MNUM-PROJEKT zadanie 4.2

Tomasz Jakubczyk

Obliczanie trajektorii ruchu punktu opisanego równaniami

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

na przedziale [0, 20].

1) Metoda Rungego-Kutty czwartego rzędu ze stałym krokiem polega na wyznaczaniu kolejnych wartości funkcji za pomocą wartości pochodnej w czterech następnych pobliskich punktach, a następnie obliczenie średniej ważonej tych pochodnych i wykorzystanie jej do zwiększenia odpowiednio wartości dla nowego punktu.
   Ogólny wzór metody Rungego-Kutty dowolnego rzędu:

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

$$\begin{aligned} k_1 &= f(x_n, y_n), \\ k_i &= f\left(x_n + c_i h, y_n + h \cdot \sum_{j=1}^{i-1} a_{ij} k_j\right), \qquad i = 2, 3, \ldots, m \end{aligned}$$

   Wzór na metodę RK4:

$$\begin{aligned} y_{n+1} &= y_n + \frac{1}{6} h (k_1 + 2k_2 + 2k_3 + k_4), \\ k_1 &= f(x_n, y_n), \\ k_2 &= f(x_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_1), \\ k_3 &= f(x_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_2), \\ k_4 &= f(x_n + h, y_n + hk_3). \end{aligned}$$

   Błąd w kroku metody Rungego-Kutty szacuje się za pomocą zasady podwójnego kroku opisanego wzorem:

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

   Polega to na tym, że oblicza się różnicę wartości następnego punktu obliczonego z normalnym krokiem i tegoż punktu obliczonego w dwóch o połowę krótszych krokach.

Implementacja metody RK4:

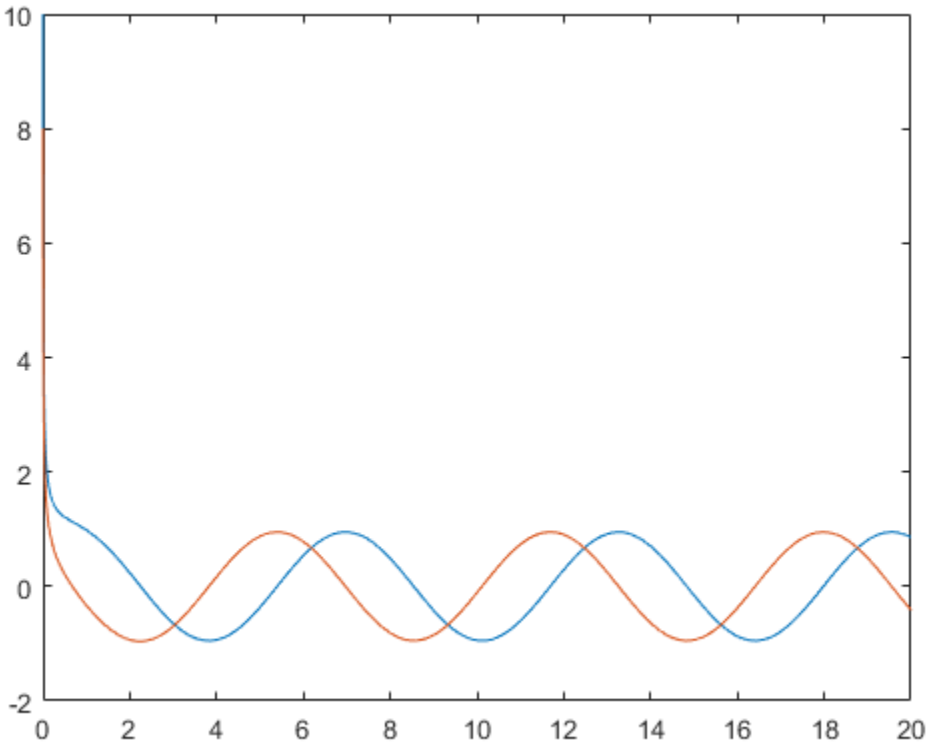
```
function [ x, T, errx, t ] = RK4_staly_krok( dx, x0, przedzial, krok )
%RK4_staly_krok Metoda Rungego-Kutty czwartego rzêdu ze sta³ym krokiem
%   dx - uk³ad równañ ró¿niczkowych pierwszego rzêdu
%   x0 - punkt pocz¹tkowy
%   przedzia³ - zadany przedzia³ (od zera)
%   x - rozwi¹zanie
%   T - interwa³y przedzia³u (potrzebne do wykresu itp)
%   errx - b³¹d rozwi¹zania
%   t - czas obliczeñ
tic;
yn=x0;%y0
h=krok;
h2=0.5*h;
j=1;
x=zeros(length(przedzial(1):krok:przedzial(2)),2);
errx=zeros(length(przedzial(1):krok:przedzial(2)),2);
T=zeros(length(przedzial(1):krok:przedzial(2)),1);
x(j,:)=yn';
T(j)=0;
for i=przedzial(1):krok:przedzial(2)
    j=j+1;
    k1=dx(yn);%zmiana oznaczeñ w celu zgodnoœci z podrêcznikiem
    k2=dx(yn+0.5*h*k1);
    k3=dx(yn+0.5*h*k2);
    k4=dx(yn+h*k3);
    ynp1=yn+1/6*h*(k1+2*k2+2*k3+k4);
    x(j,:)=ynp1';
    for i2=1:2%obliczenie yn+1 w dwóch o po³owê mniejszych krokach na potrzeby
oszacownaia b³êdu
        k1=dx(yn);
        k2=dx(yn+0.5*h2*k1);
        k3=dx(yn+0.5*h2*k2);
        k4=dx(yn+h2*k3);
        yn=yn+1/6*h2*(k1+2*k2+2*k3+k4);
    end
    errx(j,:)=abs(16/15*(yn-ynp1));
    yn=ynp1;
    T(j)=i+h;
end
t=toc;
end
```

Zadany układ równań zdefiniowałem tak:

```
dx = @(x) [  x(2)+x(1)*(0.9-x(1)^2-x(2)^2) , ...
            -x(1)+x(2)*(0.9-x(1)^2-x(2)^2) ];
```

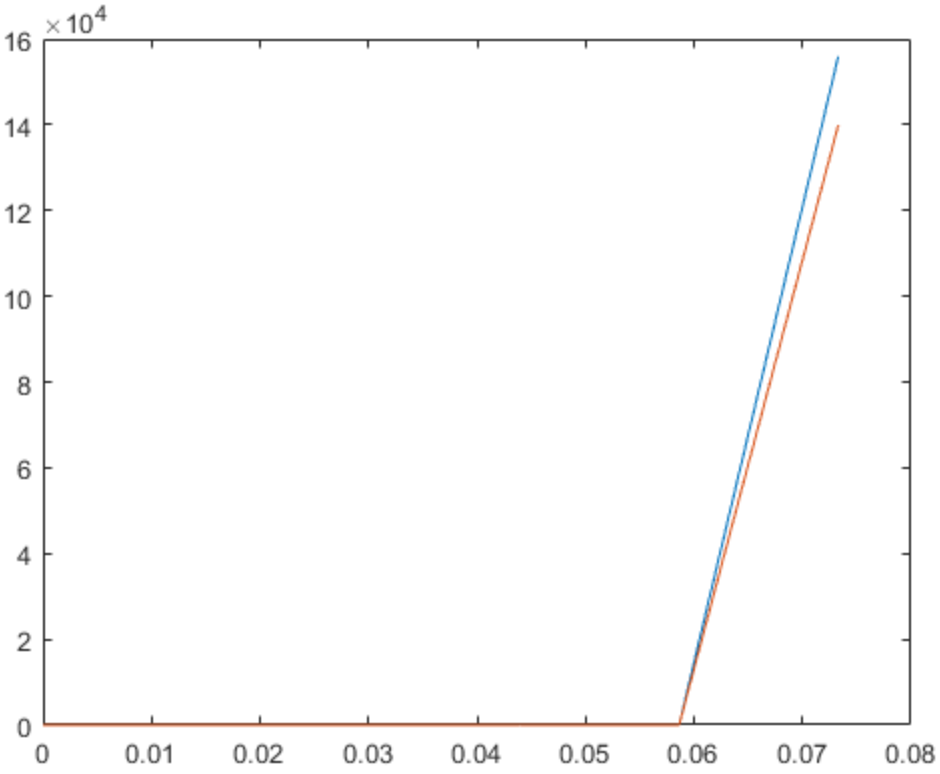
Punkt początkowy a [10 8]

Rozwiązanie ode45:

