Institute of Mathematics and Image Computing, Lübeck

MA4030: Optimierung, Sommersemester 2025

Übungsblatt 3

Abgabe von: *Fynn-Ole Claussen, 770712 und Mika Kohlhammer, 779098 und VJohann Fischer, 779072*

A6/Code/P6.py

```python
import numpy as np
import time as t
import matplotlib.pyplot as plt

## TODO: Aufgabenteil 6a. Fehlerhaften Code debuggen
print('### 6a ###')
# Fehlerhafter Code
def summe_und_produkt(a, b):
    c1 = a + b
    c2 = a * b
    c = [c1, c2]
    return c


def summe_plus_produkt(a, b):
    c1, c2 = summe_und_produkt(a, b)
    result = c1 + c2
    return result


result = summe_plus_produkt(2, 3)
print(f'Result = {result}')


#Ein


## TODO: Aufgabenteil 6b. Primzahlen bestimmen
print('\n### 6b ###')
def prime(n):
    #Nutzen das Sieb des Erastosthenes
    primelist = []
    # BEGIN SOLUTION
    #Iteriere über alle zahlen greosser zwei und kleiner n
    for i in range (2, n):
        primcount = len(primelist)
        isprime = True

        #ueberpruefe, ob die aktuelle Zahl restlos durch alle bisher gefundenen primzahlen teilbar ist.
        for j in range(0, primcount):
            if(i % primelist[j] == 0):
                isprime = False
                break

        #Falls die Zahl nicht durch eine bisher gefundene Primzahl teilbar ist, wird sie zum Primzahlarray
```

```
        if(isprime):
            primelist.append(i)

    return primelist


    # END SOLUTION

print(f'Alle Primzahlen kleiner als  5: {prime(5)}')
print(f'Alle Primzahlen kleiner als 15: {prime(15)}')
print(f'Alle Primzahlen kleiner als 50: {prime(50)}')
print(f'Alle Primzahlen kleiner als  0: {prime(0)}')


## TODO: Aufgabenteil 6c. Annäherung der Zahl pi
print('\n### 6c ###')
# (i) Funktionen
def leibniz_mit_for_schleifen(N):
    leibniz_pi, leibniz_time = 0, 0
    start_time = t.time()
    # BEGIN SOLUTION
    for k in range(0,N+1):
        leibniz_pi = leibniz_pi + (-1)**k / (2*k + 1)

    leibniz_time = t.time() - start_time

    # END SOLUTION
    return 4 * leibniz_pi, leibniz_time


def leibniz_ohne_for_schleifen(N):
    leibniz_pi, leibniz_time = 0, 0
    # BEGIN SOLUTION
    start_time = t.time()

    #erstellen ein Array A = [1,...,N]
    i = np.arange(N)
    leibniz_arr = (-1)**i / (2*i+1)

    leibniz_pi = 4*np.sum(leibniz_arr)

    leibniz_time = t.time()-start_time
    # END SOLUTION
    return leibniz_pi, leibniz_time


def euler_mit_for_schleifen(N):
    euler_pi, euler_time = 0, 0
    # BEGIN SOLUTION
    start_time = t.time()
    for i in range(1, N+1):
        euler_pi = euler_pi + 1/(i**2)

    euler_time = t.time() - start_time
    euler_pi = (6*euler_pi)**(1/2)
    # END SOLUTION
```

```
    return euler_pi, euler_time


