1. [*Präsenz*] **Alternative Koordinatensysteme I: Zylinder**

Für viele Probleme ist es zuträglich, diese nicht in kartesischen Koordinaten zu berechnen, sondern ein alternatives Koordinatensystem zu wählen. Für ein klassisches Fadenpendel beschreiben wir beispielsweise die Position des Schwingers über einen Abstand bzw. Radius ρ zur Ursprung (fest vorgeschrieben durch die Pendellänge ℓ) und eine Auslenkung φ. Diese Darstellung bezeichnet man als Polarkoordinaten. Eine Erweiterung auf drei Raumdimensionen stellen die *Zylinderkoordinaten* dar, wobei der Radius ρ nun den senkrechten Abstand zur z-Achse angibt. Bezüglich des kartesischen Systems sind die Zylinderkoordinaten gegeben durch

$$\mathbf{r} = \begin{pmatrix} x(\rho,\varphi,z) \\ y(\rho,\varphi,z) \\ z(\rho,\varphi,z) \end{pmatrix} = \begin{pmatrix} \rho\,\cos(\varphi) \\ \rho\,\sin(\varphi) \\ z \end{pmatrix}$$

Anmerkung: Bezüglich des kartesischen Koordinatensystems meint, dass wir immer noch die Basisvektoren

$$\mathbf{e}_x = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad \mathbf{e}_y = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \text{ und } \quad \mathbf{e}_z = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

annehmen, ein Ortsvektor also die Form $\mathbf{r} = x\,\mathbf{e}_x + y\,\mathbf{e}_y + z\,\mathbf{e}_z$ hat.

(a) Zunächst wollen wir die Basisvektoren des zylindrischen Koordinatensystems in der kartesischen Darstellung berechnen. Deren Definition lautet:

$$\mathbf{e}_\rho = \frac{\frac{\partial \mathbf{r}}{\partial \rho}}{\left|\frac{\partial \mathbf{r}}{\partial \rho}\right|}, \quad \mathbf{e}_\varphi = \frac{\frac{\partial \mathbf{r}}{\partial \varphi}}{\left|\frac{\partial \mathbf{r}}{\partial \varphi}\right|}, \quad \mathbf{e}_z = \frac{\frac{\partial \mathbf{r}}{\partial z}}{\left|\frac{\partial \mathbf{r}}{\partial z}\right|}.$$

(b) Stelle den Ortsvektor $\mathbf{r}$ mittels der neuen Basisvektoren dar, d.h. $\mathbf{r} = A\,\mathbf{e}_\rho + B\,\mathbf{e}_\varphi + C\,\mathbf{e}_z$.

Die kartesischen Basisvektoren $\mathbf{e}_x$, $\mathbf{e}_y$ und $\mathbf{e}_z$ sind ortsunabhängig. Dies bedeutet insbesondere für zeitliche Ableitungen

$$\mathbf{v} = \dot{\mathbf{r}} = \dot{x}\,\mathbf{e}_x + \dot{y}\,\mathbf{e}_y + \dot{z}\,\mathbf{e}_z \quad \text{und} \quad \mathbf{a} = \ddot{\mathbf{r}} = \ddot{x}\,\mathbf{e}_x + \ddot{y}\,\mathbf{e}_y + \ddot{z}\,\mathbf{e}_z\,.$$

In der zylindrischen Basisdarstellung ist dies nicht mehr gültig und die Basisvektoren selbst besitzen eine Zeitableitung.

(c) Bestimme die zeitliche Ableitung der Basisvektoren $\mathbf{e}_\rho$, $\mathbf{e}_\varphi$ und $\mathbf{e}_z$. Identifiziere die Ergebnisse mit den Basisvektoren.

(d) Bestimme nun die Vektoren der Geschwindigkeit und Beschleunigung, ausgedrückt durch die zylindrischen Basisvektoren, indem du die Zeitableitung auf dein Ergebnis in Teil b) anwendest.
Hinweis: *Beachte die Produktregel!*

(e) In der Lagrange-Funktion taucht die kinetische Energie $T = \frac{1}{2}m|\mathbf{v}|^2$ auf. Berechne diese in Zylinderkoordinaten, d.h. bestimme $T \equiv T(\rho,\varphi,z,\dot{\rho},\dot{\varphi},\dot{z})$.
Hinweis: *Du kannst ausnutzen, dass* $\mathbf{e}_i \cdot \mathbf{e}_j = \delta_{ij}$.

