

Blatt 10

Vincent Kümmerle und Elvis Gnaglo

2. Januar 2026

1 Problem des Handlungsreisenden

```
1 import itertools
2 import math
3 import random
4 import matplotlib.pyplot as plt
5 import time
6
7 def distanz(p1, p2): # Euklidische Distanz berechnen
8     return math.sqrt((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2)
9
10 def pfad_laenge(pfad): # Gesamtlänge eines Pfades berechnen (Liste
    von Punkten)
11     laenge = 0
12     # Iteration von 0 bis zum vorletzten Punkt und Addition der
        Distanz zum Nachfolger
13     for i in range(len(pfad) - 1):
14         laenge += distanz(pfad[i], pfad[i+1])
15     return laenge
16
17 def main():
18     n = 10 # Anzahl der Punkte
19     random.seed() # Setzt den Seed für reproduzierbare Ergebnisse,
        ohne Wert ist das Ergebnis für jede Ausführung des Programms
        unterschiedlich
20
21     # Zufällige Punkte für x und y zwischen 0 und 100 generieren
22     punkte = [(random.uniform(0, 100), random.uniform(0, 100)) for _
        in range(n)]
23
24     print(f"Berechne kürzesten Pfad für {n} Punkte...")
25     print(f"Zu prüfende Permutationen: {math.factorial(n):,}")
26
```

```

27 start_time = time.time()
28
29 # Alle Permutationen berechnen und die mit der kürzesten Strecke
    finden
30 kuerzeste_strecke = float('inf')
31 bester_pfad = None
32
33 # itertools.permutations erstellt alle möglichen Reihenfolgen
34 for perm in itertools.permutations(punkte):
35     aktuelle_laenge = pfad_laenge(perm)
36
37     if aktuelle_laenge < kuerzeste_strecke:
38         kuerzeste_strecke = aktuelle_laenge
39         bester_pfad = perm
40
41 end_time = time.time()
42 print(f"Fertig in {end_time - start_time:.2f} Sekunden.")
43 print(f"Kürzeste Strecke: {kuerzeste_strecke:.2f}")
44
45 if bester_pfad:
46     # Koordinaten für den Plot entpacken
47     x_coords = [p[0] for p in bester_pfad]
48     y_coords = [p[1] for p in bester_pfad]
49
50     plt.figure(figsize=(8, 6))
51
52     # Weg zeichnen und Punkte anzeigen
53     plt.plot(x_coords, y_coords, color='black', linestyle='-',
        linewidth=2, zorder=1, label='Kürzester Pfad')
54     plt.scatter(x_coords, y_coords, color='green', s=100, zorder
        =2, label='Orte')
55
56     # Start- und Endpunkt beschriften
57     plt.text(x_coords[0], y_coords[0], 'Start',
        verticalalignment='bottom', fontweight='bold')
58     plt.text(x_coords[-1], y_coords[-1], 'Ende',
        verticalalignment='bottom', fontweight='bold')
59     plt.title(f'Kürzester Pfad durch {n} zufällige Punkte\nLänge:
        {kuerzeste_strecke:.2f}')
60     plt.xlabel('X-Koordinate')
61     plt.ylabel('Y-Koordinate')
62     plt.grid(True)
63     plt.legend()
64     plt.savefig("Reisender2.pdf")
65     plt.show()

```

```

66 if __name__ == "__main__":
67     main()