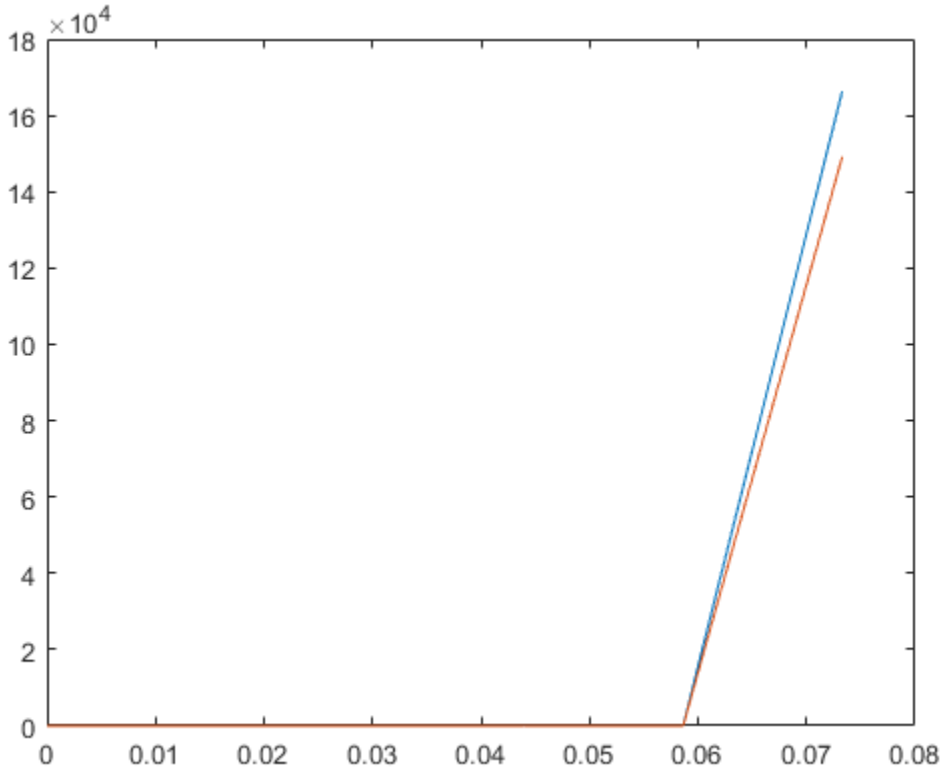


Rozwiązanie RK4 z najmniejszym jaki znalazłem krokiem który jest zbyt duży

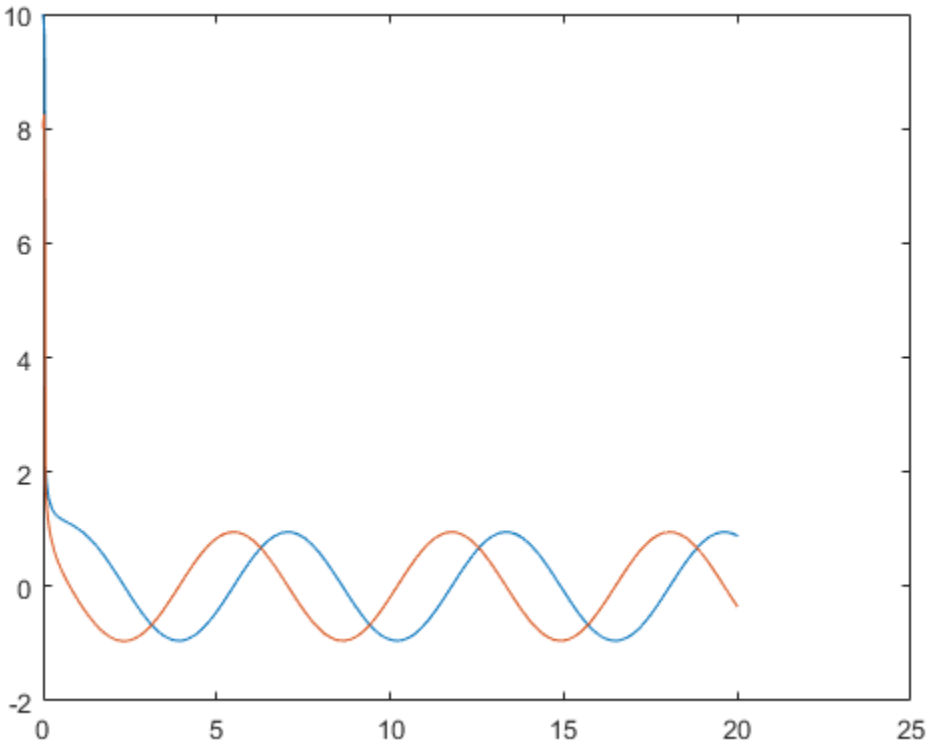
```
krok=0.0146583
```



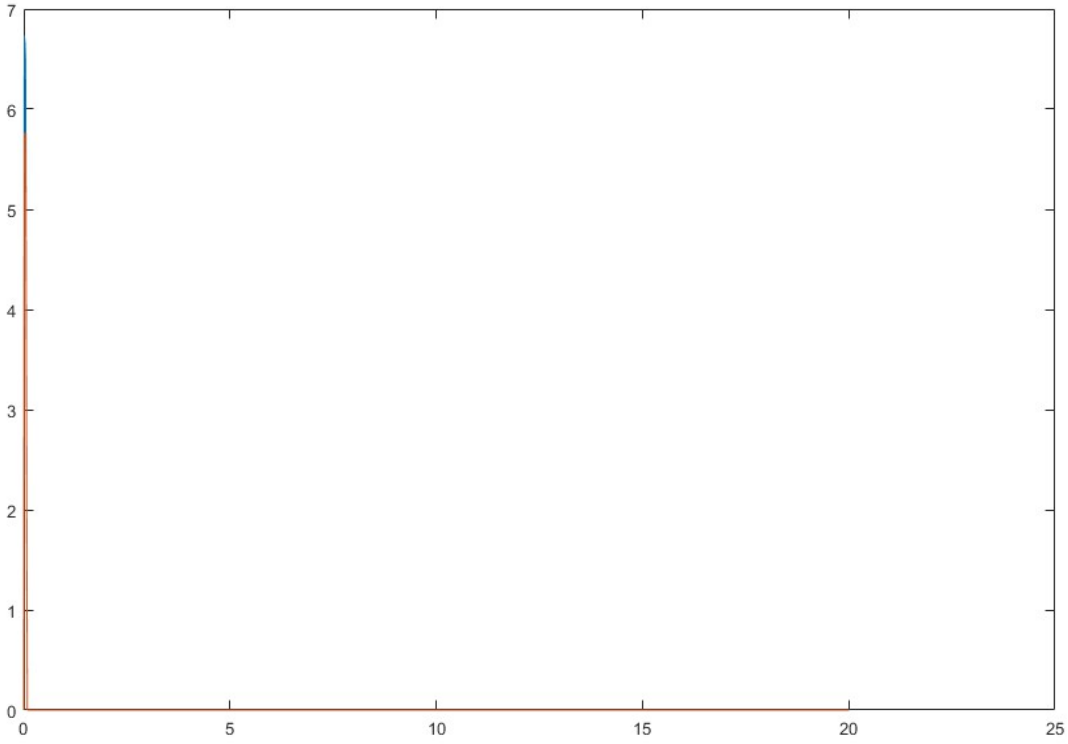
Ze względu na dużą wartość pochodnej w punkcie początkowym algorytm podawał coraz mniej dokładne rozwiązania, aż zwrócił nieskończoność w kroku 7.
Oszacowane błędy wyglądały tak:



Nieco większy `krok=0.0146582`
zwraca wykres wyglądający identycznie z ode45:



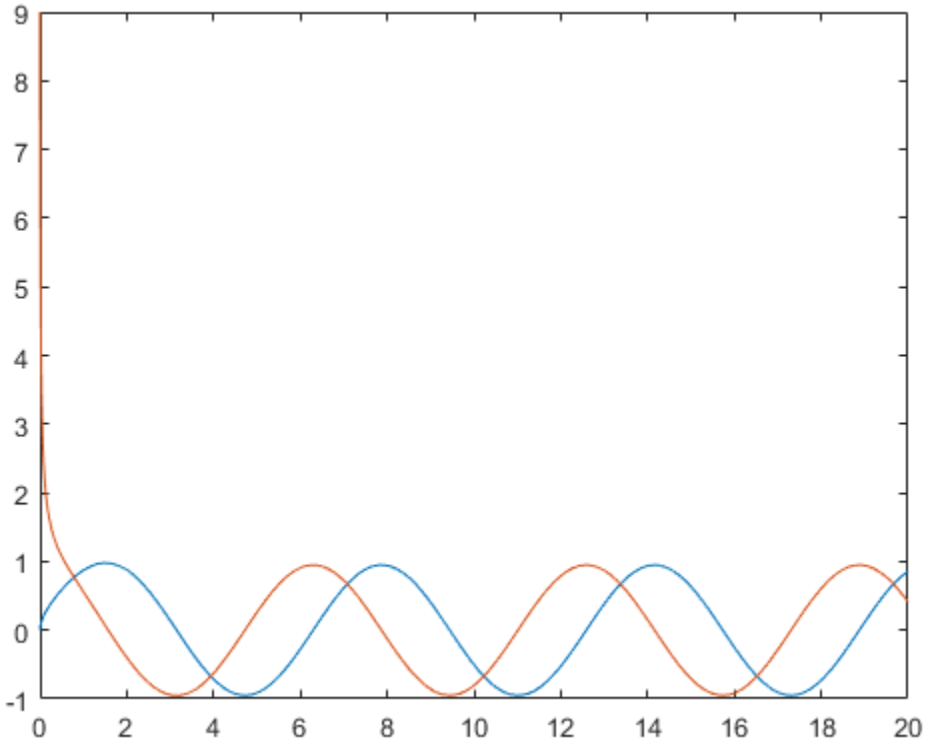
szacowane błędy:



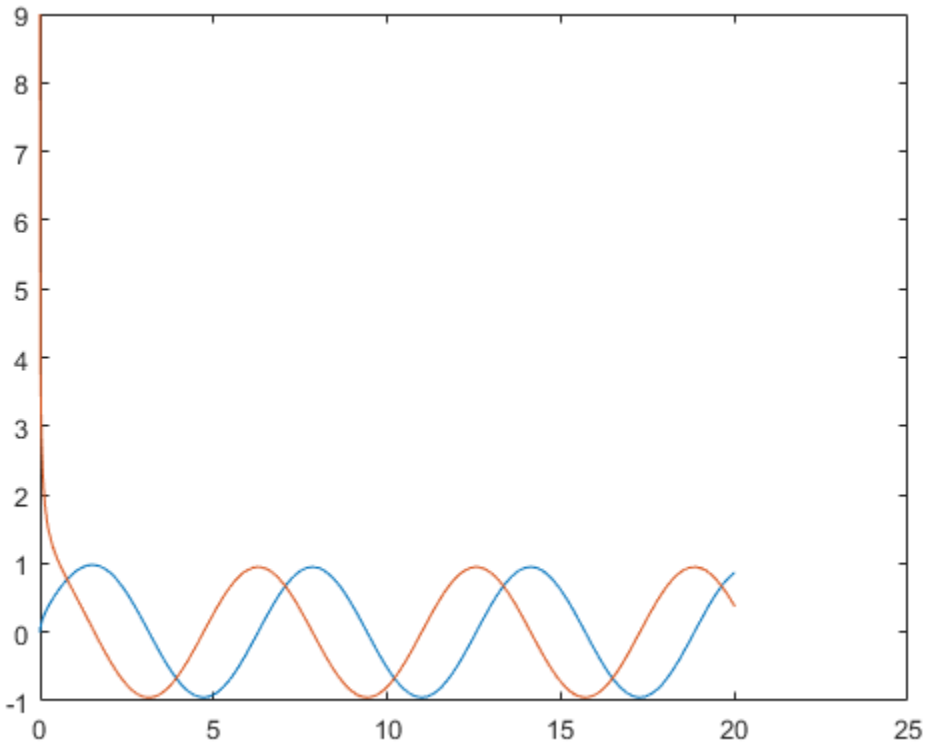
Wyższa wartość po pierwszym kroku wnioskuję z obserwacji działania programu, że

wynika z tego, że pochodna jest w tym miejscu bardzo duża i z działania oszacowania. Rząd wielkości tego błędu spada ze zmniejszeniem korku. Dla h=0.001 błąd jest już rzędu 10^-5.

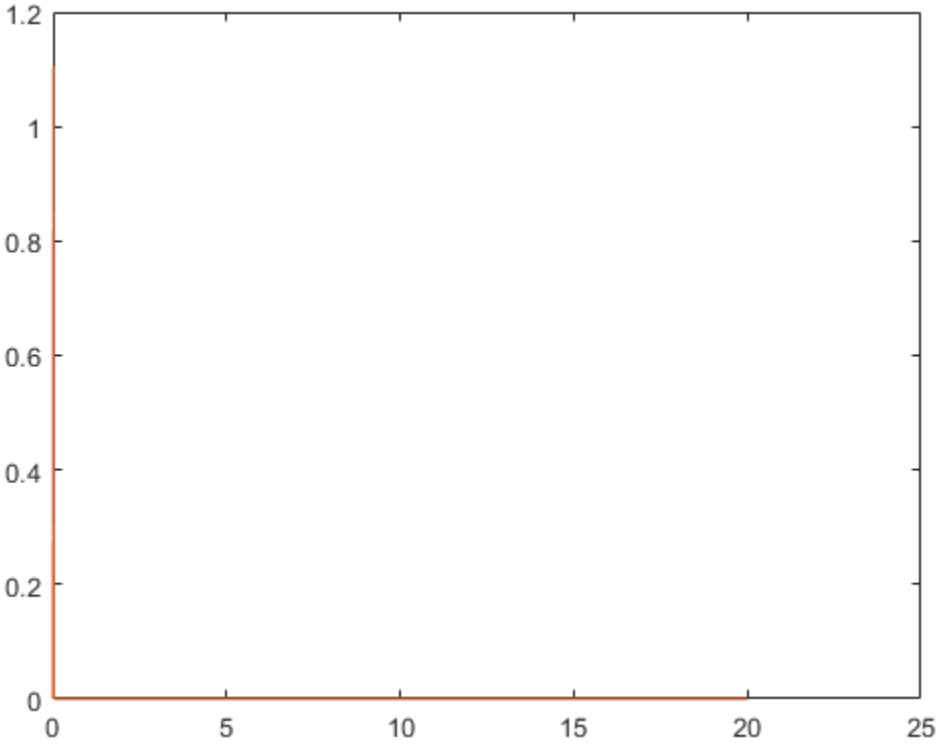
Punkt początkowy b [0 9]
ode45:



RK4 krok `0.0146582` znów okazał się pierwszym wystarczającą krótkim, punkt b jest bardzo podobny do a:

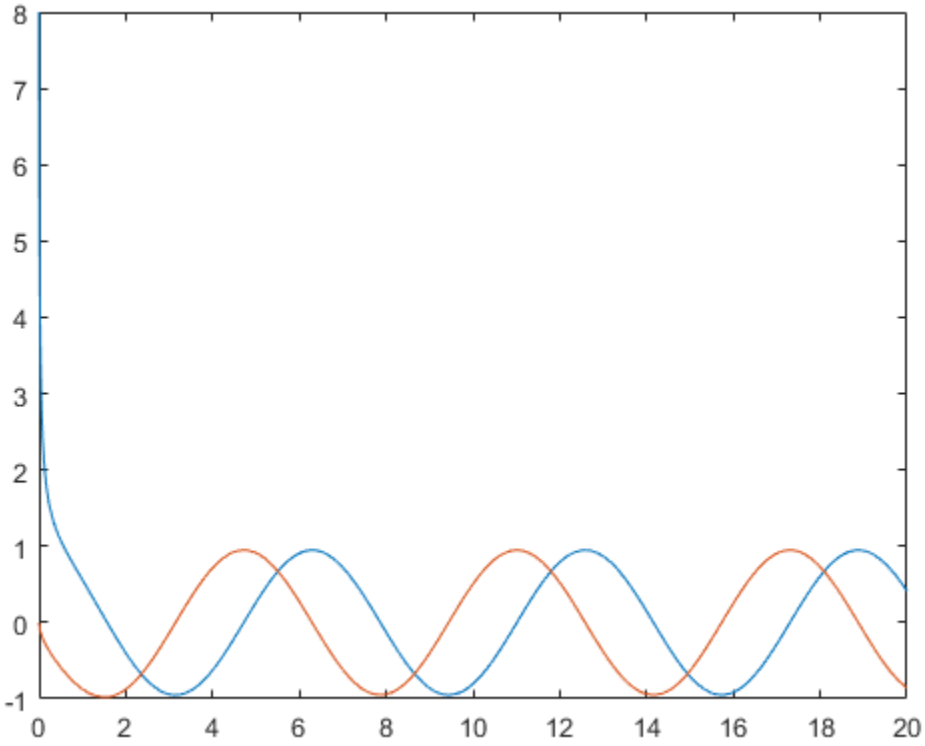


Szacowane błędy wyglądały tak:

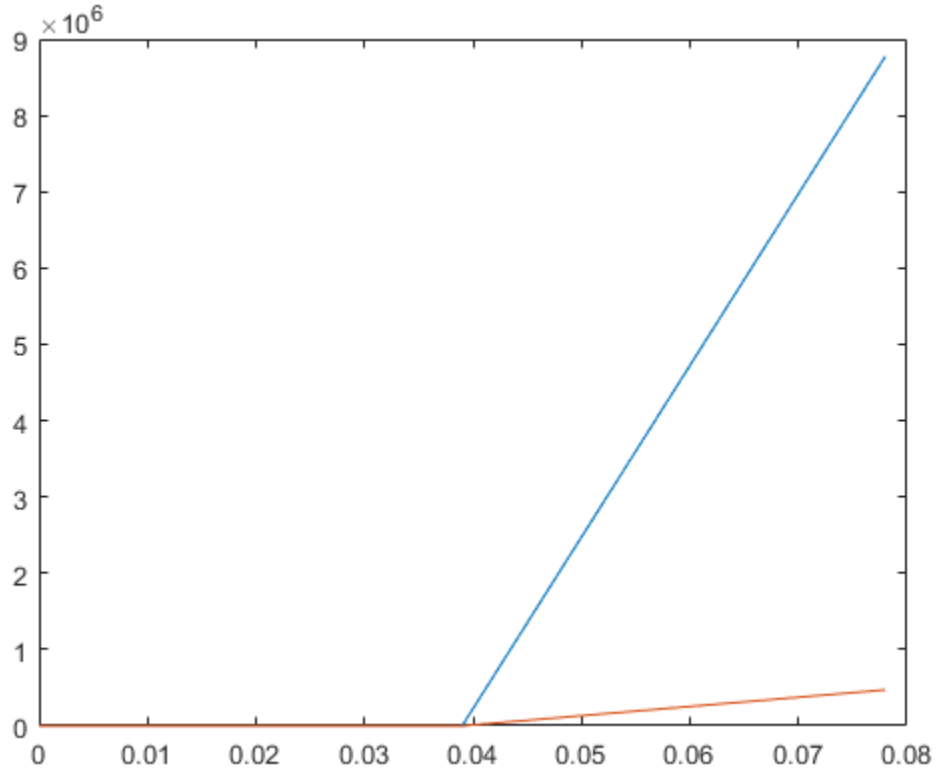


Punkt początkowy c [8 0]

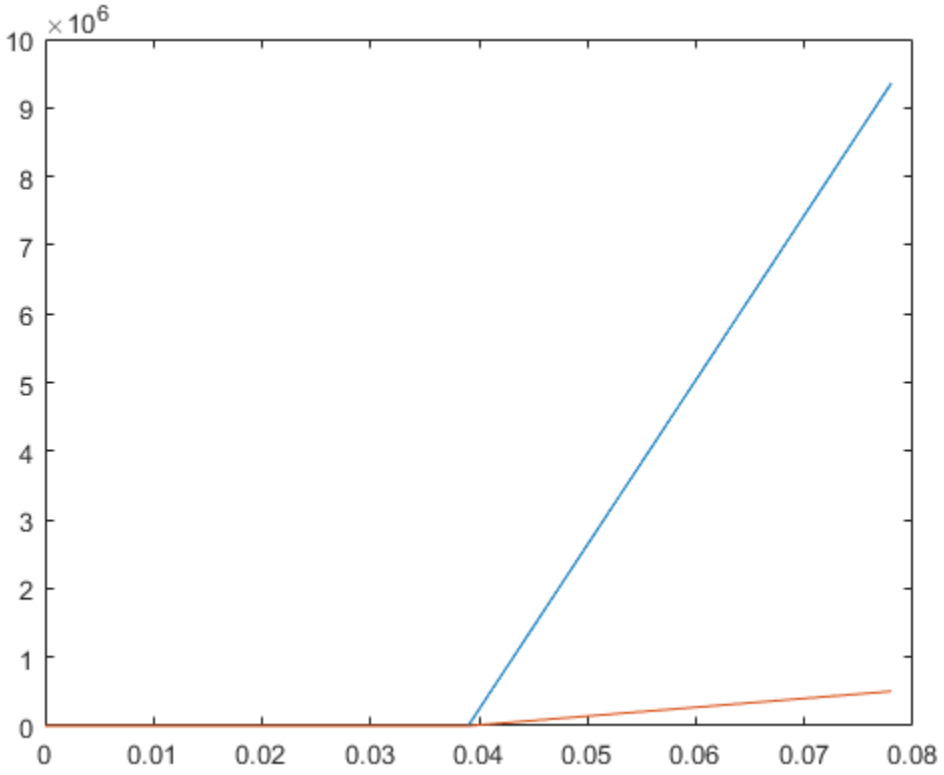
ode45:



RK4 dla kroku 0.039 który okazał się w przybliżeniu pierwszym za dużym wyszło tak:

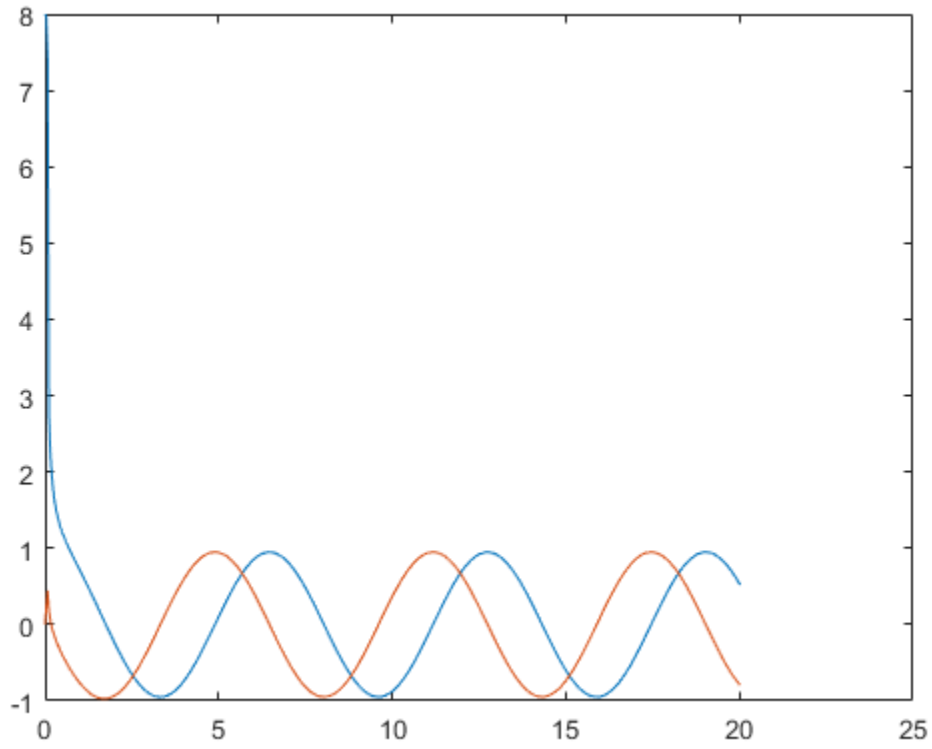


Szacowane błędy:

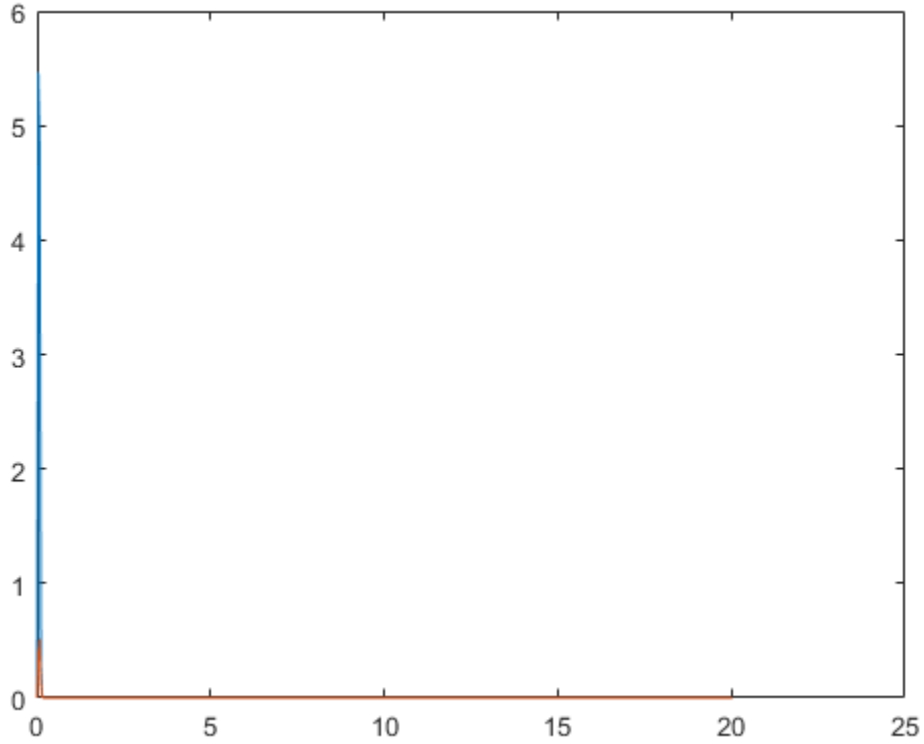


Jak rozwiązanie szybko się psuje, to szacowane błędy też.

