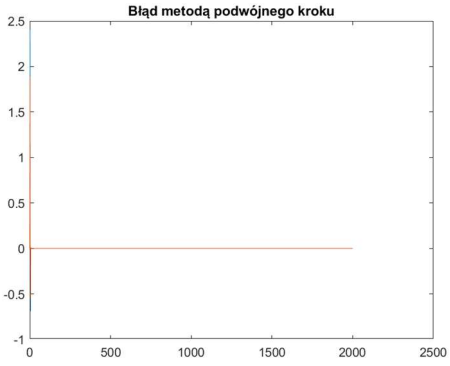
Dla kroku $h = 0.15$:



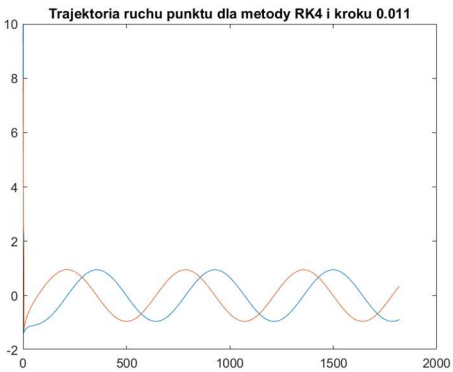


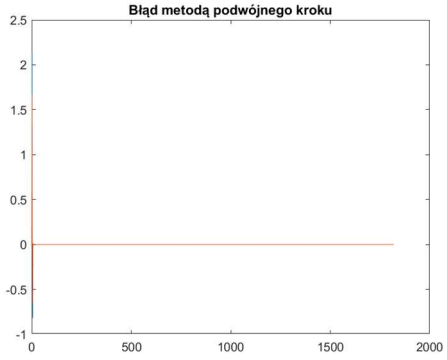
Dla kroku $h = 0.01$:





Dla kroku $h = 0.011$:

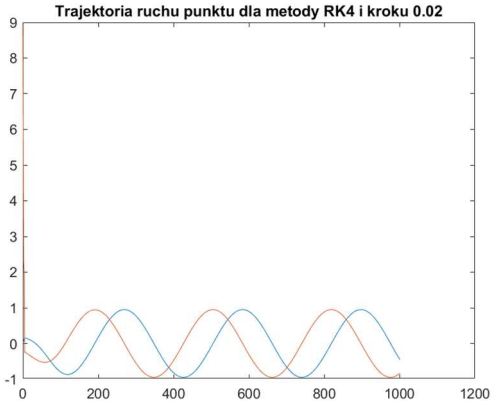


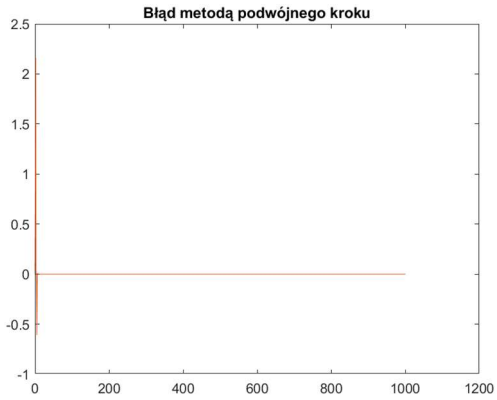


Tak więc za najlepszy krok uznałem $\boldsymbol{h = 0.01}$.

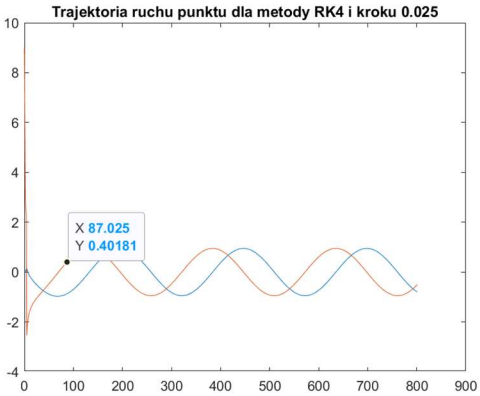
3.2 PUNKT $\boldsymbol{X_1(0) = 0, X_2(0) = 9}$

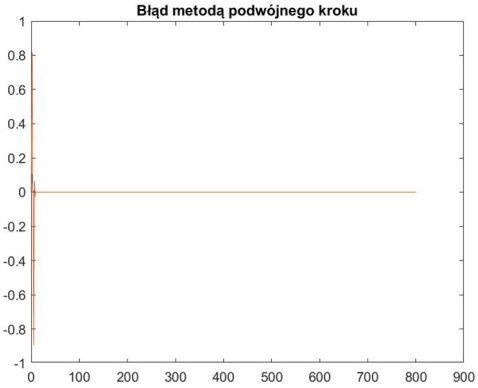
Dla kroku $\boldsymbol{h = 0.02}$:



