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 = True  # Shows more debugging information
fix_steigung = False  # Disregards permanent sensor changes
show_graphs = False  # If disabled, the Plots will not be shown.


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

def run_analysis(input_file):

    if(verbose):
        print(14*"-", "BEGIN OF CALCULATION", 14*"-")

    # Einlesen der Daten
    if(verbose):
        print(f'[{1}/{3}] Einlesen von "{input_file}"', end="\r")
    with open(os.path.join("data", input_file), "r") as f:
        data = f.readlines()

    # Datenbereinigung
    if(verbose):
        print(f'[{2}/{3}] Einlesen von "{input_file}"', 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())

    # Umwandeln in Datenreihen
    if(verbose):
        print(f'[{3}/{3}] Einlesen von "{input_file}"', 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]
    if(verbose):
        print("")
    if(show_graphs):
        plot_xyzs(datenreihen_ohne_zeit, f'Messreihe "{input_file}"')

    # Berechnung der linearen Regression von Sensor 1
    x = np.array(datenreihen[0])
```

```
081     y = np.array(datenreihen[1])
082     steigung_1 = (len(x) * np.sum(x*y) - np.sum(x) * np.sum(y)) / (len(x)*np.sum(x*x) - np.sum(x) ** 2)
083     offset_1 = (np.sum(y) - steigung_1 *np.sum(x)) / len(x)
084     if(verbose):
085         print(f'[X] Trend: ({steigung_1:.10f})x + ({offset_1:+.6f})')
086 
087     # Berechnung der linearen Regression von Sensor 2
088     x = np.array(datenreihen[0])
089     y = np.array(datenreihen[2])
090     steigung_2 = (len(x) * np.sum(x*y) - np.sum(x) * np.sum(y)) / (len(x)*np.sum(x*x) - np.sum(x) ** 2)
091     offset_2 = (np.sum(y) - steigung_2 *np.sum(x)) / len(x)
092     if(verbose):
093         print(f'[Y] Trend: ({steigung_2:.10f})x + ({offset_2:+.6f})')
094 
095     # Berechnung der linearen Regression von Sensor 3
096     x = np.array(datenreihen[0])
097     y = np.array(datenreihen[3])
098     steigung_3 = (len(x) * np.sum(x*y) - np.sum(x) * np.sum(y)) / (len(x)*np.sum(x*x) - np.sum(x) ** 2)
099     offset_3 = (np.sum(y) - steigung_3 *np.sum(x)) / len(x)
100     if(verbose):
101         print(f'[Z] Trend: ({steigung_3:.10f})x + ({offset_3:+.6f})')
102 
103     # Erstellung Lineare Regression zum Plotten (Plot-Punkte)
104     linearisierung = [[], [], []]
105     for i in datenreihen[0]:
106         linearisierung[0].append(i*steigung_1+offset_1)
107         linearisierung[1].append(i*steigung_2+offset_2)
108         linearisierung[2].append(i*steigung_3+offset_3)
109     # Plot der linearen Regression
110     if(show_graphs):
111         plot_werte_s([datenreihen[1], linearisierung[0]], ["X", "Linearisierung"])
112         plot_werte_s([datenreihen[2], linearisierung[1]], ["Y", "Linearisierung"])
113         plot_werte_s([datenreihen[3], linearisierung[2]], ["Z", "Linearisierung"])
114 
115     # Wenn Steigung nicht bereinigt werden soll
116     if(fix_steigung):
117         steigung_1 = 0
118         steigung_2 = 0
119         steigung_3 = 0
120 
121     # Bereinigung des Trends aller Sensorreihen
122     datenreihen_ohne_trend = [[], [], []]
123     if(verbose):
124         print(f"[{1}/{len(datenreihen_ohne_trend)}] Trendbereinigung", end="\r")
125     for i, e in enumerate(datenreihen[1]):
126         datenreihen_ohne_trend[0].append(e - (steigung_1*(datenreihen[0][i])+offset_1))
127     if(verbose):
128         print(f"[{2}/{len(datenreihen_ohne_trend)}] Trendbereinigung", end="\r")
129     for i, e in enumerate(datenreihen[2]):
130         datenreihen_ohne_trend[1].append(e - (steigung_2*(datenreihen[0][i])+offset_2))
131     if(verbose):
132         print(f"[{3}/{len(datenreihen_ohne_trend)}] Trendbereinigung", end="\r")
133     for i, e in enumerate(datenreihen[3]):
134         datenreihen_ohne_trend[2].append(e - (steigung_3*(datenreihen[0][i])+offset_3))
135     if(verbose):
136         print("")
137     # Plot der vom Trend bereinigten Sensorreihen
138     if(show_graphs):
139         plot_xyzs(datenreihen_ohne_trend, f'Messreihe "{input_file}" (ohne Trend)')
140 
141     # Low-Pass-Filterung der Sensorreihen
142     low_pass_strength = 20
143     datenreihen_low_pass = []
144     for i, e in enumerate(datenreihen_ohne_trend):
145         if(verbose):
146             print(f"[{i+1}/{len(datenreihen_ohne_trend)}] Low-Pass-Filterung (Strength
      {low_pass_strength})", end="\r")
147         datenreihen_low_pass.append(low_pass_filter(e, low_pass_strength))
148     if(verbose):
149         print("")
150     # Plot der low-pass Sensorreihen
151     if(show_graphs):
152         plot_xyzs(datenreihen_low_pass, f'Messreihe "{input_file}" (mit Low-Pass-Filter)')
153 
154     # Hoch-Pass-Filterung der Sensorreihen
155     datenreihen_hoch_pass = [[], [], []]
156     if(verbose):
157         print(f"[{1}/{len(datenreihen_hoch_pass)}] Hoch-Pass-Filterung", end="\r")
158     for i, e in enumerate(datenreihen_low_pass[0]):
159         datenreihen_hoch_pass[0].append(datenreihen_ohne_trend[0][i] - e)
160     if(verbose):
161         print(f"[{2}/{len(datenreihen_hoch_pass)}] Hoch-Pass-Filterung", end="\r")
162     for i, e in enumerate(datenreihen_low_pass[1]):
```

```
        datenreihen_hoch_pass[1].append(datenreihen_ohne_trend[1][i] - e)
    if(verbose):
        print(f"[{3}/{len(datenreihen_hoch_pass)}] Hoch-Pass-Filterung", end="\r")
    for i, e in enumerate(datenreihen_low_pass[2]):
        datenreihen_hoch_pass[2].append(datenreihen_ohne_trend[2][i] - e)
    if(verbose):
        print("")
    # Plot der hoch-pass Sensorreihen
    if(show_graphs):
        plot_xyzs(datenreihen_hoch_pass, f'Messreihe "{input_file}" (mit Hoch-Pass-Filter)')

    # Fourier-Transformation

    # Weitere Informationen:
    # https://docs.scipy.org/doc/scipy/reference/tutorial/fft.html
    sample_frequenz = datenreihen[0][2] - datenreihen[0][1]
    N = len(datenreihen[0])*2
    if(verbose):
        print(f"[{1}/{len(datenreihen_ohne_trend)}] Fast-Fourier-Transformation", end="\r")
    yf_1 = fft(datenreihen_low_pass[0])
    xf_1 = fftfreq(len(datenreihen_low_pass[0]), 1/sample_frequenz)
    if(verbose):
        print(f"[{2}/{len(datenreihen_ohne_trend)}] Fast-Fourier-Transformation", end="\r")
    yf_2 = fft(datenreihen_low_pass[1])
    xf_2 = fftfreq(len(datenreihen_low_pass[1]), 1/sample_frequenz)
    if(verbose):
        print(f"[{3}/{len(datenreihen_ohne_trend)}] Fast-Fourier-Transformation", end="\r")
    yf_3 = fft(datenreihen_low_pass[2])
    xf_3 = fftfreq(len(datenreihen_low_pass[2]), 1/sample_frequenz)
    if(verbose):
        print("")
    # Plot der Fourier-Transformation
    if(show_graphs):
        plt.plot(xf_1, 2.0/N * np.abs(yf_1[0:N//2]))
        # plt.xlim(0, 0.0001)
        plt.grid()
        plt.xlabel("Frequenz [s/Sample]")
        plt.ylabel("Amplitude [1]")
        plt.title("Fast-Fourier-Transformation (X-Achse)")
        plt.tight_layout()
        plt.show()

        plt.plot(xf_2, 2.0/N * np.abs(yf_2[0:N//2]))
        # plt.xlim(0, 0.0001)
        plt.grid()
        plt.xlabel("Frequenz [s/Sample]")
        plt.ylabel("Amplitude [1]")
        plt.title("Fast-Fourier-Transformation (Y-Achse)")
        plt.tight_layout()
        plt.show()

        plt.plot(xf_3, 2.0/N * np.abs(yf_3[0:N//2]))
        # plt.xlim(0, 0.0001)
        plt.grid()
        plt.xlabel("Frequenz [s/Sample]")
        plt.ylabel("Amplitude [1]")
        plt.title("Fast-Fourier-Transformation (Z-Achse)")
        plt.tight_layout()
        plt.show()

    if(verbose):
        print(15*"-", "END OF CALCULATION", 15*"-")


def plot_werte(datenreihen, name=["Messwerte"]):
    """
    Diese Funktion nimmt Datenreihen und plottet diese in ein Diagramm.
    """
    for i, datenreihe in enumerate(datenreihen):
        zeit = range(len(datenreihe))
        plt.plot(zeit, datenreihe)
    plt.legend(name)
    plt.grid()
    plt.xlabel("")
    plt.ylabel("")
    plt.title(name[0])
    plt.show()


def plot_werte_s(datenreihen, name=["Messwerte"]):
    """
    Diese Funktion plottet Werte als Samples.

```

```
246     Args:
247         datenreihen ([type]): Datenreihen zum Plotten.
248         name (list, optional): Angaben zur Beschriftung. Defaults to ["Messwerte"].
249     """
250     for i, datenreihe in enumerate(datenreihen):
251         zeit = range(len(datenreihe))
252         plt.plot(zeit, datenreihe)
253     plt.legend(name)
254     plt.grid()
255     plt.xlabel("Sample [1]")
256     plt.ylabel("Koordinate [mm]")
257     plt.title(name[0])
258     plt.tight_layout()
259     plt.show()
260 
261 
262 def plot_xyzs(datenreihen, name="Messwerte"):
263     """
264     Diese Funktion nimmt genau drei Datenreihen und plottet diese in ein Diagramm.
265 
266     Args:
267         datenreihen ([list]): Drei Datenreihen zum Plotten.
268         name (list, optional): Dies ist der Titel. Defaults to "Messwerte".
269     """
270     for i, datenreihe in enumerate(datenreihen):
271         zeit = range(len(datenreihe))
272         plt.plot(zeit, datenreihe)
273     plt.legend(["x", "y", "z"])
274     plt.grid()
275     plt.xlabel("Sample [1]")
276     plt.ylabel("Koordinate [mm]")
277     plt.title(name)
278     plt.tight_layout()
279     plt.show()
280 
281 
282 def plot_xy(datenreihen, name=["Messwerte"]):
283     """
284     Diese Funktion nimmt je zwei Datenreihen und plottet diese in Abhängigkeit
285     zueinander in ein Diagramm.
286     """
287     for i, datenreihe in enumerate(datenreihen):
288         plt.plot(datenreihe[0], datenreihe[1])
289     plt.legend(name)
290     plt.grid()
291     plt.xlabel("Y")
292     plt.ylabel("X")
293     plt.title(name[0])
294     plt.show()
295 
296 
297 def fill_nan(A):
298     '''
299     interpolate to fill nan values
300     '''
301     inds = np.arange(A.shape[0])
302     good = np.where(np.isfinite(A))
303     f = interpolate.interp1d(inds[good], A[good],bounds_error=False)
304     B = np.where(np.isfinite(A),A,f(inds))
305     return B
306 
307 
308 def low_pass_filter(datenreihe, filterungsgrad):
309     """
310     Diese Funktion macht einen vereinfachten Low-Pass-Filter, indem die letzten
311     x Sensorwerte gemittelt werden.
312     """
313     ausgabe = []
314     for i, e in enumerate(datenreihe):
315         divisor = filterungsgrad
316         summe = 0
317         for j in range(filterungsgrad):
318             ji = i-j
319             if(ji < 0):  # Wenn Wert ausserhalb der Datenreihe, ändern des Divisors
320                 divisor = divisor - 1
321             else:
322                 summe = summe + float(datenreihe[ji])
323         temp = summe/divisor
324         ausgabe.append(temp)
325     return(ausgabe)
326 
327 
328 def cross_correlation(x_red, y_red):
```

```
329         Nx = len(x_red)
330         if Nx != len(y_red):
331             raise ValueError('x and y must be equal length')
332         c = np.correlate(x_red, y_red, mode=2)
333         c /= np.sqrt(np.dot(x_red, x_red) * np.dot(y_red, y_red))
334         maxlags = Nx - 1
335         if maxlags >= Nx or maxlags < 1:
336             raise ValueError('maglags must be None or strictly '
337                              'positive < %d' % Nx)
338         lags = np.arange(-maxlags, maxlags + 1)
339         c = c[Nx - 1 - maxlags:Nx + maxlags]
340         return lags, c
341 
342 
343 # Classes
344 # ------------------------------------------------------------------------------
345 
346 
347 # Beginning of the Programm
348 # ------------------------------------------------------------------------------
349 
350 if __name__ == '__main__':
351     print("Running schwingungsanalyse_main.py...")
352 
353     for i in range(4):
354         i += 1
355         run_analysis(f"Schwingungsanalyse_{i:02d}.txt")
```