

schwingungsanalyse_main.py

```
001 # Schwingungsanalyse
002 # #####
003
004 # This python script calculates the frequency of the occilation.
005
006 # Parts of this python script come from the following github-repository made by
007 # Christopher Mahn:
008 # https://github.com/c-mahn/test-sensor-data-analysis
009
010 # Authors:
011 # Joshua Wolf
012 # Silas Teske
013 # Lasse Zeh
014 # Christopher Mahn
015
016 # #####
017
018 # Import of Libraries
019 # -----
020
021 # import string as st
022 # import random as r
023 # import re
024 import matplotlib.pyplot as plt
025 from scipy import interpolate
026 import numpy as np
027 import math as m
028 import sys
029 import os
030 from scipy.fft import fft, fftfreq
031 from scipy import signal
032
033
034 # -----
035 # Settings
036
037 verbose = False # Shows more debugging information
038 fix_steigung = False # Disregards permanent sensor changes
039
040
041 # Functions
042 # -----
043
044 def run_analysis(input_file):
045
046     # Einlesen der Daten
047     print(f"[{1}/{1}] Einlesen der Daten...", end="\r")
048     with open(os.path.join("data", input_file), "r") as f:
049         data = f.readlines()
050     print("")
051
052     # Datenbereinigung
053     print(f"[{1}/{1}] Datenbereinigung...", end="\r")
054     for i, e in enumerate(data):
055         data[i] = e.split(";")
056         for j, e in enumerate(data[i]):
057             data[i][j] = float(e.strip())
058     print("")
059
060     # Umwandeln in Datenreihen
061     print(f"[{1}/{1}] Datenkonvertierung...", end="\r")
062     datenreihen = [[], [], [], []]
063     for i in data:
064         datenreihen[0].append(i[0])
065         datenreihen[1].append(i[1])
066         datenreihen[2].append(i[2])
067         datenreihen[3].append(i[3])
068     datenreihen_ohne_zeit = datenreihen[1:4]
069     print("")
070
071     plot_xyz_t(datenreihen_ohne_zeit, f'Messreihe "{input_file}"')
072
073     # Berechnung der linearen Regression von Sensor 1
074     x = np.array(datenreihen[0])
075     y = np.array(datenreihen[1])
076     steigung_1 = (len(x) * np.sum(x*y) - np.sum(x) * np.sum(y)) / (len(x)*np.sum(x*x) - np.sum(x) ** 2)
077     offset_1 = (np.sum(y) - steigung_1 * np.sum(x)) / len(x)
078     print(f'[Sensor 1] Trend: ({steigung_1:.10f})x + ({offset_1:.6f})')
079
080     # 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]], ["X", "Linearisierung"])
102 plot_werte([datenreihen[2], linearisierung[1]], ["Y", "Linearisierung"])
103 plot_werte([datenreihen[3], linearisierung[2]], ["Z", "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_xyz(datenreihen_ohne_trend, f'Messreihe "{input_file}" (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_xyz(datenreihen_low_pass, f'Messreihe "{input_file}" (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_xyz(datenreihen_hoch_pass, f'Messreihe "{input_file}" (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_xyz(datenreihen, name="Messwerte"):
200     """
201     Diese Funktion nimmt genau drei Datenreihen und plottet diese in ein Diagramm.
202
203     Args:
204         datenreihen ([list]): Drei Datenreihen zum Plotten.
205         name (list, optional): Dies ist der Titel. Defaults to "Messwerte".
206     """
207     for i, datenreihe in enumerate(datenreihen):
208         zeit = range(len(datenreihe))
209         plt.plot(zeit, datenreihe)
210     plt.legend(["x", "y", "z"])
211     plt.grid()
212     plt.xlabel("")
213     plt.ylabel("")
214     plt.title(name)
215     plt.show()
216
217
218 def plot_xy(datenreihen, name=["Messwerte"]):
219     """
220     Diese Funktion nimmt je zwei Datenreihen und plottet diese in Abhängigkeit
221     zueinander in ein Diagramm.
222     """
223     for i, datenreihe in enumerate(datenreihen):
224         plt.plot(datenreihe[0], datenreihe[1])
225     plt.legend(name)
226     plt.grid()
227     plt.xlabel("Y")
228     plt.ylabel("X")
229     plt.title(name[0])
230     plt.show()
231
232
233 def fill_nan(A):
234     """
235     interpolate to fill nan values
236     """
237     inds = np.arange(A.shape[0])
238     good = np.where(np.isfinite(A))
239     f = interpolate.interp1d(inds[good], A[good], bounds_error=False)
240     B = np.where(np.isfinite(A), A, f(inds))
241     return B
242
243
244 def low_pass_filter(datenreihe, filterungsgrad):
245     """

```

```

246 Diese Funktion macht einen vereinfachten Low-Pass-Filter, indem die letzten
247 x Sensorwerte gemittelt werden.
248 """
249 ausgabe = []
250 for i, e in enumerate(datenreihe):
251     divisor = filterungsgrad
252     summe = 0
253     for j in range(filterungsgrad):
254         ji = i-j
255         if(ji < 0): # Wenn Wert ausserhalb der Datenreihe, ändern des Divisors
256             divisor = divisor - 1
257         else:
258             summe = summe + float(datenreihe[ji])
259     temp = summe/divisor
260     ausgabe.append(temp)
261 return(ausgabe)
262
263
264 def cross_correlation(x_red, y_red):
265     Nx = len(x_red)
266     if Nx != len(y_red):
267         raise ValueError('x and y must be equal length')
268     c = np.correlate(x_red, y_red, mode=2)
269     c /= np.sqrt(np.dot(x_red, x_red) * np.dot(y_red, y_red))
270     maxlags = Nx - 1
271     if maxlags >= Nx or maxlags < 1:
272         raise ValueError('maglags must be None or strictly '
273                          'positive < %d' % Nx)
274     lags = np.arange(-maxlags, maxlags + 1)
275     c = c[Nx - 1 - maxlags:Nx + maxlags]
276     return lags, c
277
278
279 # Classes
280 # -----
281
282
283 # Beginning of the Programm
284 # -----
285
286 if __name__ == '__main__':
287     print("Running schwingungsanalyse_main.py...")
288
289     for i in range(4):
290         i += 1
291         run_analysis(f"Schwingungsanalyse_{i:02d}.txt")

```