Projekt.

Abgabe bis: **30.06.2022**

Simulation eines Basketballwurfs in 2D

Um ein Basketball in ein Korb zu werfen, müssen verschiedene Faktoren beachtet werden. In diesem Projekt werden wir diese Faktoren untersuchen und simulieren. Der Ball unterliegt der Bewegungsgleichung:

$$\begin{aligned} \dot{\boldsymbol{x}}(t) &= \boldsymbol{v}(t), \\ \dot{\boldsymbol{v}}(t) &= \boldsymbol{a}\big(\boldsymbol{v}(t)\big). \end{aligned} \tag{1}$$

Hierbei bezeichnet $\boldsymbol{x} \in \mathbb{R}^2$ den Positionsvektor, $\boldsymbol{v} \in \mathbb{R}^2$ den Geschwindigkeitsvektor und $\boldsymbol{a} \in \mathbb{R}^2$ den Beschleunigungsvektor, alle im zweidimensionalen Raum mit einer x-Komponente und einer y-Komponente versehen.
Am Anfang des Wurfes wird der Ball von einer Anfangsposition $\boldsymbol{x}_0$ mit Anfangsgeschwindigkeit $\boldsymbol{v}_0$ abgeworfen. Diese kann durch den Abwurfwinkel und den Betrag der Abwurfgeschwindigkeit festgelegt werden (Polarkoordinaten).
Auch die Beschleunigung kann in x- und y-Richtung aufgeteilt werden. Diesbezüglich ist zuallererst die Schwerkraft zu betrachten, die zu jedem Zeitpunkt gleichermassen auf den Ball wirkt und eine solche Beschleunigung erzeugt. Sie lässt sich approximieren als $\mathrm{g} = -9.81\ \mathrm{m/s^2}$ in y-Richtung.
Um die Trajektorie $\boldsymbol{x}(t)$ für $t \in [0, T_{end}]$ des Balles zu beschreiben, muss die Bewegungsgleichung (1) über die Zeit integriert werden. Dies soll in diesem Projekt mit drei unterschiedlichen Methoden implementiert werden.

Explizites Euler-Verfahren

Für das Euler-Verfahren wird die Zeitableitung formuliert als

$$\begin{aligned} \lim_{\Delta t \to 0} \frac{\boldsymbol{x}(t+\Delta t) - \boldsymbol{x}(t)}{\Delta t} &= \boldsymbol{v}(t), \\ \lim_{\Delta t \to 0} \frac{\boldsymbol{v}(t+\Delta t) - \boldsymbol{v}(t)}{\Delta t} &= \boldsymbol{a}\big(\boldsymbol{v}(t)\big) \end{aligned}$$

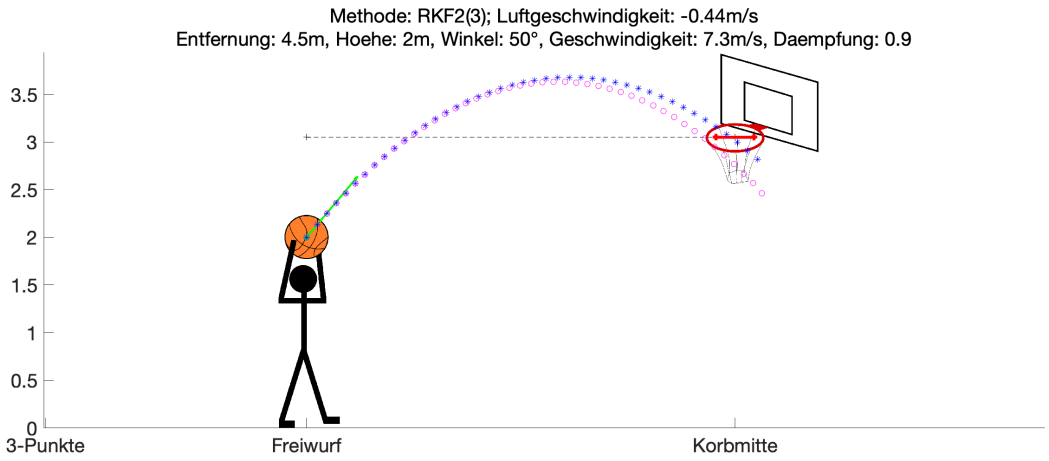

Abbildung 1: Basketballwurf einmal ohne Luftwiderstand (*) und einmal mit (o).