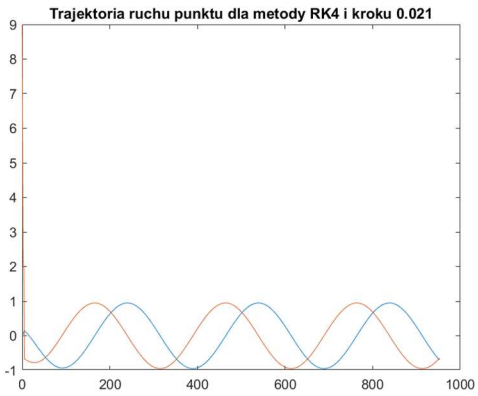


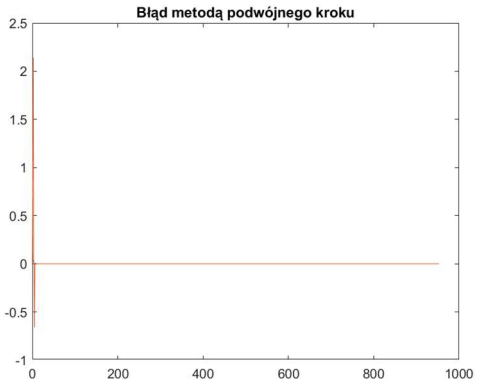
Dla kroku $\boldsymbol{h = 0.025}$:





Dla kroku $\boldsymbol{h = 0.021}$:

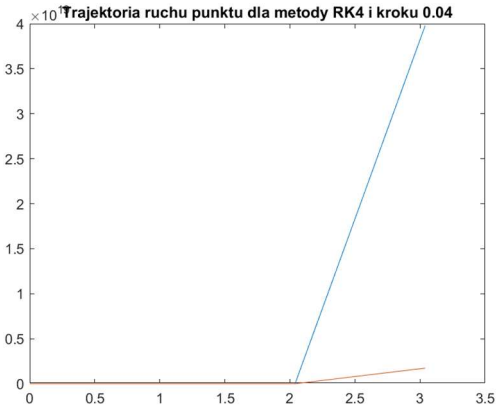


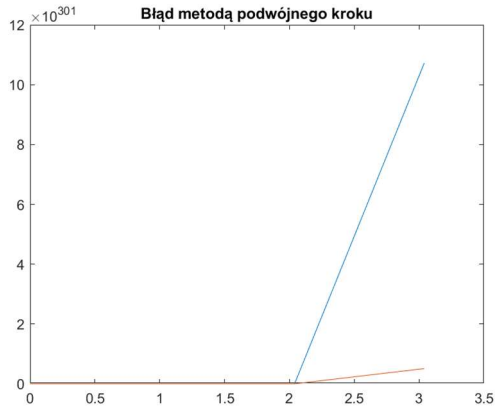


Tutaj uznałem, że najlepszy krok to $\boldsymbol{h = 0.021}$.

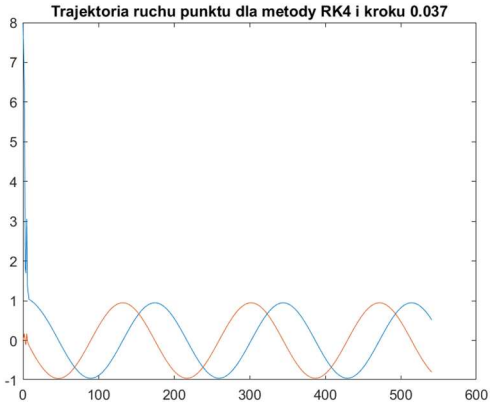
3.3 PUNKT $X_1(0) = 8$, $X_2(0) = 0$

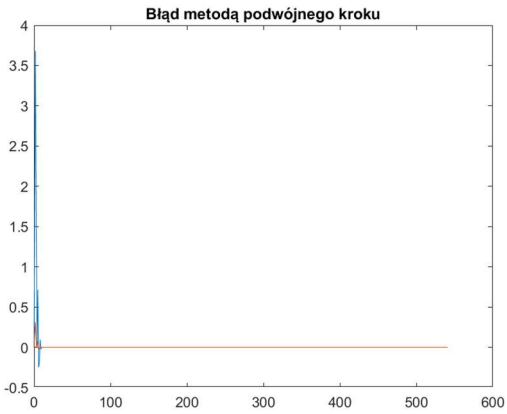
Dla $h = 0.04$:



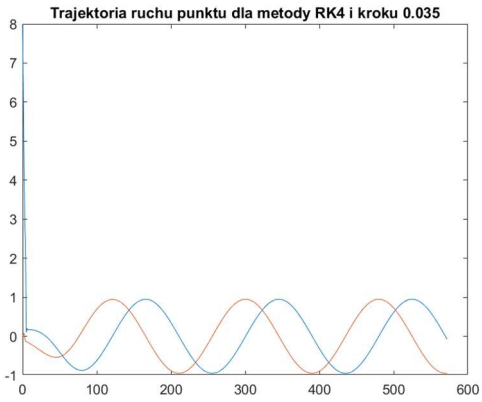


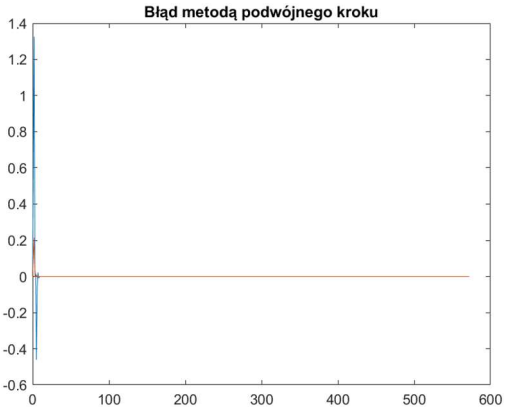
Dla $h = 0.037$:





Dla $h = 0.035$:

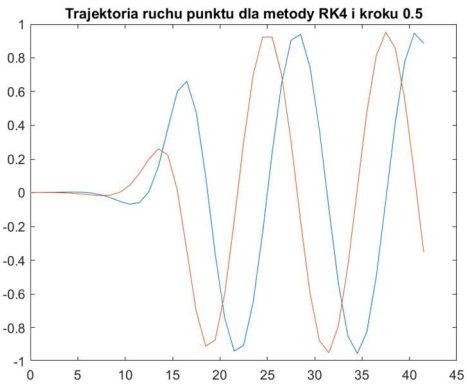


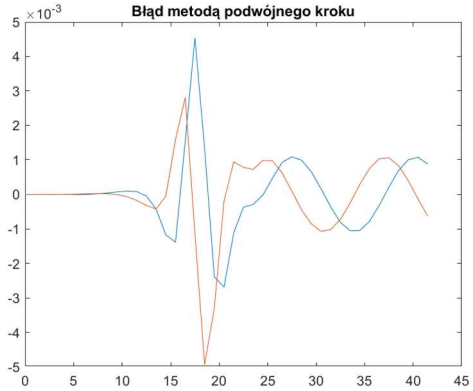


Tutaj za najlepszy krok uznałem $h = 0.035$.

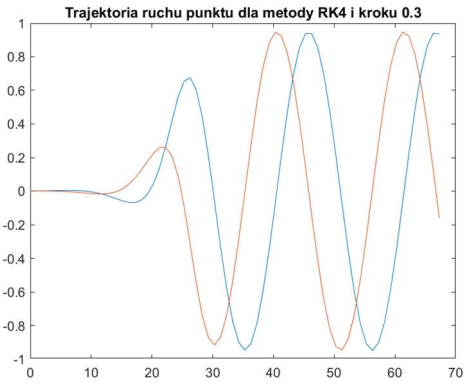
3.4 PUNKT $X_1(0) = 0.001$, $X_2(0) = 0.001$

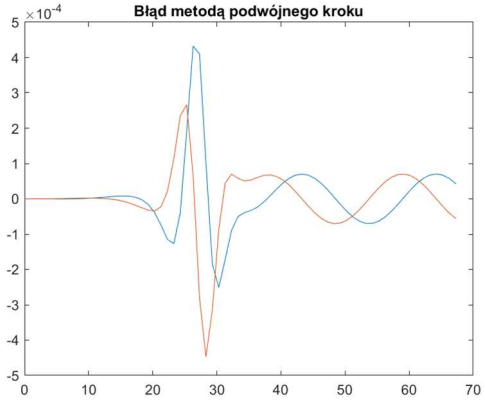
Dla $h = 0.5$:





Dla $h = 0.3$:





Dla $h = 0.3$ uznałem, że krok jest wystarczający.

