# Resultate aus dem Projekt zu “Einführung in die Numerik”, FS22

von Alicia Martinelli

27. Juni 2022

## Aufgabe 1: Konvergenz in 1D

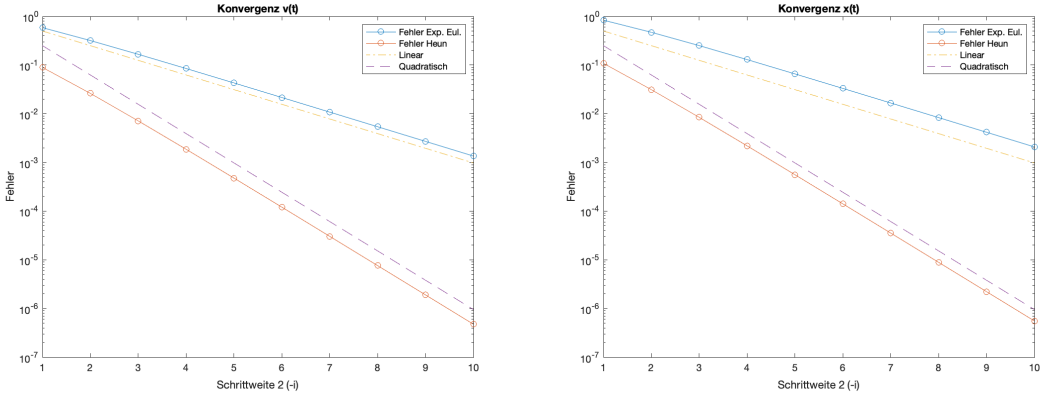


Abbildung 1: In den beiden Abbildungen sind die Konvergenz der Geschwindigkeit $v(t)$ (links) und die Konvergenz der Position $x(t)$(rechts) zu sehen. Hierzu wurde der Fehler der jeweiligen Schrittweiten $\Delta t^{(1)}$ eingezeichnet. Der Fehler von des Expliziten Euler-Verfahren ist in blau. Der Fehler des Heun-Verfahren ist in rot. Dazu wurden noch zwei Vergleichsgeraden, einmal linear und einmal quadratisch, eingezeichnet. Die X-Achse ist linear und die Y-Achse logarithmisch eingestellt.

Bei der Betrachtung der beiden Plots fällt auf, dass das Explizite Euler-Verfahren eine lineare Konvergenz aufzeigt und das Heun-Verfahren eine quadratische Konvergenz. Im Gegensatz zum expliziten Euler-Verfahren erfolgt die Näherung beim Heun-Verfahren über ein Trapez und nicht über ein Rechteck. Das explizite Euler-Verfahren hat Konvergenzordnung 1. Der Fehler geht mit $\Delta t^1$ gegen Null. Das Verfahren von Heun hat Konvergenzordnung 2, da der Fehler mit $\Delta t^2$ gegen Null geht. Dies sieht man in der Abbildung auch, da die beiden Verfahren jeweils den beiden Vergleichsgeraden folgen.

## Aufgabe 2: Einfacher Wurf

Bei dem einfachen Wurf, habe ich die Schrittweite $\Delta t^{(1)} = 0.1$ gewählt. Es fällt auf, dass das Explizite Euler-Verfahren und das Adaptive Runge-Kutta-Verfahren visuell ähnliche Trajektorien aufweisen. Das Heun-Verfahren jedoch gibt eine Trajektorie heraus, die ein wenig tiefer ist. Die Richtigkeit der Implementierung der Schrittweitenanpassung beim Adaptiven Runge-Kutta-Verfahren wurde visuell überprüft, also ob eine ähnliche Trajektorie herausgegen wird, sowie auch die berechneten Werte.

## Aufgabe 3: Ein springender Ball

Bei dem springenden Ball wurde ein Algorithmus eingeführt, der die Richtung der Y-Geschwindigkeit, sowie auch die Richtung der Y-Position ändert. Dies wurde mit einem ”if”-Statement gemacht mit der Bedingung, dass die Y-Position des Balles kleiner Null sei, was soviel bedeutet wie, dass der Ball den Boden erreicht hat. Die Y-Geschwindigkeit wurde nicht nur mit dem Vorzeichen gedreht, sondern auch mit

der Dämpfungskonstante $\alpha$ gedämpft. Wenn Dämpfungskonstante $\alpha = 1$ ist, dann wird nicht gedämpft und der Ball hüpft ohne "Verlust"weiter.

