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.	

```
# if the length is an odd number (modulo-two calculation != 0), then drop the last value
if len(change_indices) % 2 == 1:
    change_indices = change_indices[XYZ]  # omit last value



# Until now the indices are one after the other
# We always want two in one line
# first column: pulse start index, second column: pulse end index)
XYZ = XYZ.reshape(-1, 2) # -1  means: "choose the row number such that it fits"

# increase all values in the first column by one, because the index refers to the
# last value before the jump.

XYZ[XYZ, XYZ] += 1

print(XYZ)


######### task 4

⋮
    # unpack the 4. row into two scalar values
    idx1, idx2 = change_indices[3, :]

    plt.hist(XYZ[XYZ])
    plt.show()
    sys.exit()


######### task 5

# Take the average of the current

ii_mean = 0*ii # # create new ('empty') array


# Iterate over change_indices line by line
for XYZ, XYZ in change_indices:
    XYZ[XYZ] = np.mean(XYZ) # Calculate mean values (and save them)


if 0:

    plt.plot(tt, XYZ) # current-values with noise
    plt.plot(tt, XYZ) # mean values of the current

    plt.show()
    sys.exit()
⋮
if 0:
    plt.figure()

    start_idcs = XYZ[:, 0] # first column: indices where a block (ore section) starts

    # Determine a voltage-current value pair for each current block:
```

```
# Until now the indices are one after the other
# We always want two in one line
# first column: pulse start index, second column: pulse end index)
change_indices = change_indices.reshape(-1, 2) # -1  means: "choose the row number such that it fits"

# increase all values in the first column by one, because the index refers to the
# last value before the jump.

change_indices[:, 0] += 1

print(change_indices)

######### task 4

⋮
    # unpack the 4. row into two scalar values
    idx1, idx2 = change_indices[3, :]

    plt.hist(I[idx1:idx2])
    plt.show()
    sys.exit()

######### task 5

# Take the average of the current

ii_mean = 0*ii # # create new ('empty') array

# Iterate over change_indices line by line
for idx1, idx2 in change_indices:
    ii_mean[idx1:idx2] = np.mean(ii[idx1:idx2]) # Calculate mean values (and save them)


if 0:

    plt.plot(tt, ii)  # current-values with noise
    plt.plot(tt, ii_mean)  # mean values of the current

    plt.show()
    sys.exit()
⋮
if 0:
    plt.figure()

    start_idcs = change_indices[:, 0] # first column: indices where a block (ore section) starts

    # Determine a voltage-current value pair for each current block:
```

```python
    ii2 = ii_mean[XYZ]
    uu2 = uu[XYZ]

    plt.plot(uu2, ii2, 'bx', ms=7)  # big blue crosses (x)

    a1, a0 =  sc.polyfit(XYZ, XYZ, XYZ)  # lineare regression

    plt.plot(XYZ, XYZ, 'g-')  # Evaluate and plot polynomial (straight line equation)
    # alternatively use: sc.polyval [a1, a0]

    print("conductivity (inverse resistance):", a1)
⋮
    sys.exit()




######### task 7

# step size of the time array (assuming it starts at 0)
dt = tt[1]

# calc velocity and acceleration via np.diff
xd = XYZ
xdd = XYZ


######### task 8
if 0:
⋮
        if uu[idx1] < 0:
            continue  # this continues with the next iteration (omitting the plot)

        plt.plot(XYZ[idx1:idx2-1], XYZ[XYZ])

    plt.show()
    sys.exit()


######### task 9