```

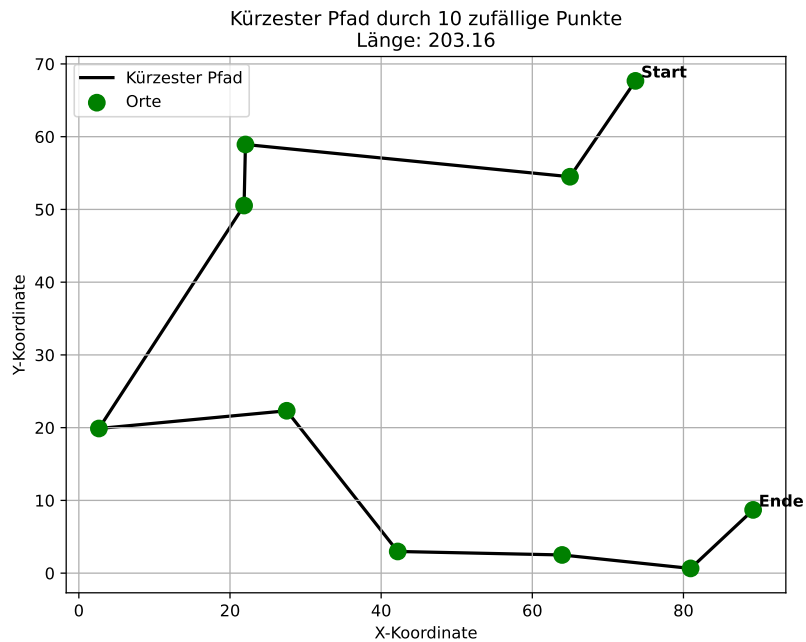


Abbildung 1: Plot einer zufälligen Reisestrecke

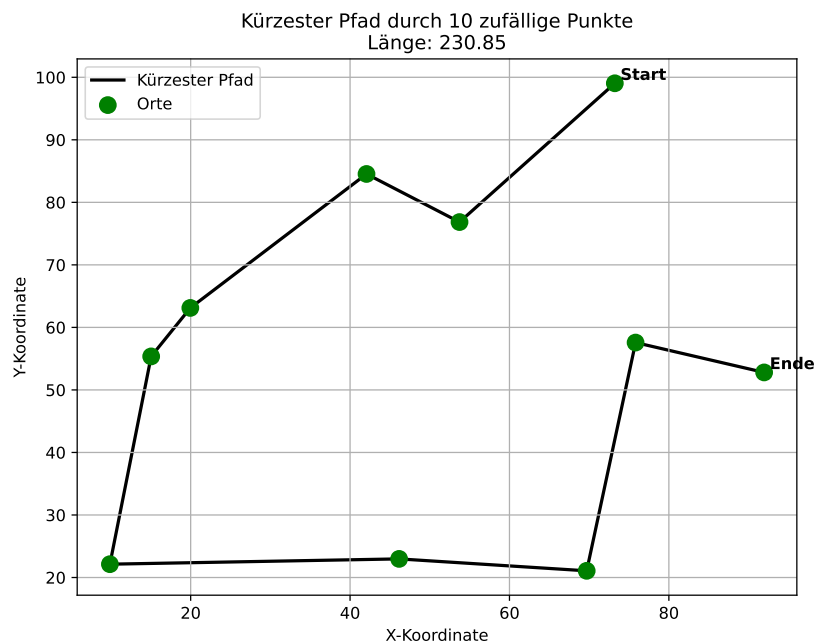


Abbildung 2: Plot einer anderen zufälligen Reisestrecke

2 Fehleranalyse einer gedämpften Schwingung

2.1 Symbolische Berechnung

```
1 import sympy as sp
2 # Variablen und Fehler definieren
3 t, A, gamma, omega = sp.symbols('t A gamma omega')
4 dA, dgamma, domega = sp.symbols('dA dgamma domega')
5
6 # Funktion
7 x = A * sp.exp(-gamma * t) * sp.cos(omega * t)
8
9 # Partielle Ableitungen
10 dx_dA = sp.diff(x, A)
11 dx_dgamma = sp.diff(x, gamma)
12 dx_domega = sp.diff(x, omega)
13
14 # Fehlerfortpflanzung
15 dx = dA * sp.Abs(dx_dA) + dgamma * sp.Abs(dx_dgamma) + domega * sp.
    Abs(dx_domega)
16
17 # Lambdify für numerische Funktion
18 x_func = sp.lambdify((t, A, gamma, omega), x, "numpy")
19 dx_func = sp.lambdify((t, A, gamma, omega, dA, dgamma, domega), dx, "
    numpy")
```

2.2 Numerische Simulation und Plot

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # Skript aus Teil A
5
6 # Zeitvektor
7 t_num = np.linspace(0, 20, 400)
8
9 # Fall 1: dA = 1.0
10 x_num = x_func(t_num, 10, 0.1, 2.0) # numerische x-Werte
11 dx_num = dx_func(t_num, 10, 0.1, 2.0, 1.0, 0.0, 0.0)
12
13 plt.plot(t_num, x_num)
14 plt.fill_between(t_num, x_num - dx_num, x_num + dx_num, alpha=0.4)
15 plt.xlabel("t")
16 plt.ylabel("x(t)")
17 plt.legend(["x(t)", "Unsicherheitsbereich"])
```

