#Scalarer Optimierer mit Brent-Algo. res=optimize.minimize_scalar(f_1) print(res)> Gibt gleiches Ergebniss aus
#Minimierung mithilfe von "Brent" minimizer_brent_f_1 = optimize.brent(f_1, brack=(-10,10),full_output=True) #Brack: If bracket consists of two numbers (a,c) then they are assumed to be a starting interval for a downhill bracket search minimizer_brent_f_2 = optimize.brent(f_2, brack=(-10,10),full_output=True) #Brack legt Start Bereich zwischen (a,c) der Suche fest #Output : (a,b,c,d) mit a=xmin , b=f(xmin) , c=Anzahl der Iterationen , d=Anzahl der Funktionsbewertungen
print(f'Minimierung der Funktion f_1(x) mithilfe von "Brent"-Algrithmus: Optimum bei x={minimizer_brent_f_1[0]} und dem Funktionswert f(x)={minimizer_brent_f_1[1]}' ) print(f'Minimierung der Funktion f_2(x) mithilfe von "Brent"-Algrithmus: Optimum bei x={minimizer_brent_f_2[0]} und dem Funktionswert f(x)={minimizer_brent_f_2[1]}' )  #Minimierung mithilfe von "BFGS" minimizer_BFGS_f_1= optimize.minimize(f_1,x_0,method='BFGS') #Startwert x_0
minimizer_BFGS_f_2= optimize.minimize(f_2,x_0,method='BFGS') #Startwert x_0 #Ausgabe f_1 Ausgabe_BFGS_f_1=minimizer_BFGS_f_1 Ausgabe_BFGS_f_1_x_wert= str(Ausgabe_BFGS_f_1.x).strip('[]') Ausgabe_BFGS_f_1_f_werte=Ausgabe_BFGS_f_1.fun
#Ausgabe f_2 Ausgabe_BFGS_f_2=minimizer_BFGS_f_2 Ausgabe_BFGS_f_2_x_wert= str(Ausgabe_BFGS_f_2.x).strip('[]') Ausgabe_BFGS_f_2_f_werte=Ausgabe_BFGS_f_2.fun  print(f'Minimierung der Funktion f_1(x) mithilfe von "BFGS"-Algrithmus: Optimum bei x={Ausgabe_BFGS_f_1_x_wert} und dem Funktionswert f(x)={Ausgabe_BFGS_f_1.fun}')
print(f'Minimierung der Funktion f_2(x) mithilfe von "BFGS"-Algrithmus: Optimum beix={Ausgabe_BFGS_f_2_x_wert} und dem Funktionswert f(x)={Ausgabe_BFGS_f_2.fun}')  Erklärungsabschnitt zum eigenen Verständnis über BFGS:  BFGS nutzt 2.Ableitung, um Optimum zu finden> Mithilfe von Inverser Hesse-Matrix> (n x n) - Näherungen gespeichert mit n=Anzahl Variablen , Line-Search in bestimmte Richtung
#Minimierung mithilfe von "L-BFGS-B" minimizer_L_BFGS_B_f_1= optimize.minimize(f_1,x_0,method='L-BFGS-B') #Startwert x_0 minimizer_L_BFGS_B_f_2= optimize.minimize(f_2,x_0,method='L-BFGS-B') #Startwert x_0 #Ausgabe Ausgabe_L_BFGS_f_1=minimizer_L_BFGS_B_f_1
Ausgabe_L_BFGS_f_1_x_wert= str(Ausgabe_L_BFGS_f_1.x).strip('[]') Ausgabe_L_BFGS_f_2=minimizer_L_BFGS_B_f_2 Ausgabe_L_BFGS_f_2_x_wert= str(Ausgabe_L_BFGS_f_2.x).strip('[]')  print(f'Minimierung der Funktion f_1(x) mithilfe von "L-BFGS-B"-Algrithmus: Optimum bei x={Ausgabe_L_BFGS_f_1_x_wert} und dem Funktionswert f(x)={Ausgabe_L_BFGS_f_1.fun}')
print(f'Minimierung der Funktion f_2(x) mithilfe von "L-BFGS-B"-Algrithmus: Optimum bei x={Ausgabe_L_BFGS_f_2_x_wert} und dem Funktionswert f(x)={Ausgabe_L_BFGS_f_2.fun}' ) #Wert unterschiedlich ?  #Erklärungsabschnitt zum eigenen Verständnis über L-BFGS-B:
#Limited-memory BFGS: Analog zu BFGS> Mithilfe von Inverser Hesse-Matrix> Nur wenige Vektoren gespeichert> Linearer Speicher  Minimierung der Funktion f_1(x) mithilfe von "Brent"-Algrithmus: Optimum bei x=-5.0 und dem Funktionswert f(x)=0.0  Minimierung der Funktion f_2(x) mithilfe von "Brent"-Algrithmus: Optimum bei x=-2.5209409400751377 und dem Funktionswert f(x)=-4.766829111587686  Minimierung der Funktion f_1(x) mithilfe von "BFGS"-Algrithmus: Optimum bei x=-5.00000003 und dem Funktionswert f(x)=6.914540208852864e-16  Minimierung der Funktion f_2(x) mithilfe von "BFGS"-Algrithmus: Optimum beix=-2.52094096 und dem Funktionswert f(x)=-4.766829111587682
Minimierung der Funktion f_1(x) mithilfe von "L-BFGS-B"-Algrithmus: Optimum bei x=-4.99999954 und dem Funktionswert f(x)=2.0772310584811207e-13 Minimierung der Funktion f_2(x) mithilfe von "L-BFGS-B"-Algrithmus: Optimum bei x=-4.99449928 und dem Funktionswert f(x)=-0.9579138626591672  Erklärung
Brent-Algorithmus ist zur Bestimmung der Nullstelle, mithilfe einer Bisektion (Iterationssverfahren).  BFGS ist ein Abstiegsverfahren, welches die 2.Ableitungen nutzt, um ein Optimum zu finden. Dabei wird die inverse Hesse-Matrix gelöst, um eine initalen Start zu ermöglichen. \$\$ L-BFGS-B ist ein Speicher limitierder Algorithmus basierend auf BFGS, sodass dieser ein unterschiedliches Ergebniss produzieren kann. So ist die Minimierung von Funktion f_1(x) fast gleich, jedoch nicht für Funktion f_2(x).
<pre>In [6]: from matplotlib import ticker  #Werte für BFGS und L_BFGS-B als int abspeichern (dabei leider gerundet):     x_BFGS_f_1_round= int(float(Ausgabe_BFGS_f_1_x_wert))     f_BFGS_f_1_round=int(float(Ausgabe_L_BFGS_f_1.fun))</pre>
<pre>x_BFGS_f_2_round= int(float(Ausgabe_BFGS_f_2_x_wert)) f_BFGS_f_2_round=int(float(Ausgabe_BFGS_f_2.fun))  x_L_BFGS_f_1_round= int(float(Ausgabe_L_BFGS_f_1_x_wert)) f_L_BFGS_f_1_round=int(float(Ausgabe_L_BFGS_f_1.fun))  1.1</pre>
<pre>x_L_BFGS_f_2_round= int(float(Ausgabe_L_BFGS_f_2_x_wert)) f_L_BFGS_f_2_round=int(float(Ausgabe_L_BFGS_f_2.fun))  fig, (ax1,ax2) = plt.subplots(2,1, layout= "constrained") #Graph für Funktion f_1(x)</pre>
<pre>ax1.set_title("Funktion f\$_{1}\$(x)") ax1.set_xlabel("x") ax1.set_ylabel("f(x)") #setup(ax1, title="StrMethodFormatter('{x:.3f}')")</pre>
ax1.xaxis.set_major_formatter(ticker.StrMethodFormatter("{x:.4f}")) ax1.plot(x, f_1(x), label=r"\$(x+5)^{2}\$"); ax1.plot(minimizer_brent_f_1[0], minimizer_brent_f_1[1], "o", label=r"\$Brent\$"); ax1.plot(x_BFGS_f_1_round, f_BFGS_f_1_round, "o", label=r"\$BFGS\$")  ax1.plot(x_L_BFGS_f_1_round, f_L_BFGS_f_1_round, "o", label=r"\$L-BFGS-B\$")  ax1.plot(x_L_BFGS_f_1_round, f_L_BFGS_f_1_round, "o", label=r"\$L-BFGS-B\$")  #Minimum des L-BFGS-B-Algorithmus ax1.legend()
#Graph für Funktion f_2(x) ax2.set_title("Funktion f\$_{2}\$(x)") ax2.set_xlabel("x") ax2.set_ylabel("f(x)") ax2.plot(x, f_2(x), label=r"\$(x+3)^{2} - 5\cos(5x)\$");
<pre>ax2.plot(minimizer_brent_f_2[0], minimizer_brent_f_2[1], "o", label=r"\$Brent\$"); #Minimum des Brent-Algorithmus ax2.plot(x_BFGS_f_2_round, f_BFGS_f_2_round, "o" , label=r"\$BFGS\$") #Minimum des BFGS-Algorithmus ax2.plot(x_L_BFGS_f_2_round, f_L_BFGS_f_2_round, "o" , label=r"\$L-BFGS-B\$") #Minimum des L-BFGS-B-Algorithmus ax2.legend() plt.show()</pre>