Info: Adam Wysocki, E2 6, Zi. 4.26, a.wysocki@lusi.uni-sb.de
Marc Thome, E2 6, Zi. 4.23, thome.marc@lusi.uni-sb.de
Joshua Weißenfels, s9joweis@uni-saarland.de
Nic Collingro, nico00001@uni-saarland.de

2. [*Computer*] **Einfaches Fadenpendel**

Die Bewegungsgleichung des einfachen Fadenpendels mit Pendellänge ℓ lautet

$$\ddot{\varphi} = \frac{g}{\ell} \sin(\varphi) \,.$$

Wir stoßen also schon bei diesem simplem Problem an die Grenzen der analytischen Berechenbarkeit und müssen uns einer Näherung für kleine Winkel bedienen, um eine Lösung zu finden. Wollen wir jedoch die exakte Differenzialgleichung lösen, können wir uns der Numerik behelfen. Die Programmiersprache PYTHON bietet dafür die Funktion `odeint` (Ordenary Differential Equation Integration), welche im Paket `scipy.integrate` zu finden ist.
`odeint` erwartet ein System von Differenzialgleichung der Form

$$\frac{d\mathbf{y}}{dt} = \mathbf{F}(\mathbf{y}, t) \,,$$

weshalb wir zunächst die Differenzialgleichung zweiten Grades des Pendels in ein gekoppeltes System ersten Grades überführen müssen.

(a) Wir definieren $\theta = \dot{\varphi}$. Nutze dies, um dies um ein gekoppeltes System aus Differenzialgleichung für das Pendel aufzustellen, d.h. finde eine vektorielle Funktion $\mathbf{F}$, so dass

$$\begin{pmatrix} \dot{\varphi} \\ \dot{\theta} \end{pmatrix} = \mathbf{F}(\varphi, \theta, t) \,.$$

(b) Implementiere die Funktion `derivatives_pendulum` indem du $\mathbf{F}(\varphi, \theta, t)$ in Python-Code übersetzt. Die Funktion erwartet dabei als Input eine Liste `x`, wobei wir `x[0]` $=\varphi$ und `x[1]` $= \theta$ identifizieren und sollte die Liste $[\dot{\varphi}, \ddot{\varphi}]$ zurückgeben.
Hinweis: *Die Pendelgleichungen werden nicht explizit vom Zeitpunkt t abhängen,* `odeint` *erwartet jedoch eine Input Funktion im der Signatur* `f(x, t, ...)`*, weshalb der Parameter in der Funktionen-Definition vorkommt.*

Info: Adam Wysocki, E2 6, Zi. 4.26, a.wysocki@lusi.uni-sb.de
Marc Thome, E2 6, Zi. 4.23, thome.marc@lusi.uni-sb.de
Joshua Weißenfels, s9joweis@uni-saarland.de
Nic Collingro, nico00001@uni-saarland.de

Präsenzübung 4

1) Alternative Koordinatensysteme I: Zylinder

a) $\vec{r} = \begin{pmatrix} r\cos(\varphi) \\ r\sin(\varphi) \\ z \end{pmatrix}$

$\Rightarrow \frac{\partial \vec{r}}{\partial r} = \begin{pmatrix} \cos(\varphi) \\ \sin(\varphi) \\ 0 \end{pmatrix}$ $\quad \left|\frac{\partial \vec{r}}{\partial r}\right|^2 = \cos^2(\varphi) + \sin^2(\varphi) = 1$ ✓

$\vec{e}_r = \begin{pmatrix} \cos(\varphi) \\ \sin(\varphi) \\ 0 \end{pmatrix}$

$\Rightarrow \frac{\partial \vec{r}}{\partial \varphi} = \begin{pmatrix} -r\sin(\varphi) \\ r\cos(\varphi) \\ 0 \end{pmatrix}$ $\quad \left|\frac{\partial \vec{r}}{\partial \varphi}\right|^2 = r^2\left(\sin^2(\varphi) + \cos^2(\varphi)\right) = r^2$