# see
⋮
# We work first with lists (can be concatenated more easily).
# At the end we convert the lists into arrays.


points_vel = []
points_acc = []
voltage = []

for idx1, XYZ in change_indices:

    # ignore negative values
    if uu[idx1] < 0:
        continue

    points_vel += list(XYZ)
    points_acc += list(XYZ)
```

```python
    ii2 = ii_mean[start_idcs]
    uu2 = uu[start_idcs]

    plt.plot(uu2, ii2, 'bx', ms=7)  # big blue crosses (x)

    a1, a0 =  sc.polyfit(uu2, ii2, 1)  # lineare regression

    plt.plot(uu2, a1*uu2+a0, 'g-')  # Evaluate and plot polynomial (straight line equation)
    # alternatively use: sc.polyval [a1, a0]

    print("conductivity (inverse resistance):", a1)
⋮
    sys.exit()


######### task 7

# step size of the time array (assuming it starts at 0)
dt = tt[1]

# calc velocity and acceleration via np.diff
xd = np.diff(xx1)/dt
xdd = np.diff(xx1, 2)/dt**2

######### task 8
if 0:
⋮
        if uu[idx1] < 0:
            continue  # this continues with the next iteration (omitting the plot)

        plt.plot(xd[idx1:idx2-1], xdd[idx1:idx2-1])

    plt.show()
    sys.exit()

######### task 9

# see
⋮
# We work first with lists (can be concatenated more easily).
# At the end we convert the lists into arrays.

points_vel = []
points_acc = []
voltage = []

for idx1, idx2 in change_indices:

    # ignore negative values
    if uu[idx1] < 0:
        continue

    points_vel += list(xd[idx1:idx2-1])
    points_acc += list(xdd[idx1:idx2-1])
```

```python

    # List of the appropriate length in which all elements have the same value.
    # namely the matching voltage value
    length = (idx2-1-idx1)
    voltage += [XYZ[XYZ]]*length


# Workaround for interpolation:
# Add pseudo-measurement values at the boundary to avoid nan-values ("not-a-number").
# Assumption: at 3V still (almost) nothing moves.
points_vel = [0, 0,  0,  7,  7] + points_vel
points_acc = [0, 3, 14,  0, 14] + points_acc
voltage =    [3, 3, 12, 12, 12] + voltage


# Pack lists together as arrays:
points = np.array([points_vel, points_acc]).T
⋮
interp_voltage[:, 0] = interp_voltage[:, 1]  # first column := second column
interp_voltage[0, :] = interp_voltage[1, :]  # first row:= second row

if 0:
    # Display 2d array graphically, see https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.imshow.html
    plt.figure()
    plt.imshow(interp_voltage,
⋮
    # The faster the car is, the more engine power is needed for
    # maintaining the speed -> acceleration decreases.

    plt.show()
    sys.exit()

⋮
    idx_a= int( a/xdd_max*N_grid )

    # evaluate the 2d array containing the interpolated values at those indices
    return interp_voltage[XYZ, XYZ]*s

###### task 11

# load swingup data
data = np.load('XYZ')

tt, x1, x2, x3, x4, acc = XYZ

uu_s = acc*0

⋮
    a = acc[idx]
    v = x2[idx]

    uu_s[idx] = calc_voltage(XYZ, XYZ)

if 1:
```

```python

    # List of the appropriate length in which all elements have the same value.
    # namely the matching voltage value
    length = (idx2-1-idx1)
    voltage += [uu[idx1]]*length


# Workaround for interpolation:
# Add pseudo-measurement values at the boundary to avoid nan-values ("not-a-number").
# Assumption: at 3V still (almost) nothing moves.
points_vel = [0, 0,  0,  7,  7] + points_vel
points_acc = [0, 3, 14,  0, 14] + points_acc
voltage =    [3, 3, 12, 12, 12] + voltage

# Pack lists together as arrays:
points = np.array([points_vel, points_acc]).T
⋮
interp_voltage[:, 0] = interp_voltage[:, 1]  # first column := second column
interp_voltage[0, :] = interp_voltage[1, :]  # first row:= second row

if 1:
    # Display 2d array graphically, see https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.imshow.html
    plt.figure()
    plt.imshow(interp_voltage,
⋮
    # The faster the car is, the more engine power is needed for
    # maintaining the speed -> acceleration decreases.


    plt.savefig("res.pdf")

    plt.show()
    sys.exit()

⋮
    idx_a= int( a/xdd_max*N_grid )

    # evaluate the 2d array containing the interpolated values at those indices
    return interp_voltage[idx_v, idx_a]*s

###### task 11

# load swingup data
data = np.load('../data/swingup.npy')

tt, x1, x2, x3, x4, acc = data.T

uu_s = acc*0

⋮
    a = acc[idx]
    v = x2[idx]

    uu_s[idx] = calc_voltage(v, a)

if 1:
```

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