schwingungsanalyse_main.py

```python
# Schwingungsanalyse
# ##############################################################################

# This python script calculates the frequency of the occilation.

# Parts of this python script come from the following github-repository made by
# Christopher Mahn:
# https://github.com/c-mahn/test-sensor-data-analysis

# Authors:
# Joshua Wolf
# Silas Teske
# Lasse Zeh
# Christopher Mahn

# ##############################################################################

# Import of Libraries
# ------------------------------------------------------------------------------

# import string as st
# import random as r
# import re
import matplotlib.pyplot as plt
from scipy import interpolate
import numpy as np
import math as m
import sys
import os
from scipy.fft import fft, fftfreq
from scipy import signal


# ------------------------------------------------------------------------------
# Settings

verbose = False  # Shows more debugging information
fix_steigung = False # Disregards permanent sensor changes


# Functions
# ------------------------------------------------------------------------------

def run_analysis(input_file):


    # Einlesen der Daten
    print(f"[{1}/{1}] Einlesen der Daten...", end="\r")
    with open(os.path.join("data", input_file), "r") as f:
        data = f.readlines()
    print("")

    # Datenbereinigung
    print(f"[{1}/{1}] Datenbereinigung...", end="\r")
    for i, e in enumerate(data):
        data[i] = e.split(";")
        for j, e in enumerate(data[i]):
            data[i][j] = float(e.strip())
    print("")

    # Umwandeln in Datenreihen
    print(f"[{1}/{1}] Datenkonvertierung...", end="\r")
    datenreihen = [[], [], [], []]
    for i in data:
        datenreihen[0].append(i[0])
        datenreihen[1].append(i[1])
        datenreihen[2].append(i[2])
        datenreihen[3].append(i[3])
    datenreihen_ohne_zeit = datenreihen[1:4]
    print("")
    plot_werte(datenreihen_ohne_zeit, ["Sensor 1", "Sensor 2", "Sensor 3"])

    # Berechnung der linearen Regression von Sensor 1
    x = np.array(datenreihen[0])
    y = np.array(datenreihen[1])
    steigung_1 = (len(x) * np.sum(x*y) - np.sum(x) * np.sum(y)) / (len(x)*np.sum(x*x) - np.sum(x) ** 2)
    offset_1 = (np.sum(y) - steigung_1 *np.sum(x)) / len(x)
    print(f'[Sensor 1] Trend: ({steigung_1:.10f})x + ({offset_1:+.6f})')

    # Berechnung der linearen Regression von Sensor 2
```

```
081     x = np.array(datenreihen[0])
082     y = np.array(datenreihen[2])
083     steigung_2 = (len(x) * np.sum(x*y) - np.sum(x) * np.sum(y)) / (len(x)*np.sum(x*x) - np.sum(x) ** 2)
084     offset_2 = (np.sum(y) - steigung_2 *np.sum(x)) / len(x)
085     print(f'[Sensor 2] Trend: ({steigung_2:.10f})x + ({offset_2:+.6f})')
086 
087     # Berechnung der linearen Regression von Sensor 3
088     x = np.array(datenreihen[0])
089     y = np.array(datenreihen[3])
090     steigung_3 = (len(x) * np.sum(x*y) - np.sum(x) * np.sum(y)) / (len(x)*np.sum(x*x) - np.sum(x) ** 2)
091     offset_3 = (np.sum(y) - steigung_3 *np.sum(x)) / len(x)
092     print(f'[Sensor 2] Trend: ({steigung_3:.10f})x + ({offset_3:+.6f})')
093 
094     # Erstellung Lineare Regression zum Plotten (Plot-Punkte)
095     linearisierung = [[], [], []]
096     for i in datenreihen[0]:
097         linearisierung[0].append(i*steigung_1+offset_1)
098         linearisierung[1].append(i*steigung_2+offset_2)
099         linearisierung[2].append(i*steigung_3+offset_3)
100     # Plot der linearen Regression
101     plot_werte([datenreihen[1], linearisierung[0]], ["Sensor 1", "Linearisierung"])
102     plot_werte([datenreihen[2], linearisierung[1]], ["Sensor 2", "Linearisierung"])
103     plot_werte([datenreihen[3], linearisierung[2]], ["Sensor 3", "Linearisierung"])
104 
105     # Wenn Steigung nicht bereinigt werden soll
106     if(fix_steigung):
107         steigung_1 = 0
108         steigung_2 = 0
109         steigung_3 = 0