$\vec{e}_\varphi = \begin{pmatrix} -\sin(\varphi) \\ \cos(\varphi) \\ 0 \end{pmatrix}$

$\Rightarrow \frac{\partial \vec{r}}{\partial z} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$ $\quad \left|\frac{\partial \vec{r}}{\partial z}\right|^2 = 1$ ✓

$\vec{e}_z = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$

b) $\vec{r} = r\vec{e}_r + z\vec{e}_z$

c) $\dot{\vec{e}}_r = \begin{pmatrix} -\dot{\varphi}\sin(\varphi) \\ \dot{\varphi}\cos(\varphi) \\ 0 \end{pmatrix} = \dot{\varphi}\vec{e}_\varphi$

$\dot{\vec{e}}_\varphi = \begin{pmatrix} -\dot{\varphi}\cos(\varphi) \\ -\dot{\varphi}\sin(\varphi) \\ 0 \end{pmatrix} = -\dot{\varphi}\vec{e}_r$

$\dot{\vec{e}}_z = 0$

d) $\vec{v} = \dot{\vec{r}} = \dot{r}\vec{e}_r + r\dot{\vec{e}}_r + \dot{z}\vec{e}_z$

$= \dot{r}\vec{e}_r + r\dot{\varphi}\vec{e}_\varphi + \dot{z}\vec{e}_z$

$\vec{a} = \ddot{\vec{r}} = \ddot{r}\vec{e}_r + \dot{r}\dot{\vec{e}}_r + \dot{r}\dot{\varphi}\vec{e}_\varphi + r\ddot{\varphi}\vec{e}_\varphi + r\dot{\varphi}\dot{\vec{e}}_\varphi + \ddot{z}\vec{e}_z$

($\dot{\vec{e}}_r = \dot{\varphi}\vec{e}_\varphi$, $\dot{\vec{e}}_\varphi = -\dot{\varphi}\vec{e}_r$)

$= (\ddot{r} - r\dot{\varphi}^2)\vec{e}_r + (2\dot{r}\dot{\varphi} + r\ddot{\varphi})\vec{e}_\varphi + \ddot{z}\vec{e}_z$

e) $T = \frac{1}{2}m|\vec{v}|^2 = \frac{1}{2}m(\dot{r}^2 + r^2\dot{\varphi}^2 + \dot{z}^2)$

TPIb_Uebung04_coding_SS25

May 5, 2025

1 Theoretische Physik Ib - Analytische Mechanik

Marc Thome, Philipp Hövel, Saarland University
Date: **May 05, 2025**

1.0.1 Tasks

- Getting started with Python
- Pendulum equation

1.0.2 Importing helpful Python modules

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt

        from scipy.integrate import odeint
```

1.0.3 Defining the right-hand side of the differential equation

Equations of motion:

$$\ddot{\varphi} = -\omega^2 \sin\varphi$$

```
In [2]: def derivatives_pendulum(x, t, omega):
            # x contains the values [phi, d/dt phi]
            phi, phi_dot = x

            # returns [d/dt phi, d^2/dt^2 phi]
            return [phi_dot, -omega**2 * np.sin(phi)]
```

1.0.4 Defining the solution for the small-angle approximation

```
In [3]: def solution_small_angle_approx(t, omega, amp):
            return amp * np.sin(omega * t), amp * omega * np.cos(omega * t)
```

1.0.5 Here comes the main part

1.0.6 Define parameters

In [4]:
```
# define Parameters
g = 9.81
# pendulum length
L = 5.0
# pendulum frequency
omega = np.sqrt(g / L)
# period length
T = 2.0 * np.pi / omega