```

18 plt.show()
19
20 # Fall 2: domega = 0.2
21 dx_num = dx_func(t_num, 10, 0.1, 2.0, 0.0, 0.0, 0.2)
22
23 plt.plot(t_num, x_num)
24 plt.fill_between(t_num, x_num - dx_num, x_num + dx_num, alpha=0.4)
25 plt.xlabel("t")
26 plt.ylabel("x(t)")
27 plt.legend(["x(t)", "Unsicherheitsbereich"])
28 plt.show()

```

Das Plot für Fall 1 mit $\Delta A = 1.0$ ist in Abbildung 3 dargestellt. Der Verlauf des Plots mit Abnahme des Unsicherheitsbereichs lässt sich durch die Ableitung von x nach A erklären, die nur eine Exponentialfunktion ohne Faktor t enthält, wodurch ΔA mit der Zeit stark abnimmt. Somit nimmt auch der Gesamtfehler ab, wenn die Schwingung abklingt.

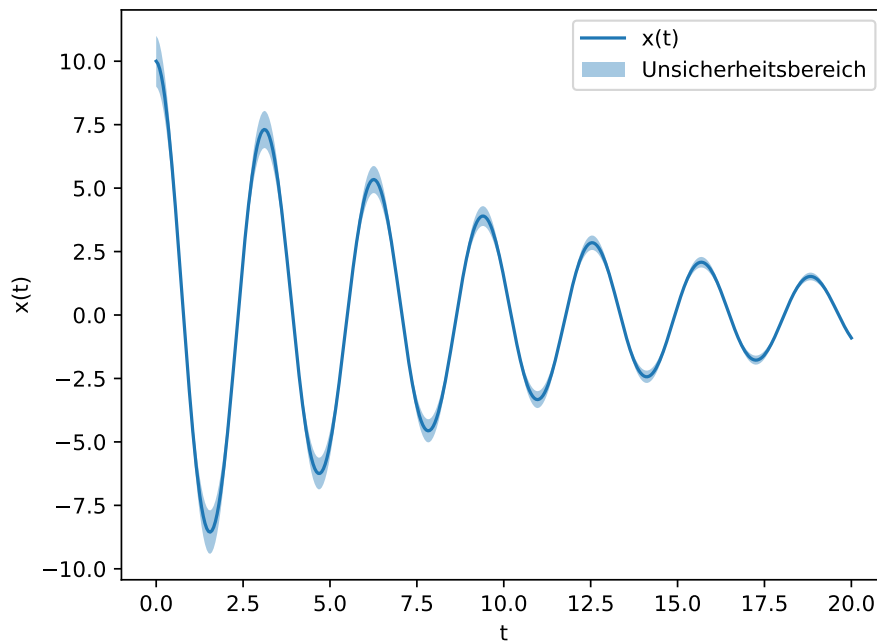


Abbildung 3: Plot der gedämpften Schwingung mit $\Delta A = 1.0$.

Das Plot für Fall 2 mit $\Delta\omega = 0.2$ ist in Abbildung 4 dargestellt. Der Verlauf des Plots mit Zunahme des Unsicherheitsbereichs lässt sich dadurch erklären, dass in der Ableitung von x nach ω der Faktor t enthalten ist, wodurch der Gesamtfehler proportional zur Zeit ist. Somit nimmt der Fehler mit der Zeit zu, wenn die Schwingung abklingt.

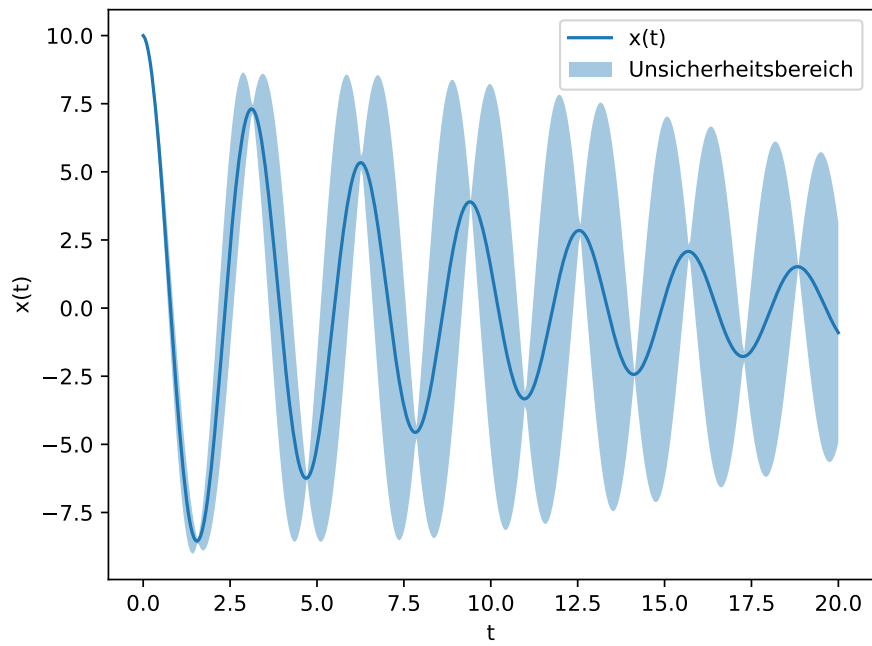


Abbildung 4: Plot der gedämpften Schwingung mit $\Delta\omega = 0.2$.