Funktion $f_1(x)$ $ 200 - (x+5)^2 $ Brent  BFGS  BFGS
E 100 - L−BFGS−B  50 - 0 - 10
$-10.0000 -7.5000 -5.0000 -2.5000 0.0000 2.5000 5.0000 7.5000 10.0000$ $x$ Funktion $f_2(x)$ $- (x+3)^2 - 5\cos(5x)$ Proof:
100 - BFGS • L-BFGS-B
$\begin{array}{c ccccccccccccccccccccccccccccccccccc$
Aufgabe 2.2  Informationen x = x-Wert , Start x_start s = Schrittweite , Start s_start k = Iterationszahl (iters) theta = Faktor zwischen (0,1)> s(k+1)=theta * s(k) : d = Richtung mit : Es existiert mindestens ein d aus D D = Einheitsvektoren (positiv/negativ) für n Dimensionen
<pre>In [7]: 'Funktionen definieren'  def f_a(x):     return (x+5)**2  def f_b(x):</pre>
<pre>def f_b(x):     return (x+3)**2 - 5*np.cos(5*x)  def f_c(x,y):     return x**2 + y**2  def f_d(x,y):</pre>
return x*np.sin(x) + 3*(y**2)  In [8]: 'Definiert Kompasssuche'  def kompasssuche(f,x0,s0,theta,iters,showit=False,getpath=False): #Alle Parameter wie in Aufgabe gewünscht mit 2 zusätzlichen optionalen Parametern,  dim=len(signature(f).parameters) #showit um Anzahl der Iterationen zu printen, und getpath um ein Array des Weges zurückzugeben
<pre>d=np.concatenate((np.identity(dim), np.negative(np.identity(dim)))) if dim==1:</pre>
<pre>f_current=f(x0 + s0 * d0) if f_current&lt; f_old:     x0=x0+s0*d0     path.append([x0, s0, f_current])     break if d0 == d[-1]:</pre>
<pre>s0=theta*s0 if showit:     print("Iterationen: " + str(len(path)-1)+ ", \t Abstand zum Optimum: ",end = " ") if getpath:     return path else:</pre>
return x0  else:     path=[[x0,s0,f(x0[0],x0[1])]]     for k in range(0,iters):         f_old=f(x0[0],x0[1])         for do in d:
<pre>x1=x0 + s0 * d0 f_current=f(x1[0],x1[1]) if f_current &lt; f_old:</pre>
<pre>if all(d0 == d[-1]):</pre>
return path else: return x0  In [9]: 'Betrachte alle möglichen Kombinationen aller Parameter' print(-5-kompasssuche(f_a,3,0.5,0.3,20,True))
<pre>print(-5-kompasssuche(f_a,9,0.5,0.3,20,True)) print(-5-kompasssuche(f_a,3,4,0.3,20,True)) print(-5-kompasssuche(f_a,9,4,0.3,20,True)) print(-5-kompasssuche(f_a,3,0.5,0.8,20,True)) print(-5-kompasssuche(f_a,3,0.5,0.8,20,True))</pre>
print(-5-kompasssuche(f_a,3,4,0.8,20,True)) print(-5-kompasssuche(f_a,9,4,0.8,20,True)) print("Wähle für f_a die Parameter 3, 4, 0.3 für jeweils x0, s0 und theta \n") #Auswahl aufgrund vom Abstand zum Optimum und Iterationsanzahl  print(2.521-kompasssuche(f_b,3,0.5,0.3,20,True)) print(2.521-kompasssuche(f_b,9,0.5,0.3,20,True))
<pre>print(2.521-kompasssuche(f_b,3,4,0.3,20,True)) print(2.521-kompasssuche(f_b,9,4,0.3,20,True)) print(2.521-kompasssuche(f_b,3,0.5,0.8,20,True)) print(2.521-kompasssuche(f_b,9,0.5,0.8,20,True)) print(2.521-kompasssuche(f_b,9,0.5,0.8,20,True)) print(2.521-kompasssuche(f_b,3,4,0.8,20,True)) print(2.521-kompasssuche(f_b,9,4,0.8,20,True))</pre>