# Aufgabe 4: Luftwiderstand

Nun soll der Luftwiderstand implementiert werden. Bei allen drei Verfahren ist gut ersichtlich, dass die Trajektorie bei dem Wurf ohne Luftwiderstand höher liegt, als die Trajektorie bei dem Wurf mit Luftwiderstand.

# Aufgabe 5: Körbe werfen

| $\Delta t^{(1)}$ | i | $\mathbf{x}^{(i)} - x^{(1)}$ | $\mathbf{v}^{(i)}$ |
|---|---|---|---|
| 0.1 | 13 | 4.2426m | 8.9632m/s |
| 0.01 | 112 | 3.9244m | 8.1593m/s |
| 0.001 | 1096 | 3.8714m | 8.0270m/s |

Tabelle 1: Anzahl an Iterationen, Wurfdistanz, Endgeschwindigkeit und Schrittweite für das explizite Euler-Verfahren zum Zeitpunkt an dem der Ball zum ersten Mal den Boden berührt. Beim Abwurf: Entfernung 11 m, Höhe 2 m, Winkel 45° und Geschwindigkeit 5 m/s.

| $\Delta t^{(1)}$ | i | $\mathbf{x}^{(i)} - x^{(1)}$ | $\mathbf{v}^{(i)}$ |
|---|---|---|---|
| 0.1 | 12 | 3.8891m | 8.0710m/s |
| 0.01 | 111 | 3.8891m | 8.0710m/s |
| 0.001 | 1095 | 3.8679m | 8.0182m/s |

Tabelle 2: Anzahl an Iterationen, Wurfdistanz, Endgeschwindigkeit und Schrittweite für das Heun-Verfahren zum Zeitpunkt an dem der Ball zum ersten Mal den Boden berührt. Beim Abwurf: Entfernung 11 m, Höhe 2 m, Winkel 45° und Geschwindigkeit 5 m/s.

| $\Delta t^{(1)}$ | i | $\mathbf{x}^{(i)} - x^{(1)}$ | $\mathbf{v}^{(i)}$ |
|---|---|---|---|
| 0.1 | 15 | 4.5075m | 9.1549m/s |
| 0.01 | 16 | 4.4896m | 9.1203m/s |
| 0.001 | 16 | 4.4622m | 9.0499m/s |

Tabelle 3: Anzahl an Iterationen, Wurfdistanz, Endgeschwindigkeit und Schrittweite für das RKF2(3)-Verfahren zum Zeitpunkt an dem der Ball zum ersten Mal den Boden berührt. Beim Abwurf: Entfernung 11 m, Höhe 2 m, Winkel 45° und Geschwindigkeit 5 m/s.

Bei den erhaltenen Werten fällt auf, dass alle Verfahren nicht so präzise sind. Der exakte Wert von $x$, wenn der Ball das erste Mal auf dem Boden landet ist $x_{Exakt} = 2.95278m$. Dies wurde wie folgt berechnet, wobei $t_w$ die Wurfzeit und $x_{Exakt}$ die Wurfweite ist bis zum Aufprall auf den Boden.

$$t_w = \frac{v_0 \cdot \sin(\alpha_0)}{g} + \frac{\sqrt{(v_0 \cdot \sin(\alpha_0))^2 + 2 \cdot g \cdot h}}{g}$$

$$x_{Exakt} = v_0 \cdot \cos(\alpha_0) \cdot t_w$$

Mit den Werten von $v_0 = 5\frac{m}{s}$, $\alpha_0 = 45°$, $g = 9.81\frac{m}{s^2}$ und $h = 2m$ ergibt sich der Wert für die exakte Wurfweite $x_{Exakt} = 2.95278m$.