def euler_ohne_for_schleifen(N):
    euler_pi, euler_time = 0, 0
    # BEGIN SOLUTION
    start_time = t.time()

    #erstellen ein Array A = [1,...,N]
    i = np.arange(N)
    euler_arr = 1/((i+1)**2)

    euler_pi = (6*np.sum(euler_arr))**(1/2)

    euler_time = t.time()-start_time
    # END SOLUTION
    return euler_pi, euler_time

#Eine Methode, die wie vorgegeben die Ergebnisse der Berechnungen von Pi ausgibt
def pi_berechnungsmethoden_auswertung(value, time, method, iteration_count):
    print(f"Die verwendete Methode war: {method}")
    print(f"Die benötigte Zeit war {time} mit einer Iterationsanzahl von {iteration_count}")
    print(f"Der Fehler zum maschinengenauen Wert lag bei {abs(value - np.pi)} \n")


# (ii) Testen
N1 = int(1e2)
N2 = int(1e7)
# BEGIN SOLUTION
#Für den nachfolgenden Code ist eine Entschuldigung fällig: SORRY!


[value, time] = euler_mit_for_schleifen(N1)
pi_berechnungsmethoden_auswertung(value, time, "Euler, For", N1)

[value, time] = euler_mit_for_schleifen(N2)
pi_berechnungsmethoden_auswertung(value, time, "Euler, For", N2)

[value, time] = euler_ohne_for_schleifen(N1)
pi_berechnungsmethoden_auswertung(value, time, "Euler, Ohne-For", N1)

[value, time] = euler_ohne_for_schleifen(N2)
pi_berechnungsmethoden_auswertung(value, time, "Euler, Ohne-For", N2)

[value, time] = leibniz_mit_for_schleifen(N1)
pi_berechnungsmethoden_auswertung(value, time, "Leibniz, For", N1)

[value, time] = leibniz_mit_for_schleifen(N2)
pi_berechnungsmethoden_auswertung(value, time, "Leibniz, For", N2)

[value, time] = leibniz_ohne_for_schleifen(N1)
pi_berechnungsmethoden_auswertung(value, time, "Leibniz, Ohne-For", N1)

[value, time] = leibniz_ohne_for_schleifen(N2)
pi_berechnungsmethoden_auswertung(value, time, "Leibniz, Ohne-For", N2)
```

```
print("Die Varianten, die keine for-Schleife in Python nutzen, sind deutlich schneller, da hier eine inter


# END SOLUTION

# (iii) Plots
# BEGIN SOLUTION
N = [5,10, 50, 100, 500, 1000]
approx_euler = []
approx_leibniz = []



#Iteriere über die Liste N und speichere die Approximationen der jeweiligen Methoden ab
for i in range(len(N)):
    [value, time] = euler_ohne_for_schleifen(N[i])
    approx_euler.append(abs(value - np.pi))

    [value, time] = leibniz_ohne_for_schleifen(N[i])
    approx_leibniz.append(abs(value - np.pi))


#Plotte die Approximationen
plt.plot(N, approx_euler)
plt.plot(N, approx_leibniz)
plt.title("Approximationsfehler von Euler und Leibniz für Pi")
plt.xlabel("Anzahl Summanden")
plt.ylabel("Approximation")
plt.show()

print("Der Plot zeigt, dass beide Varianten gleich schnell gute Ergebnisse liefern. Lässt man sich den Feh

# END SOLUTION

#TERMINAL OUTPUT:
```