print("Wähle für f_b die Parameter 3, 0.5, 0.8 für jeweils x0, s0 und theta \n")  print(np.linalg.norm(kompasssuche(f_c,[3,3],0.5,0.3,20,True)))  print(np.linalg.norm(kompasssuche(f_c,[9,9],0.5,0.3,20,True)))  print(np.linalg.norm(kompasssuche(f_c,[3,3],4,0.3,20,True)))  print(np.linalg.norm(kompasssuche(f_c,[9,9],4,0.3,20,True)))  print(np.linalg.norm(kompasssuche(f_c,[9,9],4,0.3,20,True)))
print(np.linalg.norm(kompasssuche(f_c,[3,3],0.5,0.8,20,True))) print(np.linalg.norm(kompasssuche(f_c,[9,9],0.5,0.8,20,True))) print(np.linalg.norm(kompasssuche(f_c,[3,3],4,0.8,20,True))) print(np.linalg.norm(kompasssuche(f_c,[9,9],4,0.8,20,True))) print("Wähle für f_c die Parameter (3,3), 0.5, 0.3 für jeweils x0, s0 und theta \n")
<pre>print(np.linalg.norm(kompasssuche(f_d,[3,3],0.5,0.3,20,True)-(4.71,0))) print(np.linalg.norm(kompasssuche(f_d,[9,9],0.5,0.3,20,True)-(4.71,0))) print(np.linalg.norm(kompasssuche(f_d,[3,3],4,0.3,20,True)-(4.71,0))) print(np.linalg.norm(kompasssuche(f_d,[9,9],4,0.3,20,True)-(4.71,0))) print(np.linalg.norm(kompasssuche(f_d,[3,3],0.5,0.8,20,True)-(4.71,0)))</pre>
print(np.linalg.norm(kompasssuche(f_d,[9,9],0.5,0.8,20,True)-(4.71,0))) print(np.linalg.norm(kompasssuche(f_d,[3,3],4,0.8,20,True)-(4.71,0))) print(np.linalg.norm(kompasssuche(f_d,[9,9],4,0.8,20,True)-(4.71,0))) print("Wähle für f_d die Parameter (3,3), 0.5, 0.8 für jeweils x0, s0 und theta \n")  Iterationen: 16, Abstand zum Optimum: [0.] Iterationen: 20, Abstand zum Optimum: [-4.]
Iterationen: 2, Abstand zum Optimum: [0.] Iterationen: 11, Abstand zum Optimum: [-6.79999994e-08] Iterationen: 16, Abstand zum Optimum: [0.] Iterationen: 20, Abstand zum Optimum: [-4.] Iterationen: 2, Abstand zum Optimum: [0.] Iterationen: 10, Abstand zum Optimum: [0.]
Wähle für f_a die Parameter 3, 4, 0.3 für jeweils x0, s0 und theta  Iterationen: 10, Abstand zum Optimum: [0.097487]  Iterationen: 13, Abstand zum Optimum: [-2.3690995]  Iterationen: 10, Abstand zum Optimum: [5.04195037]
Iterationen: 11, Abstand zum Optimum: [5.04197655] Iterationen: 7, Abstand zum Optimum: [0.08428534] Iterationen: 12, Abstand zum Optimum: [-2.4118256] Iterationen: 4, Abstand zum Optimum: [6.27172] Iterationen: 5, Abstand zum Optimum: [6.25828] Wähle für f_b die Parameter 3, 0.5, 0.8 für jeweils x0, s0 und theta
Iterationen: 12, Abstand zum Optimum: 0.0 Iterationen: 20, Abstand zum Optimum: 8.0 Iterationen: 14, Abstand zum Optimum: 0.004355777772109083 Iterationen: 15, Abstand zum Optimum: 0.013165348457219006 Iterationen: 12, Abstand zum Optimum: 0.0
Iterationen: 20, Abstand zum Optimum: 8.0 Iterationen: 10, Abstand zum Optimum: 0.23841390591264072 Iterationen: 10, Abstand zum Optimum: 0.36898619408256955 Wähle für f_c die Parameter (3,3), 0.5, 0.3 für jeweils x0, s0 und theta Iterationen: 15, Abstand zum Optimum: 0.2037649999999975
