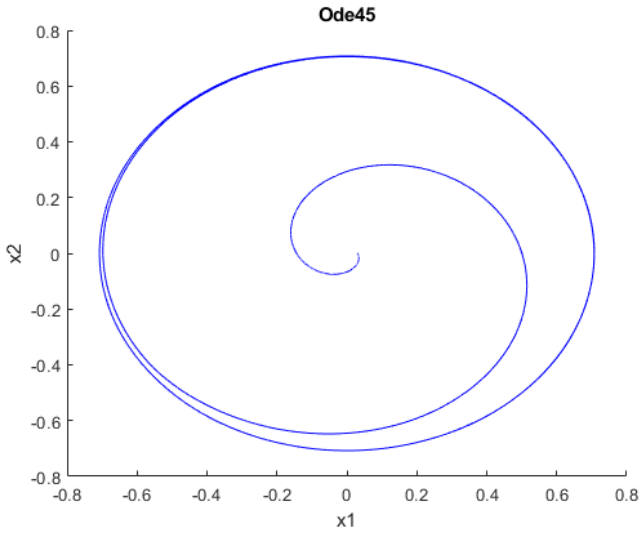
Feliks Bańka, 323733
MNUM – Projekt 4

Ruch punktu jest opisany równaniami:

$$dx_1/dt = x_2 + x_1(0{,}5 - x_1^2 - x_2^2),$$
$$dx_2/dt = -x_1 + x_2(0{,}5 - x_1^2 - x_2^2).$$

Należy obliczyć przebieg trajektorii ruchu na przedziale [0, 20] dla warunków początkowych: $x_1(0) = 0{,}03$, $x_2(0) = 0{,}001$. Rozwiązanie proszę znaleźć korzystając z zaimplementowanych **przez siebie** w języku Matlaba (w formie funkcji) metod:

Trajektoria uzyskana za pomocą wbudowanej w Matlab funkcji ode45:



Zadanie 1

Rungego–Kutty czwartego rzędu (RK4) oraz wielokrokowej predyktor–korektor Adamsa czwartego rzędu ze stałym krokiem. Proszę przy tym wykonać tyle prób (kilka – kilkanaście), ile będzie potrzebnych do znalezienia takiego kroku, którego zmniejszanie nie wpływa znacząco na rozwiązanie, podczas gdy zwiększanie – już wpływa;

Równania różniczkowe są często wykorzystywane do tworzenia tak zwanych układów dynamicznych. Są to modele matematyczne, które opisują zmianę jakiegoś zjawiska względem stanu początkowego. Równania różniczkowe układów dynamicznych, które zazwyczaj są nieliniowe, są przeważnie niemożliwe do rozwiązania w sposób analityczny, a jedynym sposobem na ich rozwiązanie jest wyznaczenie ich rozwiązania za pomocą metod numerycznych, które pozwalają wyznaczyć wynik z dużą dokładnością.

Metoda Rungego-Kutty ze stałym krokiem

Metoda Rungego-Kutty jest metodą jednokrokową, której krok w każdej iteracji jest stały.
Można ja opisać następującym wzorem:

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

Gdzie,

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

Metoda ta polega na obliczeniu wartości prawych stron równania różniczkowego m razy (w naszym przypadku m = 4). Wraz ze wzrostem m zwiększa się dokładność tej metody, ale jednocześnie rośnie też nakład obliczeń na jedna iteracje, a w związku tym zwiększę się też wpływ błędów zaokrągleń na końcowy wynik oraz czas potrzebne na wykonanie obliczeń. Dlatego też najczęściej używa się 4-etapowej metody, która jest idealnym kompromisem.
W 4 krokowej metodzie wzory na kolejne współczynniki prezentują się następująco:

$$k_1 = f(x_n, y_n)$$
$$k_2 = f\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_1\right)$$
$$k_3 = f\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_2\right)$$
$$k_4 = f(x_n + h, y_n + hk_3)$$

Współczynniki k_i możemy interpretować jako pochodne rozwiązania w kolejnych punktach (dla k_1 w (x_n, y_n)).