RK4 z krokiem 0.038 wygląda już jak ode45:

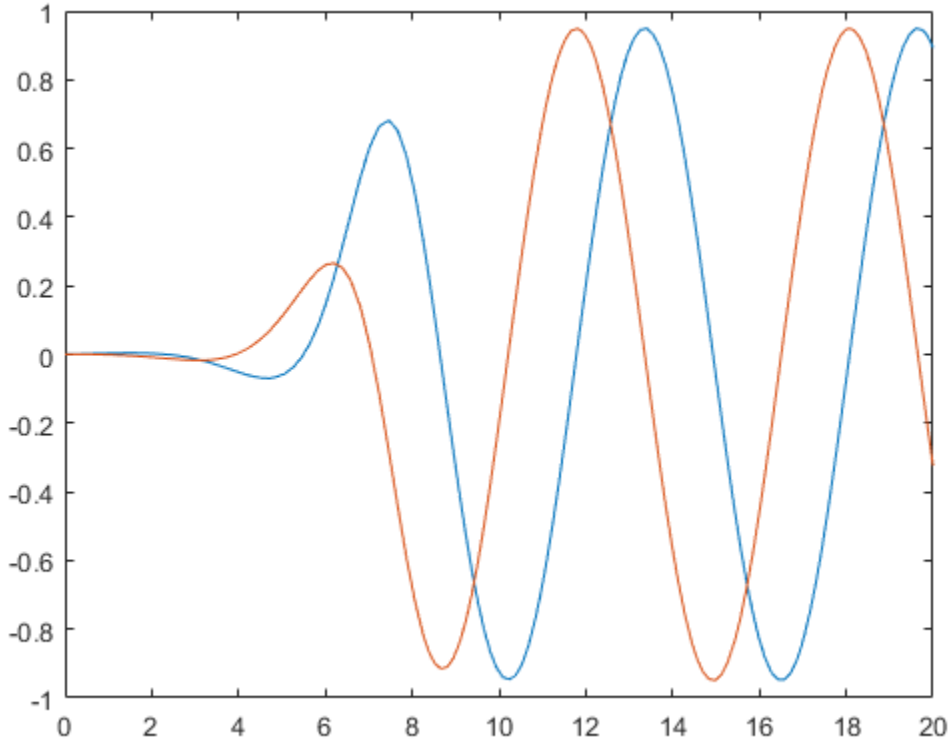


Szacowane błędy:

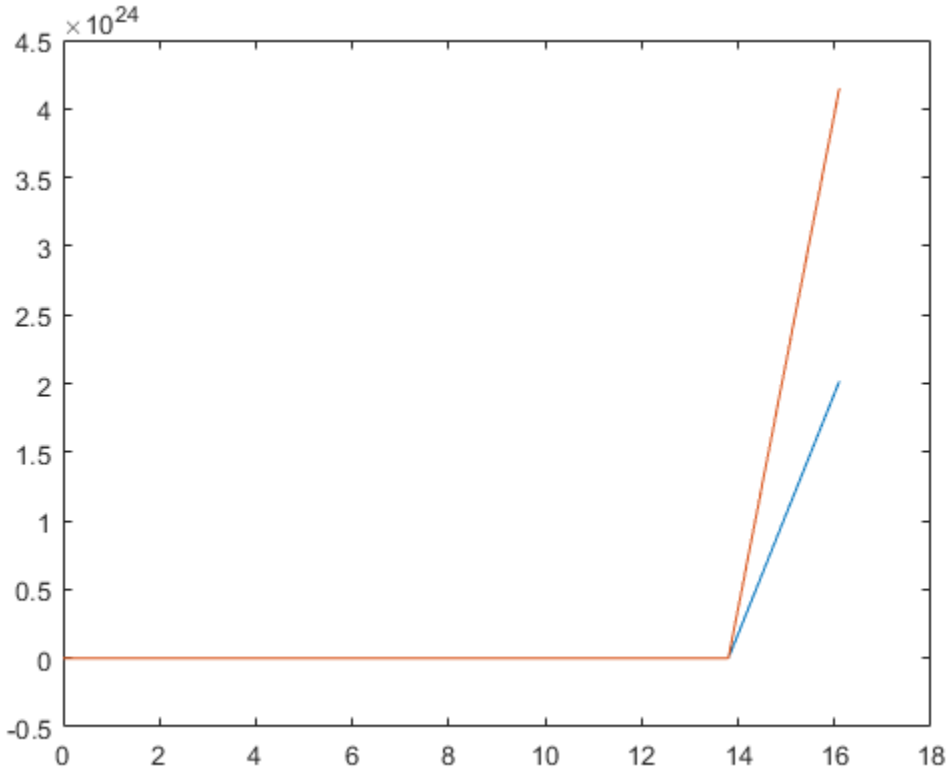


Punkt początkowy d [0.001 0.001]

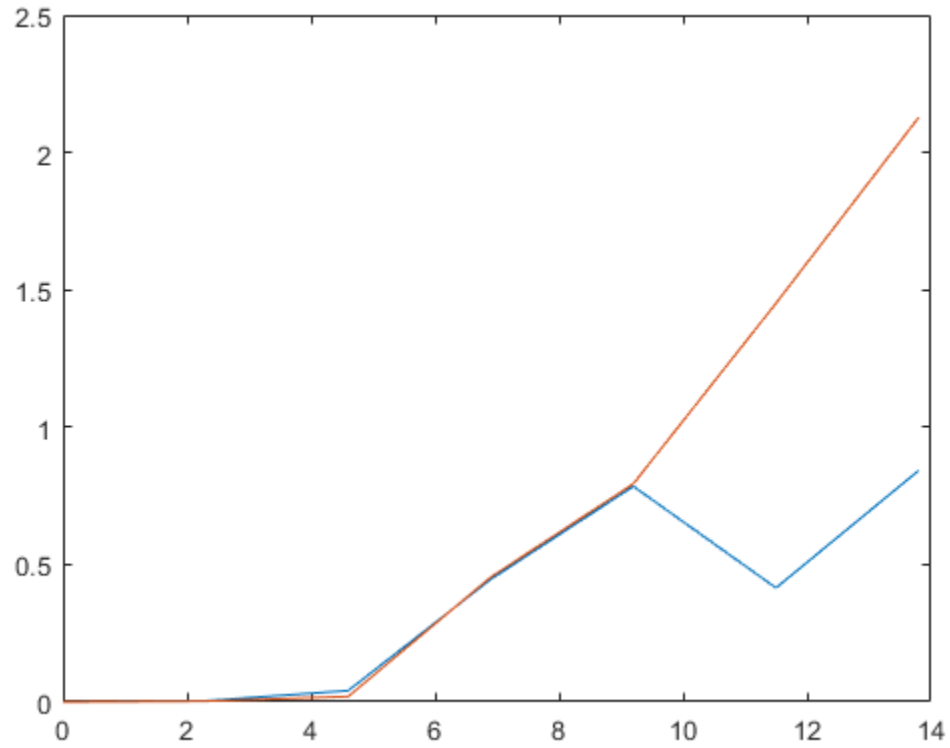
ode45:



RK4 przy kroku 2.3 definitywnie przestaje działać:

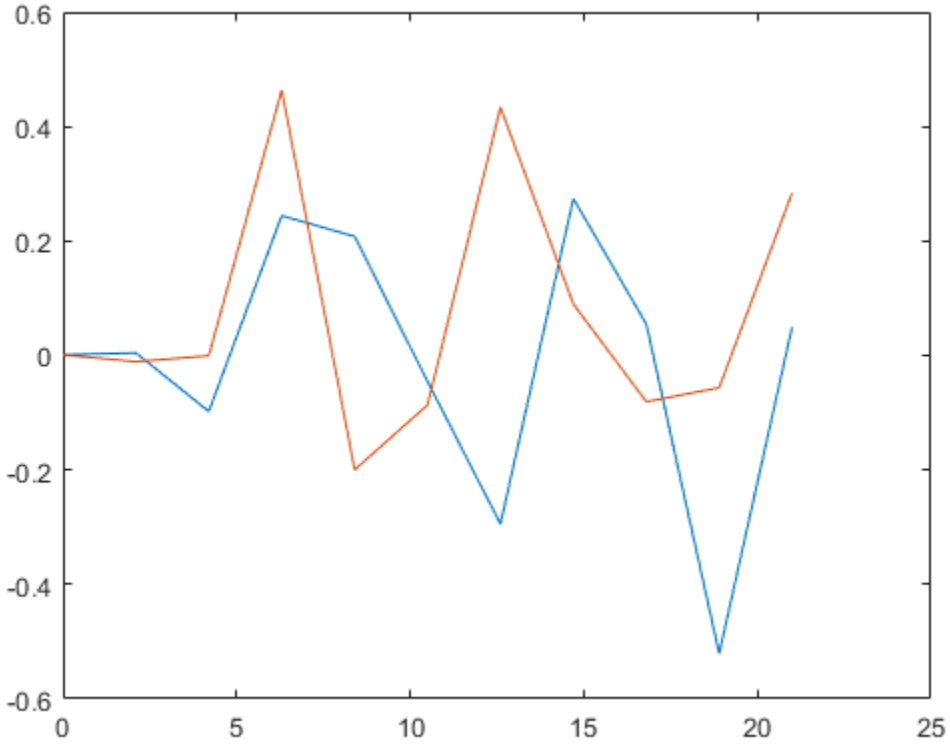


a szacowane błędy wyglądają tak:

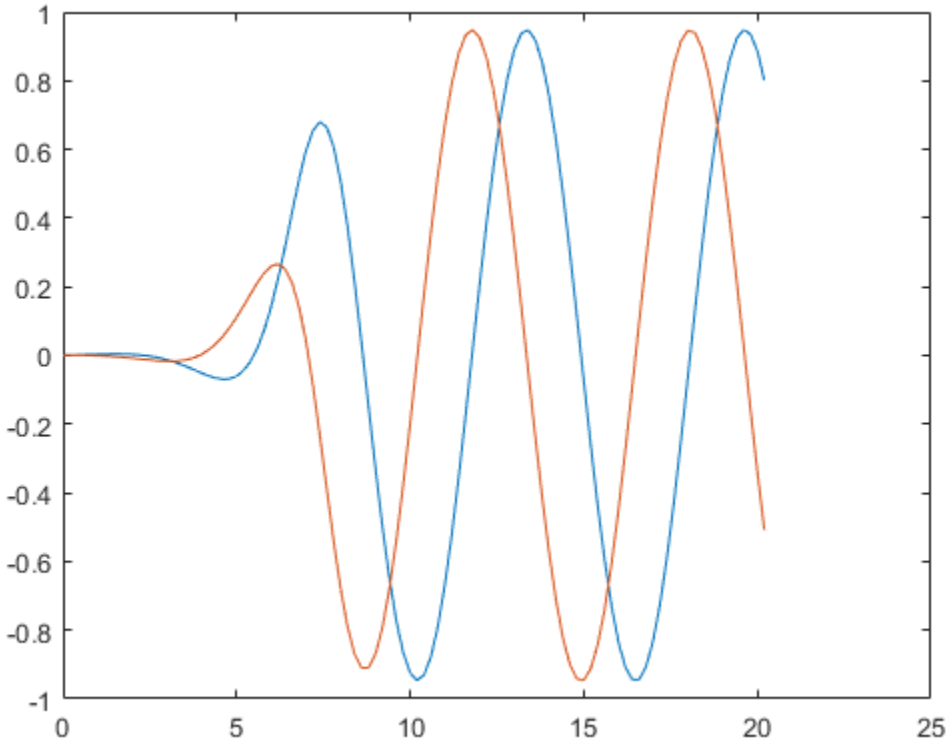


Powodem tego, że metoda nie działa przy tak dużym kroku jest to, że pochodne są brane z na tyle oddalonych o d oryginalnego punktach, że trudno powiedzieć żeby opisywały oryginalną funkcję w tym punkcie.

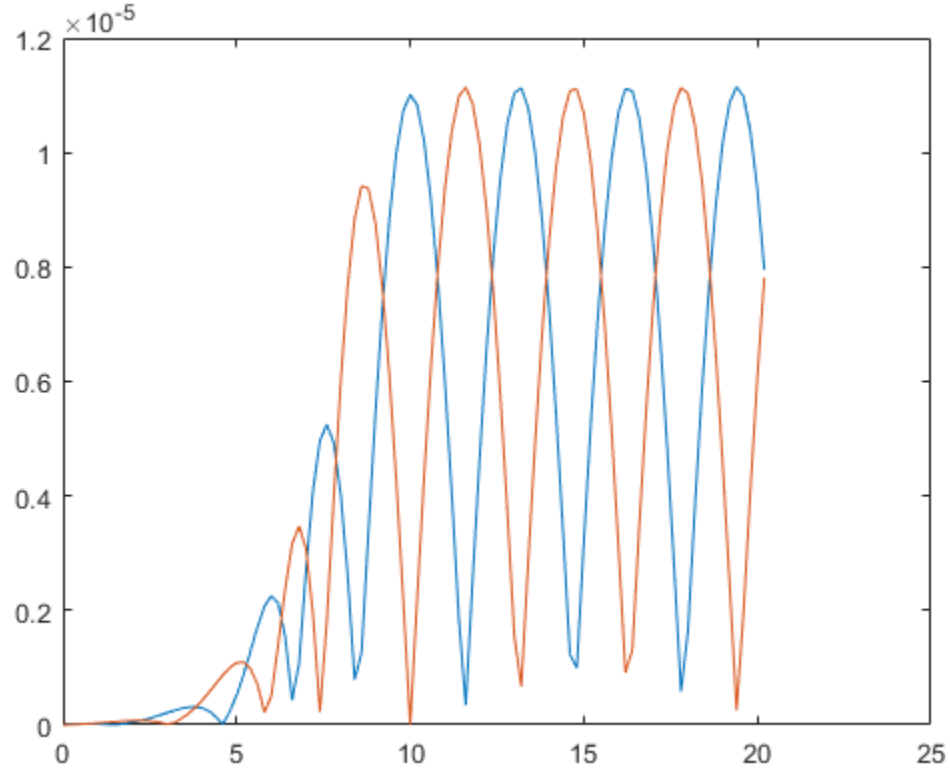
RK4 z krokiem 2.1 można uznać, że przypomina rozwiązanie z ode45 ale jest wybitnie nie gładkie:



RK4 z krokiem 0.2 jest pierwszym wyglądającym gładko:



Szacowane błędy wyglądają tak:



Metoda RK4 dla zadanego układu z punktu początkowego d zachowuje się bardzo

stabilnie ze zmianą kroku, prawdopodobnie ze względu na łagodniejsze zmiany funkcji niż w pozostałych punktach.

