

## Diff:

### Differences between given skeleton and solution

In order to make the sample solution easier to understand, the differences between it and the given skeleton source code were highlighted with the help of the program `diff`.

### Legend:

- Gray: unchanged text (only excerpts).
- Green: new lines
- Yellow: changed lines
- Red: deleted lines

Note: Files not listed have not been changed.

This document was created with the help of [diff2html](#) erstellt.

```
diff -u ../course06-data-processing-and-analysis/exercise/code/01_excercise.py ../course06-data-processing-and-analysis/exercise/solution/01_excercise.py
```

../course06-data-processing-and-analysis/exercise/code/01_excercise.py	../course06-data-processing-and-analysis/exercise/solution/01_excercise.py
:	:
7	7
8	8
9	9
10 # replace `XYZ` by some meaningful code	
11	
12 # to avoid runtime errors use `sys.exit()`	
13 # (and do not forget to move that line further down as you proceed)	
14	
15	
16	
17 ##### task 1	10 ##### task 1
18	11
19 data = np.loadtxt('../data/measurementdata.dat')	12 data = np.loadtxt('../data/measurementdata.dat')
:	:
41 # Note: if voltage == 0 then sign = 0, thus current is also set to 0	34 # Note: if voltage == 0 then sign = 0, thus current is also set to 0
42	35
43	36
	37 # alternatively use: ii[uu < 0 ] *= -1
	38
	39
44 # Visualize data	40 # Visualize data
45 if 0: #( 0 -> dont execute this block, 1 -> execute this block)	41 if 0: #( 0 -> dont execute this block, 1 -> execute this block)
46 plt.plot(tt, ii)	42 plt.plot(tt, ii)
:	:
48 plt.show()	44 plt.show()
49 sys.exit()	45 sys.exit()
50	46
51	
52 ##### task 3	47 ##### task 3
53	48
54 # goal: find out indices of voltage pulses	49 # goal: find out indices of voltage pulses
:	:
57 udiff = np.diff(uu)	52 udiff = np.diff(uu)
58	53
59 # create array for all indices	54 # create array for all indices
60 all_indices = np.arange(len(uu)-1)	55 all_indices = np.arange(len(uu)-1)
61	
62	56
63 # select those indices where the derivative does not vanish (using boolean indexing)	57 # select those indices where the derivative does not vanish (using boolean indexing)
64 # (if this is hard to understand have a look to the array idcs = (udiff != 0)	58 # (if this is hard to understand have a look to the array idcs = (udiff != 0)
65 idcs = (udiff != 0)	59 change_indices = all_indices[udiff != 0]
66 change_indices = all_indices[XYZ]	60
	61 # Prevent the possibility that a "half pulse" at the end is also detected.
	62 # if the length is an odd number (modulo-two calculation != 0), then drop the last value
	63 if len(change_indices) % 2 == 1:
	64 change_indices = change_indices[:-1] # omit last value
	65
67	66
68 if 0:	67 if 0:
69 # plot the derivative (x-axis: all_indices, automatically chosen)	68 # plot the derivative (x-axis: all_indices, automatically chosen)
:	:
75 sys.exit()	74 sys.exit()
76	75
77	76
78 # Prevent the possibility that a "half pulse" at the end is also detected.	

```

79 # if the length is an odd number (modulo-two calculation != 0), then drop the last value
80 if len(change_indices) % 2 == 1:
81     change_indices = change_indices[XYZ] # omit last value
82
83
84
85 # Until now the indices are one after the other
86 # We always want two in one line
87 # first column: pulse start index, second column: pulse end index)
88 XYZ = XYZ.reshape(-1, 2) # -1 means: "choose the row number such that it fits"
89
90 # increase all values in the first column by one, because the index refers to the
91 # last value before the jump.
92
93 XYZ[XYZ, XYZ] += 1
94
95 print(XYZ)
96
97
98 ##### task 4
99
:
:
102 # unpack the 4. row into two scalar values
103 idx1, idx2 = change_indices[3, :]
104
105 plt.hist(XYZ[XYZ])
106 plt.show()
107 sys.exit()
108
109
110 ##### task 5
111
112 # Take the average of the current
113
114 ii_mean = 0*ii # # create new ('empty') array
115
116
117 # Iterate over change_indices line by line
118 for XYZ, XYZ in change_indices:
119     XYZ[XYZ] = np.mean(XYZ) # Calculate mean values (and save them)
120
121
122 if 0:
123
124     plt.plot(tt, XYZ) # current-values with noise
125     plt.plot(tt, XYZ) # mean values of the current
126
127     plt.show()
128     sys.exit()
129
:
:
132 if 0:
133     plt.figure()
134
135     start_idcs = XYZ[:, 0] # first column: indices where a block (ore section) starts
136
137     # Determine a voltage-current value pair for each current block:

```

```

77 # Until now the indices are one after the other
78 # We always want two in one line
79 # first column: pulse start index, second column: pulse end index)
80 change_indices = change_indices.reshape(-1, 2) # -1 means: "choose the row number such that
it fits"
81
82 # increase all values in the first column by one, because the index refers to the
83 # last value before the jump.
84
85 change_indices[:, 0] += 1
86
87 print(change_indices)
88
89 ##### task 4
90
:
:
93 # unpack the 4. row into two scalar values
94 idx1, idx2 = change_indices[3, :]
95
96 plt.hist(I[idx1:idx2])
97 plt.show()
98 sys.exit()
99
100 ##### task 5
101
102 # Take the average of the current
103
104 ii_mean = 0*ii # # create new ('empty') array
105
106 # Iterate over change_indices line by line
107 for idx1, idx2 in change_indices:
108     ii_mean[idx1:idx2] = np.mean(ii[idx1:idx2]) # Calculate mean values (and save them)
109
110
111 if 0:
112
113     plt.plot(tt, ii) # current-values with noise
114     plt.plot(tt, ii_mean) # mean values of the current
115
116     plt.show()
117     sys.exit()
118
:
:
121 if 0:
122     plt.figure()
123
124     start_idcs = change_indices[:, 0] # first column: indices where a block (ore section)
starts
125
126     # Determine a voltage-current value pair for each current block:

```

138 ii2 = ii_mean[XYZ]	127 ii2 = ii_mean[start_idcs]
139 uu2 = uu[XYZ]	128 uu2 = uu[start_idcs]
140	129
141 plt.plot(uu2, ii2, 'bx', ms=7) # big blue crosses (x)	130 plt.plot(uu2, ii2, 'bx', ms=7) # big blue crosses (x)
142	131
143 a1, a0 = sc.polyfit(XYZ, XYZ, XYZ) # lineare regression	132 a1, a0 = sc.polyfit(uu2, ii2, 1) # lineare regression
144	133
145 plt.plot(XYZ, XYZ, 'g-') # Evaluate and plot polynomial (straight line equation)	134 plt.plot(uu2, a1*uu2+a0, 'g-') # Evaluate and plot polynomial (straight line equation)
146 # alternatively use: sc.polyval [a1, a0]	135 # alternatively use: sc.polyval [a1, a0]
147	136
148 print("conductivity (inverse resistance):", a1)	137 print("conductivity (inverse resistance):", a1)
:	:
152 sys.exit()	141 sys.exit()
153	142
154	143
155	
156##### task 7	144##### task 7
157	145
158# step size of the time array (assuming it starts at 0)	146# step size of the time array (assuming it starts at 0)
159dt = tt[1]	147dt = tt[1]
160	148
161# calc velocity and acceleration via np.diff	149# calc velocity and acceleration via np.diff
162xd = XYZ	150xd = np.diff(xx1)/dt
163xdd = XYZ	151xdd = np.diff(xx1, 2)/dt**2
164	
165	152
166##### task 8	153##### task 8
167if 0:	154if 0:
:	:
172 if uu[idx1] < 0:	159 if uu[idx1] < 0:
173 continue # this continues with the next iteration (omitting the plot)	160 continue # this continues with the next iteration (omitting the plot)
174	161
175 plt.plot(XYZ[idx1:idx2-1], XYZ[XYZ])	162 plt.plot(xd[idx1:idx2-1], xdd[idx1:idx2-1])
176	163
177 plt.show()	164 plt.show()
178 sys.exit()	165 sys.exit()
179	166
180	
181##### task 9	167##### task 9
182	168
183# see	169# see
:	:
191# We work first with lists (can be concatenated more easily).	177# We work first with lists (can be concatenated more easily).
192# At the end we convert the lists into arrays.	178# At the end we convert the lists into arrays.
193	179
194	
195points_vel = []	180points_vel = []
196points_acc = []	181points_acc = []
197voltage = []	182voltage = []
198	183
199for idx1, XYZ in change_indices:	184for idx1, idx2 in change_indices:
200	185
201 # ignore negative values	186 # ignore negative values
202 if uu[idx1] < 0:	187 if uu[idx1] < 0:
203 continue	188 continue
204	189
205 points_vel += list(XYZ)	190 points_vel += list(xd[idx1:idx2-1])
206 points_acc += list(XYZ)	191 points_acc += list(xdd[idx1:idx2-1])

207	192
208 # List of the appropriate length in which all elements have the same value.	193 # List of the appropriate length in which all elements have the same value.
209 # namely the matching voltage value	194 # namely the matching voltage value
210 length = (idx2-1-idx1)	195 length = (idx2-1-idx1)
211 voltage += [XYZ[XYZ]]*length	196 voltage += [uu[idx1]]*length
212	197
213	198
214# Workaround for interpolation:	199# Workaround for interpolation:
215# Add pseudo-measurement values at the boundary to avoid nan-values ("not-a-number").	200# Add pseudo-measurement values at the boundary to avoid nan-values ("not-a-number").
216# Assumption: at 3V still (almost) nothing moves.	201# Assumption: at 3V still (almost) nothing moves.
217points_vel = [0, 0, 0, 7, 7] + points_vel	202points_vel = [0, 0, 0, 7, 7] + points_vel
218points_acc = [0, 3, 14, 0, 14] + points_acc	203points_acc = [0, 3, 14, 0, 14] + points_acc
219voltage = [3, 3, 12, 12, 12] + voltage	204voltage = [3, 3, 12, 12, 12] + voltage
220	
221	205
222# Pack lists together as arrays:	206# Pack lists together as arrays:
223points = np.array([points_vel, points_acc]).T	207points = np.array([points_vel, points_acc]).T
:	:
243interp_voltage[:, 0] = interp_voltage[:, 1] # first column := second column	227interp_voltage[:, 0] = interp_voltage[:, 1] # first column := second column
244interp_voltage[0, :] = interp_voltage[1, :] # first row:= second row	228interp_voltage[0, :] = interp_voltage[1, :] # first row:= second row
245	229
246if 0:	230if 1:
247 # Display 2d array graphically, see https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.imshow.html	231 # Display 2d array graphically, see https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.imshow.html
248 plt.figure()	232 plt.figure()
249 plt.imshow(interp_voltage,	233 plt.imshow(interp_voltage,
:	:
268 # The faster the car is, the more engine power is needed for	252 # The faster the car is, the more engine power is needed for
269 # maintaining the speed -> acceleration decreases.	253 # maintaining the speed -> acceleration decreases.
270	254
	255
	256 plt.savefig("res.pdf")
	257
271 plt.show()	258 plt.show()
272 sys.exit()	259 sys.exit()
273	260
:	:
288 idx_a= int( a/xdd_max*N_grid )	275 idx_a= int( a/xdd_max*N_grid )
289	276
290 # evaluate the 2d array containing the interpolated values at those indices	277 # evaluate the 2d array containing the interpolated values at those indices
291 return interp_voltage[XYZ, XYZ]*s	278 return interp_voltage[idx_v, idx_a]*s
292	279
293##### task 11	280##### task 11
294	281
295# load swingup data	282# load swingup data
296data = np.load('XYZ')	283data = np.load('../data/swingup.npy')
297	284
298tt, x1, x2, x3, x4, acc = XYZ	285tt, x1, x2, x3, x4, acc = data.T
299	286
300uu_s = acc*0	287uu_s = acc*0
301	288
:	:
304 a = acc[idx]	291 a = acc[idx]
305 v = x2[idx]	292 v = x2[idx]
306	293
307 uu_s[idx] = calc_voltage(XYZ, XYZ)	294 uu_s[idx] = calc_voltage(v, a)
308	295
309if 1:	296if 1:

```
310 plt.figure()
```

```
297 plt.figure()
```

```
Nur in ../course06-data-processing-and-analysis/exercise/solution/: res.pdf.
```