W taki sposób otrzymujemy 4 wartości pochodnych rozwiązania: po jednej na końcach przedziału i dwie w środku. Natomiast aproksymacja pochodnej dla pewnego kroku wyznaczania jest jako średnia wazony tych wartości, przy czym k_1 oraz k_4 mają wagę 1, a k_2 i k_3 mają wagę 2.

$$y_{n+1} = y_n + \frac{1}{6}h(k_1 + 2k_2 + 2k_3 + k_4)$$

Dla układu 2 równań (f i g) współczynniki prezentują się następująco (analogicznie dla funkcji g):

$$k_1 = f(x_n, y_n, z_n)$$
$$k_2 = f\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_{y1}, z_n + \frac{1}{2}hk_{z1}\right)$$
$$k_3 = f\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_{y2}, z_n + \frac{1}{2}hk_{z2}\right)$$
$$k_4 = f\left(x_n + h, y_n + hk_{y3}, z_n + \frac{1}{2}hk_{z3}\right)$$

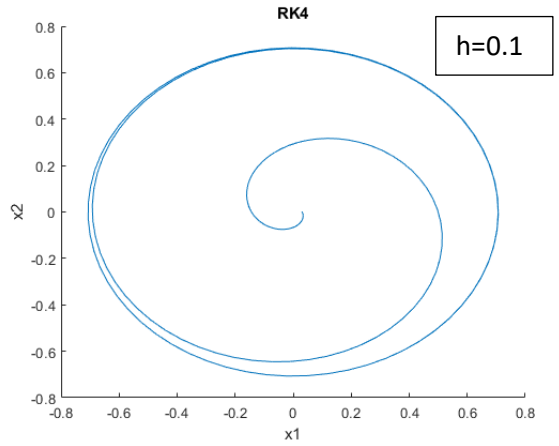
Metoda ta wymaga od nas założenia długości kroku. Nie może on być za mały, ponieważ wtedy obliczenia zajmą bardzo dużo czasu, a dokładność maszynowa wpłynie znacząco na wynik. Jednocześnie musi być na tyle mały, aby metoda była zbieżna. Metoda ta wymaga od nas najwcześniej wielokrotnego zmieniania długości kroku i testowania wpływu tej zmiany na efekt końcowy, a więc ostateczną długość kroku wybieramy niejako metoda prób i błędów.

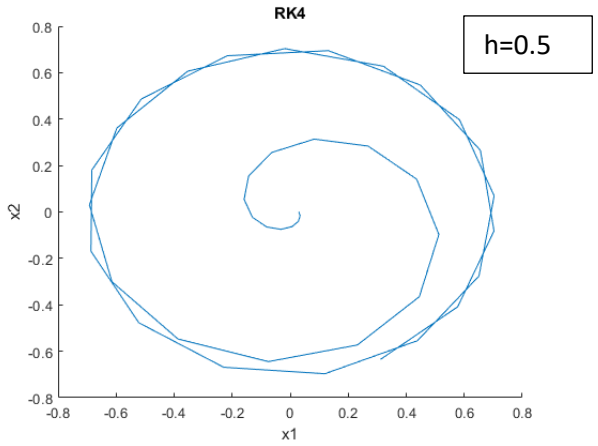
Implementacja w Matlab'ie

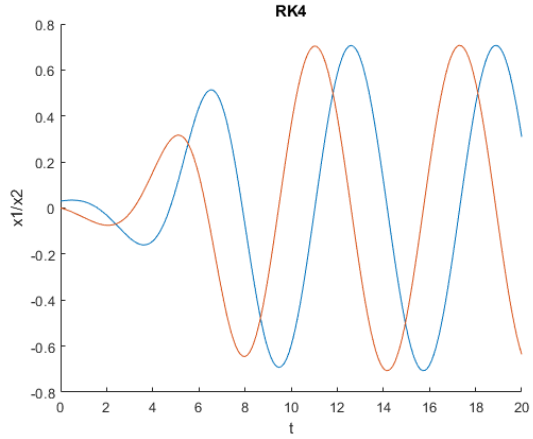
```
function [y] = RK4(dx1,dx2,x1,x2,h,a,b)
% Returns y, which represents a list of points (x1,x2) calculated using ...
    tic;
    t=a:h:b;
    halfstep=h/2;
    X1(:,1) = x1;
    X2(:,1) = x2;
    for i=1:(length(t)-1)
        k11=dx1(x1,x2);
        k21=dx2(x1,x2);

        k12=dx1(x1+halfstep*k11,x2+halfstep*k21);
        k22=dx2(x1+halfstep*k11,x2+halfstep*k21);

        k13=dx1(x1+halfstep*k12,x2+halfstep*k22);
        k23=dx2(x1+halfstep*k12,x2+halfstep*k22);

        k14=dx1(x1+h*k13,x2+h*k23);
        k24=dx2(x1+h*k13,x2+h*k23);

        x1=x1+(h/6)*(k11+2*(k12+k13)+k14);
        x2=x2+(h/6)*(k21+2*(k22+k23)+k24);

        X1(:,i+1)=x1;
        X2(:,i+1)=x2;
    end
    toc;
    y = [X1; X2];
end
```

