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 ../course07-advanced-programming-techniques/exercise/code/data1.txt ../course07-advanced-programming-techniques/exercise/solution/data1.txt

../course07-advanced-programming-techniques/exercise/code/data1.txt	../course07-advanced-programming-techniques/exercise/solution/data1.txt
⋮	⋮
-15	-15
2	2
-7	-7
	# stabiles System (schwingungsfähig)

diff -u ../course07-advanced-programming-techniques/exercise/code/data3.txt ../course07-advanced-programming-techniques/exercise/solution/data3.txt

../course07-advanced-programming-techniques/exercise/code/data3.txt	../course07-advanced-programming-techniques/exercise/solution/data3.txt
⋮	⋮
2	2
5.1	5.1
-6	-6
	# instabiles System

diff -u ../course07-advanced-programming-techniques/exercise/code/data4.txt ../course07-advanced-programming-techniques/exercise/solution/data4.txt

../course07-advanced-programming-techniques/exercise/code/data4.txt	../course07-advanced-programming-techniques/exercise/solution/data4.txt
⋮	⋮
0	0
2	2
-7	-7
	# stabiles System (nicht schwingungsfähig)

diff -u ../course07-advanced-programming-techniques/exercise/code/data_tools.py ../course07-advanced-programming-techniques/exercise/solution/data_tools.py

../course07-advanced-programming-techniques/exercise/code/data_tools.py

```
# this file is initially empty and is to be completed in the course of the exercise (cf. task 10)
```

../course07-advanced-programming-techniques/exercise/solution/data_tools.py

```python
"""
Module with auxilliary functions

"""

import numpy as np



def create_rhs_from_1darr(arr):
    n = arr.shape[0]
    n2 = int(np.sqrt(n))
    arr2 = arr.reshape(n2, -1)

    return rhs_factory(arr2)


def rhs_factory(A):
    """
    factory function, to "produce" a `solve_ivp`-compatible
    rhs function based on a matrix `A`.
    """

    n, m = A.shape
    # ensure that A is a square matrix
    assert n == m

    # define the new function (this is the 'product' of the factory)
    def rhs(time, state):
        # ODE: derivative of the state is Matrix A times state vector
        x_dot = np.dot(A, state)  # alternative: A@state
```

```
        return x_dot

    # add the state dimension as additional attribute to the function object
    rhs.state_dimension = n


    # return the procuct of the fatory (the created rhs function)
    return rhs
```

Nur in ../course07-advanced-programming-techniques/exercise/solution/: __pycache__.

diff -u ../course07-advanced-programming-techniques/exercise/code/simulation.py ../course07-advanced-programming-techniques/exercise/solution/simulation.py

../course07-advanced-programming-techniques/exercise/code/simulation.py	../course07-advanced-programming-techniques/exercise/solution/simulation.py
⋮	⋮

../course07-advanced-programming-techniques/exercise/code/simulation.py

```


# In this exercise the order of the given code snippets is arbitrary
# For each task you select the appropriate the block(s), uncomment and
# make your adaptions at `(...)`

# Use the ability of Spyder (or another IDE), to move blocks
# with multiple lines and to blockwise (un)comment!


#import numpy as ...
#from scipy.integrate import odeint
#import matplotlib.pyplot as plt

# optional debugging tool
#from ipydex import IPS

# ###############################################################

# np.loadtxt(...)

# ###############################################################

#for k in range(1, ...):
#    fname = f"data{k}.txt"
#    print(fname)
#    try:
#        x = np.loadtxt(...)
#    except ValueError as ve:
#        print("Error:", ve)
#    else:
#        # Task 3:
#        rhs = create_rhs_from_1darr(x)
#        rhs_list.append(rhs)
```