betrachtet. Wenn nun Δt als eine konstante, kleine Zahl gewählt wird und alle bekannten Einträge auf die rechte Seite gebracht werden, erhält man das explizite Euler-Verfahren:

$$\begin{aligned} \boldsymbol{x}^{(i+1)} &= \boldsymbol{x}^{(i)} + \Delta t\, \boldsymbol{v}^{(i)}, \\ \boldsymbol{v}^{(i+1)} &= \boldsymbol{v}^{(i)} + \Delta t\, \boldsymbol{a}(\boldsymbol{v}^{(i)}), \\ \boldsymbol{x}^{(i+1)} \approx \boldsymbol{x}(t^{(i+1)}),\ \boldsymbol{v}^{(i+1)} &\approx \boldsymbol{v}(t^{(i+1)}),\ t^{(i+1)} = t^{(i)} + \Delta t. \end{aligned}$$

Dabei entspricht $i = 1$ dem Anfangszeitpunkt $t^{(1)} = 0$, $i = 2$ dem Zeitpunkt $t^{(2)} = \Delta t$, usw. Der Zeitschritt Δt wird am Anfang der Simulation festgelegt. Wir werden in den Verfahren ohne Schrittweitenanpassung die Schrittweiten $\Delta t = 0.1, 0.01, 0.001$ betrachten.

Runge-Kutta-Verfahren zweiter Ordnung (Verfahren von Heun)

Eine andere Methode das Problem (1) zu lösen, bieten die Runge-Kutta-Verfahren. Das Runge-Kutta-Verfahren zweiter Ordnung, oder auch das Verfahren von Heun genannt, impliziert das folgende diskrete Problem:

$$\begin{aligned} \boldsymbol{x}^{(i+1)} &= \boldsymbol{x}^{(i)} + 0.5\,\Delta t \cdot \left(\boldsymbol{v}^{(i)} + \left(\boldsymbol{v}^{(i)} + \Delta t\, \boldsymbol{a}(\boldsymbol{v}^{(i)})\right)\right), \\ \boldsymbol{v}^{(i+1)} &= \boldsymbol{v}^{(i)} + 0.5\,\Delta t \cdot \left(\boldsymbol{a}(\boldsymbol{v}^{(i)}) + \boldsymbol{a}\left(\boldsymbol{v}^{(i)} + \Delta t\, \boldsymbol{a}(\boldsymbol{v}^{(i)})\right)\right). \end{aligned}$$

Wird diese Methode auf die Gleichung (1) mit konstanter Beschleunigungsfunktion $\boldsymbol{a}(\boldsymbol{v}) = \tilde{\boldsymbol{a}} \in \mathbb{R}^2$ für alle $\boldsymbol{v} \in \mathbb{R}^2$ übertragen, so vereinfacht sich die Methode zu:

$$\begin{aligned} \boldsymbol{x}^{(i+1)} &= \boldsymbol{x}^{(i)} + 0.5\,\Delta t \cdot (\boldsymbol{v}^{(i)} + \boldsymbol{v}^{(i)} + \Delta t\, \tilde{\boldsymbol{a}}), \\ \boldsymbol{v}^{(i+1)} &= \boldsymbol{v}^{(i)} + \Delta t \cdot \tilde{\boldsymbol{a}}. \end{aligned}$$

Adaptives Runge-Kutta-Verfahren (RKF2(3))