Verglichen mit den drei Verfahren und der Schrittweite $\Delta t^{(1)} = 0.001$ kommen folgende Fehler raus:

| Verfahren | $x_{Verfahren}$ | Abweichung $= x_{Exakt} - x_{Verfahren}$ |
|---|---|---|
| Exp. Euler-Verfahren | 3.8714m | 0.91862m |
| Heun-Verfahren | 3.8679m | 0.91512m |
| Runge-Kutta-Verfahren | 4.4622m | 1.50942m |

Tabelle 4: Der Fehler der drei Verfahren bei Schrittweite $\Delta t^{(1)} = 0.001$ mit dem exakten Wert von $x_{Exakt} = 2.95278m$.

Dem zufolge ist das Heun-Verfahren das genauste Verfahren bei der Schrittweite $\Delta t^{(1)} = 0.001$, da es die kleinste Abweichung aufweist.
Nun wurde für die Luftgeschwindigkeit eine Zufallszahl zwischen -2 und 2 implementiert und geschaut, wie schnell man einen Ball in den Korb trifft mit einem Wurf mit Luftwiderstand. Bei dem Expliziten Euler-Verfahren gelang mir der Treffer schon nach zwei Würfen. Bei dem Heun-Verfahren und Runge Kutta-Verfahren schon sogar nach einem Wurf! Dies ist sicherlich nicht nur Glück, sondern hat auch damit zu tun, dass ich schon sehr viel mit der Applikation getestet habe. Somit wusste ich schon einige Einstellungen, die den Ball sehr Nahe zum Korb brachten. Die dennoch vorhandene Abweichung mit der zufälligen Luftgeschwindigkeit war Glück. Daher wäre es sicherlich interessant eine Person testen zu lassen, die die Applikation noch nie benutzt hat.

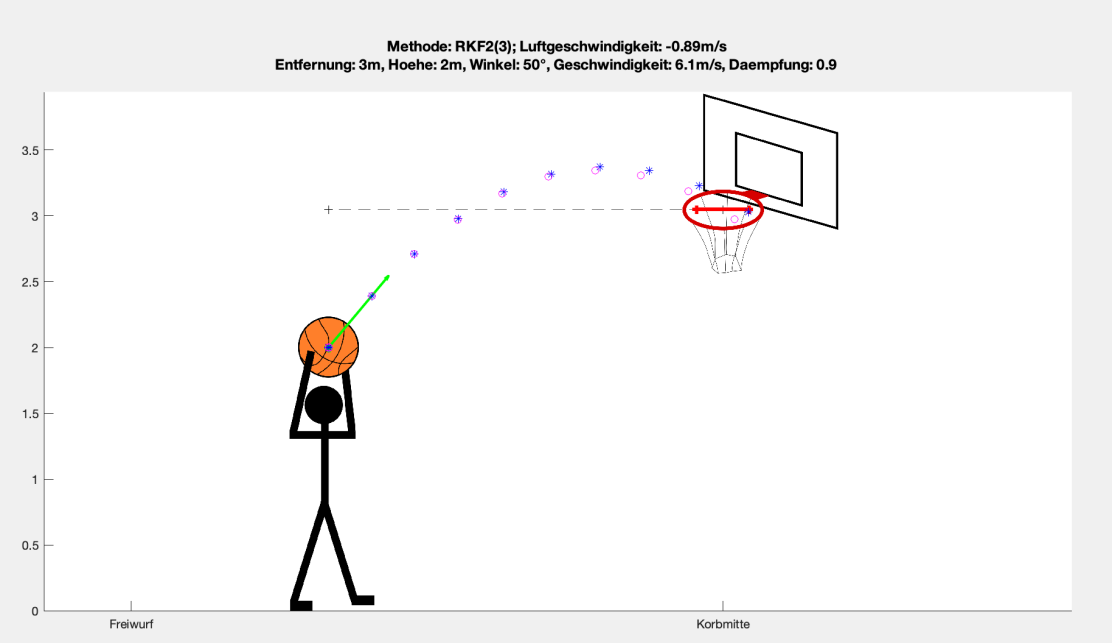


Abbildung 2: Basketballwurf einmal ohne Luftwiderstand (*) und einmal mit (o) mit dem Runge Kutta Verfahren.

# Quellcode

Listing 1: codeFolder/Aufgabe$_1$.$m$

```
%Aufgabe 1:

clear all;
clc;
close all;

%Approximationen

j = (1:10);
delta_t = (2.^(-j));

n = 1 ./ delta_t;

for k = 1:10
    for i = 1:(n(k)-1)

        %Euler-Verfahren:

        x_euler(1,k) = 0;
        v_euler(1,k) = 1;

        x_euler((i+1),k) = x_euler(i,k) + delta_t(k) * v_euler(i,k);
        v_euler((i+1),k) = v_euler(i,k) + delta_t(k) * 2 * sqrt(v_euler(i,k))
            ;

        %Heun-Verfahren:

        x_heun(1,k) = 0;
        v_heun(1,k) = 1;

        x_heun((i+1),k) = x_heun(i,k) + 0.5 * delta_t(k) * (v_heun(i,k) + (
            v_heun(i,k) + delta_t(k) * 2 * sqrt(v_heun(i,k))));
        v_heun((i+1),k) = v_heun(i,k) + 0.5 * delta_t(k) * (2 * sqrt(v_heun(i
            ,k)) + 2 * sqrt(v_heun(i,k) + delta_t(k) * 2 * sqrt(v_heun(i,k)))
            );

    end


    %Geschwindigkeit:

    Err_v_euler(k) = 4 - v_euler(n(k),k);
    Err_v_heun(k) = 4 - v_heun(n(k),k);

    %Ort:

    Err_x_euler(k) = (7/3) - x_euler(n(k),k);
    Err_x_heun(k) = (7/3) - x_heun(n(k),k);

end

%Plot Geschwindigkeit
figure(1)
semilogy(1:10,Err_v_euler,'-o',1:10,Err_v_heun,'-o',1:10,delta_t,'-.',1:10,(
    delta_t).^2,'--')
%axis([1 10 10^-7 1])
legend({'Fehler Exp. Eul.','Fehler Heun','Linear','Quadratisch'})
xlabel('Schrittweite 2^ (-i)')
ylabel('Fehler')
title('Konvergenz v(t)')
```

```
56
57 %Plot Ort
58 figure(2)
59 semilogy(1:10,Err_x_euler,'-o',1:10,Err_x_heun,'-o',1:10,delta_t,'-.',1:10,(
       delta_t).^2,'--')
60 %axis([1 10 10^-7 1])
61 legend({'Fehler␣Exp.␣Eul.','Fehler␣Heun','Linear','Quadratisch'})
62 xlabel('Schrittweite␣2^␣(-i)')
63 ylabel('Fehler')
64 title('Konvergenz␣x(t)')
```

Listing 2: `codeFolder/expEul.m`

```
1  function TrajPkte = expEul(initX,initY,initAng,initVel,daempf)
2
3  % Trajektorien Berechnung eines Basketballwurfs ohne Luftwiderstand
4  % Input:  initX .... Abwurfentfernung vom Mittelpunkt des Korbs
5  %         initY .... Abwurfh he
6  %         initAng .. Abwurfwinkel
7  %         initVel .. Abwurfgeschwindigkeit
8  %         daempf ... Daempfung (ab Aufgabe 3 relevant)
9  % Output: TrajPkte . 2-zeilige Matrix aller Punkte die der Ball beim Wurf
10 %                    durchlaueft
11
12 %Anzahl Punkte
13 n = 100;
14
15 x = zeros(2,n);
16 v = zeros(2,n);
17
18 g = -9.81;
19 a = [0 g]';
20
21 %Anfangsbedigungen Ort- und Geschwindigkeitsspaltenvektor
22 x(:,1) = [initX; initY];
23 v(:,1) = [initVel * cos(initAng); initVel * sin(initAng)];
24
25 delta_t = 0.1; %Schrittweite (0.1, 0.01, 0.001)
26
27 i = 1;
28
29 %__________________AUFGABE 2_________________%
30
31 %     while (x(2,i) > 0) && (i < n)
32 %
33 %         x(:,i+1) = x(:,i) + delta_t * v(:,i);
34 %         v(:,i+1) = v(:,i) + delta_t * a;
35 %         i = i+1;
36 %     end
37 %
38 % TrajPkte = x(:,(1:i));
39
40 %__________________AUFGABE 3_________________%
41
42 Boden = false;
43
44     while (x(1,i) < x(1,end)) && (i < n)
45
46         x(:,i+1) = x(:,i) + delta_t .* v(:,i);
47         v(:,i+1) = v(:,i) + delta_t .* a;
48         i = i+1;
49
50         if x(2,i) < 0
51
```

```
52                  if (Boden == false)
53
54                  Iterationen = i
55                  Wurfdistanz = (x(1,i) - initX(1,1))
56                  Geschwindigkeit = norm(v(:,i))
57
58                  Boden = true;
59
60                  end
61
62              x(2,i) = -x(2,i);
63              v(2,i) = -v(2,i) * daempf;
64          end
65
66      end
67
68  TrajPkte = x(:,(1:i));
69
70  %_____________________________________________%
71
72  end
```

