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 ../course09-symbolic-computation/exercise/code/lagrange.py ../course09-symbolic-computation/exercise/solution/lagrange.py

../course09-symbolic-computation/exercise/code/lagrange.py	../course09-symbolic-computation/exercise/solution/lagrange.py
⋮	⋮
import sys	import sys
## this file contains a skeleton and has to be amended	
# the variable XXX is only a placeholder and has to be replaced appropriately in each case	
# task 1	# task 1
params = sp.symbols("m1, m2") # incomplete (parameters are missing)	params = sp.symbols("m1, m2, l, g")
m1, m2 = params	m1, m2, l, g = params
sys.exit() # move this line further down as you proceed with the exercise	
# task 2	# task 2
t = sp.Symbol("XXX") # create the symbol for the time	t = sp.Symbol("t") # create the symbol for the time
xt = Function("x")(t) # x(t)	xt = Function("x")(t) # x(t)
phit = Function(XXX)(XXX) # phi(t)	phit = Function("phi")(t) # phi(t)
# task 3 (construct four time derivatives)	# task 3 (construct four time derivatives)
xdt = xt.diff(t)	xdt = xt.diff(t)
## ...	phidt = phit.diff(t)
xddt = xt.diff(t, 2)	xddt = xt.diff(t, 2)
## ...	phiddt = phit.diff(t, 2)
# task 4	# task 4
# auxiliary quantities	# auxiliary quantities
x2t = XXX	x2t = xt + l*sin(phit)
y2t = XXX	y2t = -l*cos(phit)
x2dt = XXX.diff(t)	x2dt =x2t.diff(t)
## ...	y2dt =y2t.diff(t)
# task 5	# task 5
# kinetic energy	# kinetic energy
T = (m1*xdt**2 + XXX)/2	T = (m1*xdt**2 + m2*(x2dt**2 + y2dt**2))/2
# potential energy	# potential energy
U = XXX	U = y2t*g*m2
L = T - U # Lagrange funktion	L = T - U # Lagrange funktion
⋮	⋮
# auxiliary expressions:	# auxiliary expressions:
L_d_x = L.diff(xt)	L_d_x = L.diff(xt)
L_d_phi = XXX	L_d_phi = L.diff(phit)
L_d_xd = XXX	L_d_xd = L.diff(xdt)
L_d_phid = XXX	L_d_phid = L.diff(phidt)
# task 7	# task 7
DL_d_xd = XXX.diff(t)	DL_d_xd = L_d_xd.diff(t)

```
DL_d_phid = XXX


# task 8
F = sp.Symbol("F")  # external force (translatoric)

# right hand side of the equations of motion (left hand side is zero)
Eq1 =  XXX - XXX - F
Eq2 =  XXX

# useful for debugging: pretty printing
# sp.pprint(Eq1)
# sp.pprint(Eq2)


# task 9
# list of accelerations
acc = [xddt, XXX]

# solve equations for acceleration symbols
res = sp.solve([XXX, XXX], acc)

# task 10

msg = f"\nThe variable `res` is of type: {XXX} and has the following value:\n"
print(msg)
sp.pprint(res)

xdd_expr = res[xddt]
phidd_expr = XXX


# task 11
⋮
         (xt, x), (phit, phi)]

# step 2:
params_values = [(m1, 0.8), XXX]


# perform subsitution and save result in variables
xdd_expr_num = xdd_expr.subs(rplmts+params_values)
phidd_expr_num = XXX


# preparation done; now we can create the python functions

# generation of the Python functions using sp.lambdify
xdd_fnc = sp.lambdify([x, phi, xd, phid, F], xdd_expr_num, modules="numpy")
phidd_fnc = sp.lambdify(XXX)
```

```
DL_d_phid = L_d_phid.diff(t)


# task 8
F = sp.Symbol("F")  # external force (translatoric)



# right hand side of the equations of motion (left hand side is zero)
Eq1 =  DL_d_xd - L_d_x - F
Eq2 =  DL_d_phid - L_d_phi


# useful for debugging: pretty printing
sp.pprint(Eq1)
sp.pprint(Eq2)


# task 9
# list of accelerations
acc = [xddt, phiddt]

# solve equations for acceleration symbols
res = sp.solve([Eq1, Eq2], acc)

# task 10

msg = f"\nThe variable `res` is of type: {type(res)} and has the following value:\n"
print(msg)
sp.pprint(res)

xdd_expr = res[xddt]
phidd_expr = res[phiddt]


# task 11
⋮
         (xt, x), (phit, phi)]

# step 2:
params_values = [(m1, 0.8), (m2, 0.3), (l, 0.5), (g, 9.81)]


# perform subsitution and save result in variables
# note: a sum of lists is a new list containing all elements
xdd_expr_num = xdd_expr.subs(rplmts+params_values)
phidd_expr_num = phidd_expr.subs(rplmts+params_values)

# preparation done; now we can create the python functions

# generation of the Python functions using sp.lambdify
xdd_fnc = sp.lambdify([x, phi, xd, phid, F], xdd_expr_num, modules="numpy")
phidd_fnc = sp.lambdify([x, phi, xd, phid, F], phidd_expr_num, modules="numpy")
```