Eine naheliegende Idee ist es, eine Anpassung des Zeitschrittes während der Simulation zu machen. Ein grösserer Zeitschritt kann zwar eine geringere Genauigkeit bedeuten, aber auch weniger Rechenaufwand. Wir werden hier ein adaptives Zeitschrittverfahren für das Verfahren von Heun implementieren. Generell kann man Runge-Kutta-Verfahren als eine Familie von Methoden beschreiben, welche mit der Schrittweite Δt, Stufenzahl m, den Inkrementen $\boldsymbol{k}_l^{(x)}$ und $\boldsymbol{k}_l^{(v)}$ und den Gewichten γ_l für $l = 1, \ldots, m$ das Zeitintegral aus (1) mittels einer Quadraturregel der folgenden Form approximieren:

$$\begin{aligned} \boldsymbol{x}^{(i+1)} &= \boldsymbol{x}^{(i)} + \Delta t \sum_{l=1}^{m} \gamma_l \boldsymbol{k}_l^{(x)}, \\ \boldsymbol{v}^{(i+1)} &= \boldsymbol{v}^{(i)} + \Delta t \sum_{l=1}^{m} \gamma_l \boldsymbol{k}_l^{(v)}. \end{aligned}$$

Das adaptive Runge-Kutta-Fehlberg-Verfahren verwendet folgende Zwischenschritte:

$$\begin{aligned} \boldsymbol{k}_1^{(x)} &= \boldsymbol{v}^{(i)}, & \boldsymbol{k}_1^{(v)} &= \boldsymbol{a}(\boldsymbol{v}^{(i)}), \\ \boldsymbol{k}_2^{(x)} &= \boldsymbol{v}^{(i)} + \Delta t \cdot \boldsymbol{k}_1^{(x)}, & \boldsymbol{k}_2^{(v)} &= \boldsymbol{a}\left(\boldsymbol{v}^{(i)} + \Delta t \cdot \boldsymbol{k}_1^{(v)}\right), \\ \boldsymbol{k}_3^{(x)} &= \boldsymbol{v}^{(i)} + \Delta t/4 \cdot \boldsymbol{k}_1^{(x)} + \Delta t/4 \cdot \boldsymbol{k}_2^{(x)}, & \boldsymbol{k}_3^{(v)} &= \boldsymbol{a}\left(\boldsymbol{v}^{(i)} + \Delta t/4 \cdot \boldsymbol{k}_1^{(v)} + \Delta t/4 \cdot \boldsymbol{k}_2^{(v)}\right). \end{aligned}$$

Betrachte nun folgende Gewichte γ_l für das Runge-Kutta-Verfahren:

$(\gamma_1, \gamma_2, \gamma_3) = (0.5, 0.5, 0)$ (Gewichte des Verfahrens zur Lösungsapproximation).

Mit dieser Wahl der Gewichte erhalten wir wieder das Verfahren von Heun.
Die Simpson-Regel, welche als Kontrollverfahren benutzt wird, verwendet dieselben Zwischenschritte $\boldsymbol{k}_l$, wie oben angegeben, aber folgende Gewichte:

$$(\hat{\gamma}_1, \hat{\gamma}_2, \hat{\gamma}_3) = (1/6, 1/6, 4/6) \qquad \text{(Gewichte für das Kontrollverfahren).}$$

In unserem Fall ergibt sich also folgende Iterationsvorschrift:

$$\begin{aligned} \boldsymbol{x}^{(i+1)} &= \boldsymbol{x}^{(i)} + 0.5\Delta t\boldsymbol{k}_1^{(x)} + 0.5\Delta t\boldsymbol{k}_2^{(x)}, \\ \boldsymbol{v}^{(i+1)} &= \boldsymbol{v}^{(i)} + 0.5\Delta t\boldsymbol{k}_1^{(v)} + 0.5\Delta t\boldsymbol{k}_2^{(v)}. \end{aligned} \tag{2}$$

Für das Kontrollverfahren erhalten wir analog:

$$\begin{aligned} \hat{\boldsymbol{x}}^{(i+1)} &= \boldsymbol{x}^{(i)} + \frac{1}{6}\Delta t\boldsymbol{k}_1^{(x)} + \frac{1}{6}\Delta t\boldsymbol{k}_2^{(x)} + \frac{4}{6}\Delta t\boldsymbol{k}_3^{(x)}, \\ \hat{\boldsymbol{v}}^{(i+1)} &= \boldsymbol{v}^{(i)} + \frac{1}{6}\Delta t\boldsymbol{k}_1^{(v)} + \frac{1}{6}\Delta t\boldsymbol{k}_2^{(v)} + \frac{4}{6}\Delta t\boldsymbol{k}_3^{(v)}. \end{aligned} \tag{3}$$