Listing 3: codeFolder/expEulLW.m

```
1   function [TrajPkte,luftG] = expEulLW(initX,initY,initAng,initVel,daempf)
2   % Trajektorien Berechnung eines Basketballwurfs mit Luftwiderstand
3   % Input:  initX .... Abwurfentfernung vom Mittelpunkt des Korbs
4   %         initY .... Abwurfh he
5   %         initAng .. Abwurfwinkel
6   %         initVel .. Abwurfgeschwindigkeit
7   %         daempf ... Daempfung
8   % Output: TrajPkte . 2-zeilige Matrix aller Punkte die der Ball beim Wurf
9   %                    durchlaueft
10  %         luftG .... Luftgeschwindigkeit (kann bis Aufgabe 5 gleich 0
11  %                    gesetzt bleiben)
12
13  %Luftgeschwindigkeit zuf llig
14  luftG = (-2 + (2+2) * rand(1,1));
15
16  %Anzahl Punkte
17  n = 100;
18
19  x = zeros(2,n);
20  v = zeros(2,n);
21  a = zeros(2,n);
22
23  g = [0 -9.81]';
24
25  %Anfangsbedigungen Ort- und Geschwindigkeitsspaltenvektor
26  x(:,1) = [initX; initY];
27  v(:,1) = [initVel * cos(initAng); initVel * sin(initAng)];
28
29  delta_t = 0.1; %Schrittweite (0.1, 0.01, 0.001)
30
31  i = 1;
32
33  %Luftwiderstand
34  p  = 1.21; %Luftdichte [kg/m^3]
35  C_L = 0.45; %Widerstrandskonstante
36  A = 0.04; %Querschnittsfl che des Balles [m^2]
37
38  %Resultierende Beschleunigung mit Newton's zweitem Gesetz:
39  m = 0.6; %Masse des Balles [kg]
40
```

```
41      i = 1;
42
43      while (x(2,i) > 0) && (i < n)
44
45      z = sign(v(:,i) - [luftG; 0]) %Vorzeichen rel. Luftgeschwind.
46
47      x(:,i+1) = x(:,i) + delta_t .* v(:,i);
48      a(:,i+1) = (0.5 .* p .* (v(:,i) - [luftG; 0]).^2 .* C_L .* A) ./ m;
49      v(:,i+1) = v(:,i) + delta_t .* (g - z .* a(:,i+1));
50      i = i+1;
51
52      end
53
54  TrajPkte = x(:,(1:i));
55
56  end
```

