

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 ../course03-numerical-computation/exercise/code/01_simulation.py ../course03-numerical-computation/exercise/solution/01_simulation.py
```

```
../course03-numerical-computation/exercise/code/01_simulation.py
```

```
../course03-numerical-computation/exercise/solution/01_simulation.py
```

```
1 import sys
2 import numpy as np
3 from numpy import r_, pi
4 from matplotlib import pyplot as plt # used for the plotting at the end
5
6
7 # Move the following line further down as you are advancing.
8 # Background: actively exiting the program here prevents errors,
9 # due to usage of undefined names like `XXX`.
10
11
12 sys.exit() # Ends the program here. Otherwise: error messages
13
14
15
16
17 # Task 1:
18
19 # import the function solve_ivp from the package scipy.integrate
20 from scipy.XXX import XXX
21
22 # import two functions for calculating the accelerations
23 # (look inside the file `equations_of_motion.py`!)
24 from equations_of_motion import xdd_fnc #, XXX
25
26
27 # Task 2:
28
29 def rhs(XXX, XXX):
30     # This function calculates the time derivative z_dot from the state z
31     # the 1st argument (the time t) is not needed here
32
33     x, phi, xd, phid = z # unpacking (see overview slides in course01)
34     F = 0
35
36     xdd = xdd_fnc(XXX, XXX, ...)
37     phidd = XXX
38
39
40     # Return the derivative of the state vector
41     z_dot = r_[xd, phid, XXX, XXX]
42     return z_dot
43
44
45
46
47
48 # Task 3:
49
50 zz0 = np.array([XXX, pi*0.5, XXX, XXX])
51
52
53 # Task 4:
54
55 # do the numerical integration
56 res = solve_ivp(XXX, (tt[0], tt[-1]), XXX, t_eval=tt, rtol=1e-5)
```

```
1 import sys
2 import numpy as np
3 from numpy import r_, pi
4 from matplotlib import pyplot as plt # used for the plotting at the end
5
6
7 # Task 1:
8
9 # import the function solve_ivp from the package scipy.integrate
10 from scipy.integrate import solve_ivp
11
12 # import two functions for calculating the accelerations
13 # (look inside the file `equations_of_motion.py`!)
14 from equations_of_motion import xdd_fnc, phidd_fnc
15
16
17 # Task 2:
18
19 def rhs(t, z):
20     # This function calculates the time derivative z_dot from the state z
21     # the 1st argument (the time t) is not needed here
22
23     x, phi, xd, phid = z # unpacking (see overview slides in course01)
24     F = 0
25
26     xdd = xdd_fnc(x, phi, xd, phid, F)
27     phidd = phidd_fnc(x, phi, xd, phid, F)
28
29     # Return the derivative of the state vector
30     z_dot = r_[xd, phid, xdd, phidd]
31     return z_dot
32
33
34
35
36
37 # Task 3:
38
39 zz0 = np.array([0, pi*0.5, 0, 0])
40
41
42 # Task 4:
43
44 # do the numerical integration
45 res = solve_ivp(rhs, (tt[0], tt[-1]), zz0, t_eval=tt, rtol=1e-5)
```

```

57
58 # res: result container
59 # res.y: result array with shape (4, 1001)
60 # rows -> state components, columns -> time steps.
61

```

```

62
63 # Task 5:
64
65 # Unpacking of individual state components.
66 # Arrays are always unpacked along the 1st axis (rows).
67 x, phi, xd, phid = XXX

```

```

68
69 # visualization (more on this in course04):
70 plt.plot(tt, x)
71 plt.plot(XYZ)
72 plt.show()

```

```

46 # res: result container
47 # res.y: result array with shape (4, 1001)
48 # rows -> state components, columns -> time steps.
49
50

```

```

51 # Task 5:
52
53 # Unpacking of individual state components.
54 # Arrays are always unpacked along the 1st axis (rows).
55 x, phi, xd, phid = res.y

```

```

56
57
58 # visualization (more on this in course04):
59 plt.plot(tt, x)
60 plt.plot(tt, phi)
61 plt.show()

```

```

62
63
64 ## The following code is not part of the exercise03.1 but conveniently stored here
65 ## by the supervisor. It is used to generate "pseudo-measurement data"
66 ## for exercise03.2.
67 ## The block will not be executed, but can be quickly converted to "active code"
68 ## by replacing `0` with `1` in the if-statement.
69
70 if 0:
71     # binary format:
72     np.save('measurement-data.npy', res.y)
73     # text format (human readable, but needs more memory):
74     np.savetxt('measurement-data.txt', res.y)
75
76     print ("Files written.")

```

```
diff -u ../course03-numerical-computation/exercise/code/02_identification.py ../course03-numerical-computation/exercise/solution/02_identification.py
```

```
../course03-numerical-computation/exercise/code/02_identification.py
```

```
../course03-numerical-computation/exercise/solution/02_identification.py
```

```

:
:
1
2 # Move the following line further down as you are advancing.
3 # Background: actively exiting the program here prevents errors,
4 # due to usage of undefined names like `XXX`.
5
6

```

```

7 sys.exit() # Ends the program here. Otherwise: error messages
8
9

```

```

15 import sys
16 import numpy as np
17

```

```

19 # Task 1:
20
21 # import the function solve_ivp from the package scipy.integrate
22
23

```

```

1 import sys
2 import numpy as np
3

```

```

5 # Task 1:
6
7 # import the function solve_ivp from the package scipy.integrate
8 from scipy.integrate import solve_ivp
9

```