Wyniki

Metoda ta jest nieskuteczna dla h większego niż 0.2







Metoda predyktor – korektor Adamsa czwartego rzędu ze stałym krokiem

Jest to metoda wielokrokowa. Działa ona na podstawie algorytmu typu predyktor-korektor, który jest polaczeniem metod jawnych (Adamsa-Bashfortha) i niejawnych (Adamsa-Moultona), dzięki czemu łącza one zalety obu tych metod, mianowicie ma ona:

1. wysoki rząd i małą stałą błędu (metody niejawne)
2. możliwie duży obszar absolutnej stabilności (metody niejawne)
3. możliwie małą liczbę obliczeń na iterację (metody jawne)

Dla k-kroków metoda ta ma realizację w postaci struktury predyktor-korektor $P_k E K_k E$:

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

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

K: $y_n = \sum_{j=1}^{k} \alpha_j^* y_{n-j} + h \sum_{j=1}^{k} \beta_j^* f_{n-j} + h\beta_0^* f_n^{[0]}$ (K – korekcja)

E: $y_n = f(x_n, y_n)$ (E – ewaluacja)

Iteracja predyktora to obliczenie dobrego punktu początkowego dla iteracji algorytmu korelatora, które rozwiązuje nieliniowe równanie algebraiczne metody niejawnej korektora metoda iteracji prostej.

Dla metody Adamsa algorytm $P_k E K_k E$ ma postać:

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

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

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

E: $y_n = f(x_n, y_n)$

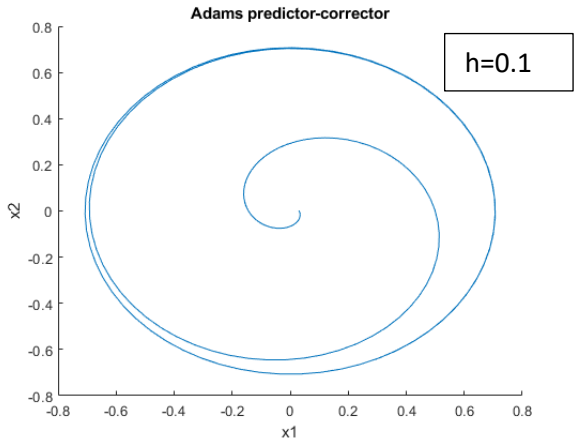
Jeżeli predyktor jest dostatecznie dokładny (jego rząd jest nie mniejszy od rzędu korektora), to dla dostatecznie małych wartości kroku h uzyskanie maksymalnego rzędu (tj. rzędu metody korektora) następuje w algorytmie PK już po jednej iteracji korektora. W innym wypadku niezbędne będzie więcej iteracji.