Listing 4: codeFolder/heun.m

```
1   function TrajPkte = heun(initX,initY,initAng,initVel,daempf)
2   % Trajektorien Berechnung eines Basketballwurfs ohne Luftwiderstand
3   % Input:  initX .... Abwurfentfernung vom Mittelpunkt des Korbs
4   %         initY .... Abwurfh he
5   %         initAng .. Abwurfwinkel
6   %         initVel .. Abwurfgeschwindigkeit
7   %         daempf ... Daempfung (ab Aufgabe 3 relevant)
8   % Output: TrajPkte . 2-zeilige Matrix aller Punkte die der Ball beim Wurf
9   %                    durchlaueft
10
11  %Anzahl Punkte
12  n = 100;
13
14  x = zeros(2,n);
15  v = zeros(2,n);
16
17  g = -9.81;
18  a = [0 g]';
19
20  %Anfangsbedigungen Ort- und Geschwindigkeitsspaltenvektor
21  x(:,1) = [initX; initY];
22  v(:,1) = [initVel * cos(initAng); initVel * sin(initAng)];
23
24  delta_t = 0.1; %Schrittweite (0.1, 0.01, 0.001)
25
26  i = 1;
27
28  %__________________AUFGABE 2_________________%
29
30  % while (x(2,i) > 0) && (i < n)
31  %
32  %     x(:,(i+1)) = x(:,i) + 0.5 .* delta_t .* (v(:,i) + v(:,i) + delta_t .* a
        );
33  %     v(:,(i+1)) = v(:,i) + delta_t .* a;
34  %     i = i+1;
35  %
36  % end
37  %
38  % TrajPkte = x(:,(1:i));
39
40  %__________________AUFGABE 3_________________%
41
42  Boden = false;
43
44      while (x(1,i) < x(1,end)) && (i < n)
```

```
45
46          x(:,(i+1)) = x(:,i) + 0.5 .* delta_t .* (v(:,i) + v(:,i) + delta_t .*
                a);
47          v(:,(i+1)) = v(:,i) + delta_t .* a;
48          i = i+1;
49
50               if x(2,i) < 0
51
52                   if (Boden == false)
53
54                       Iterationen = i
55                       Wurfdistanz = (x(1,i) - initX(1,1))
56                       Geschwindigkeit = norm(v(:,i))
57
58                       Boden = true;
59
60                   end
61
62                   x(2,i) = -x(2,i);
63                   v(2,i) = -v(2,i) * daempf;
64               end
65      end
66
67  TrajPkte = x(:,(1:i));
68
69  %_____________________________________________%
70
71  end
```

Listing 5: codeFolder/heunLW.m

```
1   function [TrajPkte,luftG] = heunLW(initX,initY,initAng,initVel,daempf)
2   % Trajektorien Berechnung eines Basketballwurfs mit Luftwiderstand
3   % Input:  initX .... Abwurfentfernung vom Mittelpunkt des Korbs
4   %         initY .... Abwurfh he
5   %         initAng .. Abwurfwinkel
6   %         initVel .. Abwurfgeschwindigkeit
7   %         daempf ... Daempfung
8   % Output: TrajPkte . 2-zeilige Matrix aller Punkte die der Ball beim Wurf
9   %                    durchlaueft
10  %         luftG .... Luftgeschwindigkeit (kann bis Aufgabe 5 gleich 0
11  %                    gesetzt bleiben)
12
13  %Luftgeschwindigkeit zuf llig
14  luftG = (-2 + (2+2) * rand(1,1));
15
16  %Anzahl Punkte
17  n = 100;
18
19  x = zeros(2,n);
20  v = zeros(2,n);
21  a = zeros(2,n);
22
23  g = [0 -9.81]';
24
25  %Anfangsbedigungen Ort- und Geschwindigkeitsspaltenvektor
26  x(:,1) = [initX; initY];
27  v(:,1) = [initVel * cos(initAng); initVel * sin(initAng)];
28
29  delta_t = 0.1; %Schrittweite (0.1, 0.01, 0.001)
30
31  i = 1;
32
33  %Luftwiderstand
```

```
p  = 1.21; %Luftdichte [kg/m^3]
C_L = 0.45; %Widerstrandskonstante
A = 0.04; %Querschnittsfl che des Balles [m^2]

%Resultierende Beschleunigung mit Newton's zweitem Gesetz:
m = 0.6; %Masse des Balles [kg]

    i = 1;

    while (x(2,i) > 0) && (i < n)

    x(:,(i+1)) = x(:,i) + 0.5 .* delta_t .* (v(:,i) + v(:,i) + delta_t .* g);
    a(:,i+1) = (0.5 .* p .* (v(:,i) - [luftG; 0]).^2 .* C_L .* A) ./ m;
    v(:,(i+1)) = v(:,i) + delta_t .* (g - a(:,i+1));
    i = i+1;

    end

TrajPkte = x(:,(1:i));

end
```