# amplitude of the oscillation
init_angle = 15.0   # degrees
# init_angle = 80.0  # degrees (this angle will violate the small angle app
A = np.pi / 180.0 * init_angle # radians
# initial values [phi, d/dt phi]
x_init = [0.0, omega * A]
```

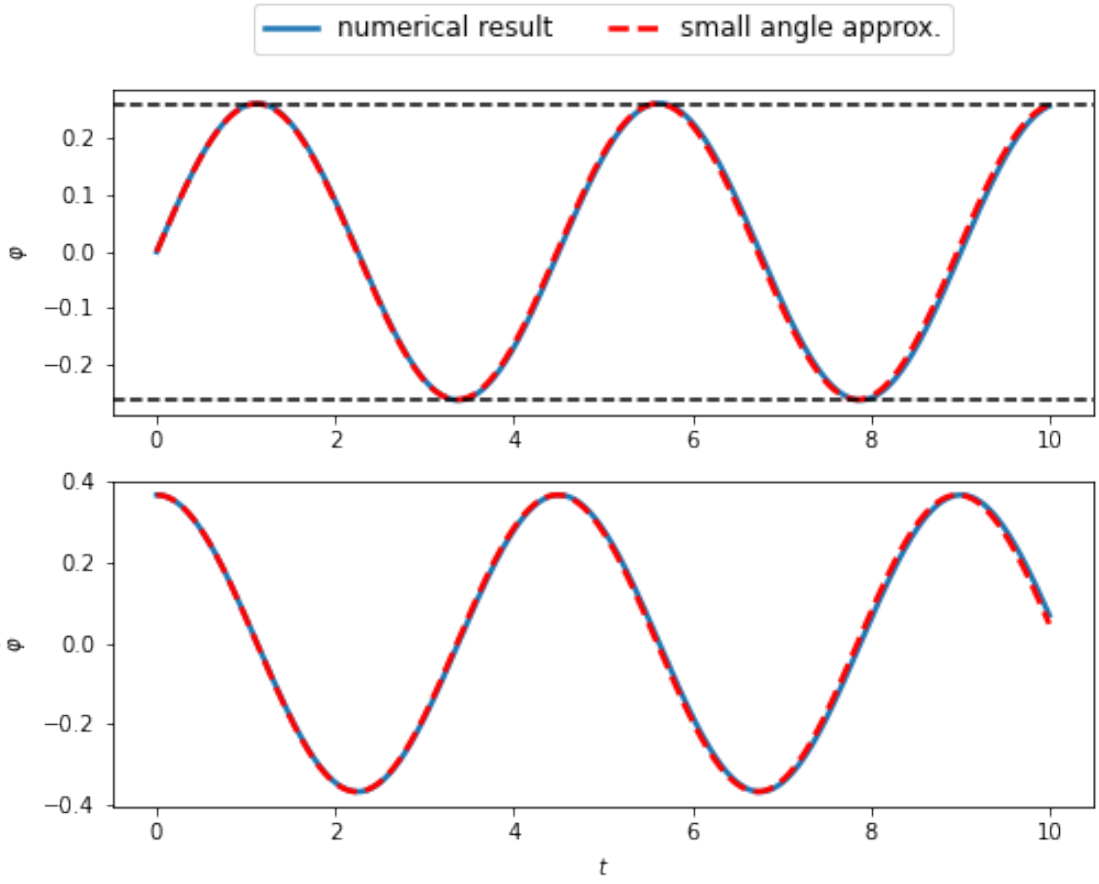
1.0.7 Let the magic begin

In [5]:
```
# this cell does the integration of out equations of motion by calling odei

# time axis
time = np.arange(0.0, 10.0, 1e-3)
# phi and d/dt phi for the small angle approximation
sas, sas_dot = solution_small_angle_approx(time, omega, A)

# integrate the exact equations of motion numerically
result = odeint(derivatives_pendulum, x_init, time, args=(omega,))
# split solution array into two arrays
phi, phi_dot = result.transpose()
```

1.0.8 Plotting the time series

In [6]:
```
# plot the results in two panels underneath each other
fig, axs = plt.subplots(nrows=2, ncols=1, figsize=(8,6))
# plot the numerical result for phi
axs[0].set_ylabel(r'$\varphi$')
axs[0].plot(time, phi, lw=2.5)
# plot the small angle approximation as red dashes line
axs[0].plot(time, sas, '--r', lw=2.5)

axs[0].legend(['numerical result', 'small angle approx.'],
              fontsize=12, ncol=2, loc='upper center', bbox_to_anchor=(0.5,

x_min, x_max = axs[0].get_xlim()
# plot  +/- maximum amplitude of the small angle solution as black dashed l
```