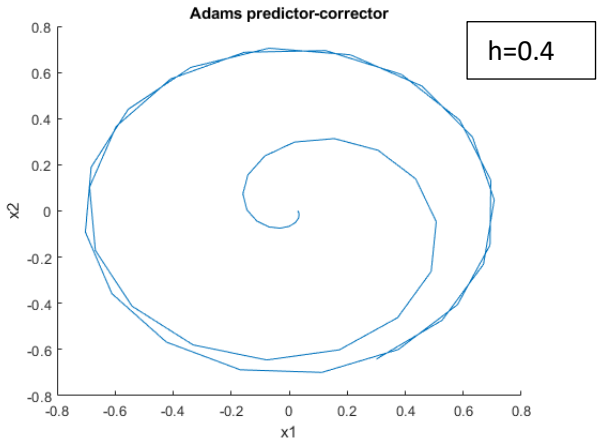
Implementacja w Matlab'ie

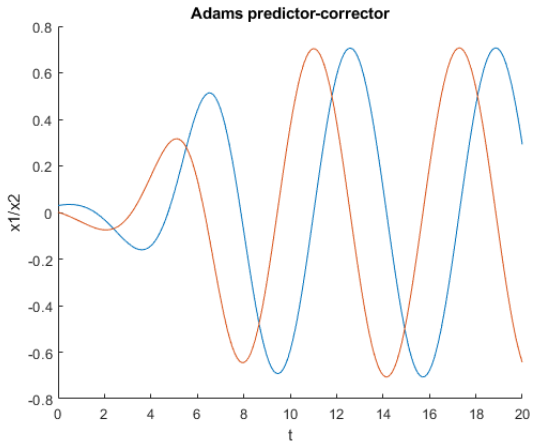
```
function [y] = P_K_Adams(dx1,dx2,x1,x2,h,a,b)
% Returns y, which represents a list of points (x1,x2) calculated using ...
    tic;
    t=a:h:b;
    X1(:,1) = x1;
    X2(:,1) = x2;
    % initializing
    for i=1:3
        [x1,x2]=rk4(dx1,dx2,x1,x2,h); % calculating using rk4 algorythm (k1,k2,k3,k4)

        X1(:,i+1)=x1;
        X2(:,i+1)=x2;
    end
    % constants
    B_bashforth = [55/24,-59/24,37/24,-9/24];
    B_moulton = [252/720,646/720,-264/720,106/720,-19/720];
    for i = 4:(length(t)-1)
        % prediction
        sum1 = 0;
        sum2 = 0;
        for j=0:3
            sum1 = sum1 + B_bashforth(j+1)*dx1(X1(i-j),X2(i-j));
            sum2 = sum2 + B_bashforth(j+1)*dx2(X1(i-j),X2(i-j));
        end
        pred1 = x1 + sum1*h;
        pred2 = x2 + sum2*h;
        % corection
        sum1 = 0;
        sum2 = 0;
        for j=0:3
            sum1 = sum1 + B_moulton(j+2)*dx1(X1(i-j),X2(i-j));
            sum2 = sum2 + B_moulton(j+2)*dx2(X1(i-j),X2(i-j));
        end
        x1 = x1 + (sum1+B_moulton(1)*dx1(pred1, pred2))*h;
        x2 = x2 + (sum2+B_moulton(1)*dx2(pred1, pred2))*h;

        X1(:,i+1)=x1;
        X2(:,i+1)=x2;
    end
    toc;
    y = [X1; X2];
end
```

Wyniki

Metoda ta jest nieskuteczna dla h większego niż 0.25







Zadanie 2

Rungego–Kutty czwartego rzędu (RK4) ze zmiennym krokiem, zmienianym automatycznie przez algorytm wykorzystujący estymację błędu aproksymacji według zasady podwójnego kroku.

Metoda Rungego-Kutty (RK4) ze zmiennym krokiem

Ze względu na liczne problemy z ustaleniem stałego kroku (metoda wykonuje małe kroki nawet w miejscach, gdzie nie jest to konieczne) często stosuje się metody ze zmiennym krokiem, co pozwala na zmniejszenie tego problemu.

W miejscach, w których nie jest potrzebna duża ilość kroków po prostu zwiększa się długość kroku, a w związku z czym zmniejsza się liczbę iteracji.

W każdej iteracji wyliczany jest błąd aproksymacji, a na jego podstawie współczynnik zmiany długości kolejnego kroku:

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

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

$$h_{n+1}^* = s\alpha h_n$$

, gdzie:

$(y_i)^{(2)}$ jest punktem uzyskanym po krokach $h/2$,

$(y_i)^{(1)}$ jest punktem uzyskanym po kroku h,

p jest rzędem metody obliczeniowej (u nas p=4)