Listing 6: codeFolder/rkf23.m

```
function TrajPkte = rkf23(initX,initY,initAng,initVel,daempf)
% Trajektorien Berechnung eines Basketballwurfs ohne Luftwiderstand
% Input:  initX .... Abwurfentfernung vom Mittelpunkt des Korbs
%         initY .... Abwurfh he
%         initAng .. Abwurfwinkel
%         initVel .. Abwurfgeschwindigkeit
%         daempf ... Daempfung (ab Aufgabe 3 relevant)
% Output: TrajPkte . 2-zeilige Matrix aller Punkte die der Ball beim Wurf
%                    durchlaueft

%Anzahl Punkte
n = 100;

x = zeros(2,n);
v = zeros(2,n);

%Anfangsbedigungen Ort- und Geschwindigkeitsspaltenvektor
x(:,1) = [initX; initY];
v(:,1) = [initVel * cos(initAng); initVel * sin(initAng)];

delta_t = zeros(1,n);
delta = zeros(1,n);

delta_t(1) = 0.1; %Schrittweite (0.1, 0.01, 0.001)

i = 1;

%__________________AUFGABE 2_________________%

% while (x(2,i) > 0) && (i < n)
%
% [x, v, delta_t, delta] = swap(x, v, delta_t, delta, i);
% i = i+1;
%
% end
%
% TrajPkte = x(:,(1:i));

%__________________AUFGABE 3_________________%

```

```
41 Boden = false;
42
43     while (x(1,i) < x(1,end)) && (i < n)
44
45          [x, v, delta_t, delta] = swap(x, v, delta_t, delta, i);
46          i = i+1;
47
48             if x(2,i) < 0
49
50                 if (Boden == false)
51
52                 Iterationen = i
53                 Wurfdistanz = (x(1,i) - initX(1,1))
54                 Geschwindigkeit = norm(v(:,i))
55
56                 Boden = true;
57
58                 end
59
60                 x(2,i) = -x(2,i);
61                 v(2,i) = -v(2,i) * daempf;
62
63             end
64     end
65
66 TrajPkte = x(:,(1:i));
67
68 %_____________________________________________%
69
70 end
```

Listing 7: codeFolder/swap.m

```
1  function [x, v, delta_t, delta] = swap(x, v, delta_t, delta, i)
2
3  tau = 0.8;
4  epsilon = 10^(-3);
5
6  delta(1) = tau^3 * epsilon;
7
8  delta_t(i+1) = delta_t(i);
9  delta(i+1) = delta(i);
10
11 a = [0 -9.81]';
12
13 gamma = [0.5,0.5,0];
14 gamma_tilde = [1/6,1/6,4/6];
15
16 gammas = gamma - gamma_tilde;
17
18 while 1
19     delta_t(i+1) = tau * (epsilon / delta(i+1))^(1/3) * delta_t(i+1);
20
21     k1x = v(:,i);
22     k2x = v(:,i) + delta_t(i+1) .* k1x;
23     k3x = v(:,i) + (delta_t(i+1)/4) .* k1x + (delta_t(i + 1)/4) .* k2x;
24
25     k1v = a;
26     k2v = a;
27     k3v = a;
28
29     delta(i+1) = delta_t(i+1) .* norm(gammas(1) .* [k1x; k1v] + gammas(2) .*
           [k2x; k2v] + gammas(3) .* [k3x; k3v]);
30
```

```
    disp(delta_t(i+1)); %While do Schleife
    if delta(i+1) <= epsilon
        break;
    end
end

x(:,i+1) = x(:,i) + 0.5 * delta_t(i+1) .* k1x + 0.5 * delta_t(i+1) .* k2x;
v(:,i+1) = v(:,i) + 0.5 * delta_t(i+1) .* k1v + 0.5 * delta_t(i+1) .* k2v;

end
```

Listing 8: codeFolder/rkf23LW.m

```
function [TrajPkte,luftG] = rkf23LW(initX,initY,initAng,initVel,daempf)
% Trajektorien Berechnung eines Basketballwurfs mit Luftwiderstand
% Input:  initX .... Abwurfentfernung vom Mittelpunkt des Korbs
%         initY .... Abwurfh he
%         initAng .. Abwurfwinkel
%         initVel .. Abwurfgeschwindigkeit
%         daempf ... Daempfung
% Output: TrajPkte . 2-zeilige Matrix aller Punkte die der Ball beim Wurf
%                    durchlaueft
%         luftG .... Luftgeschwindigkeit (kann bis Aufgabe 5 gleich 0
%                    gesetzt bleiben)

%Luftgeschwindigkeit zuf llig
luftG = (-2 + (2+2)*rand(1,1));

%Anzahl Punkte
n = 100;

x = zeros(2,n);
v = zeros(2,n);
a = zeros(2,n);

%Anfangsbedigungen Ort- und Geschwindigkeitsspaltenvektor
x(:,1) = [initX; initY];
v(:,1) = [initVel * cos(initAng); initVel * sin(initAng)];

delta_t = zeros(1,n);
delta_t(1) = 0.1; %Schrittweite (0.1, 0.01, 0.001)

delta = zeros(1,n);

i = 1;

while (x(2,i) > 0) && (i < n)

    [x, v, delta_t, delta] = swapLW(x, v, a, delta_t, delta, i, luftG);

i = i+1;

end

TrajPkte = x(:,(1:i));

end
```

Listing 9: codeFolder/swapLW.m

```
function [x, v, delta_t, delta] = swapLW(x, v, a, delta_t, delta, i, luftG)

tau = 0.8;
epsilon = 10^(-3);
```

```
5
6  delta(1) = tau^3 * epsilon;
7
8  delta_t(i+1) = delta_t(i);
9  delta(i+1) = delta(i);
10
11 g = [0 -9.81]';
12
13 gamma = [0.5,0.5,0];
14 gamma_tilde = [1/6,1/6,4/6];
15 gammas = gamma - gamma_tilde;
16
17 %Luftwiderstand
18 p  = 1.21; %Luftdichte [kg/m^3]
19 C_L = 0.45; %Widerstrandskonstante
20 A = 0.04; %Querschnittsfl che des Balles [m^2]
21
22 %Resultierende Beschleunigung mit Newton's zweitem Gesetz:
23 m = 0.6; %Masse des Balles [kg]
24
25
26 while 1
27     delta_t(i+1) = tau * (epsilon / delta(i+1))^(1/3) * delta_t(i+1);
28
29     k1x = v(:,i);
30     k2x = v(:,i) + delta_t(i+1) .* k1x;
31     k3x = v(:,i) + (delta_t(i+1)/4) .* k1x + (delta_t(i+1)/4) .* k2x;
32
33     a(:,i+1) = (0.5 .* p .* (v(:,i) - [luftG; 0]).^2 .* C_L .* A) ./ m;
34     z1 = sign(v(:,i) - [luftG; 0]);
35     k1v = g - z1 .* a(:,i+1);
36
37     a(:,i+1) = (0.5 .* p .* (v(:,i) - [luftG; 0]).^2 .* C_L .* A) ./ m;
38     z2 = sign((v(:,i) + delta_t(i+1) * k1v) - [luftG; 0]);
39     k2v = g - z2 .* a(:,i+1);
40
41     a(:,i+1) = (0.5 .* p .* (v(:,i) - [luftG; 0]).^2 .* C_L .* A) ./ m;
42     z3 = sign((v(:,i) + (delta_t(i+1)/4) * k1v + (delta_t(i+1)/4) * k2v) - [
           luftG; 0]);
43     k3v = g - z3 .* a(:,i+1);
44
45     delta(i+1) = delta_t(i+1) .* norm(gammas(1) .* [k1x; k1v] + gammas(2) .*
           [k2x; k2v] + gammas(3) .* [k3x; k3v]);
46     disp(delta_t(i+1)); %While do Schleife
47     if delta(i+1) <= epsilon
48         break;
49     end
50 end
51
52 x(:,i+1) = x(:,i) + 0.5 * delta_t(i+1) .* k1x + 0.5 * delta_t(i+1) .* k2x;
53 a(:,i+1) = (0.5 .* p .* (v(:,i) - [luftG; 0]).^2 .* C_L .* A) ./ m;
54 v(:,i+1) = v(:,i) + 0.5 * delta_t(i+1) .* k1v + 0.5 * delta_t(i+1) .* k2v;
55
56 end
```