6a

Result = 11

6b

Alle Primzahlen kleiner als 5: [2, 3]

Alle Primzahlen kleiner als 15: [2, 3, 5, 7, 11, 13]

Alle Primzahlen kleiner als 50: [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]

Alle Primzahlen kleiner als 0: []

6c

Die verwendete Methode war: Euler, For

Die benötigte Zeit war 1.049041748046875e-05 mit einer Iterationsanzahl von 100

Der Fehler zum maschinengenauen Wert lag bei 0.009516121780687836

Die verwendete Methode war: Euler, For

Die benötigte Zeit war 0.7834091186523438 mit einer Iterationsanzahl von 10000000

Der Fehler zum maschinengenauen Wert lag bei 9.549389057283975e-08

Die verwendete Methode war: Euler, Ohne-For

Die benötigte Zeit war 0.00010228157043457031 mit einer Iterationsanzahl von 100

Der Fehler zum maschinengenauen Wert lag bei 0.009516121780686948

Die verwendete Methode war: Euler, Ohne-For

Die benötigte Zeit war 0.0724942684173584 mit einer Iterationsanzahl von 10000000

Der Fehler zum maschinengenauen Wert lag bei 9.549296065003432e-08

Die verwendete Methode war: Leibniz, For

Die benötigte Zeit war 2.6941299438476562e-05 mit einer Iterationsanzahl von 100

Der Fehler zum maschinengenauen Wert lag bei 0.009900747481198291

Die verwendete Methode war: Leibniz, For

Die benötigte Zeit war 2.732234001159668 mit einer Iterationsanzahl von 10000000

Der Fehler zum maschinengenauen Wert lag bei 9.999998829002266e-08

Die verwendete Methode war: Leibniz, Ohne-For

Die benötigte Zeit war 8.654594421386719e-05 mit einer Iterationsanzahl von 100

Der Fehler zum maschinengenauen Wert lag bei 0.009999750031241206

Die verwendete Methode war: Leibniz, Ohne-For

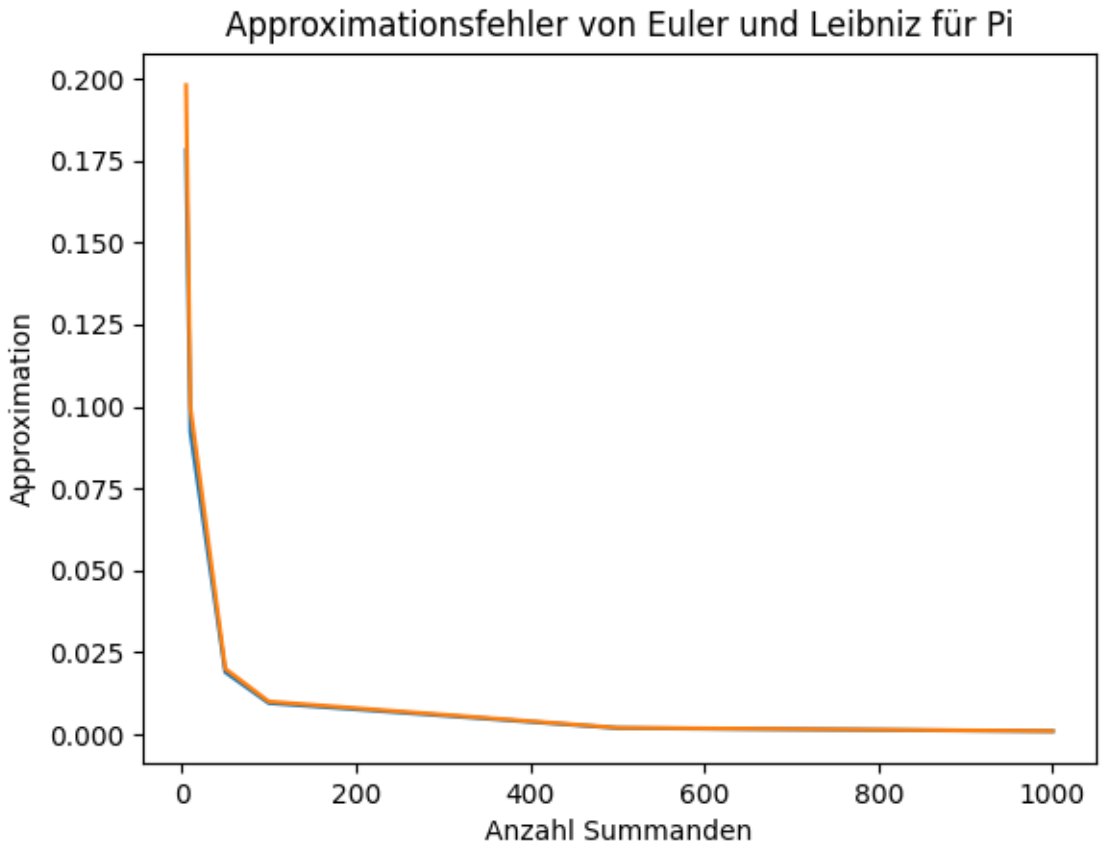
Die benötigte Zeit war 0.5972936153411865 mit einer Iterationsanzahl von 10000000

Der Fehler zum maschinengenauen Wert lag bei 9.999999539545001e-08

Die Varianten, die keine for-Schleife in Python nutzen, sind deutlich schneller, da hier eine interne Schleife genutzt wird, die in c geschrieben ist

Der Plot zeigt, dass beide Varianten gleich schnell gute Ergebnisse liefern. Lässt man sich den Fehler in der Konsole ausgeben,sieht man, dass die Variante von Euler minimal genauer ist.

```
#PLOTS:
```


Approximationsfehler von Euler und Leibniz für Pi
Approximation
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0
200
400
600
800
1000
Anzahl Summanden