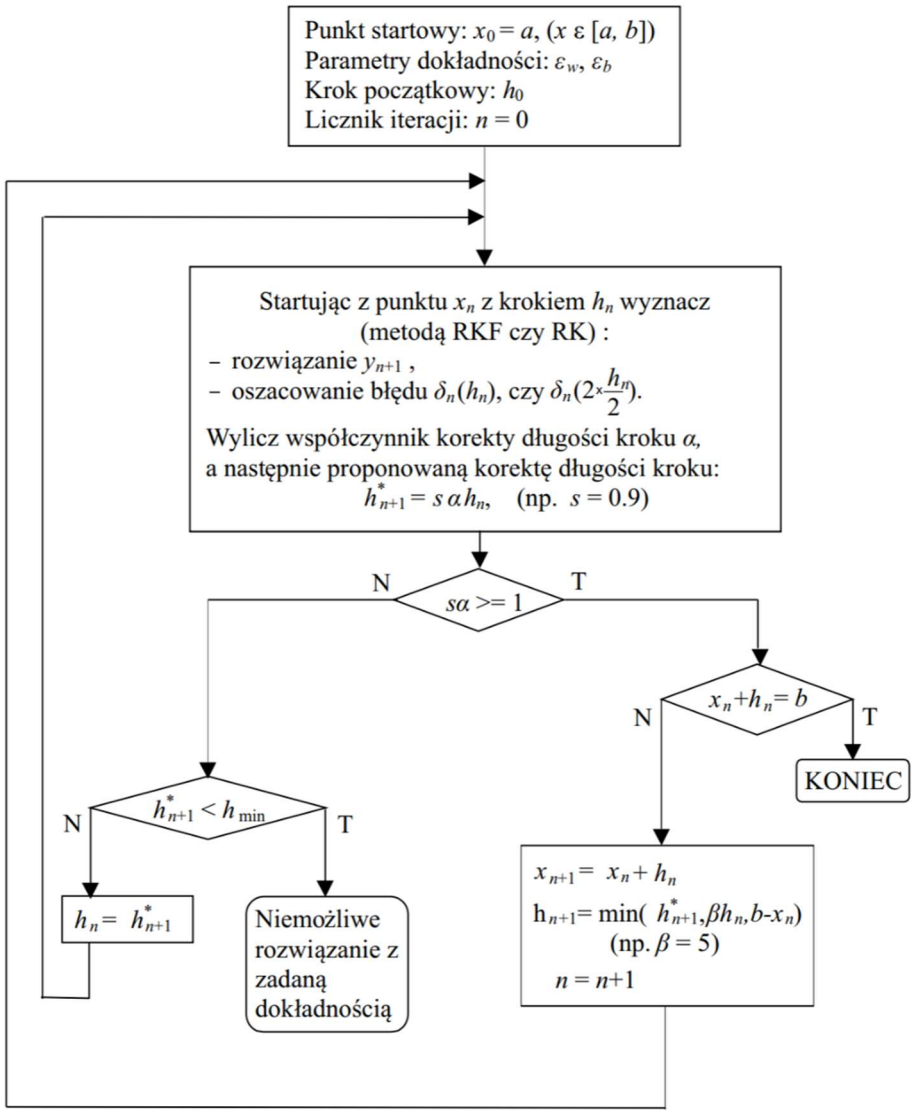
Teraz czasy rozwiązywania ostatniego z punktów dla różnych kroków:

```
>> predcorvis(f, [0.001, 0.001], [0, 20], 5);
    "Rozwiązanie zajęło: "    "0.0001061"    " s"
>> predcorvis(f, [0.001, 0.001], [0, 20], 2);
    "Rozwiązanie zajęło: "    "0.0001325"    " s"
>> predcorvis(f, [0.001, 0.001], [0, 20], 1);
    "Rozwiązanie zajęło: "    "0.0002983"    " s"
>> predcorvis(f, [0.001, 0.001], [0, 20], 0.2);
    "Rozwiązanie zajęło: "    "0.0005452"    " s"
>> predcorvis(f, [0.001, 0.001], [0, 20], 0.15);
    "Rozwiązanie zajęło: "    "0.0008495"    " s"
```

Widać, że w porównaniu do metody RK4, algorytm rozwiązuje nieco szybciej zadanie z tą samą dokładnością, dla tych samych warunków początkowych.

4 METODA RK ZE ZMIENNYM KROKIEM

Metoda Rungego-Kutty czwartego rzędu (RK4) istnieje również w formie **ze zmiennym krokiem**. Działa ona **tak samo jak RK4** ze **stałym krokiem**, z tym, że **wykorzystuje oszacowanie błędu do korekty kroku**, a także **zadawana jest jej oczekiwana dokładność**. Podstawowy schemat realizacji metody Rungego-Kutty został przedstawiony na poniższym schemacie blokowym:



Gdzie **współczynnik korekty długości kroku α** jako **współczynnik wyliczony dla najbardziej krytycznego równania** – dla **metod RK** dzieje się to z **szacowaniem błędu** wg **zasady podwójnego kroku** (patrz str. 3):

$$\delta_n(h)_i = \frac{(y_i)_n^{(2)} - (y_i)_n^{(1)}}{2^p - 1}, i = 1, 2, \dots, k$$

$$\epsilon_i = \left|(y_i)_n^{(2)}\right| \cdot \epsilon_w + \epsilon_b$$

$$\alpha = \min_{1 \le i \le k} \left(\frac{\epsilon_i}{|\delta_n(h)_i|}\right)^{\frac{1}{p+1}}$$