Wenn der Unterschied zwischen den Werten $\boldsymbol{x}^{(i+1)}$ und $\boldsymbol{v}^{(i+1)}$ in (2) und den Werten $\hat{\boldsymbol{x}}^{(i+1)}$ und $\hat{\boldsymbol{v}}^{(i+1)}$ in (3) zu gross ist, muss der Zeitschritt Δt, den wir in beiden Verfahren (2) und (3) verwenden, verkleinert werden. Dies führt auf folgenden Algorithmus:

Algorithmus Eine Schrittweitenanpassung

Input: Position $\boldsymbol{x}^{(i)}$, Geschwindigkeit $\boldsymbol{v}^{(i)}$, Zeitschritt $\Delta t^{(i)}$, Fehler $\boldsymbol{\delta}^{(i)}$
Output: Position $\boldsymbol{x}^{(i+1)}$, Geschwindigkeit $\boldsymbol{v}^{(i+1)}$, Zeitschritt $\Delta t^{(i+1)}$, Fehler $\boldsymbol{\delta}^{(i+1)}$

setze τ, ϵ
setze $\Delta t^{(i+1)} = \Delta t^{(i)}$, $\boldsymbol{\delta}^{(i+1)} = \boldsymbol{\delta}^{(i)}$
repeat
  setze $\Delta t^{(i+1)} = \tau \left(\frac{\epsilon}{\boldsymbol{\delta}^{(i+1)}}\right)^{1/3} \Delta t^{(i+1)}$
  berechne $\boldsymbol{k}_1 = \left[\boldsymbol{k}_1^{(x)}; \boldsymbol{k}_1^{(v)}\right], \boldsymbol{k}_2 = \left[\boldsymbol{k}_2^{(x)}; \boldsymbol{k}_2^{(v)}\right], \boldsymbol{k}_3 = \left[\boldsymbol{k}_3^{(x)}; \boldsymbol{k}_3^{(v)}\right]$
  berechne $\boldsymbol{\delta}^{(i+1)} = \Delta t^{(i+1)} \| (\gamma_1 - \hat{\gamma}_1) * \boldsymbol{k}_1 + (\gamma_2 - \hat{\gamma}_2) * \boldsymbol{k}_2 + (\gamma_3 - \hat{\gamma}_3) * \boldsymbol{k}_3 \|_2$
until $\boldsymbol{\delta}^{(i+1)} \leq \epsilon$
berechne $\boldsymbol{x}^{(i+1)}$ und $\boldsymbol{v}^{(i+1)}$ mit den Gleichungen aus (2)

Für $\boldsymbol{\delta}$ muss ein Anfangswert gewählt werden. Wir nehmen $\boldsymbol{\delta}^{(1)} = \tau^3\epsilon$. Ferner setzen wir $\tau = 0.8$ und $\epsilon = 10^{-3}$.

Aufgabe 1 (Konvergenz in 1D).

Um die Genauigkeit der Integrationsverfahren zu analysieren, schauen wir uns im eindimensionalen Raum folgendes Gleichungssystem an:

$$\begin{aligned} \dot{x}(t) &= v(t), \\ \dot{v}(t) &= a\big(v(t)\big) = 2\sqrt{v(t)}. \end{aligned} \tag{4}$$

Wir setzen folgende Anfangsbedingungen fest:

$$x^{(1)} = x(0) = 0 \quad \text{und} \quad v^{(1)} = v(0) = 1. \tag{5}$$

Die Lösungen $v(t)$ und $x(t)$ des Gleichungssystems (4) mit Anfangsbedingungen (5) sind

$$v(t) = (t+1)^2 \quad \text{und} \quad x(t) = \tfrac{1}{3}(t+1)^3 - \tfrac{1}{3}.$$

Ziel der Aufgabe ist es nun, die Funktion $x(t)$ mittels Euler- und Heun-Verfahren von $t = 0$ bis $t = 1$ zu integrieren und den Fehler zur exakten Lösung zu beobachten bei der Wahl von verschiedenen Schrittweiten Δt in beiden Verfahren.