A6/Code/P7.py

```python
import numpy as np
import matplotlib.pyplot as plt

## TODO: Aufgabenteil 7a. 1D-Funktion plotten und Minimierer rot markieren
x = np.linspace(-10, 12, 500)
f = lambda x: (x - 1)**2 - 0.5
# Hinweis: x kann hier sowohl als einzelner Wert als auch als ein ganzes Array übergeben werden!

# Plotten und markieren
# BEGIN SOLUTION
# Plotte f(x) = (x - 1)² - 0.5
plt.plot(x, f(x), 'b-', label='f(x)')
# Plotten des Minimums
x_min = 1
y_min = (x_min - 1)**2 - 0.5
min_label = f'Minimum: f({x_min}) = {y_min}'
plt.scatter(x_min, y_min, color='red', label=min_label)
# Kosmetische Anpassungen
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('f(x) = (x - 1)² - 0.5')
plt.legend()
plt.grid(True)
plt.show()
# END SOLUTION


## TODO: Aufgabenteil 7b. Quadratische Funktion
print('### 7b ###')
def quadratic_function(x, A, b, c):
    """
    Quadratische Funktion
    Input:
        x: Variable, shape (n,)  | Zusatz: shape (n,) oder shape (m,n)
        A: Matrix, shape (n, n)
        b: Vektor, shape (n,)
        c: Skalar.

    Output:
        val: Skalar (g(x)).
    """
    # BEGIN SOLUTION
    return 0.5*np.dot(x.T, np.dot(A, x)) - np.dot(b.T, x) + c
    # END SOLUTION


# Test
A = np.array([[2, 5], [1, 4]])
b = np.array([1, 4]) # Im Blatt steht (1,4)
c = 5
x = np.array([1, 2])

result = quadratic_function(x, A, b, c)
```

```
print(f'Result quadratic function = {result}')


## TODO: Aufgabenteil 7c. Implementierung spezifischer quadratischer Funktionen
print('\n### 7c ###')
# BEGIN SOLUTION
# Erstelle Matrizen und Vektoren
A1 =  np.array([[2,0],[0,1]])
A2 =  np.array([[-2,0],[0,1]])
A3 =  np.array([[-2,1],[1,-1]])
b2 = np.array([1,3])
c2 = -1.5
x2 = np.array([1, 1])

# Erstelle die g1, g2, g3 als lambda-Funktion
g1 = lambda x: quadratic_function(x, A1, b2, c2)
g2 = lambda x: quadratic_function(x, A2, b2, c2)
g3 = lambda x: quadratic_function(x, A3, b2, c2)

# Teste die Implementierung
print(f'g1({x2}) = {g1(x2)} \n'
      f'g2({x2}) = {g2(x2)} \n'
      f'g3({x2}) = {g3(x2)}')
# Ausgabe nicht ganz sauber, x müsste in der Darstellung ein Spaltenvektor sein :D
# END SOLUTION


## TODO: Aufgabenteil 7d. Visualisierung quadratischer 2D-Funktionen
# 2D-Gitter-Punkte anlegen
# BEGIN SOLUTION
x_werte = np.linspace(-10, 10, 1000)
y_werte = np.linspace(-10, 10, 1000)
xx, yy = np.meshgrid(x_werte, y_werte)
# END SOLUTION

# zugehörige Funktionswerte bestimmen
# BEGIN SOLUTION
# Geht bestimmt einfacher:
# werte_funktion_aus wertet für eine gegebene Funktion g Stellenweise die Funktion g aus
def werte_funktion_aus(funktion):
    bestimmte_werte = np.zeros_like(xx)
    # doppelte Indizierung weil 2d
    for i in range(xx.shape[0]):
        for j in range(xx.shape[1]):
            # setze bestimmten Punkt in Funktion ein
            bestimmte_werte[i, j] = funktion(np.array([xx[i, j], yy[i, j]]))
    return bestimmte_werte

g1_werte = werte_funktion_aus(g1)
g2_werte = werte_funktion_aus(g2)
g3_werte = werte_funktion_aus(g3)
# END SOLUTION

# Visualisirung als Höhenlinien
fig, ax = plt.subplots(1, 3)
fig.suptitle('Visualisierung als Höhelinien')
```

```
fig.set_size_inches(10, 4)
fig.tight_layout(pad=3)
# BEGIN SOLUTION
# Plotten der Funktionen als Contour Plot
ax[0].contourf(xx,yy,g1_werte)
ax[1].contourf(xx,yy,g2_werte)
ax[2].contourf(xx,yy,g3_werte)

# Kosmetische Anpassungen
funktionen = ['g1', 'g2', 'g3']
for i in range(3):
    ax[i].set_title(funktionen[i])
    ax[i].set_xlabel("x")
    ax[i].set_ylabel("y")
# END SOLUTION
plt.show()

# Visualisierung als Surface-Plot
fig = plt.figure()
fig.suptitle('Visualisierung als Surface-Plot')
fig.set_size_inches(12, 4)
fig.tight_layout(pad=1.5)
# BEGIN SOLUTION
# Erstellen 3 Subplots mit jeweils 3 Achsen in 3d
ax1 = fig.add_subplot(1, 3, 1, projection='3d')
ax2 = fig.add_subplot(1, 3, 2, projection='3d')
ax3 = fig.add_subplot(1, 3, 3, projection='3d')

# Plotten der Funktionen
ax1.plot_surface(xx, yy, g1_werte, cmap='viridis')
ax1.set_title("g1")
ax1.set_xlabel("x")
ax1.set_ylabel("y")
ax1.set_zlabel("g1(x)")

ax2.plot_surface(xx, yy, g2_werte, cmap='viridis')
ax2.set_title("g2")
ax2.set_xlabel("x")
ax2.set_ylabel("y")
ax2.set_zlabel("g2(x)")

ax3.plot_surface(xx, yy, g3_werte, cmap='viridis')
ax3.set_title("g3")
ax3.set_xlabel("x")
ax3.set_ylabel("y")
ax3.set_zlabel("g3(x)")
# END SOLUTION
plt.show()
```

```
#TERMINAL OUTPUT:
```

