Prof. Dr. Stefan Funken
Dipl.-Math. Katharina Becker-Steinberger
Dipl.-Math. oec. Klaus Stolle
Institut für Numerische Mathematik
Universität Ulm

WS 2013/2014
Blatt 3
08.11.2013

Angewandte Numerik 2

Abgabetermin: Freitag 15.11.2013, **vor der Übung**

Aufgabe 7 *(Programmieraufgabe, modifiziertes Euler-Verfahren, Runge-Kutta-Verfahren)* (10 Punkte)

Lösen Sie die Anfangswertaufgabe aus Aufgabe 5

$$y' = -200ty^2, \quad y(-1) = \frac{1}{101}$$

mit den folgenden Einschrittverfahren mit konstanter Schrittweite h. Setze $n = 1/h$:

$$\begin{aligned} t_0 &= -1, \quad y_0 = \frac{1}{101}, \\ t_{j+1} &= t_j + h, \quad y_{j+1} = y_j + h\phi(t_j, y_j, h), \quad j = 0, \ldots, n-1 \end{aligned}$$

a) modifiziertes Euler-Verfahren: $\phi(t, y, h) = f(t + h/2, y + h/2 f(t, y))$

b) Runge-Kutta Verfahren $\phi(t, y, h) = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$ mit

$$\begin{aligned} k_1 &:= f(t, y), & k_2 &:= f(t + h/2, y + hk_1/2), \\ k_3 &:= f(t + h/2, y + hk_2/2), & k_4 &:= f(t + h, y + hk_3). \end{aligned}$$

Erweitern Sie Ihre Grafik aus Aufgabe 5 um die entsprechenden Daten der beiden Verfahren.

Lösung:

```
function y = modeuler (f, t, y, h)
% Verfahrensfunktion Modifiziertes Euler-Verfahren
y = feval (f, t+h/2, y+h/2*feval(f, t, y));
```

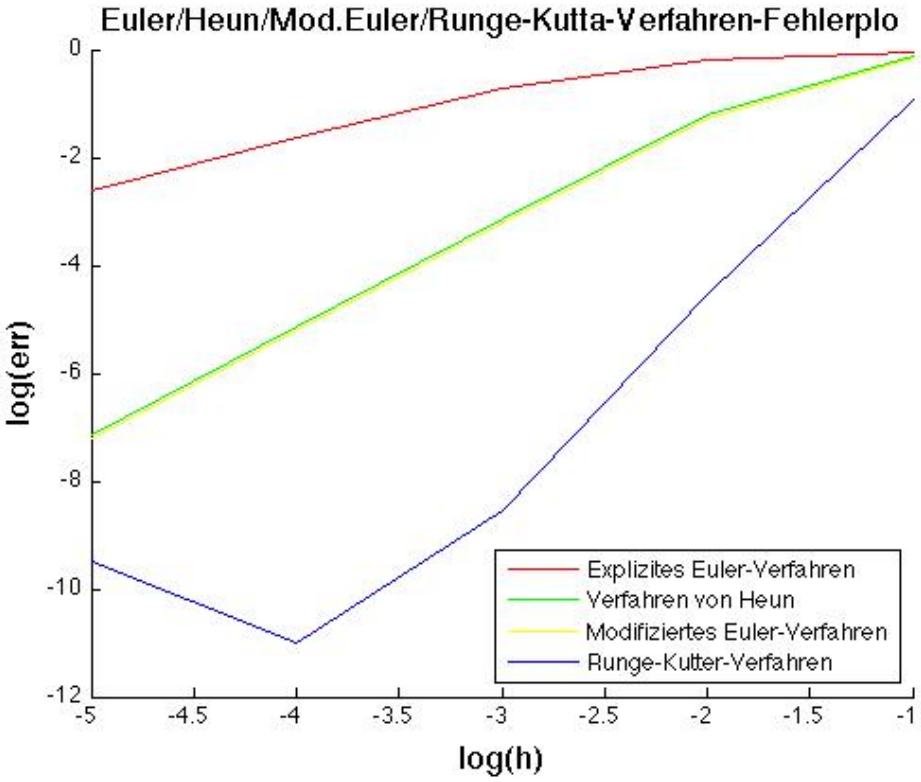
```
function y = rungekutta(f, t, y, h)
% Verfahrensfunktion Runge-Kutta-Verfahren
th2 = t + h/2;
k1 = feval (f, t, y);
k2 = feval (f, th2, y+h*k1/2);
k3 = feval (f, th2, y+h*k2/2);
k4 = feval (f, t+h, y+h*k3);
y = 1/6*(k1+2*k2+2*k3+k4);
```

```
% Skript zum Erstellen der Grafik

clear all;
close all;

t_0 = -1;
y_0 = 1/101;
```

```
8
9  N1 = logspace(1,5,5);
10
11 for k = 1:length(N1)
12     y_euler = einschritt (@euler, @fun, N1(k), t_0, y_0);
13     y_heun = einschritt (@heun , @fun, N1(k), t_0, y_0);
14     y_modeuler = einschritt (@modeuler, @fun, N1(k), t_0, y_0);
15     y_rungekutta = einschritt (@rungekutta, @fun, N1(k), t_0, y_0);
16     err_euler(k) = abs( y_euler(end) -1);
17     err_heun (k) = abs( y_heun(end) -1);
18     err_modeuler (k) = abs( y_modeuler(end) -1);
19     err_rk (k) = abs( y_rungekutta(end) -1);
20 end
21
22 hold on
23 loglog(1./N1, err_euler(:),'r', 1./N1, err_heun(:),'g', ...
24     1./N1, err_modeuler(:),'y', 1./N1,err_rk(:),'b');
25
26 title('Euler/Heun/Mod.Euler/Runge-Kutta-Verfahren-Fehlerplot','FontSize', 14)
27 xlabel('log(h)', 'FontSize', 14)
28 ylabel('log(err)', 'FontSize', 14)
29 legend ('Explizites Euler-Verfahren', 'Verfahren von Heun', ...
30 'Modifiziertes Euler-Verfahren', 'Runge-Kutter-Verfahren','Location', 'Best')
31 hold off;
```



