

Diff: Übung 03b

Unterschiede zwischen Vorgabe und Lösung

Um die Musterlösung leichter nachvollziehbar zu gestalten wurden die Unterschiede zur Quelltext-Vorgabe mit Hilfe des Programms `diff` grafisch aufbereitet.

Legende:

- Grau: unveränderter Text (nur auszugsweise)
- Grün: neue Zeilen
- Gelb: veränderte Zeilen
- Rot: gelöschte Zeilen (kommt hier nicht vor)

Hinweis: nicht aufgeführte Dateien wurden nicht verändert.

Dieses Dokument wurde mit Hilfe von [diff2html](#) erstellt.

diff -u -x messdaten.txt ../kurs03b/uebung/code/identifikation.py ../kurs03b/uebung/loesung/identifikation.py

../kurs03b/uebung/code/identifikation.py	../kurs03b/uebung/loesung/identifikation.py
:	:
3	3
4 import numpy as np	4 import numpy as np
5	5
	6 from scipy.integrate import odeint
6	7
7 # odeint importieren	
8 import sys	
9	8
10 import pylab as pl # für das Plotten am Ende	9 from lagrange_lsg import xdd_fnc2, phidd_fnc2
11	10
	11 t = np.linspace(1, 10, 101)
12	12
13	13
	14 # Daten im Binärformat laden
	15 res_target = np.load('messdaten.npy')
14	16
15 sys.exit() # Beendet das Programm hier.	17 ## alternativ: Text-Format:
	18 # res_target = np.loadtxt('messdaten.txt')
16	19
	20 x0 = res_target[0, :] # erste Zeile der Messdaten -> Anfangswert
17	21
	22 def min_target(p):
	23 m2, l = p
18	24
	25 def rhs(z, t):
	26 x, phi, xd, phid = z # Entpacken
	27 F = 0
19	28
20 #from ??? import xdd_fnc2, phidd_fnc2	29 # m2 und l kommen aus dem Namensraum eine Ebene höher
21 # Funktionen vorher anlegen!	30 xdd = xdd_fnc2(x, phi, xd, phid, F, m2, l)
22	31 phidd = phidd_fnc2(x, phi, xd, phid, F, m2, l)
23	
24	
25 #res_target = np.load(???)	
26	
27 # das sollte zu den Werten in res_target passen (bzgl. der Dimensionen):	
28 # tt = np.linspace(0, 10, ?)	
29	
30 # richtigen Anfangswert wählen	
31 # x0 = ?	
32	32
	33 return np.array([xd, phid, xdd, phidd])
33	34
	35 res = odeint(rhs, x0, t)
34	36

35	def min_target(p):	37	## Differenz der x-Positionen bilden (jeweils erste Spalte) und quadrieren
36	# m2, l = ? # Entpacken		
37		38	
38	# Funktion INNERHALB einer anderen Funktion definieren	39	err = np.sum((res[:,0]-res_target[:,0])**2)
39	# geht und ist oft sinnvoll!	40	print "simulation ready. p=", p, " Error:", err
40	# Die innere Funktion kann (lesend) auf den Namensraum	41	#x, phi, xd, phid = res.T
41	# der äußeren Funktion zugreifen (wie bei globalen Variablen).		
42			
43	# Die äußere kann nicht auf den Namensraum der inneren zugreifen.		
44			
45	def rhs(z, t):		
46	# die Parameter m2 und l werden aus dem äußeren Namensraum übernommen		
47			
48	pass # leere Anweisung; verhindert Einrückungsfehler		
49			
50	#xdd = xdd_fnc2(..., m2, l)		
51	#phidd = ...		
52			
53			
54	# return r_ [...]		
55			
56	## Ende der Funktion rhs		
57			
58			
59	## Simulation durchführen		
60			
61	#res = odeint(rhs, ...)		
62			
63	## Abweichung zu Messwerten berechnen		
64			
65	#err = np.sum((???)**2)		
66			
67	# Statusmeldung und Ausgabe (um Fortgang der Optimierung zu beurteilen)		
68	# Interessant sind zum Beispiel Werte von p und err		
69			
70		42	
71	return err	43	return err
72		44	
73		45	
74	p0 = r_[.5, .7] # Startschätzung	46	p0 = np.array([.3, .5]) # Startschätzung
75			
76	#from ??? import fmin		
77			
78			
79	# Optimierung durchführen		
80		47	
81	#m_res = fmin(zu_minimierende_Funktion, Startwert)	48	from scipy.optimize import fmin
		49	p_res = fmin(min_target, p0)
		50	print "Berechnete Parameter:", p_res

```
diff -u -x messdaten.txt ../kurs03b/uebung/code/lagrange_lsg.py ../kurs03b/uebung/loesung/lagrange_lsg.py
```

../kurs03b/uebung/code/lagrange_lsg.py	../kurs03b/uebung/loesung/lagrange_lsg.py
:	:
3	3
4 from sympy import sin, cos, Function	4 from sympy import sin, cos, Function
5 import sympy as sp	5 import sympy as sp
6 import sys	
7	6
8	7
	8 """
	9 Diese Datei beinhaltet die Lösung von Übung 3, sowie eine Erweiterung,
	10 die für Übung 4, Teil2 benötigt wird
9	11
10 # Lösung von Übung 03 (wird in Übung 04 benötigt)	12 """
11	
12	13
13	14
14 t = sp.Symbol('t')	15 t = sp.Symbol('t')
:	:
119	120
120	121
121	122
	123 ## Für Aufgabe 2:
	124 ## m2 und l nicht mit numerischen Werten ersetzen, sondern diese
	125 ## Variablen als Parameter übergeben
	126
	127 params_values2 = {m1:0.8, g:9.81}
	128 xdd_expr2 = sol[xdd_s].subs(params_values2)
	129 phidd_expr2 = sol[phidd_s].subs(params_values2)
	130
	131
	132 # Funktionen für die numerische Berechnung der Beschleunigungen
	133
	134 ## jetzt mit zusätzlicher Abhängigkeit von m2 und l
	135
	136 xdd_fnc2 = sp.lambdify([x_s, phi_s, xd_s, phid_s, F, m2, l], xdd_expr2, 'numpy')
	137 phidd_fnc2 = sp.lambdify([x_s, phi_s, xd_s, phid_s, F, m2, l], phidd_expr2, 'numpy')
	138
	139
	140
	141
122	142
123 def preview(expr, **kwargs):	143 def preview(expr, **kwargs):
124 """	144 """

Only in ../kurs03b/uebung/loesung/: lagrange_lsg.pyc

Binary files ../kurs03b/uebung/code/messdaten.npy and ../kurs03b/uebung/loesung/messdaten.npy differ

```
diff -u -x messdaten.txt ../kurs03b/uebung/code/simulation1.py ../kurs03b/uebung/loesung/simulation1.py
```

../kurs03b/uebung/code/simulation1.py	../kurs03b/uebung/loesung/simulation1.py
:	:

3		3	
4	import numpy as np	4	import numpy as np
5		5	
6	# odeint aus scipy importieren	6	from numpy import r_, pi
7	import sys	7	from scipy.integrate import odeint
8		8	
9	import pylab as pl # für das Plotten am Ende	9	import pylab as pl # für das Plotten am Ende
10		10	
11		11	
12		12	from lagrange_lsg import xdd_fnc, phidd_fnc
13	# Funktionen zur Berechnung der Beschleunigungen importieren:	13	
14	#from lagrange_lsg1 import xdd_fnc, ...	14	def rhs(z, t):
15			
16			
17			
18	sys.exit() # Beendet das Programm hier. Sonst: Fehlermeldungen		
19			
20			
21			
22			
23	def rhs(argumente):		
24			
25	x, phi, xd, phid = z # Entpacken	15	x, phi, xd, phid = z # Entpacken
26	F = 0	16	F = 0
27		17	
28	#xdd = xdd_fnc(??)	18	xdd = xdd_fnc(x, phi, xd, phid, F)
29		19	phidd = phidd_fnc(x, phi, xd, phid, F)
30			
31	#return np.array([xd, phid, ??, ??])		
32		20	
33		21	return r_[xd, phid, xdd, phidd]
34	t = np.linspace(0, 10, 101)	22	
35		23	t = np.linspace(1, 10, 101)
36	# x0 = np.array([??, np.pi, ??, ??])	24	
37		25	x0 = r_[0, pi*3/4, 0, 0]
38	# res = odeint(...) # numerische Integration ausführen	26	
39		27	res = odeint(rhs, x0, t)
40	# res: Spalten -> Zustände, Zeilen -> Zeitschritte	28	
41		29	# res: Spalten -> Zustände, Zeilen -> Zeitschritte
42	# Entpacken der einzelnen Zustände.	30	
43	# Dazu Zeilen u Spalten vertauschen (Transponieren)	31	# Entpacken der einzelnen Zustände.
44		32	# Dazu Zeilen u. Spalten vertauschen (Transponieren)
45	x, phi, xd, phid = res.T	33	
46		34	x, phi, xd, phid = res.T
		35	
		36	
		37	# Grafische Darstellung:
47	pl.plot(t, x)	38	pl.plot(t, x)

```
48 pl.plot(t, phi)
```

```
49
```

```
50
```

```
51
```

```
52 pl.show()
```

```
\ No newline at end of file
```

```
39 pl.plot(t, phi)
```

```
40 pl.show()
```

```
41
```

```
42
```

```
43 ## "Pseudo-Messdaten" für Aufgabe 2 generieren
```

```
44 ## der Block wird nicht ausgeführt, lässt sich aber schnell in "aktiven Code"
```

```
45 ## umwandeln (0 durch 1 ersetzen)
```

```
46
```

```
47 if 0:
```

```
48     # Binärformat:
```

```
49     np.save('messdaten.npy', res)
```

```
50     # Textformat (Menschenlesbar, braucht aber mehr Speicherplatz):
```

```
51     np.savetxt('messdaten.txt', res)
```

```
52
```

```
53 print "Files written."
```