../course07-advanced-programming-techniques/exercise/solution/simulation.py

```

import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt

from data_tools import create_rhs_from_1darr

def simulate(rhs):
    """
    Perform the simulation for a given rhs function object

    :param rhs:     `solve_ivp`-compatible function object

    :return:    None
    """

    np.random.seed(75)  # initialize random generator -> reproducibility
    xx0 = np.random.rand(rhs.state_dimension)

    # run the simulation
    # (tt is global variable, (tt[0], tt[-1]) is a 2-tuple with first and last time instant)
    res = solve_ivp(rhs, (tt[0], tt[-1]), xx0, t_eval=tt)

    # Extract time evolution of the first state component
    x1 = res.y[0, :]

    plt.plot(tt, x1)



# create a list for the function objects
rhs_list = []
```

```python

# ################################################################

#def rhs_factory(A):
#   """
#   factory function, to "produce" a `solve_ivp`-compatible
#   rhs function based on a matrix `A`.
#   """
#
#    # ensure that A is a square matrix
#    assert ...
#
#
#    # define the new function (this is the 'product' of the factory)
#    def rhs(..., ...):
#        # ODE: derivative of the state is Matrix A times state vector
#        ...
#        return x_dot
#
#
#    # add the state dimension as additional attribute to the function object
#    rhs.state_dimension = ...
#
#    # return the procuct of the fatory (the created rhs function)
#    return rhs

# ################################################################

#def create_rhs_from_1darr(arr):
#    n = arr.shape[0]
#    n2 = int(np.sqrt(n))
#    arr2 = arr.reshape(n2, -1)
#
#    return rhs_factory(arr2)



# ################################################################

# implement equation x_dot = A*x:
```

```python

for k in range(1, 5):
   fname = f"data{k}.txt"
   print(fname)
   try:
       x = np.loadtxt(fname)
   except ValueError as ve:
       print("Error:", ve)
   else:
       rhs = create_rhs_from_1darr(x)
       rhs_list.append(rhs)

tt = np.linspace(0, 5, int(1e3))


# two different variants to restrict the simulation to systems with
# state dimension smaller than 3 (see task 9)

if 0:  # switch filtering on/off completely
   if 0:  # distinguish between `filter`-func and list comprehension
       rhs_list = filter(lambda q: q.state_dimension < 3, rhs_list)
   else:
       # task 11 (part 1)
       rhs_list = [q for q in rhs_list if q.state_dimension < 3]


# Apply the `simulate` function from above.
# `map(...)` creates an iterator
# `list(...)` evaluates the iterator and thereby causes the actual execution
# the application of the `simulate` function:

res = list(map(simulate, rhs_list))
```

```
#def rhs(time, state):
#    x_dot = np.dot(A, state)
#
#    return x_dot
```

```
# task 11 (part 2)
res = [simulate(rhs_func) for rhs_func in rhs_list]
```

```
# ##############################################################

#rhs_list = []

# ##############################################################

# apply the `simulate` function (3 options).
# option a): classic by ordinary for-loop

#for rhs in rhs_list:
#    simulate(...)


# option b): in functional programming style with `map`
# `map(...)` creates an iterator object
# `list(...)` iterates over such an iterator object and thus causes the execution of the function
#list(map(...))


# ##############################################################

#plt.show()

# ##############################################################

# Task 9
#rhs_list = filter(lambda ...)

# ##############################################################

#tt = np.linspace(0, 5, int(1e3))

#def simulate(rhs):
#    """
#    Perform the simulation for a given rhs function object
#
#    :param rhs:     `solve_ivp`-compatible function object
#
#    :return:     None
#    """
#
#    np.random.seed(75)  # initialize random generator -> reproducibility
#    xx0 = np.random.rand(rhs.state_dimension)
#
#    # run the simulation
#    # (tt is global variable, (tt[0], tt[-1]) is a 2-tuple with first and last time instant)
#    res = solve_ivp(rhs, (tt[0], tt[-1]), xx0, t_eval=tt)
#
```

```
plt.show()
```

```
#
#     # Extract time evolution of the first state component
#     x1 = res.y[0, :]
#
#     plt.plot(tt, x1)

```