Aufgabe 8 *(Programmieraufgabe, Runge-Kutta-Verfahren mit Schrittweitensteuerung)* (12 Punkte)

Betrachten Sie das restringierte Drei-Körper-Problem in Beispiel 37 aus dem Vorlesungsskript. Um die periodischen Bewegungen eines Satelliten im Kraftfeld von Erde und Mond zu berechnen, wird dabei folgendes Szenario angenommen:

Erde und Mond bewegen sich (fast) auf Kreisbahnen um ihren gemeinsamen Schwerpunkt. Ihr Abstand ändert sich dabei nicht und sei hier auf 1 normiert. Weiter sei $\mu = 1/82.45$ die relative Mondmasse und $(1 - \mu)$ die

relative Erdmasse. Die Masse des Satelliten sei im Verhältnis zur Erde- und Mondmasse so klein, dass sie die Bewegung dieser beiden Körper nicht beeinflusst. Ausserdem verlaufe die Bewegung der drei Körper in einer Ebene. Daher lässt sich die Bewegung in dieser Ebene in einem mitrotierenden (u, v)-Koordinatensystem mit Zentrum im Schwerpunkt von Erde und Mond beschreiben. Bei geeigneter Längenskalierung befindet sich dann die Erde im Punkt $(-\mu, 0)$ und der Mond im Punkt $(1-\mu, 0)$.

Die Bewegung des Satelliten in diesem Koordinatensystem wird durch das folgende ODE System 2. Ordnung beschrieben:

$$u'' = 2v' - \frac{\partial}{\partial u}V,$$
$$v'' = -2u' - \frac{\partial}{\partial v}V,$$

mit der Potentialfunktion

$$V(u,v) = -\frac{1}{2}(u^2+v^2) - \frac{1-\mu}{\sqrt{(u+\mu)^2+v^2}} - \frac{\mu}{\sqrt{(u-1+\mu)^2+v^2}}$$

Auf der Homepage finden Sie die Matlab Funktion `adaptiv_np` aus dem Vorlesungsskript. Schreiben Sie diese zu einer Funktion `adaptiv_rkf34` um, die zur numerischen Lösung von Anfangswertaufgaben das eingebettete Runge-Kutta-Fehlberg Verfahren 3. und 4. Ordnung verwendet, welches durch das folgende Butscher-Schema gegeben ist:

0					
$\frac{1}{4}$	$\frac{1}{4}$				
$\frac{4}{9}$	$\frac{4}{81}$	$\frac{32}{81}$			
$\frac{6}{7}$	$\frac{57}{98}$	$-\frac{432}{343}$	$\frac{1053}{686}$		
1	$\frac{1}{6}$	0	$\frac{27}{52}$	$\frac{49}{156}$	
$y^{(3)}_{k+1}$	$\frac{1}{6}$	0	$\frac{27}{52}$	$\frac{49}{156}$	0
$y^{(4)}_{k+1}$	$\frac{43}{288}$	0	$\frac{243}{416}$	$\frac{343}{1872}$	$\frac{1}{12}$
$d_{k+1} \approx$	$-\frac{5}{288}$	0	$\frac{27}{416}$	$-\frac{245}{1872}$	$\frac{1}{12}$

Lösen Sie das restringierte Drei-Körper-Problem mit dieser Funktion. Testen Sie ihr Programm mit den folgenden zwei Funktionsaufrufen:

```
adaptiv_rkf34(6.1,0,[1.2;0;0;-1.049357509830350],0.1,0.001)
adaptiv_rkf34(17.1,0,[0.994;0;0;-2.0015851],0.1,0.001) .
```

Beachten Sie die unterschiedlichen Masse ($\mu_1 = 1/82.45$ und $\mu_2 = 0.012277471$). Vergleichen Sie das Ergebnis mit dem des Runge-Kutta-Verhahrens 4. Ordnung mit äquidistanten Schrittweiten. Schreiben Sie hierzu eine Funktion `rk4`. **Hinweis zur Implementierung:** Speichern Sie die Parameter aus dem Butcher-Schema in einer Verfahrensmatrix $(b_{i\ell})$ und einem Knotenvektor a.