```
axs[0].plot([x_min, x_max], [A, A], '--k')
axs[0].plot([x_min, x_max], [-A, -A], '--k')
axs[0].set_xlim(x_min, x_max)

# plot the numerical and small angle solution for d/dt phi
axs[1].set_ylabel(r'$\dot\varphi$')
axs[1].set_xlabel(r'$t$')
axs[1].plot(time, phi_dot, lw=2.5)
axs[1].plot(time, sas_dot, '--r', lw=2.5)
plt.savefig('pendulum.png', dpi=300)
```



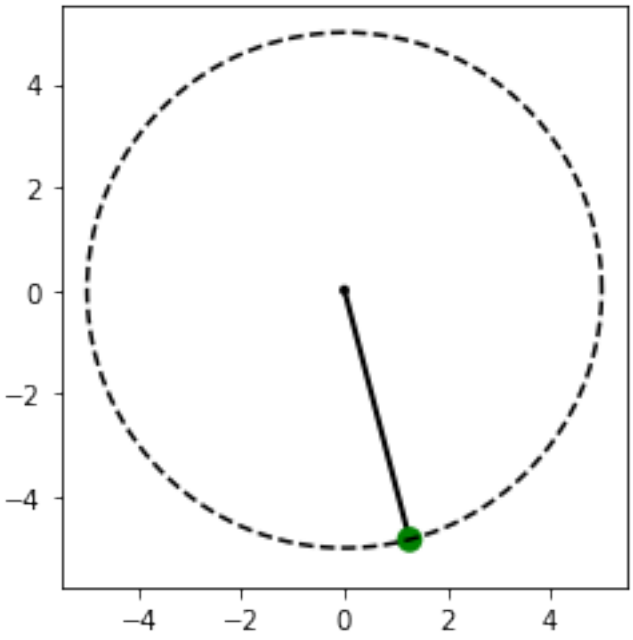
1.0.9 Let's do an animation (and save it as gif)

```
In [7]: # helpful modules
        from matplotlib.patches import Circle
        from matplotlib.animation import FuncAnimation
```

Never mind the details of the code. Just enjoy the output file.

In [8]:
```
# animation
figa, ax = plt.subplots()

# plot a black dashed circle around the origin (anker of the pendulum)
a = np.linspace(0.0, 2.0 * np.pi, 5000)
cx = L * np.cos(a)
cy = L * np.sin(a)

ax.set_aspect('equal')
ax.plot(cx, cy, '--k')

# calculate x- and y-coordinates for all values phi(t)
xp = [ L * np.sin(p) for p in phi]
yp = [-L * np.cos(p) for p in phi]

# plot the thread of the pendulum
thread, = ax.plot([0.0, xp[0]], [0.0, yp[0]], 'k', lw=2.0, zorder=-1000)

# plot a little black circle as anker for the pendulum
p_fix = Circle(xy=(0.0, 0.0),
               radius=0.1,
               facecolor='k')
ax.add_patch(p_fix)

# plot the pendulum mass aka the bob
p_bob = Circle(xy=(xp[0], yp[0]),
               radius=0.25,
               facecolor='g')
ax.add_patch(p_bob)

# since we have a lot of data points we only want to animate every skip_fra
# therefore we cut the data here
skip_frames = 20
xp = xp[::skip_frames]
yp = yp[::skip_frames]
num_frames = len(xp)

# function to update our plot objects (the pendulum bob and the thread)
def update(frame: int):
    p_bob.center = xp[frame], yp[frame]
    thread.set_xdata([0.0, xp[frame]])
    thread.set_ydata([0.0, yp[frame]])
    return (p_bob, thread)

ani = FuncAnimation(fig=figa, func=update, frames=num_frames, interval=1)

# save the animation as a gif, you can find it in the same folder as this n
ani.save('pendulum_animation.gif', fps=30)
```

```
MovieWriter ffmpeg unavailable; using Pillow instead.
```



In [ ]: