06_Uebungsaufgaben_Loesungen

0.1 Vorschläge für Übungsaufgaben

Die folgenden Übungsaufgaben sind ausführlicher als die unterwegs vorgeschlagenen Mini-Übungen. Das Ziel ist es hier, nicht nur einzelne Themen direkt auszuprobieren, sondern vielmehr verschiedene Themen zu kombinieren, um so ein verknüpfendes Verständnis der Lerninhalte zu erreichen.

0.1.1 NumPy: ndarrays

Implementieren Sie die folgenden Funktionen ohne Schleifen:

A = tridiagonal(l,d,u,N) liefert für $N \geq 2$ als Rückgabewert das Objekt A vom Typ numpy.ndarray mit A.shape = (N,N):

$$A = \begin{bmatrix} \mathrm{d} & \mathrm{u} & 0 & \dots & 0 \\ \mathrm{l} & \mathrm{d} & \mathrm{u} & \ddots & \vdots \\ 0 & \mathrm{l} & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \mathrm{u} \\ 0 & \dots & 0 & \mathrm{l} & \mathrm{d} \end{bmatrix} \in \mathbb{R}^{\mathrm{N}\times\mathrm{N}}.$$

Gegeben sei die Matrix $A \in \mathbb{R}^{I\times J}$ (A.shape = (I,J)) sowie die beiden Integer-Werte $0 \leq$ start_row $< I$ und $0 \leq$ start_col $< J$. Außerdem sei $B \in \mathbb{R}^{M\times N}$ (B.shape = (M,N)) mit $M \leq I-$ start_row und $N \leq J-$ start_col.

Die Funktion insert(B,start_row,start_col,A) befüllt die Elemente $a_{i,j}$ der Matrix A (Indizierung ab 0), für start_row $\leq i <$ start_row $+M$ und start_col $\leq j <$ start_col $+N$ mit den Elementen $b_{m,n}$ der Matrix B mit $m = 0,\dots,M-1$ und $n = 0,\dots,N-1$. Sie verhält sich also beispielsweise wie folgt:

$$B = \begin{bmatrix} 1 & 0 \\ 0 & 3 \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{3\times 2}, \quad A = \begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix} \in \mathbb{R}^{4\times 5},$$

$$\text{insert(B,1,3,A)} \quad \Longrightarrow \quad A = \begin{bmatrix} * & * & * & * & * \\ * & * & * & 1 & 0 \\ * & * & * & 0 & 3 \\ * & * & * & 0 & 1 \end{bmatrix}.$$

[6]:
```
import numpy as np

def tridiagonal(l,d,u,N):
  '''
  Erzeugt eine numpy tridiagonale Matrix mit drei verschiedenen Einträgen

          <------- N -------->

         [ d  u  0 ...    ... 0]   ^
         [ l  d  u  0 ... ... 0]   |
    A = [ 0  l  d  u  0  ... 0]   N
         [ .  0  l ...    ... u]   |
         [ 0 ...       0   l  d]   V
  '''
  return np.diagflat([u]*(N-1), 1) + np.diagflat([l]*(N-1), -1) + d * np.eye(N)

def insert(B,start_row,start_col,A):
  '''
  Befüllt Teile von A mit B
  '''
  stop_row = start_row+B.shape[0]
  stop_col = start_col+B.shape[1]
  A[start_row:stop_row,start_col:stop_col] = B[:,:]



# ============================================================
# Bitte den Code ab hier nicht mehr verändern.
# Die unten stehenden Befehle testen Ihren Code automatisch!
#
# Es ist immer eine gute Idee, ein paar bekannte Testfälle
# für Ihre Funktionen zu implementieren! Dadurch können
# Fehler sehr früh im Entwicklungsprozess ausgeschlossen
# werden!
#
# ============================================================


try:
  empty=np.arange(0)
  empty.shape=(0,0)
  D=np.diag([-4,5])
  D_h=D.copy()
  D_h.shape=(1,4)
  A=np.array([[2,-1,0,0],[-1,2,-1,0],[0,-1,2,-1],[0,0,-1,2]])
  A_mn=np.arange(15)+1.
  A_mn.shape=(5,3)
```

```
  a=np.arange(1)
  a.shape=(1,1)
  insert(empty,0,0,D)
  assert np.all(np.isclose(D,np.diag([-4,5])))
  insert(empty,0,1,D)
  assert np.all(np.isclose(D,np.diag([-4,5])))
  insert(empty,1,0,D)
  assert np.all(np.isclose(D,np.diag([-4,5])))
  insert(empty,1,1,D)
  assert np.all(np.isclose(D,np.diag([-4,5])))
  insert(D,0,0,A)
  assert np.all(np.isclose(A,np.
 ↪array([[-4,0,0,0],[0,5,-1,0],[0,-1,2,-1],[0,0,-1,2]])))
  insert(D,2,2,A)
  assert np.all(np.isclose(A,np.
 ↪array([[-4,0,0,0],[0,5,-1,0],[0,-1,-4,0],[0,0,0,5]])))
  insert(D_h,3,0,A)
  assert np.all(np.isclose(A,np.
 ↪array([[-4,0,0,0],[0,5,-1,0],[0,-1,-4,0],[-4,0,0,5]])))
  insert(D,3,1,A_mn)
  assert np.all(np.isclose(A_mn,np.
 ↪array([[1,2,3],[4,5,6],[7,8,9],[10,-4,0],[13,0,5]])))
  insert(a,0,2,A_mn)
  assert np.all(np.isclose(A_mn,np.
 ↪array([[1,2,0],[4,5,6],[7,8,9],[10,-4,0],[13,0,5]])))
  insert(D*0,0,0,D)
  assert np.all(np.isclose(D,np.zeros_like(D)))
  print("Aufgabe OK")
except:
  print("Aufgabe nicht OK")
```

```
Aufgabe OK
```

0.1.2 NumPy und matplotlib: Vektorisierung von Funktionen

Die Formel für die Berechnung der Flughöhe beim Ballwurf ist durch

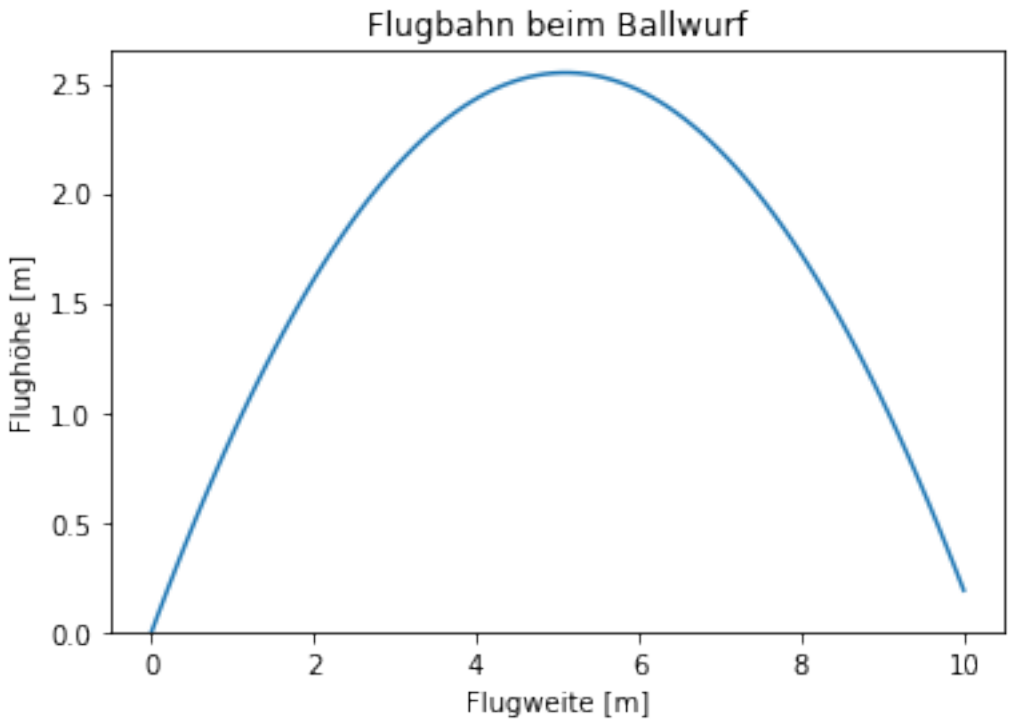
$$f(x) = x\tan(\alpha) - \frac{gx^2}{2v_0^2\cos^2(\alpha)}$$

gegeben. Hierbei ist $x \geq 0$ [m] die Flugweite, $g = 9.81$ [m/s^2] die Erdbeschleunigung, $v_0 > 0$ [m/s] die Abwurfgeschwindigkeit, $\theta \in (0, 90)$ der Abwurfwinkel in Gradmaß und $\alpha = \frac{\theta\pi}{180}$ der Abwurfwinkel in Bogenmaß. Die werfende Person befindet sich dabei im Koordinatensystem an der Stelle $(0, 0)$.

Implementieren Sie ein Programm, das zunächst die Daten θ und v_0 definiert. Stellen Sie sicher, dass die eingegebenen Daten in den entsprechenden zulässigen Bereichen liegen. Plotten Sie den Graphen der Funktion $y = f(x)$ für $x \in [0, 10]$. Verwenden Sie **keine Schleifen** zum Berechnen der Funktionswerte $f(x)$, d.h. alle Berechnungen müssen vektorisiert durchgeführt werden.

Achten Sie bei Ihrem Plot auf eine sinnvolle Achsenbeschriftung. Passen Sie zudem die Grenzen der y-Achse so an, dass die Funktionswerte nur für y-Werte im Intervall $[0, \max_{x\in[0,10]}(f(x)) + 0.1]$ angezeigt werden.

[5]:
```
import matplotlib.pyplot as plt
import numpy as np

v0 = 10
theta = 45
alpha = theta*np.pi/180
g = 9.81

# Berechne die Flugkurve fuer x aus [0,10]
x = np.linspace(start=0,stop=10, num=100)
fx = x*np.tan(alpha) - (g * x**2) / (2 * v0**2 * np.cos(alpha)**2)

# Maximale Flughoehe
max_fx = fx.max()

# Plotte die Flugkurve
plt.plot(x, fx)

# Sinnvolle Achsenbeschriftung
plt.title("Flugbahn beim Ballwurf")
plt.xlabel("Flugweite [m]")
plt.ylabel("Flughöhe [m]")

# Anpassen des dargestellten Bereichs
plt.ylim(0,max_fx+0.1)

plt.show()
```



0.1.3 NumPy: ein Textaufgaben-Generator

Wir entwerfen einen Aufgabengenerator für Lineare Gleichungssysteme (LGS) unter Verwendung des Pakets NumPy. Als Resultat soll eine Textaufgabe für ein 2×2~LGS erstellt werden, zum Beispiel wie folgt:

Dabei ist der Haupttext natürlich für Ihren Klausurentwurf, die Lösung für Sie als Kontrolle, ohne selbst nachrechnen zu müssen. Sie können sich für die folgenden Schritte auch ein beliebiges anderes Szenario ausdenken.

- Lesen Sie als Basis für den Generator die gewünschte Lösung ein (im obigen Beispiel sind dies die Ticketpreise für Kinder und Erwachsene) und erstellen Sie damit einen NumPy-Vektor.
- Erstellen Sie eine NumPy-Matrix mit zufälligen, aber für die Aufgabenstellung sinnvollen Einträgen. Im obigen Beispiel wären dies die Anzahl der Personen, also immer nichtnegative ganze Zahlen und mindestens 1 Erwachsener.
- Es kann nun passieren, dass Ihre erstellte Aufgabe nicht eindeutig lösbar ist. Prüfen Sie mit NumPy ob ihre Matrix invertierbar ist und lassen Sie solange neue Einträge der Matrix erzeugen, bis dies der Fall ist.
- Berechnen Sie die rechte Seite des LGS per Matrix-Vektor-Multiplikation und geben Sie die entsprechende Aufgabe als Textaufgabe mit Lösung aus.

Info zum Mitnehmen: Mit weiteren Abbruchbedingungen in b) kann leicht ein individuelles Mindest-Niveau für Ihre generierte Textaufgabe garantiert werden.

[7]:
```
import random
import math
import numpy as np

preis_kind = float(input("Geben Sie den Ticketpreis für ein Kind ein:"))
preis_erw = float(input("Geben Sie den Ticketpreis für einen Erwachsenen ein:"))

preis_pp = np.array([preis_kind, preis_erw]);

# Solange Matrix neu wuerfeln, bis sie invertierbar ist
while True:

    # Zufaellige Anzahl an Familienmitgliedern (ganzzahlig, mind. 1 Erwachsener)

    fam1_kind = random.randint(1,10)
    fam1_erw = random.randint(1,4)

    fam2_kind = random.randint(1,10)
    fam2_erw = random.randint(1,4)

    fam = np.array([[fam1_kind, fam1_erw], [fam2_kind, fam2_erw]]);

    # Abbruch wenn Aufgabe eindeutig loesbar und Garantie für schwierigere␣
 ↪Aufgabe.
    if(not (math.isclose(np.linalg.det(fam),0) or fam1_erw==fam2_erw or␣
 ↪fam1_kind==fam2_kind)): # hiervon ist nur die erste Bedingung notwendig. Die␣
 ↪beiden anderen garantieren eine "schwerere Aufgabe"
        break

# Berechne den Gesamtpreis an der Kinokasse
preis_ges = fam@preis_pp

# Gib die Aufgabenstellung mit Loesung aus
print ()
print ("Familie Bauer mit {} Kind(ern) und {} Erwachsenen zahlt {} Euro und".
 ↪format(fam1_kind,fam1_erw,preis_ges[0]))
print ("Familie Schneider mit {} Kind(ern) und {} Erwachsenen zahlt {} Euro an␣
 ↪der Kinokasse.".format(fam2_kind,fam2_erw,preis_ges[1]))
print ("Wie viel kostet ein Kinoticket für Kinder? Wie viel kostet ein␣
 ↪Kinoticket für Erwachsene?")
print ()
print ("(Lösung: {} Euro für Kinder, {} Euro für Erwachsene.)".
 ↪format(preis_kind,preis_erw))
```

```
Geben Sie den Ticketpreis für ein Kind ein:10
Geben Sie den Ticketpreis für einen Erwachsenen ein:20
```

```
Familie Bauer mit 2 Kind(ern) und 4 Erwachsenen zahlt 100.0 Euro und
Familie Schneider mit 3 Kind(ern) und 1 Erwachsenen zahlt 50.0 Euro an der
Kinokasse.
Wie viel kostet ein Kinoticket für Kinder? Wie viel kostet ein Kinoticket für
Erwachsene?

(Lösung: 10.0 Euro für Kinder, 20.0 Euro für Erwachsene.)
```

0.1.4 SymPy: Lagebeziehung von Geraden

Schreiben Sie ein Programm, welches die Lage von zwei Geraden im $\mathbb{R}^3$ zueinander bestimmt. Der/Die Benutzer*in soll dazu für jede Gerade jeweils zwei Punkte (x1,y1,z1) und x2,y2,z2) eingeben, durch welche die jeweilige Gerade eindeutig bestimmt ist. Sorgen Sie für adäquate Fehlermeldungen sollte dies nicht der Fall sein. Anschließend soll entschieden werden, welcher der folgenden Fälle vorliegt:

Die Geraden

- sind identisch.
- sind parallel.
- haben einen Schnittpunkt.
- sind windschief.

Das SymPy-Modul geometry könnte hilfreich sein.

[10]:
```
from sympy.geometry import Point, Line

# Punkte einlesen

while True:
        pointOneLineOne = eval(input("Bitte 1. Punkt (x,y,z) fuer Gerade 1␣
 ↪eingeben:"))
        pointTwoLineOne = eval(input("Bitte 2. Punkt (x,y,z) fuer Gerade 1␣
 ↪eingeben:"))

        # Falls die Punkte identisch sind um neue Eingabe bitten
        if pointOneLineOne == pointTwoLineOne:
            print("\nDie eingegebenen Punkte sind identisch, es kann keine \
                    eindeutige Gerade bestimmt werden. Versuchen Sie es erneut!
 ↪")
        else:
            break

while True:
        pointOneLineTwo = eval(input("Bitte 1. Punkt (x,y,z) fuer Gerade 2␣
 ↪eingeben:"))
        pointTwoLineTwo = eval(input("Bitte 2. Punkt (x,y,z) fuer Gerade 2␣
 ↪eingeben:"))
```

```
        # Falls die Punkte identisch sind um neue Eingabe bitten
        if pointOneLineTwo == pointTwoLineTwo:
            print("Die eingegebenen Punkte sind identisch, es kann keine \
                  eindeutige Gerade bestimmt werden. Versuchen Sie es erneut!")
        else:
            break


# Eingelesene Punkte in Sympy-Punkte umwandeln
p11 = Point(pointOneLineOne)
p21 = Point(pointTwoLineOne)
p12 = Point(pointOneLineTwo)
p22 = Point(pointTwoLineTwo)

# Sympy-Geraden erstellen
l1 = Line(p11, p21)
l2 = Line(p12, p22)


if l1 in l2:
    print("Die Geraden sind identisch.")
elif Line.are_concurrent(l1, l2):
    print("Die Geraden schneiden sich in einem Punkt.")
elif Line.is_parallel(l1,l2):
    print("Die Geraden sind parallel.")
else:
    print("Die Geraden sind windschief.")
```

```
Bitte 1. Punkt (x,y,z) fuer Gerade 1 eingeben:1,2,3
Bitte 2. Punkt (x,y,z) fuer Gerade 1 eingeben:4,5,6
Bitte 1. Punkt (x,y,z) fuer Gerade 2 eingeben:1,2,3
Bitte 2. Punkt (x,y,z) fuer Gerade 2 eingeben:7,8,9
Die Geraden sind identisch.
```

0.2 Impressum

0.2.1 Programmierkurs Python, Dominik Göddeke https://www.ians.uni-stuttgart.de, Universität Stuttgart

Version vom April 2023

Gefördert durch die Stiftung Innovation in der Hochschullehre.

Gefördert mit Mitteln der Deutschen Forschungsgemeinschaft (EXC 2075 - 390740016) im Rahmen der Exzellenzstrategie.

[ ]: