Pendel in linearer Näherung

Wir linearisieren die Rückstellkraft, da nur dann die DGL analytisch lösbar ist. Nachdem das Programm für die lineare DGL korrekte Ergbnisse liefert, könnte man die nichtlineare DGL 'numerisch' studieren.
Um die DGL in die für das Runge-Kutta-Verfahren in der Implementation von pylab gerforderte Form zu bringen, muss die DGL 2-Ordnung in zwei DGL 1. Ordnung überführt werden.

$$ma = -mg\sin(\alpha)$$
$$mit: \sin(\alpha) = \tan(\alpha) = x/l$$
$$ma = -mgx/l$$
$$oder$$
$$\ddot{x} + \frac{g}{l}x = 0$$
$$bzw:$$
$$\dot{x} = v$$
$$\dot{v} = -\frac{g}{l}x$$

$$mit\ Dämpfung:$$
$$\ddot{x} - \frac{r}{m} - \frac{g}{l}x = 0\ bzw:$$
$$\dot{x} = v$$
$$\dot{v} = -\frac{g}{l}x - \frac{r}{m}v$$

Die Frage bleibt, wie man/frau herausfindet, wie die Funktion zur Integration einer DGL (ordinary differntial equation- ode) in pylab heißt … Nachdem man das raus gekriegt hat:

pylab> help rk4
Help on function rk4 in module matplotlib.mlab:
rk4(derivs, y0, t)
Integrate 1D or ND system of ODEs using 4-th order Runge-Kutta. This is a toy implementation which may be useful if you find yourself stranded on a system w/o scipy. Otherwise use `scipy.integrate()`.

y0
: initial state vector

t
: sample times

derivs
: returns the derivative of the system and has the signature `dy = derivs(yi, ti)`

Example 1

```
## 2D system

def derivs6(x,t):
    d1 =  x[0] + 2*x[1]
    d2 =  -3*x[0] + 4*x[1]
    return (d1, d2)
dt = 0.0005
t = arange(0.0, 2.0, dt)
y0 = (1,2)
```

```
yout = rk4(derivs6, y0, t)
```

Example 2:

```
## 1D system
alpha = 2
def derivs(x,t):
    return -alpha*x + exp(-t)

y0 = 1
yout = rk4(derivs, y0, t)
```

Aha, dann braucht man das ja nur genau so zu machen:

Arbeit unter Windows:
Starte python(x,y)
Wähle als Editor:
SciTe (Häckchen anklicken)
Wähle als Ipython-Arbeitsumgebung:
ipython(x,y) (Klicke Knopf direkt daneben)

Schreibe DGL-Funktionen für RK4-Pylab (im Editor SciTe). Alle Python-Dateien sollen die Endung *.py haben. Die Datei kann mehrere Funktionen enthalten. Sie ist ein 'Modul'. Zudem enthält das Modul einige globale Variablen, die aus einem andern Modul/Skript heraus geändert werden können.

```
# -*- coding: utf-8 -*-
# Modul RK4  Test
from pylab import *
#
#  friction coeff:
r=0.5
#  driver freq
fd=1.2
ad=5.

def deriv_P(x,t):
        """  Pendel
        Länge so dass T=1s
        Ableitung für RK4 """
        pi=3.14159265358
        g=9.81
        L=g/(4*pi*pi)
        #
        dx=x[1]
        dv=-g/L*x[0]
        return (dx,dv)


def deriv_dP(x,t):
        """  gedämpftes Pendel
```

```
    Länge so dass T=1s
    Ableitung für RK4 """
    pi=3.14159265358
    g=9.81
    L=g/(4*pi*pi)
    #r=0.5
    #
    dx=x[1]
    dv=-g/L*x[0]-r*x[1]
    return (dx,dv)


def deriv_ddP(x,t):
    """getriebenes gedå®°fes Pendel
    Länge so dass T=1s
    Ableitung für RK4 """
    pi=3.14159265358
    g=9.81
     L=g/(4*pi*pi)
    #r=0.5
    #
    dx=x[1]
    dv=-g/L*x[0]-r*x[1]-ad*sin(2*pi*fd*t)
    return (dx,dv)
```