110 
111     # Bereinigung des Trends aller Sensorreihen
112     datenreihen_ohne_trend = [[], [], []]
113     print(f"[{1}/{len(datenreihen_ohne_trend)}] Bereinigung des Trends...", end="\r")
114     for i, e in enumerate(datenreihen[1]):
115         datenreihen_ohne_trend[0].append(e - (steigung_1*(datenreihen[0][i])+offset_1))
116     print(f"[{2}/{len(datenreihen_ohne_trend)}] Bereinigung des Trends...", end="\r")
117     for i, e in enumerate(datenreihen[2]):
118         datenreihen_ohne_trend[1].append(e - (steigung_2*(datenreihen[0][i])+offset_2))
119     print(f"[{3}/{len(datenreihen_ohne_trend)}] Bereinigung des Trends...", end="\r")
120     for i, e in enumerate(datenreihen[3]):
121         datenreihen_ohne_trend[2].append(e - (steigung_3*(datenreihen[0][i])+offset_3))
122     print("")
123     # Plot der vom Trend bereinigten Sensorreihen
124     plot_werte(datenreihen_ohne_trend, ["Sensor 1 (ohne Trend)", "Sensor 2 (ohne Trend)", "Sensor 3
        (ohne Trend)"])
125 
126     # Low-Pass-Filterung der Sensorreihen
127     low_pass_strength = 3
128     datenreihen_low_pass = []
129     for i, e in enumerate(datenreihen_ohne_trend):
130         print(f"[{i+1}/{len(datenreihen_ohne_trend)}] Low-Pass-Filterung (Strength
        {low_pass_strength})...", end="\r")
131         datenreihen_low_pass.append(low_pass_filter(e, low_pass_strength))
132     print("")
133     # Plot der low-pass Sensorreihen
134     plot_werte(datenreihen_low_pass, ["Sensor 1 (mit Low-Pass-Filter)", "Sensor 2 (mit Low-Pass-
        Filter)", "Sensor 3 (mit Low-Pass-Filter)"])
135 
136     # Hoch-Pass-Filterung der Sensorreihen
137     datenreihen_hoch_pass = [[], [], []]
138     print(f"[{1}/{len(datenreihen_hoch_pass)}] Hoch-Pass-Filterung...", end="\r")
139     for i, e in enumerate(datenreihen_low_pass[0]):
140         datenreihen_hoch_pass[0].append(datenreihen_ohne_trend[0][i] - e)
141     print(f"[{2}/{len(datenreihen_hoch_pass)}] Hoch-Pass-Filterung...", end="\r")
142     for i, e in enumerate(datenreihen_low_pass[1]):
143         datenreihen_hoch_pass[1].append(datenreihen_ohne_trend[1][i] - e)
144     print(f"[{3}/{len(datenreihen_hoch_pass)}] Hoch-Pass-Filterung...", end="\r")
145     for i, e in enumerate(datenreihen_low_pass[2]):
146         datenreihen_hoch_pass[2].append(datenreihen_ohne_trend[2][i] - e)
147     print("")
148     # Plot der hoch-pass Sensorreihen
149     plot_werte(datenreihen_hoch_pass, ["Sensor 1 (mit Hoch-Pass-Filter)", "Sensor 2 (mit Hoch-Pass-
        Filter)", "Sensor 3 (mit Hoch-Pass-Filter)"])
150 
151     # Fourier-Transformation
152 
153     # Weitere Informationen:
154     # https://docs.scipy.org/doc/scipy/reference/tutorial/fft.html
155     sample_frequenz = datenreihen[0][2] - datenreihen[0][1]
156     N = len(datenreihen[0])*2
157     print(f"[{1}/{len(datenreihen_ohne_trend)}] Fast-Fourier-Transformation...", end="\r")
158     yf_1 = fft(datenreihen_low_pass[0])
159     xf_1 = fftfreq(len(datenreihen_low_pass[0]), 1/sample_frequenz)
```

```
160     print(f"[{2}/{len(datenreihen_ohne_trend)}] Fast-Fourier-Transformation...", end="\r")
161     yf_2 = fft(datenreihen_low_pass[1])
162     xf_2 = fftfreq(len(datenreihen_low_pass[1]), 1/sample_frequenz)
163     print(f"[{3}/{len(datenreihen_ohne_trend)}] Fast-Fourier-Transformation...", end="\r")
164     yf_3 = fft(datenreihen_low_pass[2])
165     xf_3 = fftfreq(len(datenreihen_low_pass[2]), 1/sample_frequenz)
166     print("")
167     # Plot der Fourier-Transformation
168     plt.plot(xf_1, 2.0/N * np.abs(yf_1[0:N//2]))
169     # plt.xlim(0, 0.0001)
170     plt.grid()
171     plt.show()
172 
173     plt.plot(xf_2, 2.0/N * np.abs(yf_2[0:N//2]))
174     # plt.xlim(0, 0.0001)
175     plt.grid()
176     plt.show()
177 
178     plt.plot(xf_3, 2.0/N * np.abs(yf_3[0:N//2]))
179     # plt.xlim(0, 0.0001)
180     plt.grid()
181     plt.show()
182 
183 
184 def plot_werte(datenreihen, name=["Messwerte"]):
185     """
186     Diese Funktion nimmt Datenreihen und plottet diese in ein Diagramm.
187     """
188     for i, datenreihe in enumerate(datenreihen):
189         zeit = range(len(datenreihe))
190         plt.plot(zeit, datenreihe)
191     plt.legend(name)
192     plt.grid()
193     plt.xlabel("")
194     plt.ylabel("")
195     plt.title(name[0])
196     plt.show()
197 
198 
199 def plot_xy(datenreihen, name=["Messwerte"]):
200     """
201     Diese Funktion nimmt je zwei Datenreihen und plottet diese in Abhängigkeit
202     zueinander in ein Diagramm.
203     """
204     for i, datenreihe in enumerate(datenreihen):
205         plt.plot(datenreihe[0], datenreihe[1])
206     plt.legend(name)
207     plt.grid()
208     plt.xlabel("Y")
209     plt.ylabel("X")
210     plt.title(name[0])
211     plt.show()
212 
213 
214 def fill_nan(A):
215     '''
216     interpolate to fill nan values
217     '''
218     inds = np.arange(A.shape[0])
219     good = np.where(np.isfinite(A))
220     f = interpolate.interp1d(inds[good], A[good],bounds_error=False)
221     B = np.where(np.isfinite(A),A,f(inds))
222     return B
223 
224 
225 def low_pass_filter(datenreihe, filterungsgrad):
226     """
227     Diese Funktion macht einen vereinfachten Low-Pass-Filter, indem die letzten
228     x Sensorwerte gemittelt werden.
229     """
230     ausgabe = []
231     for i, e in enumerate(datenreihe):
232         divisor = filterungsgrad
233         summe = 0
234         for j in range(filterungsgrad):
235             ji = i-j
236             if(ji < 0):  # Wenn Wert ausserhalb der Datenreihe, ändern des Divisors
237                 divisor = divisor - 1
238             else:
239                 summe = summe + float(datenreihe[ji])
240         temp = summe/divisor
241         ausgabe.append(temp)
242     return(ausgabe)
```

```
243
244
245 def cross_correlation(x_red, y_red):
246         Nx = len(x_red)
247         if Nx != len(y_red):
248             raise ValueError('x and y must be equal length')
249         c = np.correlate(x_red, y_red, mode=2)
250         c /= np.sqrt(np.dot(x_red, x_red) * np.dot(y_red, y_red))
251         maxlags = Nx - 1
252         if maxlags >= Nx or maxlags < 1:
253             raise ValueError('maglags must be None or strictly '
254                              'positive < %d' % Nx)
255         lags = np.arange(-maxlags, maxlags + 1)
256         c = c[Nx - 1 - maxlags:Nx + maxlags]
257         return lags, c
258
259
260 # Classes
261 # ------------------------------------------------------------------------------
262
263
264 # Beginning of the Programm
265 # ------------------------------------------------------------------------------
266
267 if __name__ == '__main__':
268     print("Running schwingungsanalyse_main.py...")
269
270     for i in range(4):
271         i += 1
272         run_analysis(f"Schwingungsanalyse_{i:02d}.txt")
```