Lösung:

Lösungsvariante 1:

```
1 function rk4(tn,t0,y0,h,mu)
2
3 % Die Funktion rkf4param loest ein parameterabhaeniges AWP
4 % basierend auf dem 4 stufigen Runge-Kutta-Verfahren von
5 % mit einer adaptiven Schrittweitensteuerung
```

```
6  %
7  % Input:   tn    Endzeit
8  %          t0    Startzeit
9  %          y0    Anfangswerte
10 %          h     Startschrittweite
11 %          mu    problemabaengiger Parameter
12 %
13 % Output: graphische Ausgabe der Loesung
14 %
15 % Beispiele fuer Funktionsaufrufe:
16 %
17 % adaptiv_np(6.1,0,[1.2;0;0;-1.049357509830350],0.1,0.001)
18 % -> liefert periodische Loesung
19 %
20 % adaptiv_np(17.1,0,[0.994;0;0;-2.0015851],0.1,0.001)
21 % -> liefert chaotische Loesung
22
23 % Initialisierung
24 y(:,1)=y0(:);
25 t = t0;
26 T = t;
27 H = [];
28 ke =0;
29
30 % ------------------------------------------------------------------------
31 % Verfahrensmatrix
32
33 b  =  [0          0          0          0       0;
34        1/4        0          0          0       0;
35        4/81       32/81      0          0       0;
36        57/98      -432/343   1053/686   0       0;
37        1/6        0          27/52      49/156  0];
38 % Knotenvektor
39 a  =  [0 1/4 4/9 6/7 1];
40
41 % Gewichtsvektor
42 c = [43/288,0,243/416,343/1872,1/12];
43 % ------------------------------------------------------------------------
44
45 % Integrationsschleife
46 while and(t<tn, ke < 1000)
47     flag = 1;
48
49     while flag
50
51         k1 = f(t,y(:,end), mu);
52         w2 =  y(:,end) + h*(b(2,1)*k1);
53         k2 =  f(t+a(2)*h, w2, mu);
54         w3 =  y(:,end) + h*(b(3,1)*k1+b(3,2)*k2);
55         k3 =  f(t+a(3)*h,w3, mu);
56         w4 =  y(:,end) + h*(b(4,1)*k1+b(4,2)*k2+b(4,3)*k3);
57         k4 =  f(t+a(4)*h, w4, mu);
58         w5 =  y(:,end) + h*(b(5,1)*k1+b(5,2)*k2+b(5,3)*k3+b(5,4)*k4);
59         k5 =  f(t+a(5)*h, w5, mu);
```

```
60
61          % Berechnung der Verfahrensfunktion
62          phi = c(1)*k1+c(2)*k2+c(3)*k3+c(4)*k4+c(5)*k5;
63
64
65          %-------------------------------------------------------------------
66      end
67
68      y = [y,y(:,end)+h*phi];
69      t=t+h;
70      T=[T,t];
71      H=[H,h];
72  end
73
74  % Grafische Ausgabe
75  subplot(2,1,1)
76  semilogy([T(1:end-1);T(2:end)],[H;H],'g-');
77  title('Schrittweite');
78  print -depsc2 adaptiv_vp_fig01
79
80  subplot(2,1,2)
81  plot(y(1,:),y(3,:),'r-');
82  title('Phasendiagramm');
83  print -depsc2 adaptiv_vp_fig02
84
85  end
```

```
1   function adaptiv_rk34(tn,t0,y0,h,tol)
2
3   % Die Funktion adaptiv_np loest ein AWP basierend auf dem Verfahren von
4   % Kutta und der verbesserten Polygonzugmethode mit einer adaptiven
5   % Schrittweitensteuerung
6   %
7   % Input:   tn    Endzeit
8   %          t0    Startzeit
9   %          y0    Anfangswerte
10  %          h     Startschrittweite
11  %          tol   Toleranz zu Schrittweitenbestimmung
12  %
13  % Output: graphische Ausgabe der Loesung
14  %
15  % Beispiele fuer Funktionsaufrufe:
16  %
17  % adaptiv_np(6.1,0,[1.2;0;0;-1.049357509830350],0.1,0.001)
18  % -> liefert periodische Loesung
19  %
20  % adaptiv_np(17.1,0,[0.994;0;0;-2.0015851],0.1,0.001)
21  % -> liefert chaotische Loesung
22
23  % Initialisierung
24  y(:,1)=y0(:);
25  t=t0;
26  T=t;
27  H=[];
```

```
%w = y(:,1);
ke =0;

% ------------------------------------------------------------------------
% Verfahrensmatrix

b  =  [0            0           0           0       0;
       1/4          0           0           0       0;
       4/81         32/81       0           0       0;
       57/98        -432/343    1053/686    0       0;
       1/6          0           27/52       49/156  0;
       1/6          0           27/52       49/156  1/12];

% Knotenvektor
a  =  [0 1/4 4/9 6/7 1];

% Gewicht fuer Fehlerschaetzer
te = [-5/288 0 27/416 -245/1872 1/12];
% Gewichtsvektoren f¸r rk 3. und 4. Ordnung
gamma = [1/6;0;27/52;49/156;0];
delta = [43/288;0;243/416;343/1872;1/12];
% ------------------------------------------------------------------------

% Integrationsschleife
while and(t<tn, ke < 1000)
    flag = 1;

    while flag

        k1 = f(t,y(:,end));
        w2 =  y(:,end) + h*(b(2,1)*k1);
        k2 =  f(t+a(2)*h,w2);
        w3 =  y(:,end) + h*(b(3,1)*k1+b(3,2)*k2);
        k3 =  f(t+a(3)*h,w3);
        w4 =  y(:,end) + h*(b(4,1)*k1+b(4,2)*k2+b(4,3)*k3);
        k4 =  f(t+a(4)*h,w4);
        w5 =  y(:,end) + h*(b(5,1)*k1+b(5,2)*k2+b(5,3)*k3+b(5,4)*k4);
        k5 =  f(t+a(5)*h,w5);
        w6 =  y(:, end) + h*(b(6,1)*k1+b(6,2)*k2+b(6,3)*k3+b(6,4)*k4+b(6,5)*k5);
        k6 =  f(t+h,w6);

        % Berechnung der Verfahrensfunktion
        phi=gamma(1)*k1+gamma(2)*k2+gamma(3)*k3+gamma(4)*k4+gamma(5)*k5;

        % Berechnung des Fehlersch%otzers
        dk = norm(h*(te(1)*k1+te(3)*k3+te(4)*k4+te(5)*k5));

        %--Schrittweitensteuerung ------------------------------------------
        if dk > 1.2 * tol
            % nicht erfolgreicher Schritt:
            ke = ke+1;
            h = 0.9 * h;
        elseif dk < 0.8 *tol
            % nicht erfolgreicher Schritt:
```

```
82              ke = ke+1;
83              h = 1.1 * h;
84          else
85              % erfolgreicher Schritt:
86              ke = ke+1;
87              flag = 0;
88
89
90          end
91          %-------------------------------------------------------------------
92      end
93
94      y = [y,y(:,end)+h*phi];
95      t=t+h;
96      T=[T,t];
97      H=[H,h];
98  end
99
100 % Grafische Ausgabe
101 figure(1)
102 semilogy([T(1:end-1);T(2:end)],[H;H],'g-');
103 title('Schrittweite');
104 print -depsc2 adaptiv_vp_fig01
105
106 figure(2)
107 plot(y(1,:),y(3,:),'r-');
108 title('Phasendiagramm');
109 print -depsc2 adaptiv_vp_fig02
110
111 end
112
113 %------ problemabhaengige Funktion f ---------------------------------------
114
115 function wert = f(t,y)
116 % Die Funktion f beschreibt das ODE System zu 3-Koerper Problem
117 %
118 % Input:  t
119 %         y
120 %
121 % Output: wert
122
123 %mu = 1/82.45; % relative Masse fuer Beispiel 1 hard codiert
124 mu = 0.012277471 % relative Masse fuer Beispiel 2
125 z1 = ((y(1)+mu)^2+y(3)^2)^(3/2);
126 z2 = ((y(1)-1+mu)^2+y(3)^2)^(3/2);
127 wert = [y(2);
128         y(1)+2*y(4)-(1-mu)*(y(1)+mu)/z1-mu*(y(1)-1+mu)/z2;
129         y(4);
130         y(3)-2*y(2)-(1-mu)*y(3)/z1-mu*y(3)/z2];
131 end
```

```
close all
clear all


%adaptiv_rk34(6.1,0,[1.2;0;0;-1.0493575098830350],0.1,0.001)


adaptiv_rk34(17.1,0,[0.994;0;0;-2.0015851],0.1,0.001)
```