Das Euler-Verfahren führt in diesem Fall auf die Gleichung:

$$x^{(i+1)} = x^{(i)} + \Delta t\, v^{(i)},$$
$$v^{(i+1)} = v^{(i)} + \Delta t\, a(v^{(i)}).$$

Das Verfahren von Heun führt zur folgendem Iterationsvorschrift:

$$x^{(i+1)} = x^{(i)} + 0.5\,\Delta t \cdot \left(v^{(i)} + (v^{(i)} + \Delta t\, a(v^{(i)})\right),$$
$$v^{(i+1)} = v^{(i)} + 0.5\,\Delta t \cdot \left(a(v^{(i)}) + a\left(v^{(i)} + \Delta t\, a(v^{(i)})\right)\right).$$

Die gesuchten Werte zum Endzeitpunkt $t = 1$ sind $v(1) = 4$ und $x(1) = \frac{7}{3}$. Der Fehler in der Approximation der Lösung zur gegebenen Schrittweite Δt ist damit gegeben durch

$$\delta^v_{\Delta t} = 4 - v^{(n)} \quad \text{und} \quad \delta^x_{\Delta t} = \tfrac{7}{3} - x^{(n)},$$

wobei $n = 1/\Delta t$ und $v^{(n)}$ und $x^{(n)}$ die Werte der mit den beiden Verfahren approximierten Lösungen zum Endzeitpunkt 1 sind.
In einem ersten Plot sollen für die Schrittweiten $\Delta t = 2^{-1}, 2^{-2}, \ldots, 2^{-10}$ der Wert des Fehlers $\delta^v_{\Delta t}$ bei den Rechnungen mit dem Euler- und dem Heun-Verfahren dargestellt werden. Die y-Achse soll dabei logarithmisch skaliert sein. In einem zweiten Plot soll dasselbe gemacht werden für die Fehler $\delta^x_{\Delta t}$ von beiden Verfahren mit denselben Schrittweiten.
In diesen beiden Plots sollen zusätzlich eine sich mit der Schrittweite linear entwickelnde Vergleichsgerade (Gerade mit Werten Δt) und eine sich mit der Schrittweite quadratisch entwickelnde Vergleichsgerade (Gerade mit Werten $(\Delta t)^2$) eingezeichnet werden. Was fällt dir bei Betrachtung der beiden Plots auf? Das explizite Euler-Verfahren hat Konvergenzordnung 1 und das Verfahren von Heun hat Konvergenzordnung 2. Inwiefern ist das in den Plots mit den Vergleichsgeraden zu beobachten?
Zur Dokumentation: Beginne bereits hier zum Abschluss der Aufgabe 1 deine Dokumentation zum Projekt zu erstellen. Schreibe deine Antworten zu den Fragen auf und füge deine bisher geschriebenen Codes darin ein. Wie das beispielsweise gemacht werden kann, kannst du dir in der Musterabgabe und in der `.tex`-Datei dazu anschauen, die als Zusatzmaterial bereitgestellt sind.

Aufgabe 2 (Einfacher Wurf)

Ab hier betrachten wir wieder die zweidimensionalen Orts–, Geschwindikeits– und Beschleunigungsvektoren, wie auf Seite 1 eingeführt. Zum Einstieg in die Simulation des Basketballwurfs wollen wir folgenden Algorithmus in MATLAB implementieren.

Algorithmus Berechnung der Trajektorie des Wurfs bis zum Aufprall am Boden

---

Input: Abwurfentfernung d, Abwurfhöhe h, Betrag der Abwurfsgeschwindigkeit w und Abwurfwinkel θ