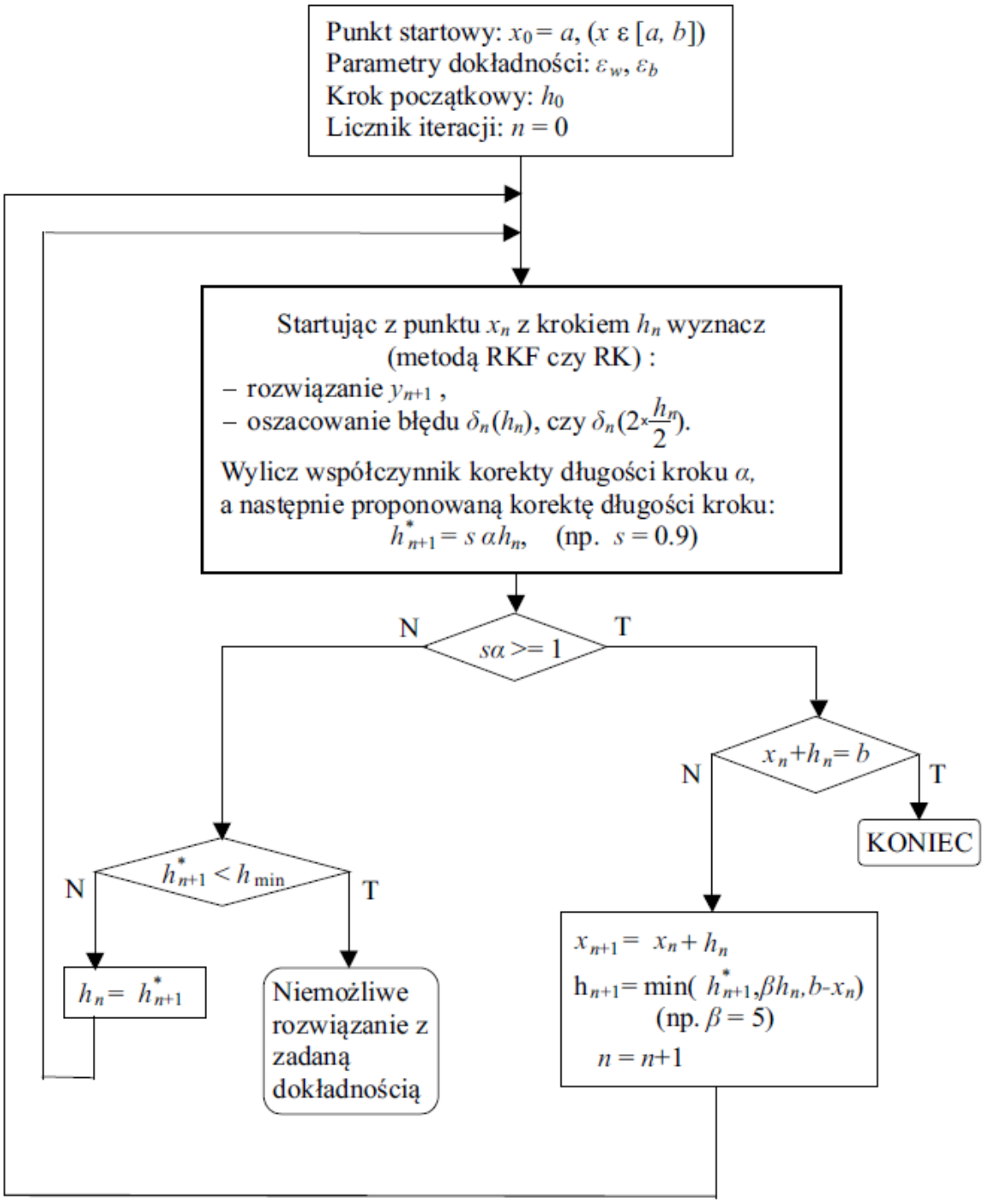
s jest tzw. współczynnikiem bezpieczeństwa (dla RK4 zakłada się $s = 0.9$),

h_{n+1}^* jest proponowana korektą długości kroku.

Dokładność obliczeń wyliczamy ze wzoru:

$$\varepsilon = |y_n| \cdot \varepsilon_w + \varepsilon_b$$

Schemat blokowy działania tej metody widoczny jest poniżej:

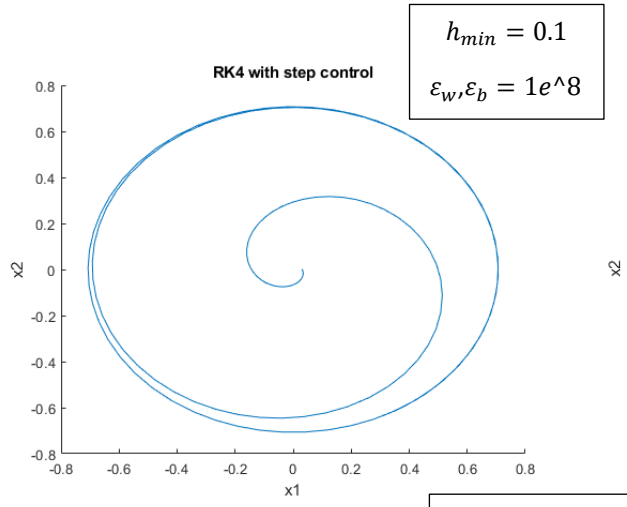


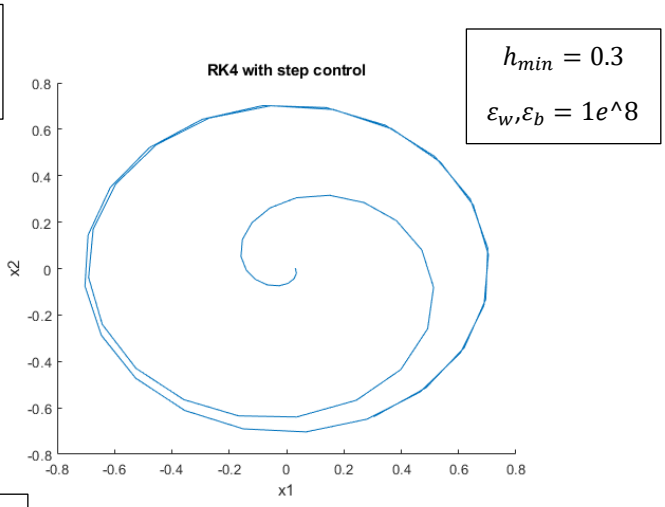
Implementacja w Matlab'ie

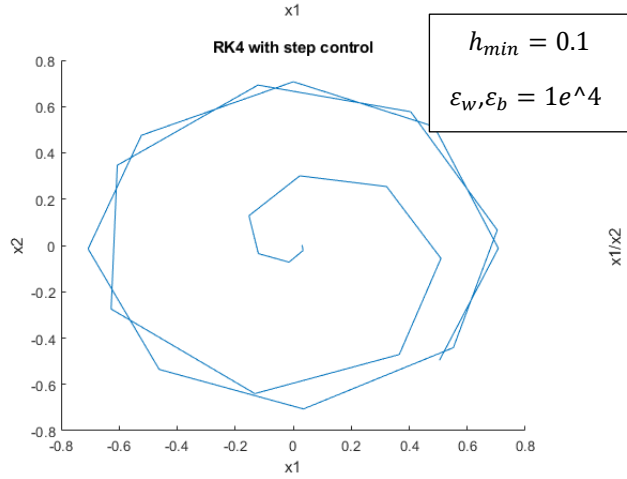
```
function [y] = RK4_with_step_control(dx1,dx2,x1,x2,h,a,b,epsW,epsB,hmin,imax)
% Returns y, which represents a list of points (x1,x2) calculated using ...
    tic
    t = zeros(imax,1); % time [a,b]
    t(1,1) = a;
    X1=zeros(imax,1);
    X2=zeros(imax,1);
    X1(1,1)=x1;
    X2(1,1)=x2;
    x1d=x1;
    x2d=x2;
    H=zeros(imax,1); % steps length H(t)
    H(1,1)=h;
    n=1;
    while(t(n,1)<b && n < imax)
        [x1,x2]=rk4(dx1,dx2,x1d,x2d,h); % calculating using rk4 algorythm (k1,k2,k3,k4)

        [x1d,x2d]=rk4(dx1,dx2,x1d,x2d,h/2);
        [x1d,x2d]=rk4(dx1,dx2,x1d,x2d,h/2);
        X1(n+1,1)=x1d;
        X2(n+1,1)=x2d;

        errorX1=(x1d-x1)/15; % calculating aproximation error
        errorX2=(x2d-x2)/15;
        eps1 = abs(x1d) * epsW + epsB; % calculating measurement error
        eps2 = abs(x2d) * epsW + epsB;

        alfa1 = (eps1/abs(errorX1))^(1/5);
        alfa2 = (eps2/abs(errorX2))^(1/5);
        alfa = min(alfa1, alfa2);
        if (0.9 * alfa >= 1)
            if (t(n,1) + h < b)
                corr = 0.9 * alfa * h;
                h = min(b - t(n,1), min(2*h, corr));
            end
        else
            corr = 0.9 * alfa * h;
            if(corr >= hmin)
                h = corr;
            end
        end
        t(n+1,1) = t(n,1)+h;
        H(n+1,1)=h;
        n=n+1;
    end
    toc;
    y = [X1(1:n), X2(1:n)];
end
```

