Blatt 10

Vincent Kümmerle und Elvis Gnaglo

3. Januar 2026

1 Problem des Handlungsreisenden

```python
import itertools
import math
import random
import matplotlib.pyplot as plt
import time

def distanz(p1, p2): # Euklidische Distanz berechnen
    return math.sqrt((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2)

def pfad_laenge(pfad): # Gesamtlänge eines Pfades berechnen (Liste von Punkten)
    laenge = 0
    # Iteration von 0 bis zum vorletzten Punkt und Addition der Distanz zum Nachfolger
    for i in range(len(pfad) - 1):
        laenge += distanz(pfad[i], pfad[i+1])
    return laenge

def main():
    n = 10  # Anzahl der Punkte
    random.seed() # Setzt den Seed für reproduzierbare Ergebnisse, ohne Wert ist das Ergebnis für jede Ausführung des Programms unterschiedlich

    # Zufällige Punkte für x und y zwischen 0 und 100 generieren
    punkte = [(random.uniform(0, 100), random.uniform(0, 100)) for _ in range(n)]

    print(f"Berechne kürzesten Pfad für {n} Punkte...")
    print(f"Zu prüfende Permutationen: {math.factorial(n):,}")

```

```
    start_time = time.time()

    # Alle Permutationen berechnen und die mit der kürzesten Strecke finden
    kuerzeste_strecke = float('inf')
    bester_pfad = None

    # itertools.permutations erstellt alle möglichen Reihenfolgen
    for perm in itertools.permutations(punkte):
        aktuelle_laenge = pfad_laenge(perm)

        if aktuelle_laenge < kuerzeste_strecke:
            kuerzeste_strecke = aktuelle_laenge
            bester_pfad = perm

    end_time = time.time()
    print(f"Fertig in {end_time - start_time:.2f} Sekunden.")
    print(f"Kürzeste Strecke: {kuerzeste_strecke:.2f}")

    if bester_pfad:
        # Koordinaten für den Plot entpacken
        x_coords = [p[0] for p in bester_pfad]
        y_coords = [p[1] for p in bester_pfad]

        plt.figure(figsize=(8, 6))

        # Weg zeichnen und Punkte anzeigen
        plt.plot(x_coords, y_coords, color='black', linestyle='-', linewidth=2, zorder=1, label='Kürzester Pfad')
        plt.scatter(x_coords, y_coords, color='green', s=100, zorder=2, label='Orte')

        # Start- und Endpunkt beschriften
        plt.text(x_coords[0], y_coords[0], ' Start', verticalalignment='bottom', fontweight='bold')
        plt.text(x_coords[-1], y_coords[-1], ' Ende', verticalalignment='bottom', fontweight='bold')
        plt.title(f'Kürzester Pfad durch {n} zufällige Punkte\nLänge: {kuerzeste_strecke:.2f}')
        plt.xlabel('X-Koordinate')
        plt.ylabel('Y-Koordinate')
        plt.grid(True)
        plt.legend()
        plt.savefig("Reisender2.pdf")
        plt.show()
```