2) Metoda wielokorkowa predyktor-korektor Adamsa czwartego rzędu ze stałym krokiem jest połączeniem metod jawnych i niejawnych. Metody jawne do obliczenia następnej wartości używają tylko poprzednich wartości, a metody nie jawne używają poprzednich wartości, oraz wartości w punkcie bieżącym. Metody Adamsa opierają się na przekształcenie równań różniczkowych w równanie całkowe:

$$y\left(x_n\right) = y\left(x_{n-1}\right) + \int_{x_{n-1}}^{x_n} f\left(t, y\left(t\right)\right) dt.$$

Metody jawne (Adamsa – Bashfortha) wykorzystują wzór interpolacyjny Lagrange'a do przybliżania wielomianem

$$f\left(x, y\left(x\right)\right) \cong W\left(x\right) = \sum_{j=1}^{k} f\left(x_{n-j}, y_{n-j}\right) \cdot L_j\left(x\right),$$

$$y_n = y_{n-1} + \sum_{j=1}^{k} f\left(x_{n-j}, y_{n-j}\right) \cdot \int_{x_{n-1}}^{x_n} L_j\left(t\right) dt,$$

wielomiany Lagrange'a:

$$L_j\left(x\right) = \prod_{m=1, m\neq j}^{k} \frac{x - x_{n-m}}{x_{n-j} - x_{n-m}}.$$

Otrzymany wzór:

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

współczynniki β są stablicowane.

Metody niejawne (Adamsa – Moultona)
Wykorzystane jest przybliżenie równania całkowego wielomianem interpolacyjnym. Dany jest wzór:

$$\begin{aligned} y_n &= y_{n-1} + h\sum_{j=0}^{k} \beta_j^* \cdot f\left(x_{n-j}, y_{n-j}\right) = \\ &= y_{n-1} + h \cdot \beta_0^* \cdot f\left(x_n, y_n\right) + h\sum_{j=1}^{k} \beta_j^* \cdot f\left(x_{n-j}, y_{n-j}\right) \end{aligned}$$

tu parametry β też są stablicowane.
Metoda predyktor-korektor Adamsa:

$$\begin{aligned}
\text{P:} \quad & y_n^{[0]} = y_{n-1} + h \sum_{j=1}^{k} \beta_j f_{n-j}, \\
\text{E:} \quad & f_n^{[0]} = f(x_n, y_n^{[0]}), \\
\text{K:} \quad & y_n = y_{n-1} + h \sum_{j=1}^{k} \beta_j^* f_{n-j} + h\beta_0^* f_n^{[0]}, \\
\text{E:} \quad & f_n = f(x_n, y_n).
\end{aligned}$$

oszacowanie błędu:

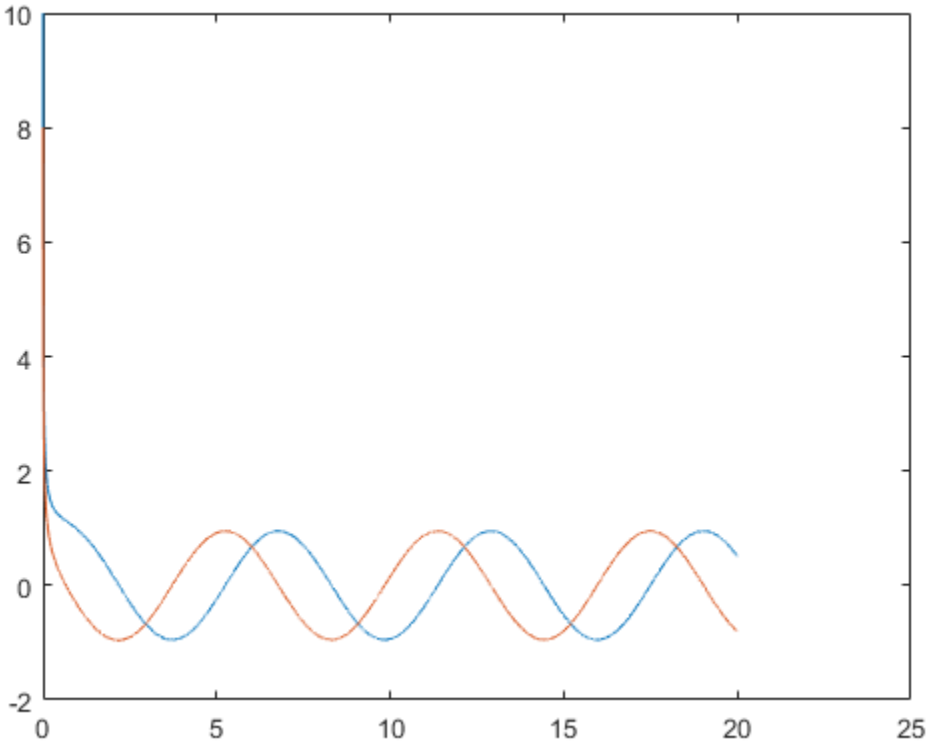
$$\delta_n(h_{n-1}) = -\frac{19}{270}(y_n{}^{[0]} - y_n)$$

Na początku metody PK znjaduję potrzebne pierwsze 4 wartości za pomocą RK4.

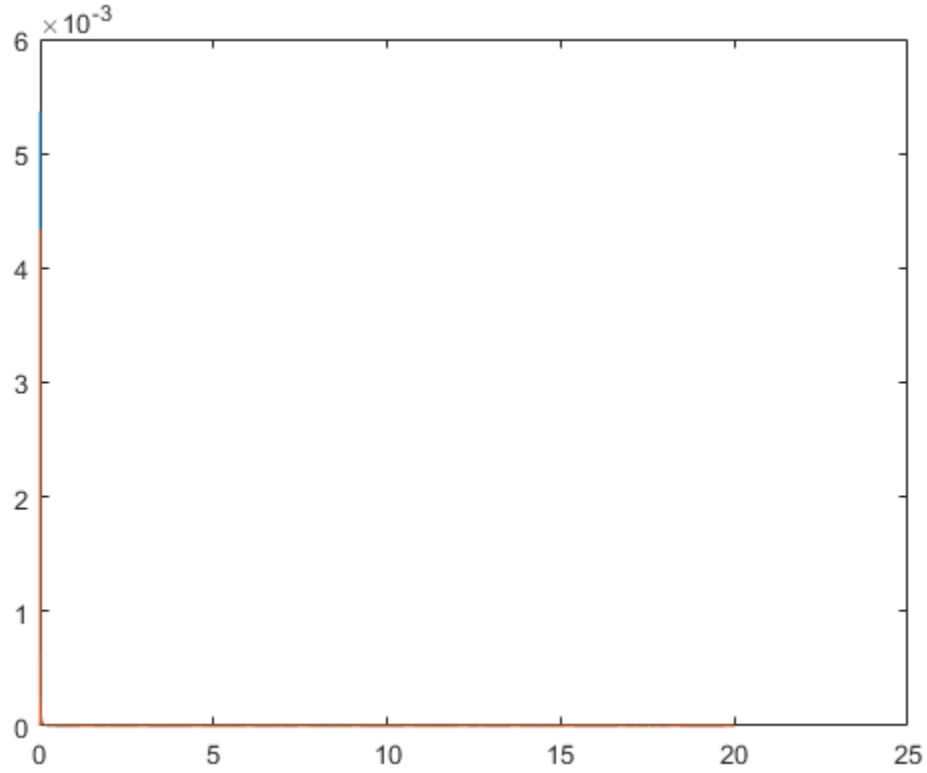
```
function [ x, T, errx, t ] = PKA( dx, x0, przedzial, krok )
%PKA Implementacja metody predyktor-korektor Adamsa ze sta³ym krokiem
%   dx - uk³ad równañ ró¿niczkowych pierwszego rzêdu
%   x0 - punkt pocz¹tkowy
%   przedzia³ - zadany przedzia³ (od zera)
%   x - rozwi¹zanie
%   T - interwa³y przedzia³u (potrzebne do wykresu itp)
%   errx - b³¹d rozwi¹zania
%   t - czas obliczeñ
tic;
yn=x0;%y0
h=krok;
h2=0.5*h;
beta=[55/24,-59/24,37/24,-9/24];
betag=[251/720,646/720,-264/720,106/720,-19/720];
x=zeros(length(przedzial(1):krok:przedzial(2)),2);
errx=zeros(length(przedzial(1):krok:przedzial(2)),2);
T=zeros(length(przedzial(1):krok:przedzial(2)),1);
x(1,:)=yn';
%obliczenie pierwszych 3 wartoœci (poza punktem pocz¹tkowym) metod¹ RK4
for i=2:4
    k1=dx(yn);
    k2=dx(yn+0.5*h*k1);
    k3=dx(yn+0.5*h*k2);
    k4=dx(yn+h*k3);
    ynp1=yn+1/6*h*(k1+2*k2+2*k3+k4);
    x(i,:)=ynp1';
    for i2=1:2
        k1=dx(yn);
        k2=dx(yn+0.5*h2*k1);
        k3=dx(yn+0.5*h2*k2);
        k4=dx(yn+h2*k3);
        yn=yn+1/6*h2*(k1+2*k2+2*k3+k4);
    end
    errx(i,:)=abs(16/15*(yn-ynp1));
    yn=ynp1;
    T(i)=i*h;
end
for i=5:length(przedzial(1):krok:przedzial(2))
    suma=[0, 0];
    for j=1:4
        suma=suma+beta(j)*dx(x(i-j,:));
```

```
    end
    xo=x(i-1,:)+h*suma;
    f=dx(xo);
    suma=[0, 0];
    for j=1:3
        suma=suma+betag(j+1)*dx(x(i-j,:));
    end
    x(i,:)=(x(i-1,:)+h*suma+h*betag(1)*f)';
    errx(i,:)=abs(19/270*(xo-x(i,:)));
    T(i)=i*h;
end
t=toc;
end
```

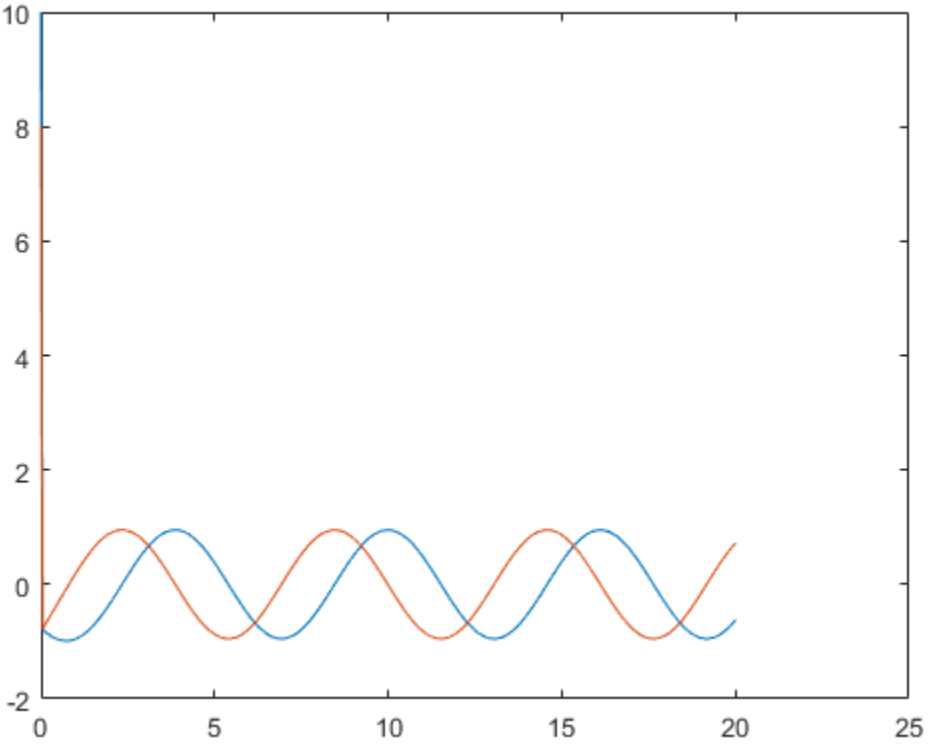
Rozwiązanie metodą PK dla punktu początkowego a, krok 0.001:



Oszacowanie błędów:

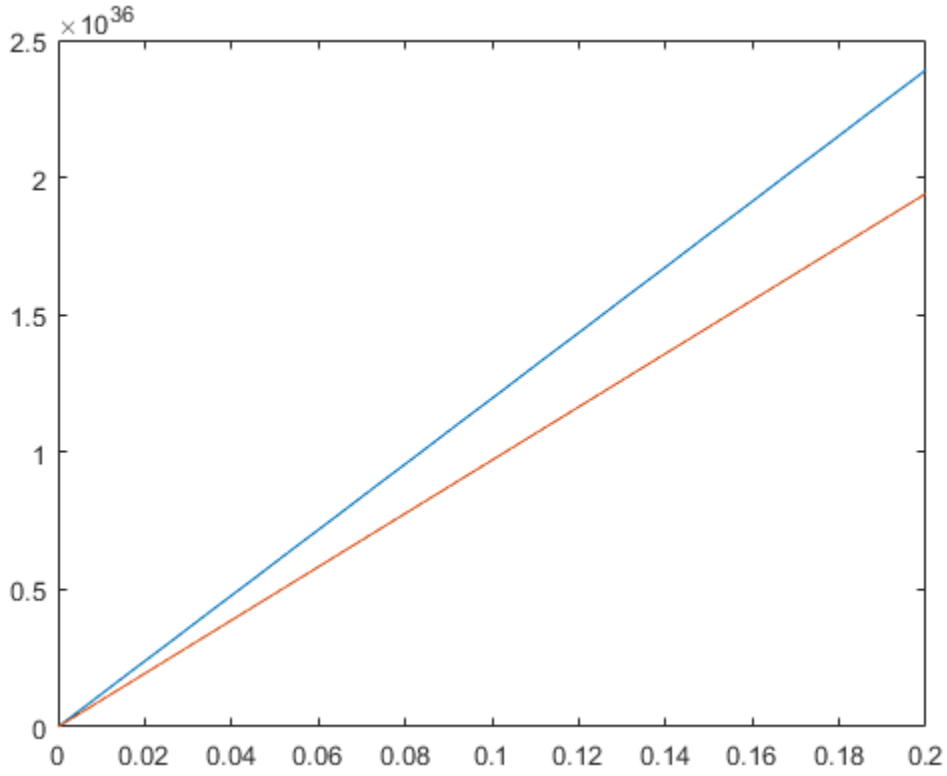


Przy kroku 0.01 zaczyna się psuć:

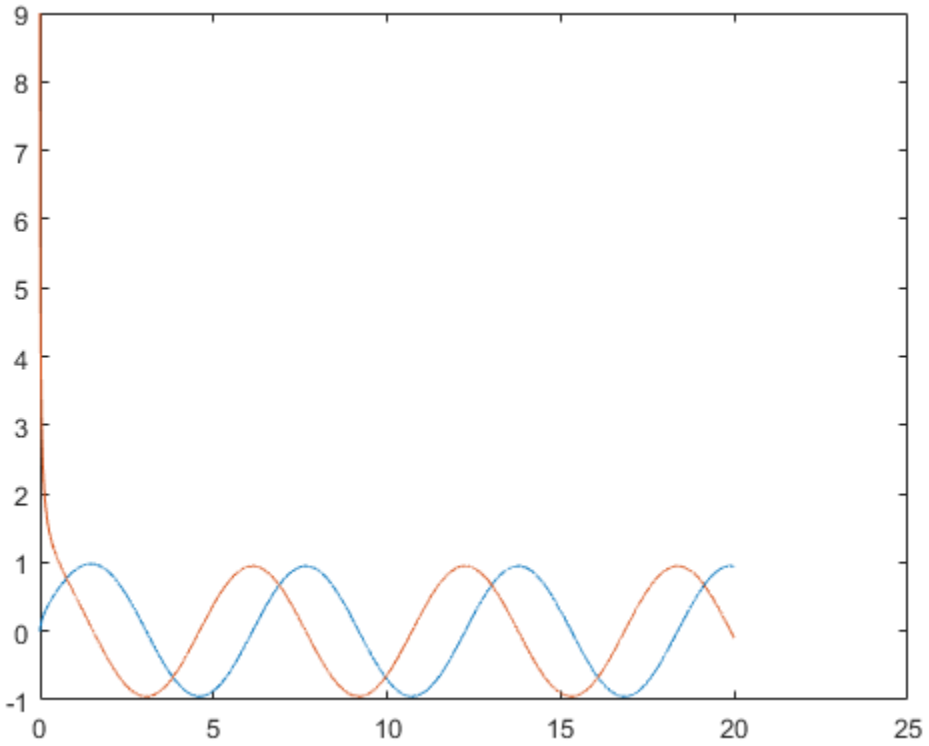


Widać nagłe przejście i brakujący fragment wykresu między dużą pochodną, a mniejszą. Z krokiem 0.1 metoda zupełnie zawodzi osiągając najpierw dużą wartość z powodu

dużej pochodnej, a potem NaN:

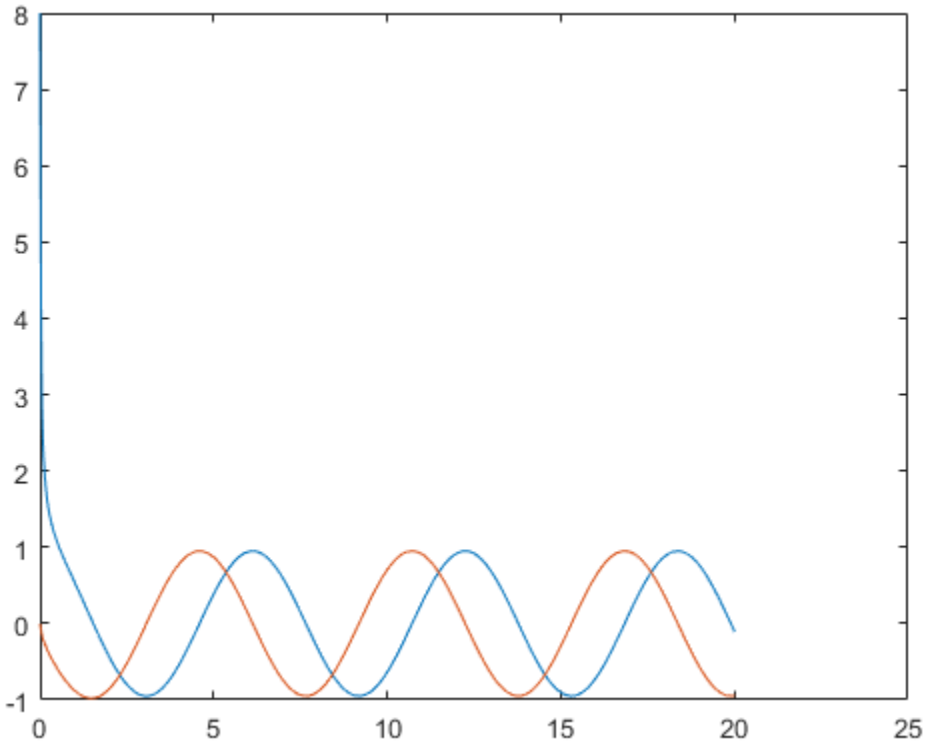


Punkt początkowy b z krokiem 0.01:



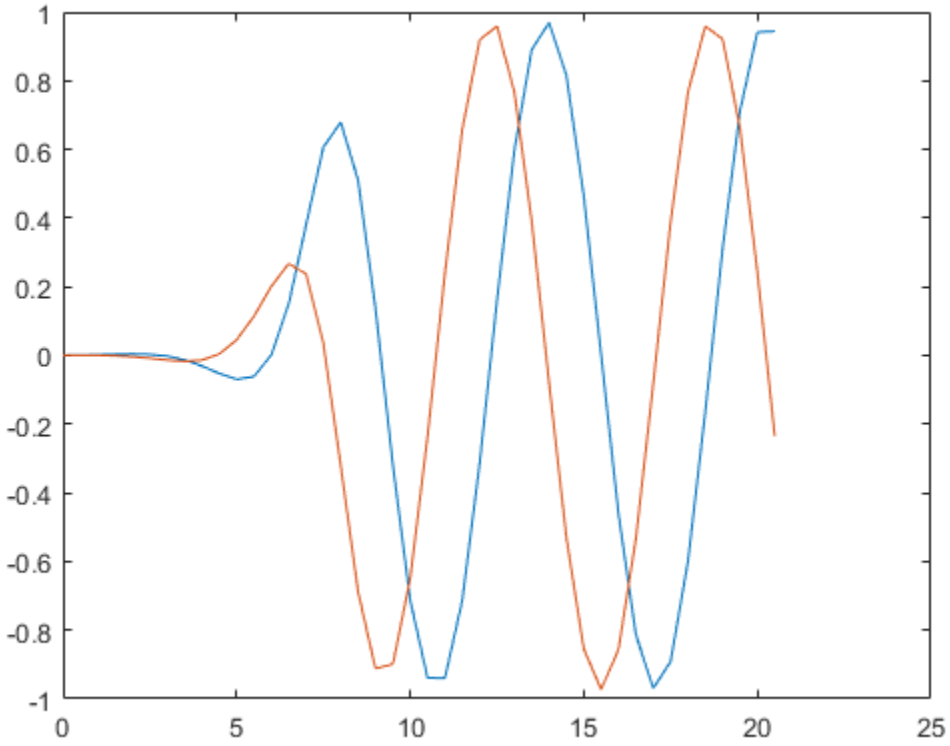
oszacowanie błędów jak w a.

Punkt początkowy c z krokiem 0.01:

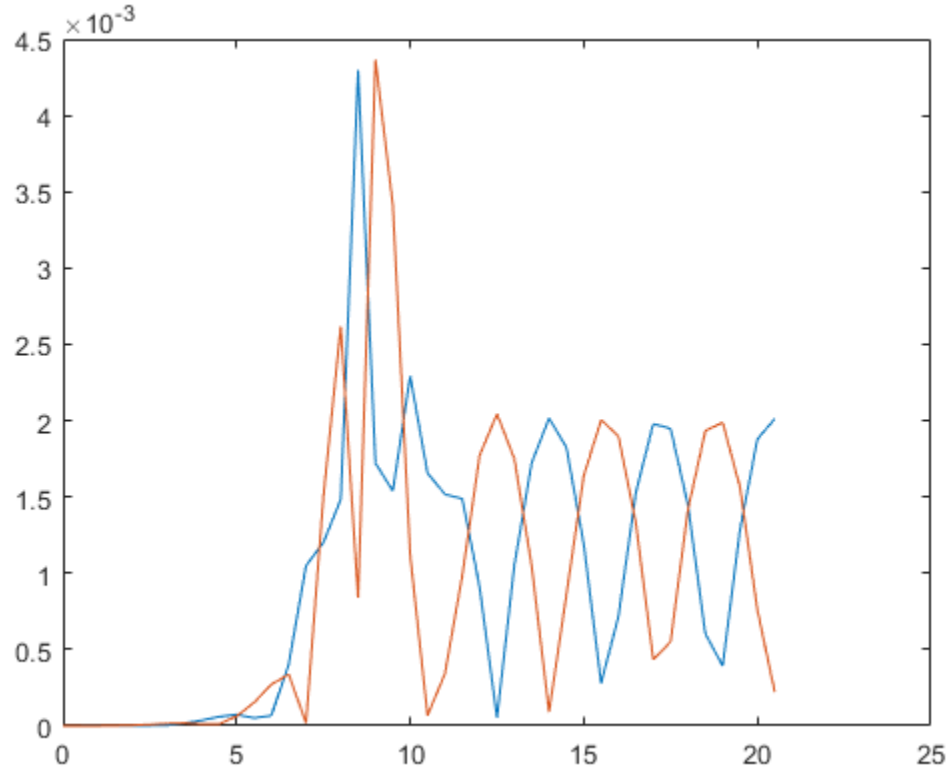


oszacowanie błędów jak w a i b wykresy te niewiele się różnią ze względu na podobieństwo punktów początkowych.

