blatt6_aufgabe2

June 12, 2024

[1]:
```
import cma
import numpy as np

# Definition der Zielfunktion
def funktion(x):
    return x[0]**3 - x[1]**3 + x[1]**2 + 1000 * np.cos(x[0]) * np.sin(x[1])

Upperbound=10
Lowerbound=-10

wiederholung=5 #Rechenzeit viel zu groß für 10 Wiederholungen

# Funktion zur Optimierung mit CMA-ES und fmin()
def optimize_cma_es(mu, lambda_,wiederholung):
    all_opt_result=[]

    for i in range(wiederholung):
        x_wert=np.linspace(-10,10,20) #x-Startwerte gleichverteilt auf -10 bis␣
 ↪10
        y_wert=np.linspace(-10,10,20) #y-Startwerte analog
        start_values = np.transpose([np.tile(x_wert, len(y_wert)), np.
 ↪repeat(y_wert, len(x_wert))])
        opt_results=[]

        for start_value in start_values:
            options = {
                'popsize': lambda_,
                'CMA_mu': mu,
                'tolfun': 1e-11,  # Stoppkriterium für die Änderung des␣
 ↪Zielfunktionswerts
                'maxfevals': 500,  #Budget von 500 Auswertungen
                'bounds': [-10, 10],
                'verb_disp': 0,  # Um die Konsolenausgabe zu minimieren
            }
            result = cma.fmin(funktion, start_value, 0.5, options=options)
            opt_results.append(result[1])
```

```
        all_opt_result.extend(opt_results)

    return all_opt_result

# Verschiedene Parameterkombinationen testen
results = []
for mu in [2,4]: #range(2,10)
    #print(mu)
    for lambda_ in [4,8]: #range(2,10)  #Lamda muss größer gleich als mu sein
        if mu<=lambda_:
            #print(lambda_)
            result = optimize_cma_es(mu, lambda_,wiederholung)
            evals_used = result[4]  # Anzahl der Funktionsevaluationen
            deviation = evals_used - 500  # Abweichung vom Soll-Wert
            results.append((mu, lambda_, result, evals_used, deviation))
        else:
            print(f'Mu muss kleiner als Lamda sein')

# Beste Kombination basierend auf dem niedrigsten Zielfunktionswert finden und␣
 ↪kleinste Abweichung zu Soll-Wert
best_result = min(results, key=lambda x: (x[2][1], abs(x[4])))
#print(best_result)
#print(results)
print(f"Beste Parameterkombination: mu = {best_result[0]}, lambda =␣
 ↪{best_result[1]}")
print(f"Minimaler Zielfunktionswert: {best_result[2][1]}")
print(f"Anzahl der Funktionsevaluationen: {best_result[3]}")
print(f"Abweichung vom Soll-Wert: {best_result[4]}")
```

```
mu = 2
mu = 2
mu = 2
mu = 2
mu = 2
mu = 2
mu = 2
mu = 2
mu = 2
mu = 2
mu = 2
mu = 2
mu = 2
mu = 2
mu = 2
mu = 2
mu = 2
mu = 2
mu = 2
mu = 2
```

```
mu = 4
mu = 4
mu = 4
mu = 4
mu = 4
Beste Parameterkombination: mu = 4, lambda = 4
Minimaler Zielfunktionswert: -356.47262536381385
Anzahl der Funktionsevaluationen: -681.1920133323786
Abweichung vom Soll-Wert: -1181.1920133323786
```

Die Berechnungszeit war trotz reduzierter Anzahl der Wiederholungen (von 10 auf 5) immernoch so hoch, dass auch die Anzahl an mu und lambda reduziert werden musste.

Ich hätte gerne mehr Paramterkombinationen von mu und lambda getestet, aber bereits mit nur 4 Möglichkeiten und 5 Wiederholungen hat der Algorithmus knapp eine Stunde gebraucht.