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

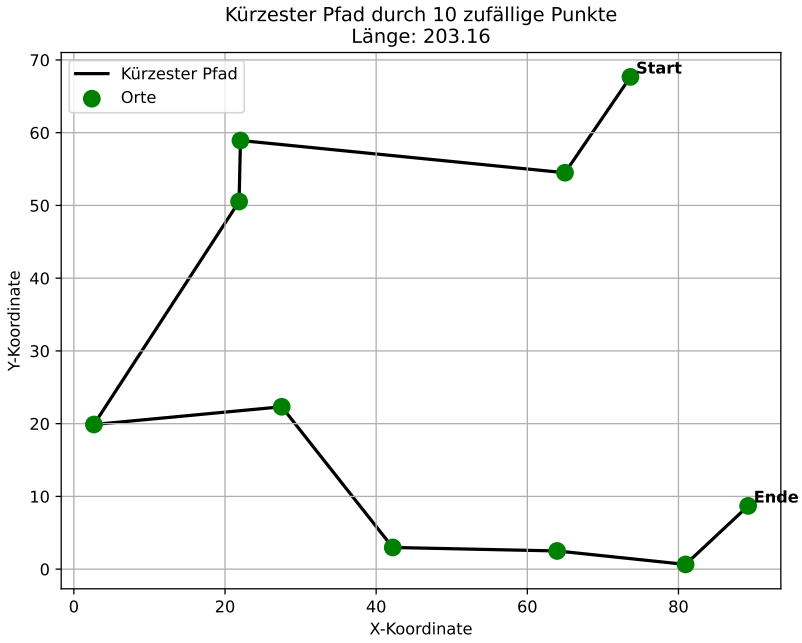

Abbildung 1: Plot einer zufälligen Reisestrecke

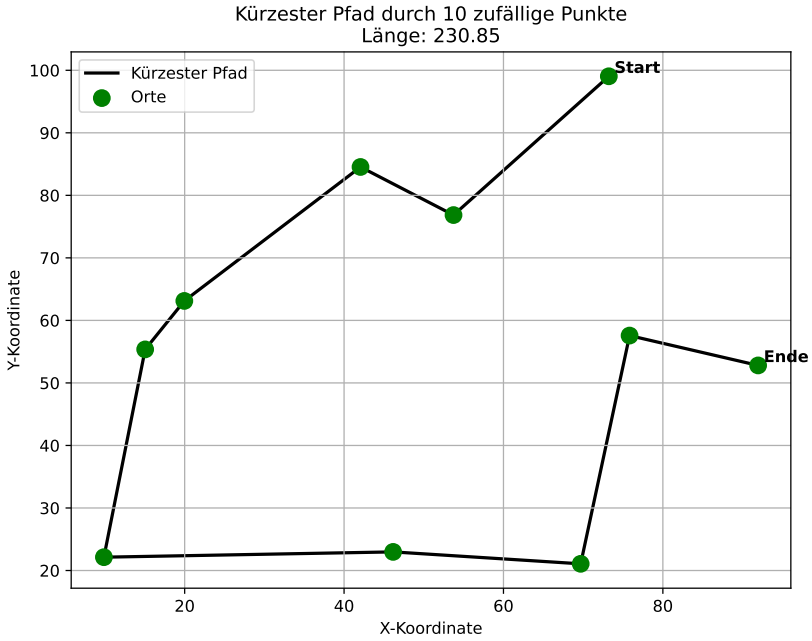

Abbildung 2: Plot einer anderen zufälligen Reisestrecke

2 Fehleranalyse einer gedämpften Schwingung

2.1 Symbolische Berechnung

```
import sympy as sp
# Variablen und Fehler definieren
t, A, gamma, omega = sp.symbols('t A gamma omega')
dA, dgamma, domega = sp.symbols('dA dgamma domega')

# Funktion
x = A * sp.exp(-gamma * t) * sp.cos(omega * t)

# Partielle Ableitungen
dx_dA = sp.diff(x, A)
dx_dgamma = sp.diff(x, gamma)
dx_domega = sp.diff(x, omega)

# Fehlerfortpflanzung
dx = dA * sp.Abs(dx_dA) + dgamma * sp.Abs(dx_dgamma) + domega * sp.
    Abs(dx_domega)

# Lambdify für numerische Funktion
x_func = sp.lambdify((t, A, gamma, omega), x, "numpy")
dx_func = sp.lambdify((t, A, gamma, omega, dA, dgamma, domega), dx, "
    numpy")
```

2.2 Numerische Simulation und Plot

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

# Skript aus Teil A

# Zeitvektor
t_num = np.linspace(0, 20, 400)

# Fall 1: dA = 1.0
x_num = x_func(t_num, 10, 0.1, 2.0) # numerische x-Werte
dx_num = dx_func(t_num, 10, 0.1, 2.0, 1.0, 0.0, 0.0)

plt.plot(t_num, x_num)
plt.fill_between(t_num, x_num - dx_num, x_num + dx_num, alpha=0.4)
plt.xlabel("t")
plt.ylabel("x(t)")
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.