Punkt początkowy d z krokiem 0.5 na granicy gładkości:

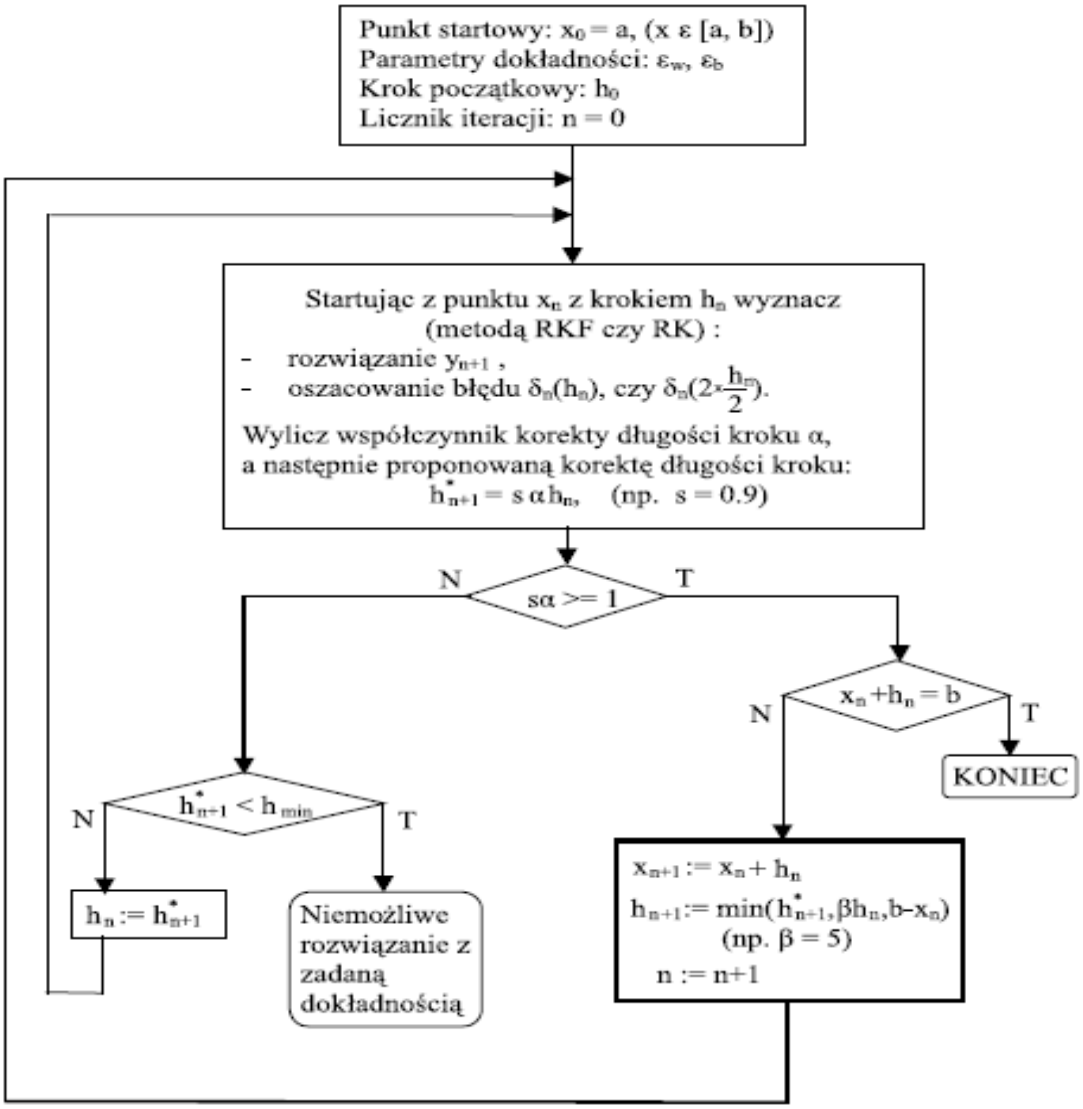


Oszacowanie błędu:



3) Metoda Rungego-Kutty ze zmiennym krokiem działa tak samo jak RK ze stałym

krokiem, tylko wykorzystywane jest oszacowanie błędu do korekty kroku, dodatkowo zadana jest oczekiwana dokładność.

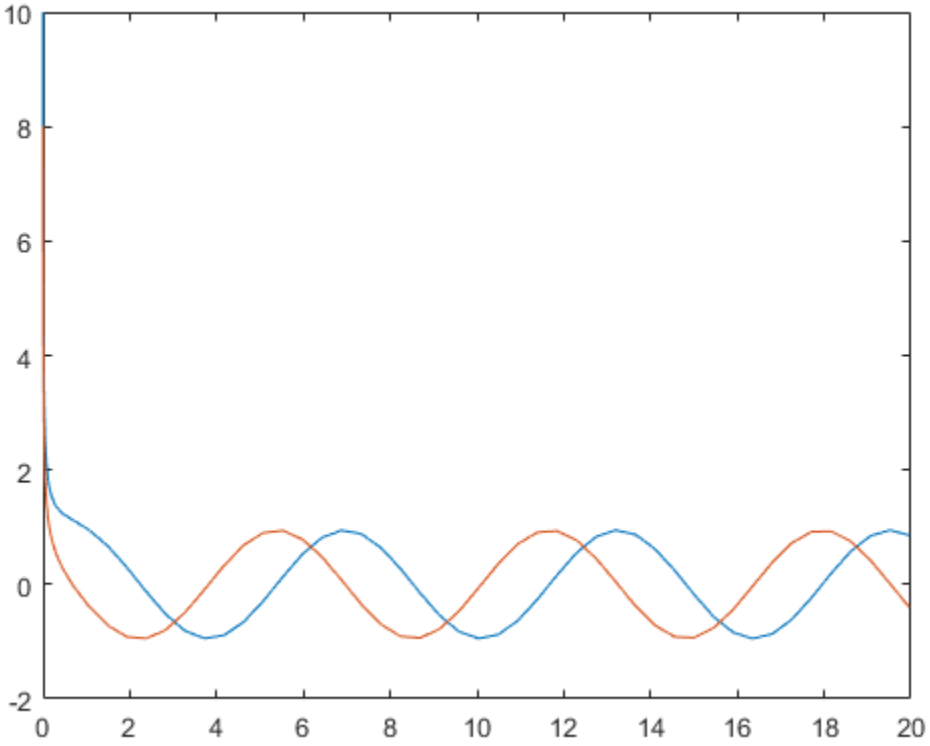



```
function [ x, T, errx, t ] = RK4_zmienny_krok( dx, x0, przedzial )
%RK4_zmienny_krok Metoda Rungego-Kutty czwartego rzêdu ze zmiennym krokiem
%   dx - uk³ad równañ ró¿niczkowych pierwszego rzêdu
%   x0 - punkt pocz¹tkowy
%   przedzia³ - zadany przedzia³ (od zera)
%   x - rozwi¹zanie
%   T - interwa³y przedzia³u (potrzebne do wykresu itp) tutaj wa¿niejsze
%   ni¿ wczeœniej ze wzglêdu na zmienny krok
%   errx - b³¹d rozwi¹zania
%   t - czas obliczeñ
tic;
yn=x0;%y0
krok=0.001;
alpha=1;%wspó³czynnik korekcji kroku
epsylon=1e-3;%dok³adnoœæ obliczeñ
epsylonWzg=1e-3;%dok³adnoœæ wzglêdna
epsylonBezWzg=1e-6;%dok³adnoœæ bezwzglêdna
```

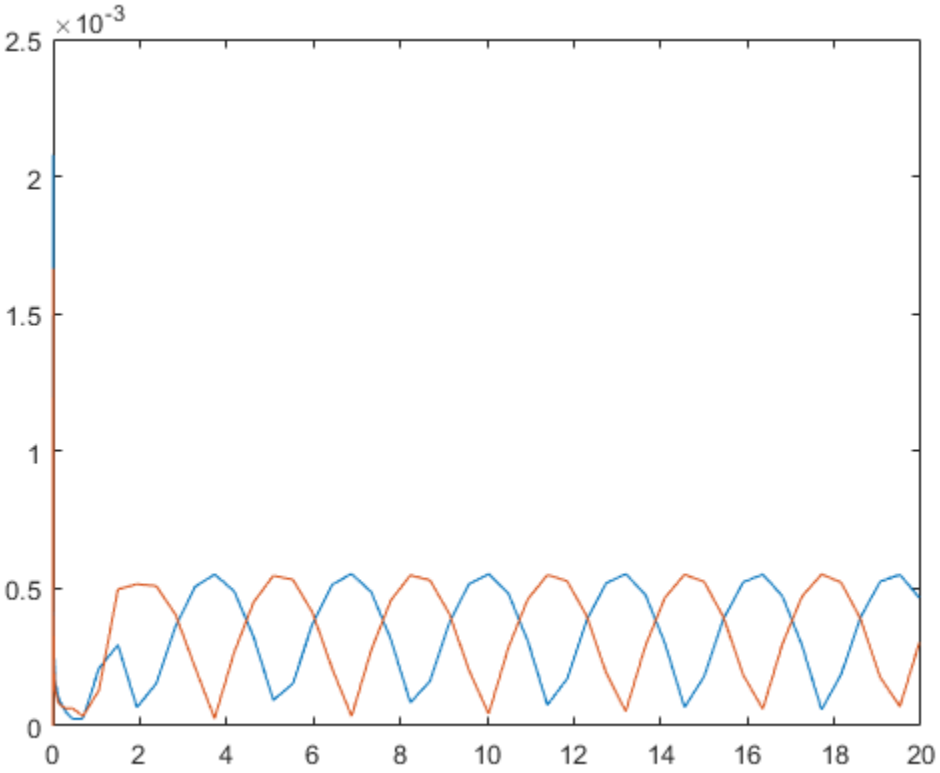
```
s=0.9;
beta=5;
maxiter=1e4;
h=krok;
h2=0.5*h;
j=1;
%x=zeros(length(przedzial(1):krok:przedzial(2)),2);
%errx=zeros(length(przedzial(1):krok:przedzial(2)),2);
%T=zeros(length(przedzial(1):krok:przedzial(2)),1);
x(j,:)=yn';
T(j)=0;
while T(j)+h<przedzial(2) && j<maxiter
    j=j+1;
    k1=dx(yn);%zmiana oznaczeñ w celu zgodnoœci z podrêcznikiem
    k2=dx(yn+0.5*h*k1);
    k3=dx(yn+0.5*h*k2);
    k4=dx(yn+h*k3);
    ynp1=yn+1/6*h*(k1+2*k2+2*k3+k4);
    x(j,:)=ynp1';
    h2=h/2;
    for i2=1:2%obliczenie yn+1 w dwóch o po³owê mniejszych krokach na potrzeby
oszacownaia b³êdu
        k1=dx(yn);
        k2=dx(yn+0.5*h2*k1);
        k3=dx(yn+0.5*h2*k2);
        k4=dx(yn+h2*k3);
        yn=yn+1/6*h2*(k1+2*k2+2*k3+k4);
    end
    errx(j,:)=abs(16/15*(yn-ynp1));
    yn=ynp1;
    T(j)=T(j-1)+h;
    epsylon=abs(yn)*epsylonWzg+epsylonBezWzg;
    alpha=min(epsylon/errx(j,:))^(1/5);
    hs=s*alpha*h;
    if s*alpha>=1
        if T(j)+h<przedzial(2)
            h=min([hs, beta*h, przedzial(2)-T(j)]);
        end
    elseif hs<1e-9
        disp('nie mozliwe rozwiazanie z zadana dokaladnoscia');
    else
        h=hs;
    end


end
t=toc;
end
```

Rozwiązanie dla punktu startowego a:

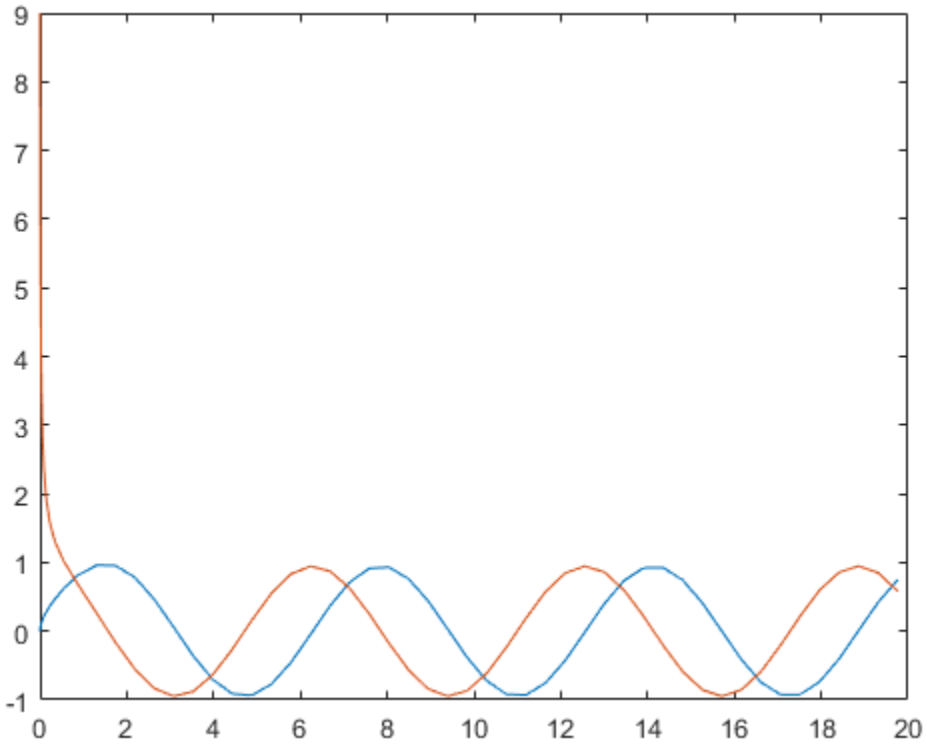


Szacowane błędy dla kroków:



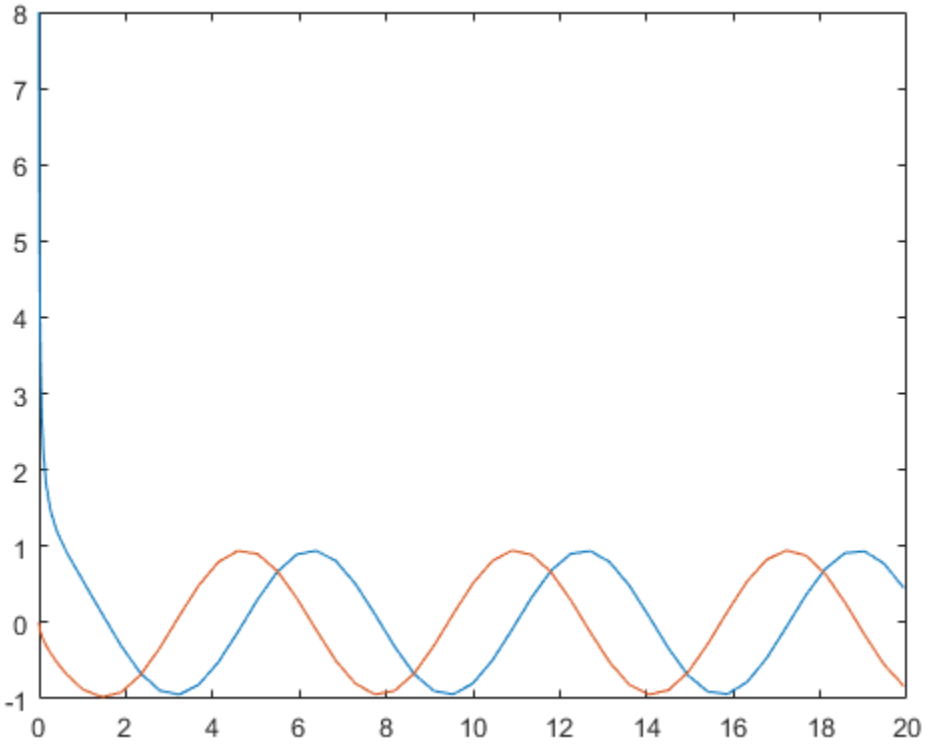
Widać, że błędy utrzymywały się w pobliżu zadanych dokładności. Rozwiązanie zajęło 0.0134 sekundy i 57 iteracji.

Punkt początkowy b:

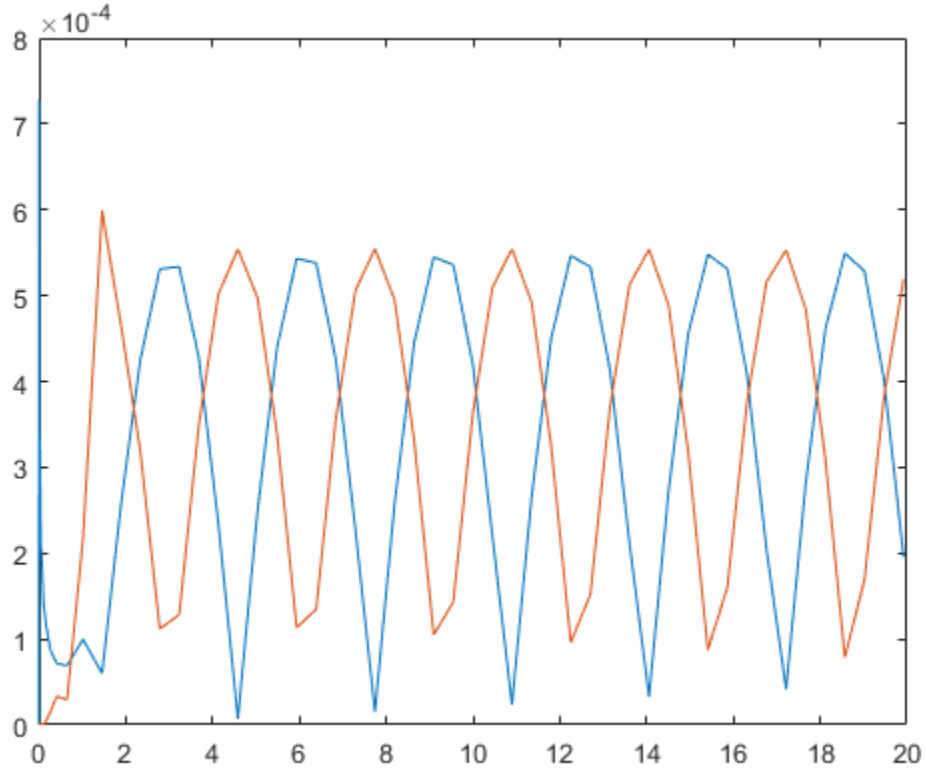


błędy wyglądy tak samo jak w a. 0.0028s, 55 iteracji.

Punkt początkowy c:

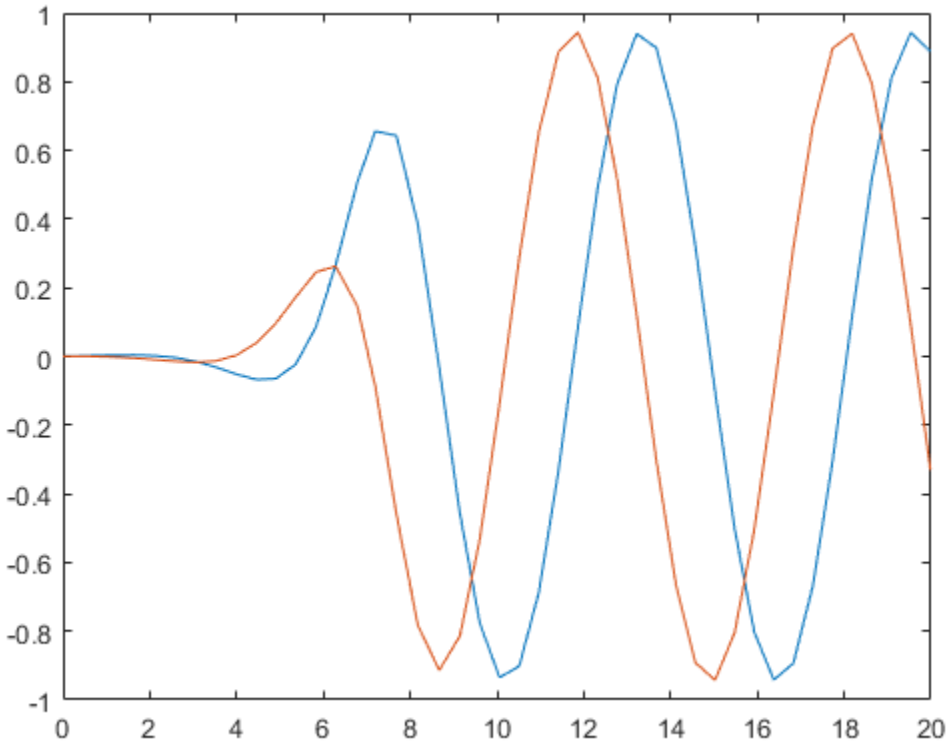


szacowane błędy:

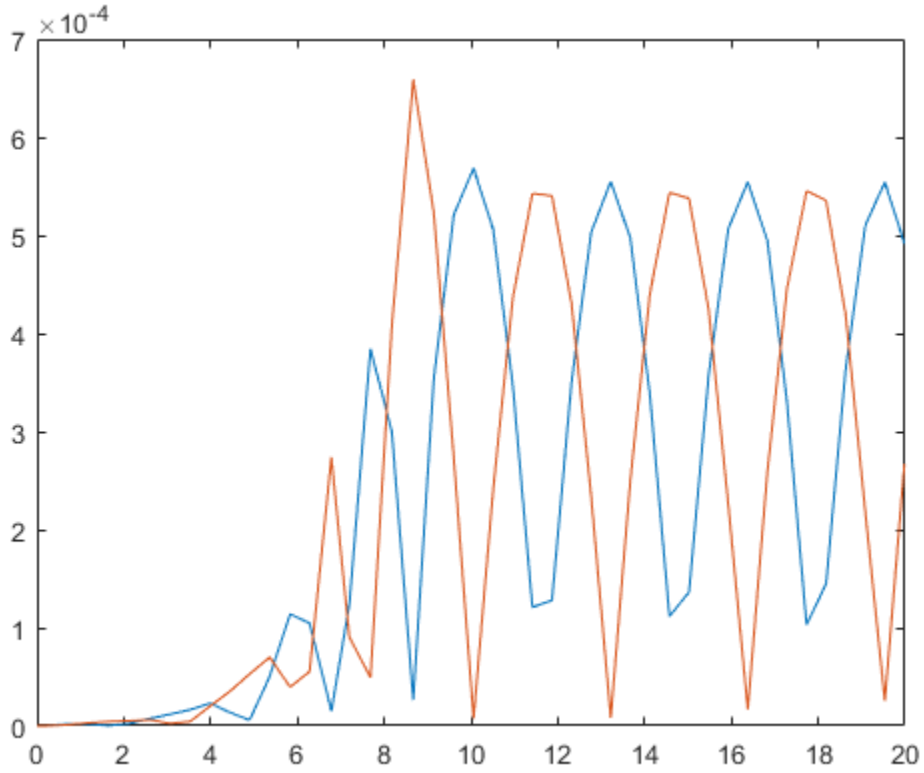


czas 0.0126s, 55 iteracji.

Punkt początkowy d:



Widać, że wyszło trochę nie gładko, jeśli by komuś na wyglądzie zależało, to można by trochę dostroić parametry.
Szacowane błędy:



czas 0.0024s, 48 iteracji.

Dla metody RK4 ze stałym krokiem czas 0.0425s i iteracji stosownie dużo (setki, tysiące) do kroku.
Dla metody PK 0.0443s porównywalna, ale raczej duża liczba kroków.
Metoda RK4 ze zmiennym korkiem stanowczo najszybsza i potrzebuje znacznie mniej kroków.
Generalnie metody ze zmiennym krokiem są lepsze.