Speichern in Datei 'pendel_ode.py'**. X[0] ist der Ort und X[1] ist die Geschwindigkeit des Pendels, beide sind die dynamischen Variablen des Systems dx ist $\dot{x}$ und dv ist $\dot{v}$.**
Nach dem Start von python(x,y) muss unter windos zunächst in das Verzeichnis navigiert werden, in dem sich die Datei 'pendel_ode.py' **befindet. Verwende dazu '**cd' **und 'pwd'. Im Arbeitsverzeichnis angekommen, importiere den Inhalt des Moduls '**pendel_ode.py':

pylab> import pendel_ode

Hilfe (help bzw ?)gibt Auskunft:

pylab> help pendel_ode

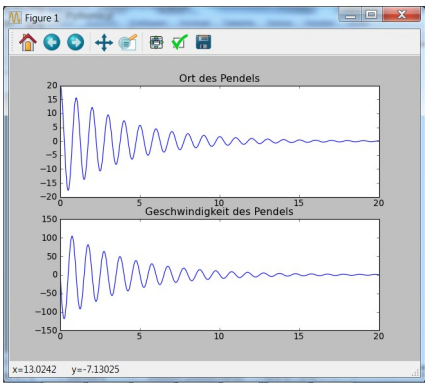
Aufrufe der Funktionen eines Moduls erfolgen in der Form 'Modulname.Funktionsname', Globale Variablen eines Moduls können mit 'Modulname.GlobaleVariable' verwendet werden.

Wenn Änderungen an 'pendel_ode.py' v**orgenommen wurden muss das Modul mit:**
pylab>reload pendel_ode
neu geladen werden.
Der Befehl

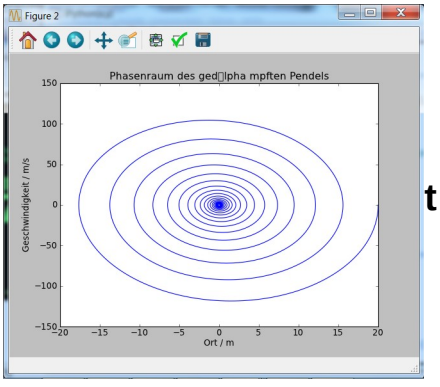
pylab> reset
löscht den gesamten „interactive workspace“, d.h. bei
pylab>whos
werden keine definierten Variablen mehr angezeigt, Module müssen nach 'reset' **mit** 'import' **neu geladen werden.**

Wird das Modul mit:
pylab> from pendl_ode import *
geladen, können die Funktionen ohne Modulname benutzt werden, z. B. derivs(...). **Das spart Schreibarbeit (insbesondere bei der interaktiven Benutzung), die Information über die Herkunft der Funktion geht aber verloren.**
So, nun zum Runge-Kutta:

Definiere einen Zeitschritt für die Intergration
> dt= 0.005 # Zeitschritt in Sec
Erzeuge damit eine Zeitvektor:
> t= arange(0,20,dt) # Zeitachse insgesamt (20sec mit dt)
Für alle diese Zeitpunkte soll ein Integrationsergbnis von RK4 erzeugt werden. Es sind size(t) viele Punkte.
> size(t) # ergibt 4000 Punkte
Setze Startwerte für die Integration. Es handelt sich hier um ein Anfangswertproblem bei dem die maximale Auslenkung bei null Geschwindigkeit vorgegeben wird:
> start= [20,0] # Startwerte für die Integration
x=20, v=0 maximale positive Auslenkung



Aufruf von rk4, Ergebnis wird in 'out' **geschrieben:**
>out=rk4(pendel_ode.derivs,start,t)



> shape(out) # gibt (400,2)
enthält als Spalte 0 die Orte, als Spalte 1 die Geschwindigkeiten des Pendels zu den Zeiten t

>figure(1)
> subplot(2,1,1)
> plot(t,out[:,0])
>title('Ort des Pendels')
>plot(t,out[:,1])
>title(Geschwindigkeit des Pendels')

>figure(2)
>plot(out[:,0],out[:,1])

> title('Phasenraum')
>xlabel('Ort / m')
>ylabel('Geschwindigkeit / m/s')

Hinweis zu Umlauten in Texten:
Im einfachsten Fall können Umlaute in TeX eingeben werden, z.B:
> xlabel('L$\\"a$nge') Länge

Naja, schön ist was anderes. Da kommen Python zu TeX-Problemen. Insbesondere muss statt '\' '\\' verwendet werden, um unter TeX das Sonderzeichen '"' drucken zu können.

Eine Berechnung sollte man in einem Skript durchführen, da dann später Kontrollen und Änderungen leicht möglich sind:

Pendel Script:

```
from pylab import *
import pendel

dt=0.01
start=[20,0]
t=arange(0,20,dt)
figure(1)
out=rk4(pendel.deriv_P,start,t)
plot(out[:,0],out[:,1])
figure(2)
#  choose friction
pendel.r=0.1
out=rk4(pendel.deriv_dP,start,t)
plot(out[:,0],out[:,1])
# start values last point
nstart=[out[size(out)/2-1,0],out[size(out)/2-1,1]]
out=rk4(pendel.deriv_dP,nstart,t)
plot(out[:,0],out[:,1])
show()
```

Das Skript kann mit
>%run scriptname

gestartet werden. Achtung: UM Graphikotuput zu erzeugen, muss am Ende des Skriptes ein 'show()' eingefügt werden.