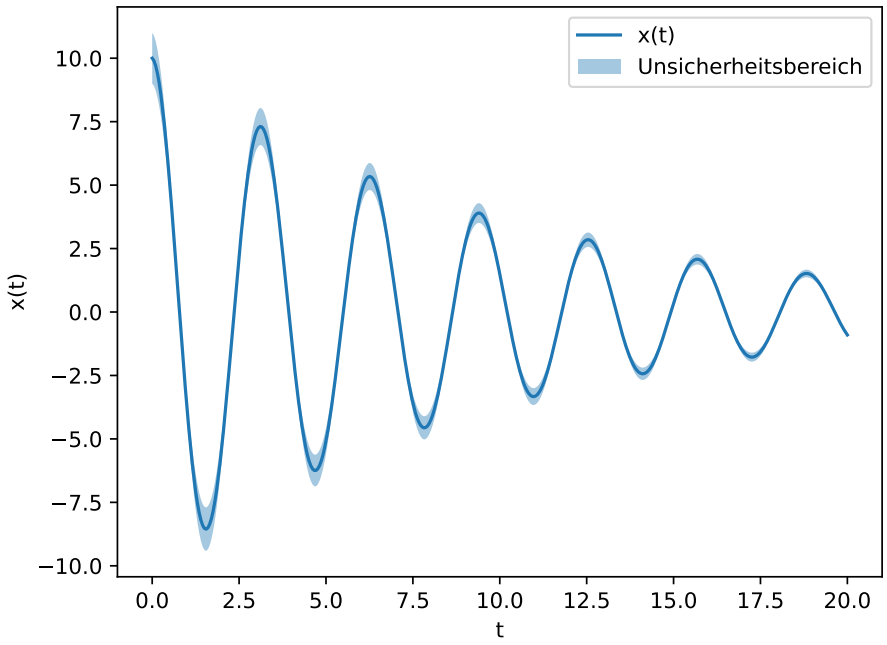

Abbildung 3: Plot der gedämpften Schwingung mit $\Delta A = 1.0$.

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.

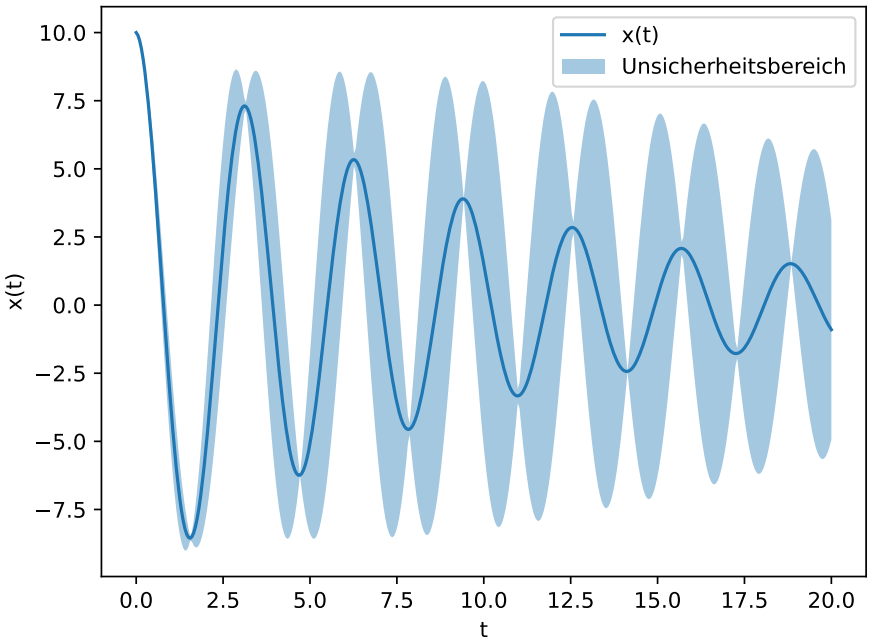

Abbildung 4: Plot der gedämpften Schwingung mit $\Delta\omega = 0.2$.