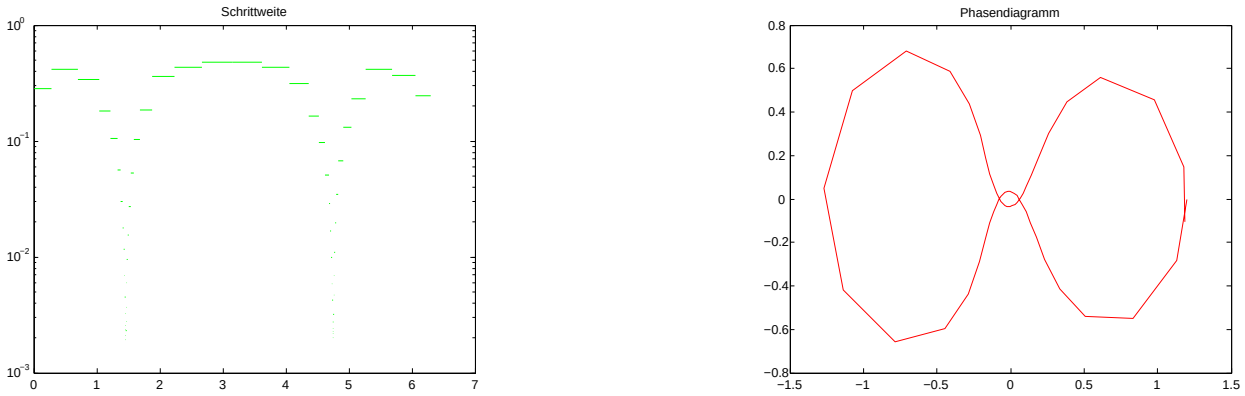


Abbildung 1: Beispiel 1($\mu = \frac{1}{82.45}$)