Output: Wurftrajektorie $\boldsymbol{x}$ bis zum Boden

Wähle $n \in \mathbb{N}$, initialisiere $\boldsymbol{x}$ und $\boldsymbol{v}$ als $2 \times n$-Matrizen, Δt und $\boldsymbol{\delta}$ als n-Vektoren

setze $\boldsymbol{x}^{(1)} = \begin{bmatrix} d \\ h \end{bmatrix}$, $\boldsymbol{v}^{(1)} = \begin{bmatrix} w \cdot \cos(\theta) \\ w \cdot \sin(\theta) \end{bmatrix}$, wähle und setze Werte für $\Delta t^{(1)}$ und $\delta^{(1)}$

setze i = 1
while $x_2^{(i)} > 0$ und $i < n$ **do**
  setze $\boldsymbol{x}^{(i+1)}, \boldsymbol{v}^{(i+1)}, \Delta t^{(i+1)}, \delta^{(i+1)} = \texttt{mthdStep}\left(\boldsymbol{x}^{(i)}, \boldsymbol{v}^{(i)}, \Delta t^{(i)}, \delta^{(i)}\right)$
  i = i+1
end while
Behalte von $\boldsymbol{x}$ bloss die $2 \times i$ relevanten Einträge

---

In diesem Algorithmus bezeichnet `mthdStep` die Implementierung eines Iterationsschrittes der jeweiligen Integrationsmethode. Die drei zuvor betrachteten Methoden, also das explizite Euler-Verfahren, das Heun-Verfahren und das adaptive Runge-Kutta-Verfahren RKF2(3), sollen so wie im Algorithmus beschrieben implementiert werden, um eine Lösungstrajektorie $\boldsymbol{x}(t)$ der Bewegungsgleichung (1) von $t = 0$ bis zum Zeitpunkt t zu finden, an dem der Ball das erste mal den Boden berührt. Hier soll die Beschleunigung konstant sein (nur durch Erdanziehung) und der Lufwiderstand vorerst vernachlässigt werden. Wir stoppen die Simulation, wenn eine bestimmte Anzahl n an Iterationsschritten durchgeführt wurde.

Um deine Codes zu testen, kannst du aus den bereitgestellten Zusatzmaterialien die neun Dateien `mainGUI.m`, `basketball.png`, `korb.png`, `expEul.m`, `heun.m`, `rkf23.m`, `expEulLW.m`, `heunLW.m` und `rkf23LW.m` in einen gemeinsamen Ordner kopieren. Die drei letzten Dateien mit Namensendung "LW" werden wir erst später brauchen wenn wir auch den Luftwiderstand berücksichtigen wollen.

Füge deinen selbstgeschriebenen Code in die drei Dateien `expEul.m`, `heun.m` und `rkf23.m` ein und passe nötigenfalls die Variablen-Namen in deinem Code an die Namen der Eingabe- und Rückgabewerte der Funktionen an. Indem du die Datei `mainGUI.m` in MATLAB nun ausführst, solltest du eine graphische Benutzeroberfläche sehen, in der du gewisse Wurfparameter einstellen kannst. Den Basketballwurf kannst du nun simulieren, indem du auf den Knopf "Wurf!" drückst. Für die Berechnung der Trajektorie wird dein Code aus den Dateien `expEul.m`, `heun.m` oder `rkf23.m` benutzt, je nachdem welche Methode du auswählst.

Wie hast du deine Anfangsschrittweite $\Delta t^{(1)}$ für die drei Verfahren gewählt? Was fällt dir bei den unterschiedlichen Methoden auf? Wie hast du die Richtigkeit der Implementierung der Schrittweitenanpassung beim RKF2(3) Verfahren überprüft?

Aufgabe 3 (Ein springender Ball).

Ein erster Schritt, um die Bewegung des Balles realistischer zu machen und über den ersten Bodenkontakt hinaus zu simulieren, ist die Richtungsänderung der Geschwindigkeit beim Aufprall. In diesem Fall muss die Schleifenbedingung auch angepasst werden: Wir stoppen die Simulation wenn der Ball in x-Richtung hinter einen bestimmten Punkt x_{end} gelangt.