7b

Result quadratic function = 11.0

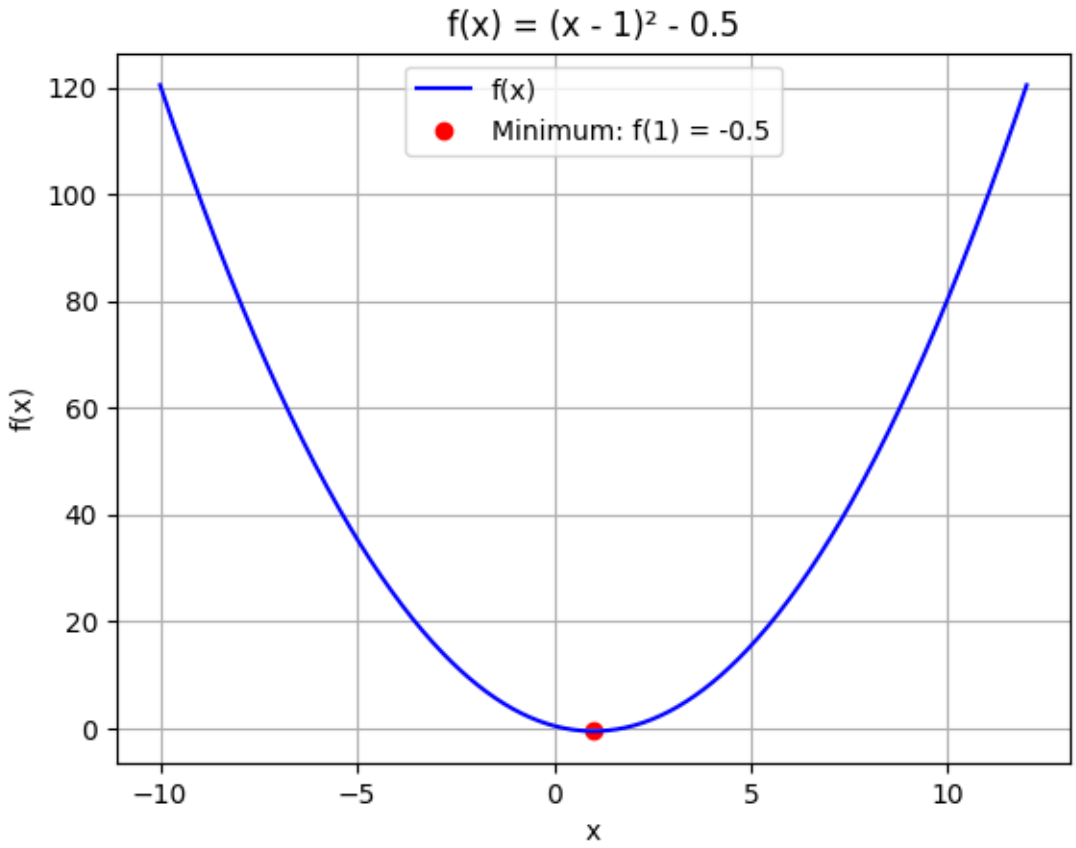
7c

g1([1 1]) = -4.0

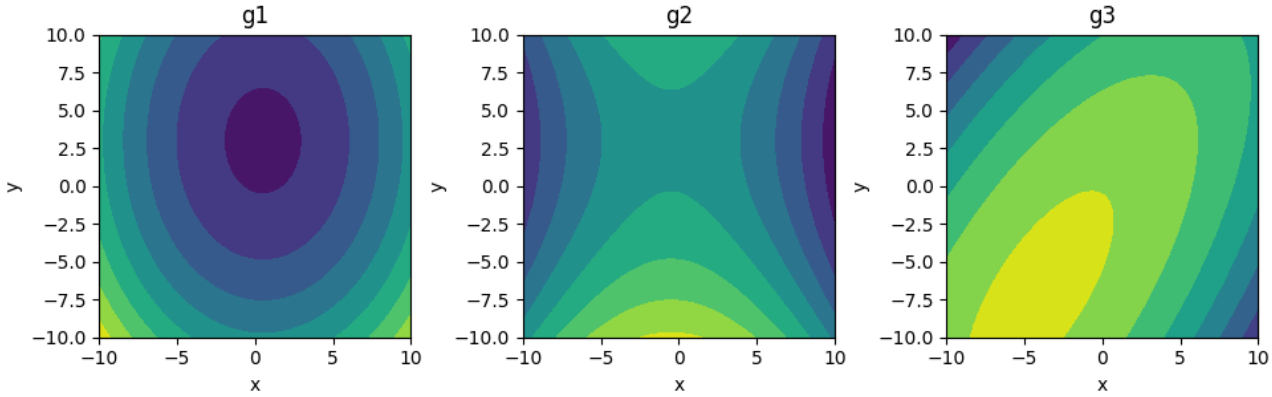
g2([1 1]) = -6.0

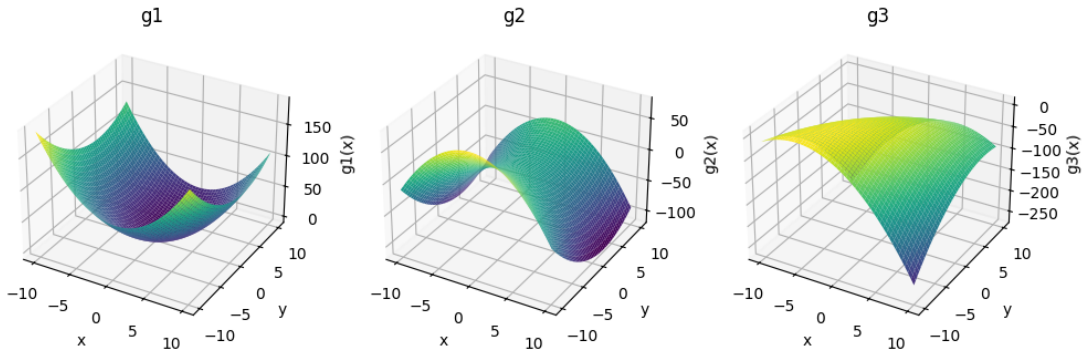
g3([1 1]) = -6.0

#PLOTS:



Visualisierung als Höhelinien




Visualisierung als Surface-Plot
g1
g2
g3
g1(x)
g2(x)
g3(x)
x
y