

```
# -*- coding: utf-8 -*-
from __future__ import division

"""
Dieses Skript besteht im wesentlichen aus der Lösung der Aufgabe 2 von Kurs03b
Es ist um Plotfunktionen erweitert erweitert.
"""

from scipy.integrate import odeint
from scipy.optimize import fmin
import matplotlib.pyplot as plt
import numpy as np

# import unserer Differentialgleichungen
from lagrange_lsg import xdd_fnc2, phidd_fnc2

# Zeitachse und Startwerte fuer den Solver
t = np.linspace(1, 10, 101)

# Referenzdatensatz, auf den das Modell optimiert werden soll
# Daten im Binärformat laden
target = np.load('messdaten.npy')
x0 = target[0, :] # erste Zeile der Messdaten -> Anfangswert

# Liste zum Sammeln der Zwischenergebnisse der Optimierung:
resList = []

def min_target(p):
    m2, l = p

    def rhs(z, t):
        x, phi, xd, phid = z # Entpacken
        F = 0

        # m2 und l kommen aus dem Namensraum eine Ebene höher
        xdd = xdd_fnc2(x, phi, xd, phid, F, m2, l)
        phidd = phidd_fnc2(x, phi, xd, phid, F, m2, l)

        return np.array([xd, phid, xdd, phidd])

    res = odeint(rhs, x0, t)

    # Zwischenergebnisse merken
    resList.append(res)

    # Differenz der x-Positionen bilden (jeweils erste Spalte) und quadrieren
    err = np.sum((res[:,0]-target[:,0])**2)
    print "simulation ready. p=", p, " Error:", err
    #x, phi, xd, phid = res.T

    return err

## Um bei wiederholter Ausführung (z.B. zur Anpassung der Plots)
## Zeit zu sparen, sollte man das Ergebnis speichern.
```

```
## Hier wird das Serialisierungsmodul pickle benutzt.
## Damit können (fast) beliebige Pythonobjekte persistent gespeichert
## werden. Alternativ bieten sich für numerische Daten natürlich auch die
## Funktionen numpy.save und numpy.load an.

## Generell ist eine Trennung von (aufwendiger) Datenerzeugung und
## Visualisierung oftmals sehr sinnvoll.

import pickle # Serialisierungsmodul
pname = "res_list.pkl" # Dateiname beliebig

## Nach dem erfolgreichen speichern hier eine 0 einsetzen:
if 0:
    # Optimierung - Ergebnis sind die zwei Parameter m2 und l

    # Startwerte fuer Optimierung
    p0 = [.5, .7]

    p_res = fmin(min_target, p0)

    myfile = open(pname, 'w') # Datei zum Schreiben öffnen
    pickle.dump(resList, myfile) # resList wird serialisiert u. gespeichert
    myfile.close()
    print "Simulationsdaten erfolgreich gespeichert:", pname
    print "Programm-Ende"
    import sys
    sys.exit()
else:

    # Berechnungsergebnisse laden

    myfile = open(pname, 'r') # Datei zum Lesen öffnen
    resList = pickle.load(myfile)
    myfile.close()

#-----
# Aufgabe 1: Messwerte, Optimierungsergebnis und Fehler (3x1 Plot)
#-----

fig = pl.figure(figsize=(12,10))

# Subplot fuer Anfangswerte
ax1 = fig.add_subplot(3,1,1)
ax1.plot(t, target[:,0], label='Messung')
ax1.plot(t, resList[0][:,0], label='Modell')
ax1.legend()
ax1.grid()
ax1.set_ylabel('Weg x [m]')

# Subplot fuer Optimierungsergebnisse
ax2 = fig.add_subplot(3,1,2)
ax2.plot(t, target[:,0], label='Messung')
ax2.plot(t, resList[-1][:,0], label='Modell optimiert')
ax2.legend()
ax2.grid()
ax2.set_ylabel('Weg x [m]')
```

```

# Subplot fuer Restfehler der Position x
ax3 = fig.add_subplot(3,1,3)
ax3.plot(t, target[:,0]-resList[-1][:,0])
ax3.grid()
ax3.set_xlabel('Zeit [s]')
ax3.set_ylabel('Fehler  $\epsilon$  [m]')
#pl.show()

#-----
# Aufgabe 2: Verlauf der Optimierung (2D)
#-----

fig = pl.figure(figsize=(12,10))
ax = fig.add_subplot(1,1,1)

# Plotten aller Zwischenergebnisse in verschiedenen Grautoenen (0.1 < col < 0.8)
for i in range(len(resList)-1):
    colStep = 0.7/len(resList)

    ax.plot(t, resList[i][:,0], color=str(0.1+(0.7-i*colStep)))

# Plotten der Messung und der Simulation
ax.plot(t, target[:,0], color='#3366FF', lw=4, label='Messung')
ax.plot(t, resList[-1][:,0], color='#FF9900', ls='--', lw=2, label='Modell optimiert')
ax.grid()
ax.legend()
ax.set_xlabel('Zeit [s]')
ax.set_ylabel('Weg x [m]')

#pl.show()

#-----
# Aufgabe 2 Zusatz: Verlauf der Optimierung (3D)
#-----

from mpl_toolkits.mplot3d import axes3d
from matplotlib import cm
from pylab import meshgrid

fig = pl.figure()
ax = fig.add_subplot(1,1,1, projection='3d')

x_res = [i[:,0] for i in resList]

Z = np.vstack(x_res).T
X, Y = meshgrid(range(len(resList)), t)

ax.plot_surface(X,Y,Z, rstride=1, cstride=1, cmap=cm.autumn)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')

pl.show()

```