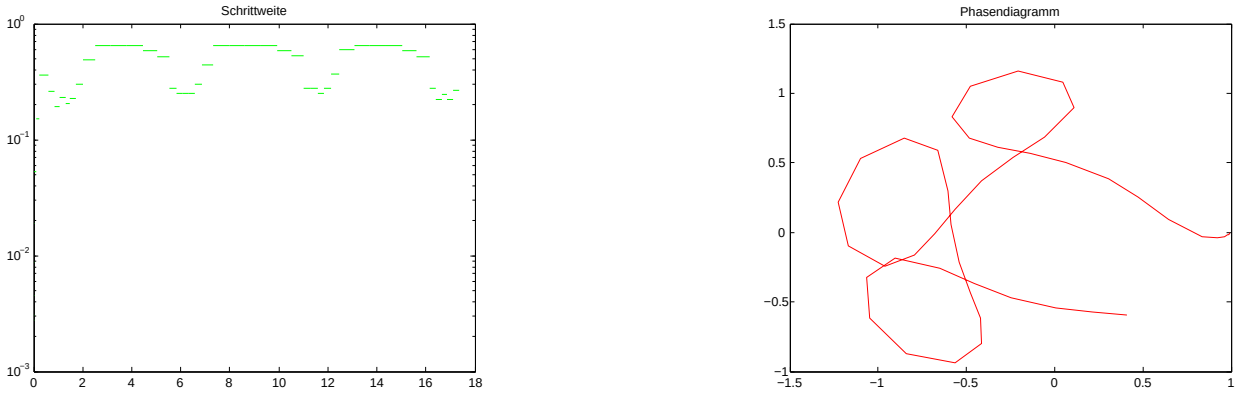


Abbildung 2: Beispiel 2 ($\mu = 0.012277471$)