```

24 # import two functions for calculating the accelerations
25 # (look inside the file `equations_of_motion.py`)
26 from equations_of_motion import xdd_fnc, XXX
27
28
29 # load pseudo measurement data in binary format
30 zz_res_target = np.load(XXX)
31
32 # -> this is a 2d array with shape = (4, 1001), i.e. 4 rows, 1001 cols.
33 # meaning of rows: x, phi, xd, phid
34 # meaning of cols: time instant
35
36
37 ## alternatively: load data in text format:
38 # zz_res_target = np.loadtxt(XXX)
39
40
41 # Task 2:
42
43 :
44 :returns: err – non-negative real valued error measure
45 :          (how "wrong the simulation result is")
46 :
47 :"""
48
49 m2, l = XXX # unpacking the parameter vector
50
51
52 # Task 3:
53
54 :
55 :
56 Righthand side of the equations of motion
57 (Note: this depends on m2 and l from the surrounding namespace).
58 """
59
60 x, phi, xd, phid = XXX # unpacking
61 F = 0
62
63 # m2 and l come from the namespace one level higher
64 # (you might want to look again into `equations_of_motion.py`)
65 # to check the signature of these functions:)
66 xdd = xdd_fnc(XXXXXXXXX, m2, l)
67 phidd = phidd_fnc(XXXXXXXXX, XXX, XXX)
68
69 # return derivative of the state vector
70 return XXX
71
72
73 # end of the inner function definition of rhs
74
75 :
76
77 # them here:
78
79 # array with evaluation times (should be consistent with the measured data)
80 tt = np.linspace(0, 10, XXX)
81
82 # select a consistent initial state (4 values) for the simulation
83 # from the measurement data (-> choose the first column)
84 zz0 = XXX
85
86 # do the simulation (get result container)
87 sim_res = solve_ivp(XXX, (XXXX, XX), YYY, t_eval=tt, rtol=1e-5)
88
89

```

```

10 # import two functions for calculating the accelerations
11 # (look inside the file `equations_of_motion.py`)
12 from equations_of_motion import xdd_fnc, phidd_fnc
13
14
15 # load pseudo measurement data in binary format
16 zz_res_target = np.load('measurement-data.npy')
17
18 # -> this is a 2d array with shape = (4, 1001), i.e. 4 rows, 1001 cols.
19 # meaning of rows: x, phi, xd, phid
20 # meaning of cols: time instant
21
22
23 ## alternatively: load data in text format:
24 # zz_res_target = np.loadtxt('measurement-data.txt')
25
26
27 # Task 2:
28
29 :
30 :returns: err – non-negative real valued error measure
31 :          (how "wrong the simulation result is")
32 :
33 :"""
34
35 m2, l = p # unpacking the parameter vector
36
37
38 # Task 3:
39
40 :
41 :
42 Righthand side of the equations of motion
43 (Note: this depends on m2 and l from the surrounding namespace).
44 """
45
46 x, phi, xd, phid = z # unpacking
47 F = 0
48
49 # m2 and l come from the namespace one level higher
50 # (you might want to look again into `equations_of_motion.py`)
51 # to check the signature of these functions:)
52 xdd = xdd_fnc(x, phi, xd, phid, F, m2, l)
53 phidd = phidd_fnc(x, phi, xd, phid, F, m2, l)
54
55 # return derivative of the state vector
56 return np.array([xd, phid, xdd, phidd])
57
58
59 # end of the inner function definition of rhs
60
61 :
62
63 # them here:
64
65 # array with evaluation times (should be consistent with the measured data)
66 tt = np.linspace(0, 10, 1001)
67
68 # select a consistent initial state (4 values) for the simulation
69 # from the measurement data (-> choose the first column)
70 zz0 = zz_res_target[:, 0]
71
72 # do the simulation (get result container)
73 sim_res = solve_ivp(rhs, (tt[0], tt[-1]), zz0, t_eval=tt, rtol=1e-5)
74
75

```

102	# select the state vector (which we call "z" but scipy calls "y")	85	# select the state vector (which we call "z" but scipy calls "y")
103	zz_res = XXX.y	86	zz_res = sim_res.y
104		87	
105	# Task 5:	88	# Task 5:
106		89	
107	# Calculate the difference of the x-positions (first line in each case)	90	# Calculate the difference of the x-positions (first line in each case)
108	# then square (...**2),	91	# then square (...**2),
109	# then add up (applying np.sum)	92	# then add up (applying np.sum)
110	err = np.sum((XXX[YYY, ZZZ] - XXX[YYY, ZZZ])**XXX)	93	err = np.sum((zz_res[0, :] - zz_res_target[0, :])**2)
111		94	
112	# Status message and output (to assess progress of optimization)	95	# Status message and output (to assess progress of optimization)
113	print("simulation ready. p =", p, " equation error:", err)	96	print("simulation ready. p =", p, " equation error:", err)
:		:	
119		102	
120	# Task 6:	103	# Task 6:
121		104	
122	p0 = np.array([.5, .7]) # Startschätzung für m2 und l	105	p0 = np.array([.5, .7]) # initial guess for m2 and l
123		106	
124	# import the function minimize from the module scipy.optimize	107	# import the function minimize from the module scipy.optimize
125	from XXX.XXX import XXX	108	from scipy.optimize import minimize
126		109	
127	# do the optimization (call the algorithm, which internally repeatedly calls min_target)	110	# do the optimization (call the algorithm, which internally repeatedly calls min_target)
128	min_res = minimize(XXX, XXX, method="Nelder-Mead")	111	min_res = minimize(min_target, p0, method="Nelder-Mead")
		112	
129		113	
130	print("\n", "minimization result (data structure):", min_res, "\n")	114	print("\n", "minimization result (data structure):", min_res, "\n")
131	print("estimated paremeters (m2, l):", min_res.x, "\n")	115	print("estimated paremeters (m2, l):", min_res.x, "\n")