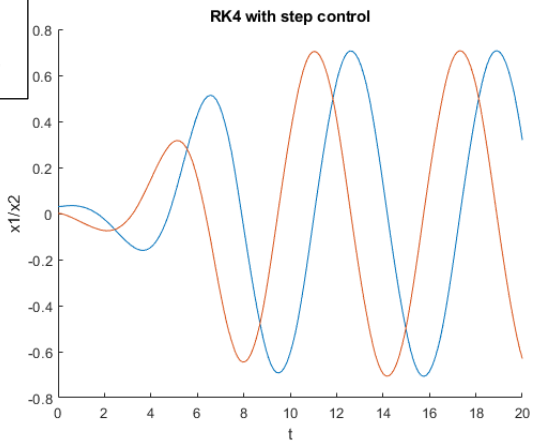
Wyniki

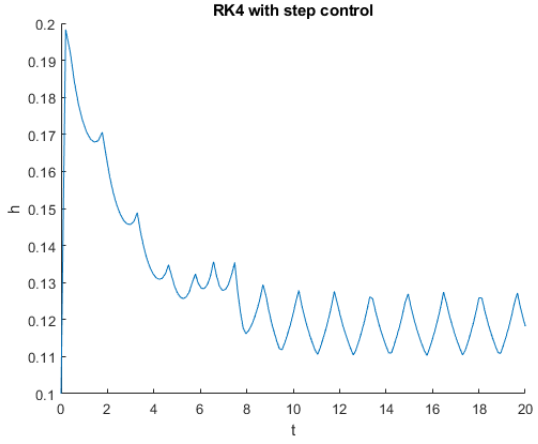
W tej metodzie w odróżnieniu do dwóch poprzednich wartość początkowej długości kroku ma znaczenie tylko dla pierwszej iteracji, gdyż potem i tak jest on zmieniany. Znaczenie ma natomiast h_{min}, ε_w, ε_b. Zwiększenie, którejkolwiek z tych wartości powoduje zwiększenie się długości kroku. W naszym przypadku granicznymi parametrami są: $h_{min} = 0.1$, ε_w oraz $\varepsilon_b = 1e\text{^}8$.

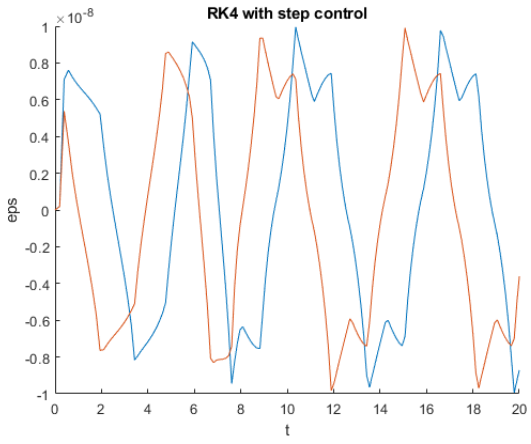












Wnioski

Tak prezentują się czasy wykonania metod (krok dla uzyskania podobnej precyzji)

Metoda	Krok	Czas
RK4	0.01	0.002760
P-K Adamsa	0.05	0.002998
RK4 zmienny krok	0.1	0.000704
Ode45	0.1	0.002519

Jak widać metoda RK4 ze zmiennym korkiem jest zdecydowanie najszybsza, jest to spowodowane faktem, iż znacznie zmniejsza ona liczbę iteracji, dzięki modyfikacji długości kroku. Co więcej uniezależnia nas ona od błędu wynikającego z błędnego wybrania kroku początkowego.

Zarówno metoda RK4 ze stałym krokiem, jak i metoda Adamsa mogą być bardzo niedokładne, jeśli tylko wybierzemy za długi krok, natomiast metoda Adamsa w niewielkim stopniu niweluje te błędy, dzięki czemu mamy większy zakres, w którym wynik będzie zadowalający.