Lösungsvariante 2 mit parameterabhängigen Funktionen:

```
function [H,T, y] = rk4param(tn,t0,y0,h,mu)

% Die Funktion rkf4 loest ein parameterabhaeniges AWP
% basierend auf dem 4 stufigen Runge-Kutta-Verfahren von
% mit einer adaptiven Schrittweitensteuerung
%
% Input:  tn    Endzeit
%         t0    Startzeit
%         y0    Anfangswerte
%         h     Startschrittweite
%         mu    problemabaengiger Parameter
%
% Output: H     Schrittweitenvektor
%         T     Zeitschrittvektor
%         Y     Vektor der Loesungen zu dem Zeischrittenvektor T
%
% Beispiele fuer Funktionsaufrufe:
%
% adaptiv_np(6.1,0,[1.2;0;0;-1.049357509830350],0.1,0.001)
% -> liefert periodische Loesung
%
% adaptiv_np(17.1,0,[0.994;0;0;-2.0015851],0.1,0.001)
% -> liefert chaotische Loesung

% Initialisierung
y(:,1)=y0(:);
t = t0;
T = t;
H = [];

% ---------------------------------------------------------------------
% Verfahrensmatrix

b = [0          0          0          0       0;
     1/4        0          0          0       0;
     4/81       32/81      0          0       0;
     57/98      -432/343   1053/686   0       0;
     1/6        0          27/52      49/156  0];

% Knotenvektor
a = [0 1/4 4/9 6/7 1];

% Gewichtsvektor
c = [43/288,0,243/416,343/1872,1/12];
% ---------------------------------------------------------------------

% Integrationsschleife
while t<tn

        k1 = f(t,y(:,end), mu);
        w2 =  y(:,end) + h*(b(2,1)*k1);
        k2 =  f(t+a(2)*h, w2, mu);
        w3 =  y(:,end) + h*(b(3,1)*k1+b(3,2)*k2);
```

```
54          k3 =  f(t+a(3)*h,w3, mu);
55          w4 =  y(:,end) + h*(b(4,1)*k1+b(4,2)*k2+b(4,3)*k3);
56          k4 =  f(t+a(4)*h, w4, mu);
57          w5 =  y(:,end) + h*(b(5,1)*k1+b(5,2)*k2+b(5,3)*k3+b(5,4)*k4);
58          k5 =  f(t+a(5)*h, w5, mu);
59
60          % Berechnung der Verfahrensfunktion
61          phi = c(1)*k1+c(2)*k2+c(3)*k3+c(4)*k4+c(5)*k5;
62
63          %-------------------------------------------------------------------
64
65
66      y = [y,y(:,end)+h*phi];
67      t=t+h;
68      T=[T,t];
69      H=[H,h];
70  end
71
72  end
73
74  %------ problemabhaengige Funktion f ---------------------------------------
75
76  function wert = f(t,y,mu)
77  % Die Funktion f beschreibt das ODE System zu 3-Koerper Problem
78  %
79  % Input:  t      Zeit
80  %         y      Vektor
81  %         mu     relative Masse
82  % Output: wert
83
84  mu = 1/82.45;
85  z1 = ((y(1)+mu)^2+y(3)^2)^(3/2);
86  z2 = ((y(1)-1+mu)^2+y(3)^2)^(3/2);
87  wert = [y(2);
88          y(1)+2*y(4)-(1-mu)*(y(1)+mu)/z1-mu*(y(1)-1+mu)/z2;
89          y(4);
90          y(3)-2*y(2)-(1-mu)*y(3)/z1-mu*y(3)/z2];
91  end
```

```
1   function [H,T,y] = adaptiv_rk34param(tn,t0,y0,h,tol,mu)
2
3   % Die Funktion adaptiv_rkf34param loest ein parameterabhaeniges AWP
4   % basierend auf dem Runge-Kutta-Fehlberg 3(4) Verfahren von
5   % mit einer adaptiven Schrittweitensteuerung
6   %
7   % Input:  tn    Endzeit
8   %         t0    Startzeit
9   %         y0    Anfangswerte
10  %         h     Startschrittweite
11  %         tol   Toleranz zu Schrittweitenbestimmung
12  %         mu    problemabaengiger Parameter
13  %
14  % Output: H     Schrittweitenvektor
15  %         T     Zeitschrittvektor
```

```
16 %          Y     Vektor der Loesungen zu dem Zeischrittenvektor T
17 %
18 % Beispiele fuer Funktionsaufrufe:
19 %
20 % adaptiv_np(6.1,0,[1.2;0;0;-1.049357509830350],0.1,0.001)
21 % -> liefert periodische Loesung
22 %
23 % adaptiv_np(17.1,0,[0.994;0;0;-2.0015851],0.1,0.001)
24 % -> liefert chaotische Loesung
25
26 % Initialisierung
27 y(:,1)=y0(:);
28 t = t0;
29 T = t;
30 H = [];
31 ke =0;
32
33 % ------------------------------------------------------------------------
34 % Verfahrensmatrix
35
36 b = [0          0          0          0       0;
37       1/4        0          0          0       0;
38       4/81       32/81      0          0       0;
39       57/98     -432/343    1053/686   0       0;
40       1/6        0          27/52      49/156  0];
41
42 % Knotenvektor
43 a = [0 1/4 4/9 6/7 1];
44
45 % Gewicht fuer Fehlerschaetzer
46 te = [-5/288 0 27/416 -245/1872 1/12];
47 % Gewichtsvektoren f¸r rk 3. und 4. Ordnung
48 gamma = [1/6;0;27/52;49/156;0];
49 % ------------------------------------------------------------------------
50
51 % Integrationsschleife
52 while and(t<tn, ke < 1000)
53     flag = 1;
54
55     while flag
56
57         k1 = f(t,y(:,end), mu);