Iterationen: 20,       Abstand zum Optimum: 6.368995211177349         Iterationen: 14,       Abstand zum Optimum: 0.20225945242682733         Iterationen: 14,       Abstand zum Optimum: 0.20176406665211793         Iterationen: 13,       Abstand zum Optimum: 0.1523743999999935         Iterationen: 20,       Abstand zum Optimum: 6.368995211177349         Iterationen: 11,       Abstand zum Optimum: 0.548568868323073
Iterationen: 8, Abstand zum Optimum: 0.3088662746328251 Wähle für f_d die Parameter (3,3), 0.5, 0.8 für jeweils x0, s0 und theta  In [11]: 'Plotten aller Funktionen mit dem jeweiligen Weg der ausgewählten Kompasssuche'
<pre>x=np.linspace(-10,10) y=np.linspace(-10,10)  fig, (ax1,ax2) = plt.subplots(2,1, layout= "constrained") #Graph für Funktion f_a(x)</pre>
<pre>ax1.set_title("Funktion f\$_{a}\$(x)") ax1.set_xlabel("x") ax1.set_ylabel("f_a(x)") #setup(ax1, title="StrMethodFormatter('{x:.3f}')") ax1.xaxis.set_major_formatter(ticker.StrMethodFormatter("{x:.4f}"))</pre>
<pre>ax1.plot(x, f_a(x), label=r"\$(x+5)^{2}\$"); path_a=kompasssuche(f_a,3,4,0.3,20,False,True) k=0 print(path_a) for j in path_a:</pre>
<pre>ax1.plot(j[0],j[2],"0"); ax1.text(j[0],j[2]+10,"x{}".format(k))     k=k+1 ax1.legend()  #Graph für Funktion f_2(x)</pre>
<pre>x=np.linspace(2.2,3) ax2.set_title("Funktion f\$_{b}\$(x)") ax2.set_xlabel("x") ax2.set_ylabel("f_b(x)") ax2.set_ylabel("f_b(x)") ax2.plot(x, f_b(x), label=r"\$(x+3)^{2} - 5\cos(5x)\$"); path_b=kompasssuche(f_b,3,0.5,0.8,20,False,True)</pre>
<pre>for j in path_b:     ax2.plot(j[0],j[2],"o");     ax2.text(j[0],j[2]+10,"x{}".format(k))     k=k+1 ax2.legend()</pre>
<pre># Gitter zum Auswerten der Funktion erzeugen x = np.linspace(-5, 5, 101) y = np.linspace(-5, 5, 101) X, Y = np.meshgrid(x, y) Z = f_c(X,Y)</pre>
# 3D-Plot erzeugen: Out[87]: In [88]: In [89]:  fig = plt.figure()  ax3 = plt.axes(projection='3d')  ax3.set_title("Funktion f\$_{c}\$(x)")  ax3.plot_surface(X, Y, Z)  path_c=kompasssuche(f_c,[3,3],0.5,0.3,20,False,True)
<pre>k=0 for j in path_c:     ax3.plot(j[0][0],j[0][1],j[2],"o");     ax3.text(j[0][0],j[0][1],j[2]+10,"{}".format(k))     k=k+1</pre>
<pre># Gitter zum Auswerten der Funktion erzeugen x = np.linspace(-6, 6, 101) y = np.linspace(-5, 5, 101) X, Y = np.meshgrid(x, y) Z = f_d(X,Y) # 3D-Plot erzeugen: Out[87]: In [88]: In [89]:</pre>
# 3D-Plot erzeugen: Out[87]: In [88]: In [89]:  fig = plt.figure()  ax4 = plt.axes(projection='3d')  ax4.set_title("Funktion f\$_{d}\$(x)")  ax4.plot_surface(X, Y, Z)  path_d=kompasssuche(f_d,[3,3],0.5,0.8,20,False,True)  k=0
<pre>for j in path_d:     ax4.plot(j[0][0],j[0][1],j[2],"o");     ax4.text(j[0][0],j[0][1],j[2]+10,"{}".format(k))     k=k+1  plt.show()</pre>
plt.show()  [[3, 4, 64], [array([-1.]), 4, array([16.])], [array([-5.]), 4, array([0.])]]  Funktion f <sub>a</sub> (x)  200 - (x+5) <sup>2</sup>
-10.0000 -7.5000 -2.5000 0.0000 2.5000 5.0000 7.5000 10.0000 x x0
Funktion $f_b(x)$ $ \underbrace{\begin{array}{c} 40 \\$
35 - XZX4X6XX5 X3X1 25 - 2.2 2.3 2.4 2.5 2.6 2.7 2.8 2.9 3.0
Funktion $f_c(x)$
50 40
30 20 10 0
Funktion $f_d(x)$
60 40 20
$\begin{array}{cccccccccccccccccccccccccccccccccccc$