Algorithmus Berechnung der Trajektorie mit Aufprall

Input: Abwurfentfernung d, Abwurfhöhe h, Betrag der Abwurfsgeschwindigkeit w, Abwurfwinkel θ, Dämpfung α

Output: Wurftrajektorie $\boldsymbol{x}$ mit Aufprall

Wähle $n \in \mathbb{N}$, initialisiere $\boldsymbol{x}$ und $\boldsymbol{v}$ als $2 \times n$-Matrizen, Δt und $\boldsymbol{\delta}$ als n-Vektoren

setze $\boldsymbol{x}^{(1)} = \begin{bmatrix} d \\ h \end{bmatrix}$, $\boldsymbol{v}^{(1)} = \begin{bmatrix} w \cdot \cos(\theta) \\ w \cdot \sin(\theta) \end{bmatrix}$, wähle und setze Werte für $\Delta t^{(1)}$ und $\delta^{(1)}$

setze i = 1
while $\boldsymbol{x}_1^{(i)} < x_{end}$ und $i < n$ **do**
  setze $\boldsymbol{x}^{(i+1)}, \boldsymbol{v}^{(i+1)}, \Delta t^{(i+1)}, \delta^{(i+1)} =$ `mthdStep`$\left(\boldsymbol{x}^{(i)}, \boldsymbol{v}^{(i)}, \Delta t^{(i)}, \delta^{(i)}\right)$
  if $\boldsymbol{x}_2^{(i)} < 0$ **then**
    $\boldsymbol{x}_2^{(i)} = -\boldsymbol{x}_2^{(i)}$
    $\boldsymbol{v}_2^{(i)} = -\boldsymbol{v}_2^{(i)} \cdot \alpha$
  end if
  i = i+1
end while
Behalte von $\boldsymbol{x}$ bloss die $2 \times i$ relevanten Einträge

---

Der Aufprall des Balles am Boden wird dadurch erkannt, dass bei einem Zeitschritt die y-Ortskoordinate negativ wird. Wir kehren diese y-Ortskoordinate und die y-Geschwindigkeit

um. Damit der Sprung nicht ideal wird, fügen wir eine Dämpfungskonstante α für die Geschwindigkeit ein. Die Dämpfung α hängt von den Sprungeigenschaften des Balles ab und wir nehmen sie als konstant an.
In der zur Verfügung gestellten Implementation `mainGUI.m` wird der Basketball vom Punkt $x \leq 0$ abgeworfen, am Punkt $x = 0$ befindet sich der Mittelpunkt des Korbes und am Punkt $x_{end} = 0.2$ der hintere Rand vom Korb.
Du kannst auch für diese Aufgabe mit den zur Verfügung gestellten Programmen deine Implementierung testen, genau so wie in Aufgabe 2 beschrieben.
Weshalb wird sowohl die Richtung der y-Geschwindigkeit geändert als auch die Richtung der y-Position? Was passiert für $\alpha = 1$?

Aufgabe 4 (Luftwiderstand).

Um die Bewegung des Balles noch realistischer zu machen, fügen wir auch den Luftwiderstand hinzu. Die Kraft, welche vom Lufwiderstand beschrieben wird, hängt von der relativen Luftgeschwindigkeit $\boldsymbol{v}_{rel}$, der Luftdichte ρ, der Querschnittsfläche des Balles A und der Widerstandskonstante C_L folgendermassen ab:

$$\boldsymbol{F}_L = \frac{1}{2}\rho \boldsymbol{v}_{rel}^2 C_L A.$$

Die Widerstandskonstante hängt von vielen Faktoren ab. In unserem Fall werden wir sie als fest vorgegeben annehmen. Somit sind folgende Konstanten gegeben:

$$\rho = 1.21\ \text{kg/m}^3, \qquad C_L = 0.45, \qquad A = \pi R^2 = 0.04\ \text{m}^2.$$

Laut Newton's zweitem Gesetz kann die resultierende Beschleunigung wie folgt berechnet werden:

$$\boldsymbol{F} = m\boldsymbol{a}.$$

Hierbei ist m die Masse des Balles und beträgt $m = 0.6$ kg. Die Beschleunigung, welche man aus dem Luftwiderstand berechnen kann, wirkt immer in entgegengesetzter Richtung zur Bewegung des Balles. Wenn der Ball also in y-Richtung eine negative Geschwindigkeit aufweist, ist diese Beschleunigung in y-Richtung positiv. Wenn der Ball aber in y-Richtung eine positive Geschwindigkeit zeigt, hat der Luftwiderstand in y-Richtung ein negatives Vorzeichen. Dasselbe gilt für die Geschwindigkeit in x-Richtung.
Es sollen wieder die Trajektorie des Basketballes unter Berücksichtigung des Luftwiderstandes mit allen drei numerischen Methoden implementiert werden. Um den Code zu testen können wieder die zur Verfügung gestellten Programme benutzt werden. Gehe dazu vor wie in Aufgabe 2 beschrieben und füge deinen Code dieses Mal passend in die Dateien `expEulLW.m`, `heunLW.m` und `rkf23LW.m` ein.
Was passiert, wenn im selben Plot die Bewegung mit und die Bewegung ohne Luftwiderstand eingezeichnet wird?