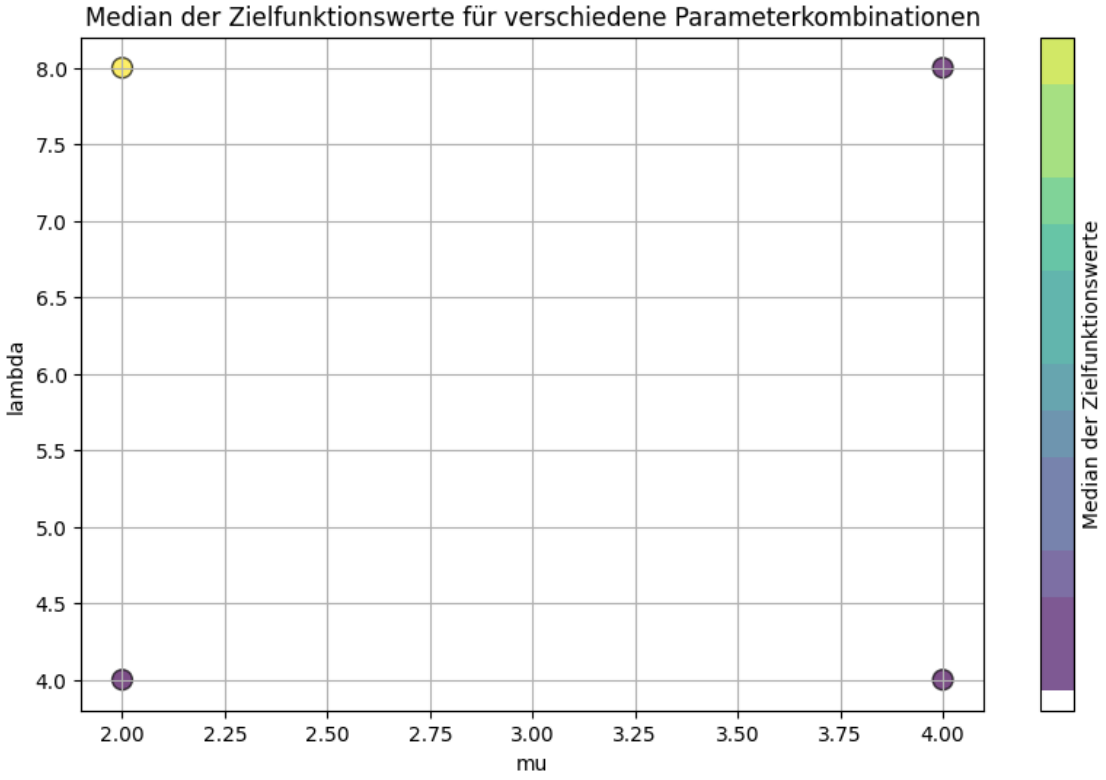
[13]:
```python
#print(results)

#Berechnung der Median der Zielfunktionswerte
for res in results:
    mu, lambda_, opt_results, evals_used, deviation = res
    median_val = np.median(opt_results)
    print(f"Parameterkombination: mu = {mu}, lambda = {lambda_}, Median der␣
 ↪Zielfunktionswerte = {median_val}")
```

```
Parameterkombination: mu = 2, lambda = 4, Median der Zielfunktionswerte =
-993.7117738741703
Parameterkombination: mu = 2, lambda = 8, Median der Zielfunktionswerte =
-993.7117738741679
Parameterkombination: mu = 4, lambda = 4, Median der Zielfunktionswerte =
-993.7117738741703
Parameterkombination: mu = 4, lambda = 8, Median der Zielfunktionswerte =
-993.7117738741703
```

[10]:
```python
#Berechnung Median der 5 Zielfunktionswerte


all_opt_results = [res[2] for res in results] #Enthält alle Zielfunktionswerte␣
 ↪für jede Parameterkombination
combined_results = [val for sublist in all_opt_results for val in sublist]␣
 ↪#Kombinierte Liste

median_val = np.median(combined_results)
print(f"Median der Zielfunktionswerte der Multi-Starts: {median_val}")
```

```
Median der Zielfunktionswerte der Multi-Starts: -993.7117738741701
```

[12]:
```
#Visualisierung als Streudiagramm
import matplotlib.pyplot as plt

mu_values = [res[0] for res in results]
lambda_values = [res[1] for res in results]
median_values = [np.median(res[2]) for res in results]

plt.figure(figsize=(10, 6))
plt.scatter(mu_values, lambda_values, c=median_values, cmap='viridis', s=100,␣
 ↪edgecolors='k', alpha=0.7)
plt.colorbar(label='Median der Zielfunktionswerte')
plt.xlabel('mu')
plt.ylabel('lambda')
plt.title('Median der Zielfunktionswerte für verschiedene␣
 ↪Parameterkombinationen')
plt.grid(True)
plt.show()
```



Die Grafik zeigt, dass eine Kombination von mu=2 mit lamda=8 , einen kleinen Median der Zielfunktionswerte ergibt.

Die Andern Parameterkombinationen erzeugen einen ähnlichen Median-Wert , welcher schlechter ist.