58         w2 =  y(:,end) + h*(b(2,1)*k1);
59         k2 =  f(t+a(2)*h, w2, mu);
60         w3 =  y(:,end) + h*(b(3,1)*k1+b(3,2)*k2);
61         k3 =  f(t+a(3)*h,w3, mu);
62         w4 =  y(:,end) + h*(b(4,1)*k1+b(4,2)*k2+b(4,3)*k3);
63         k4 =  f(t+a(4)*h, w4, mu);
64         w5 =  y(:,end) + h*(b(5,1)*k1+b(5,2)*k2+b(5,3)*k3+b(5,4)*k4);
65         k5 =  f(t+a(5)*h, w5, mu);
66
67         % Berechnung der Verfahrensfunktion
68         phi = gamma(1)*k1+gamma(2)*k2+gamma(3)*k3+gamma(4)*k4+gamma(5)*k5;
69
```

```
70          % Berechnung des Fehlersch%otzers
71          dk = norm(h*(te(1)*k1+te(3)*k3+te(4)*k4+te(5)*k5));
72
73          %--Schrittweitensteuerung ---------------------------------------
74          if dk > 1.2 * tol
75              % nicht erfolgreicher Schritt:
76              ke = ke+1;
77              h = 0.9 * h;
78          elseif dk < 0.8 *tol
79              % nicht erfolgreicher Schritt:
80              ke = ke+1;
81              h = 1.1 * h;
82          else
83              % erfolgreicher Schritt:
84              ke = ke+1;
85              flag = 0;
86
87
88          end
89          %---------------------------------------------------------------
90      end
91
92      y = [y,y(:,end)+h*phi];
93      t=t+h;
94      T=[T,t];
95      H=[H,h];
96  end
97
98  end
99
100 %------ problemabhaengige Funktion f ------------------------------------
101
102 function wert = f(t,y,mu)
103 % Die Funktion f beschreibt das ODE System zu 3-Koerper Problem
104 %
105 % Input:  t      Zeit
106 %         y      Vektor
107 %         mu     relative Masse
108 % Output: wert
109
110 mu = 1/82.45;
111 z1 = ((y(1)+mu)^2+y(3)^2)^(3/2);
112 z2 = ((y(1)-1+mu)^2+y(3)^2)^(3/2);
113 wert = [y(2);
114         y(1)+2*y(4)-(1-mu)*(y(1)+mu)/z1-mu*(y(1)-1+mu)/z2;
115         y(4);
116         y(3)-2*y(2)-(1-mu)*y(3)/z1-mu*y(3)/z2];
117 end
```

```
1  % Aufgabe 8. Drei-Koerper Problem
2
3  close all
4  clear all
5
```

```
6  % Schtittweitenvektor
7  s = [0.1, 0.01, 0.001];
8
9  %-------------------------------------------------------------------------
10 % Beispiel 1: periodischer Orbit
11 %-------------------------------------------------------------------------
12
13 figure(1)
14
15 for i = 1:3
16     % Loese Drei-Koerper Problem mit RKV 4) fuer unterschiedliche
17     % Schrittweiten
18     [H1, T1, y1] = ...
19         rk4param(6.1,0,[1.2;0;0;-1.049357509830350],s(i),1/82.45);
20
21     subplot(4,2,i+(i-1)*1)
22     semilogy([T1(1:end-1);T1(2:end)],[H1;H1],'g-');
23     title(['Schrittweite h=' num2str(s(i))]);
24     print -depsc2 adaptiv_vp_fig01
25     subplot(4,2,i+i)
26     plot(y1(1,:),y1(3,:),'r-');
27     title('Phasendiagramm: Bsp 1 mit RKV4');
28     print -depsc2 adaptiv_vp_fig02
29
30 end
31
32 % Loese Drei-Koerper Problem mit eingebettetem RKV 3(4) mit
33 % Schrittweitensteuerung
34 [H1a, T1a, y1a] = ...
35     adaptiv_rk34param(6.1,0,[1.2;0;0;-1.049357509830350],0.1,0.001,1/82.45);
36
37 subplot(4,2,7)
38 semilogy([T1a(1:end-1);T1a(2:end)],[H1a;H1a],'g-');
39 title('adaptive Schrittweite');
40 print -depsc2 adaptiv_vp_fig01
41
42 subplot(4,2,8)
43 plot(y1a(1,:),y1a(3,:),'r-');
44 title('Phasendiagramm: Bsp 1 mit RKV34');
45 print -depsc2 adaptiv_vp_fig02
46
47
48 %-------------------------------------------------------------------------
49 % Beispiel 2: Chaotischer Orbit
50 %-------------------------------------------------------------------------
51
52 figure(2)
53 for i = 1:3
54 % Loese Drei-Koerper Problem mit RKV 4) fuer unterschiedliche
55 % Schrittweiten
56 [H2, T2, y2] = ...
57     rk4param(17.1,0,[0.994;0;0;-2.0015851],s(i), 0.012277471);
58
59 subplot(4,2,i+(i-1)*1)
```

```
60 semilogy([T2(1:end-1);T2(2:end)],[H2;H2],'g-');
61 title(['Schrittweite h=' num2str(s(i))]);
62 print -depsc2 adaptiv_vp_fig01
63 subplot(4,2,i+i)
64 plot(y2(1,:),y2(3,:),'r-');
65 title('Phasendiagramm: Bsp 2 mit RKV4');
66 print -depsc2 adaptiv_vp_fig02
67
68 end
69 % Loese Drei-Koerper Problem mit eingebettetem RKV 3(4) mit
70 % Schrittweitensteuerung
71
72 [H2a, T2a, y2a] = ...
73     adaptiv_rk34param(17.1,0,[0.994;0;0;-2.0015851],0.1,0.001, 0.012277471);
74
75 subplot(4,2,7)
76 semilogy([T2a(1:end-1);T2a(2:end)],[H2a;H2a],'g-');
77 title('adaptive Schrittweite');
78 print -depsc2 adaptiv_vp_fig01
79
80 subplot(4,2,8)
81 plot(y2a(1,:),y2a(3,:),'r-');
82 title('Phasendiagramm: Bsp 2 mit RKV34');
83 print -depsc2 adaptiv_vp_fig02
```