Aufgabe 5 (Körbe werfen).

Bereite die in den Zusatzmaterialien bereitgestellten Codes so vor, wie in den Aufgaben 2 bis 4 beschrieben. Dabei soll jetzt dein eigener Code in `expEul.m` und `expEulLW.m`, in `heun.m` und `heunLW.m` und in `rkf23.m` und `rkf23LW.m` stehen. Dabei ist jeweils in der ersten Datei dein Code zur Berechnung der Trajektorie vom Wurf mit Aufprall ohne Luftwiderstand (Aufgabe 3) und in der zweiten Datei mit der Endung "LW" dein Code zur Berechnung der Trajektorie unter Berücksichtigung des Luftwiderstandes (Aufgabe 4).
Ändere deine drei Programme ohne Luftwiderstand, i.e. `expEul.m`, `heun.m` und `rkf23.m`, so ab, dass im MATLAB Command Window die Anzahl an Iterationen, Wurfdistanz und

Geschwindigkeit zur ersten Berührung des Balles mit dem Boden ausgeben wird. Fixiere eine bestimmte Parametereinstellung und führe jede dieser drei Dateien für die drei Schrittweiten $\Delta t^{(1)} = 0.1, 0.01, 0.001$ aus. Füge die Werte die du dabei erhältst zusammen mit deiner Parameterwahl in deine Dokumentation ein wie in Tabelle 1 dargestellt. Was fällt dir bei den erhaltenen Werten auf?

$\Delta t^{(1)}$	i	$x^{(i)} - x^{(1)}$	$v^{(i)}$
0.1			
0.01			
0.001			

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

Die Luftgeschwindigkeit in x-Richtung soll durch eine Zufallszahl zwischen -2 und 2 gegeben werden. Passe dementsprechend deine drei Funktionen `expEulLW.m`, `heunLW.m` und `rkf23LW.m` an. Drücke nun auf den "Wurf mit Luftwiderstand!" Knopf. Wie viele Versuche hast du gebraucht, um den Korb zu treffen? Mit welcher Integrationsmethode hattest du Glück? Hat der Luftwiderstand einen Einfluss gehabt?
Wirf nun zum Abschluss noch einmal den Basketball mit Parametereinstellungen und der Methode zur Integration deiner Wahl. Lass dir dabei die Trajektorie für den Wurf sowohl ohne wie auch mit Luftwiderstand berechnen. Drücke nun auf den Knopf "Wurf speichern!", um ein Bild ähnlich zu dem in Abbildung 1 zu erhalten, und füge zuletzt das dabei entstandene Bild in deine Dokumentation ein.

Abgabe. Als Abgabe soll eine Dokumentation in Form eines einzigen PDF-Dokumentes erstellt werden. In dieser PDF-Datei sollen enthalten sein:

- deine Plots aus Aufgabe 1,
- die Antworten auf die Fragen aus Aufgabe 1 bis 5,
- die drei Wertetabellen zu den verschiedenen Verfahren aus Aufgabe 5,
- dein Bild vom Wurf aus Aufgabe 5, und
- dein MATLAB Code zu Aufgabe 1 und die 6 von dir ergänzten Codes `expEul.m` und `expEulLW.m`, `heun.m` und `heunLW.m`, `rkf23.m` und `rkf23LW.m` beim Stand zum Ende von Aufgabe 5.

Es wurde eine mit LaTeXverfasste Musterabgabe für die Dokumentation in den Zusatzmaterialien bereitgestellt. In der `tex`-Datei wird unter anderem vorgeführt wie du deine Plots und deinen MATLAB Code in eine mit LaTeX erstellte PDF-Datei einfügen kannst.