Blatt 02 - Praktische Optimierung - Adrian Lentz, Robert

\newline

\newline

Aufgabe 2.1

Lösungen und Erklärungen für Blatt 02.

Adrian Lentz - Matrikelnummer: 258882

In [3]: #Pakete importieren

In [4]: x=np.linspace(-10,10)

def f\_1(x):
 return (x+5)\*\*2

x\_0=0

Robert Schönewald - Matrikelnummer: 188252

import numpy as np
import matplotlib.pyplot as plt
from scipy import optimize
from inspect import signature
import plotly.graph\_objects as go

#Startwert

#Definiert Funktion  $f_1(x)$ 

#Definiert Funktion  $f_2(x)$ 

Interpretation der Ergebnisse:

In der ersten Funktion wurden Parameter zufälligerweise sehr passend gewählt, sodass die Kompasssuche schnell vorbei war. Die zweite Funktion war deutlich schwieriger zu optimieren, da der gesuchte Wert nicht rational war. Im Weg der dritten Funktion sieht man, welche Achse zuerst betrachtet wurde. Nachdem auf dieser jedoch keine Verbesserung in positiver Richtung möglich war, wurde auf der nächsten Achse weitergesucht, bis schließlich das Optimum gefunden wurde. In der vierten Funktion sieht man sehr ähnliches Verhalten, es gibt wieder einen "Knick" in eine andere Richtung, nachdem dies nötig wurde.

2.2

Index der Kommentare

1 Punkt Abzug

- 1.1 Sehr gelungener Plot! Für die jeweiligen Methoden hätte noch ein anderer Marker gewählt werden können. Diese hätten sich dann "besser überlappt".

  Allerdings rundet int() nicht, sondern schneidet ab. Daher ist der x-Wert nicht mehr korrekt für den Plot. Runden ist hier nicht notwendig.
- 1 Punkt Abzug1.2 Interpretation der Ergebnisse von a) und b) fehlt.
- 1 Punkt Abzug

  1.3 Wenn man sich aus der letzten Iteration den alten Wert merkt, dann kann man diesen Funktionsaufruf sparen. Dafür muss vor Beginn der Iterationen einmal ausgewertet werden.
- Die Anzahl der Iterationen ist fehlerhaft. Wenn kein besserer Punkt gefunden wird, dann wird path nicht ergänzt, obwohl k inkrementiert wird.
- 2.1 Erste Funktion konvex, daher recht einfach. Problem ist wenn zu viele Iterationen gemacht werden.
- 2.2 Zweite Funktion hat viele lokale Minima. Da kommt es schon sehr auf x\_0 und die Schrittweite an.