Hinweise:

Die Programmieraufgaben sind in Matlab zu erstellen. Senden Sie alle Files in einer email mit dem Betreff **Loesung-Blatt3** an **angewandte.numerik@uni-ulm.de** (Abgabetermin jeweils wie beim Theorieteil). Drucken Sie zusätzlich allen Programmcode sowie die Ergebnisse aus und geben Sie diese vor der Übung ab. Der Source Code sollte strukturiert und, wenn nötig, dokumentiert sein.

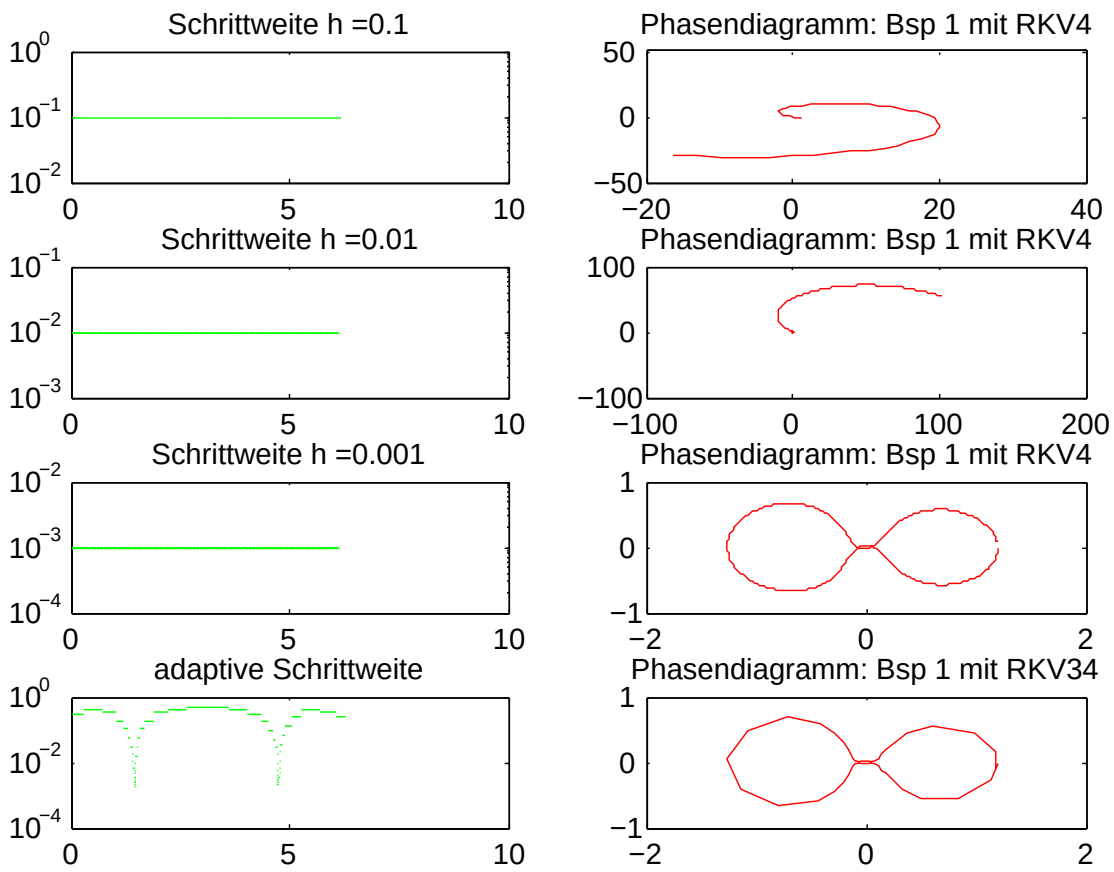


Abbildung 3: Beispiel 1: Vergleich RKV 4 mit verschiedenen Schrittweiten und eingebettetes Runge-Kutta-Fehlberg-Verfahren 3(4) mit adaptiver Schrittweitensteuerung

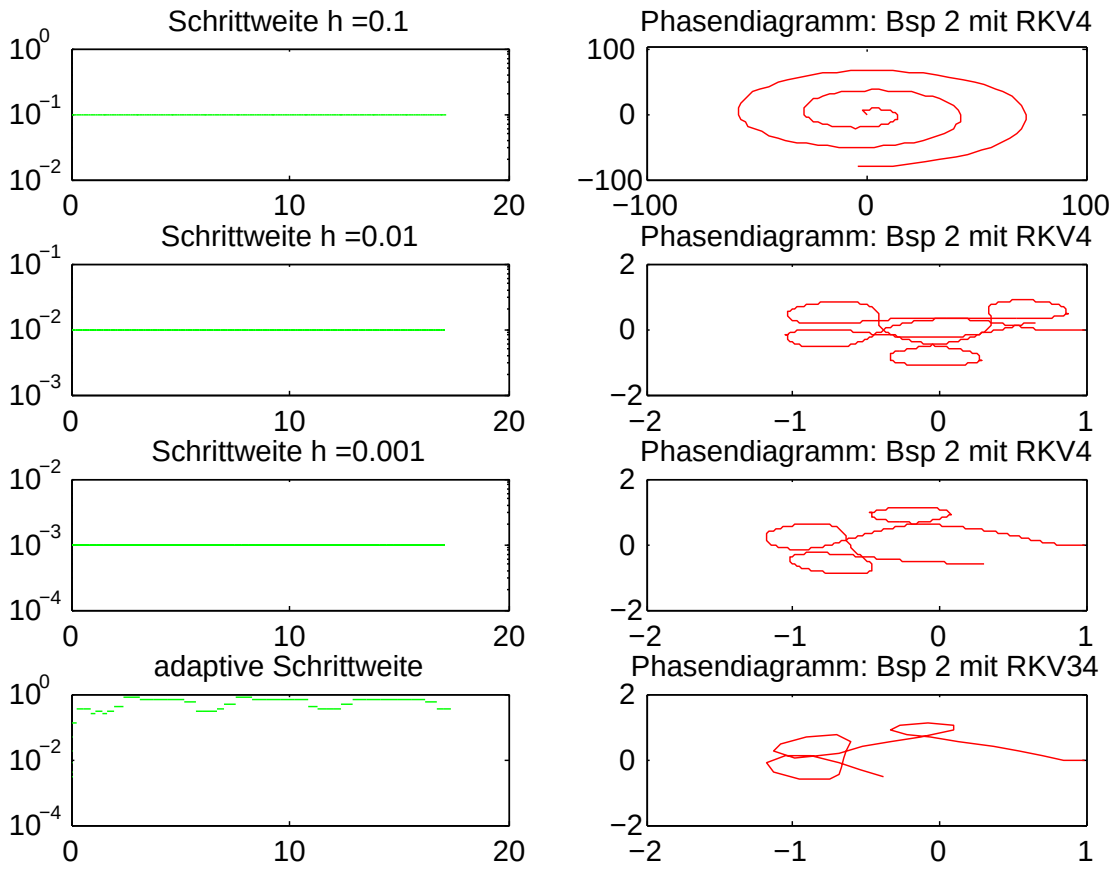


Abbildung 4: Beispiel 2: Vergleich RKV 4 mit verschiedenen Schrittweiten und eingebettetes Runge-Kutta-Fehlberg-Verfahren 3(4) mit adaptiver Schrittweitensteuerung