Einführung in Python Teil II

Bibliotheken für wissenschaftliches Rechnen

Valentin Flunkert

Institut für Theoretische Physik
Technische Universität Berlin

Fr. 28.5.2010
Nichtlineare Dynamik und Kontrolle SS2010

Wiederholung

- Dateien

```
f = open('datei.dat', 'r')
for line in f:
    pair = line.split()
```

- Listen

```
li = [1, 2, 3.4, ['ein', 'wort']]
x = li[3:5]; y = li[-3]
```

- Dictionaries

```
d = {'ich': 1, 'du': 2, 'er': 3}; d['es'] = 3
```

- Funktionen

```
def f(x, y, n=1):
    """eine Funktion"""
    return (x*y)**n

f(2.0, 3.0)
f(y=1.2, x=2.0, n=2)
```

Listen für Numerik

Nachteile von Listen für Numerik

- Elemente nicht nur Zahlen
- Listen sind langsam
- '+' hängt Listen aneinander;
  für Numerik wäre Vektor addition praktisch
- andere Arithmetische Operationen gibt es nicht (`- * /`)

Besser für Numerik: ein vektorartiger Datentyp

`numpy` arrays

Arrays – numpy Bibliothek

für Numerik besser: Arrays

- Erzeugen

```
import numpy as np
a = np.array([[1, 4], [2, 4]]) # umwandeln
b = np.zeros(100, 10)          # 100 x 10 nullen
c = np.random.rand(101)        # 101 Zufallszahlen
d = np.linspace(0, 1, 101)     # 101 Zahlen von 0 bis 1
```

- Indexing wie bei Listen;
  bei mehrdimensionalen Arrays mit Komma

```
print a[:,1], b[2:,1], c[-4] # etc
```

- für gleichlange Arrays: elementweise Arithmetik

  `c+d,  c-d,  c*d,  c/d`

Arrays – II

- Arrays haben feste Länge
- Viele eingebaute Methoden

```
x = np.random.rand(1000)
s = x.sum(); p = x.prod(); m = x.mean(); st = x.std()
```

- Array Operationen sind in C geschrieben und optimiert (sehr schnell)
- Wo möglich: Array-Operationen statt Schleifen
  Beispiel:

```
x = np.linspace(0, 1, 101)
y1 = np.sin(x)**2 * np.exp(-x)
y2 = np.cos(x)**2 + 0.1
z = y1 * y2                 # elementweise Multiplikation
s = np.dot(y1, y2)          # Skalarprodukt
```

numpy – Subpackages

```
np.random     # Zufallszahlen (verschiedene Verteilungen)
np.linalg     # lineare Algebra (Matrix operationen, etc)
np.fft        # fourier Trafo
...
```

matplotlib – 2D Plot library

Eine der besten 2D-plot Umgebungen die es gibt! Beispiel:

```
import pylab as pl  # matplotlib interface
import numpy as np

x = np.linspace(0,10,100)
y1 = np.sin(x) * np.exp(-0.05*x)
y2 = np.exp(-0.05*x)

# linestyle: 'b--' blue dashed
pl.plot(x, y1, 'b--', label=r'$\sin(x) \exp (-0.05 x)$')

# linestyle: 'g-' green solid
pl.plot(x, y2, 'g-',  label=r'envelope')

pl.xlabel(r'$x$')
pl.ylabel(r'$y$')
pl.legend()
pl.show()
# pl.savefig('meinplot.pdf')
```

Bemerkungen

- `pylab` ist die nutzerfreundliche Schnittstelle für `matplotlib`
- globale Einstellungen in `~/.matplotlib/matplotlibrc` (unter Linux)
- Latex in strings:
  - Problem: '\t' ist z.B. das tab Zeichen

    ```
    s = '$t/\tau$'
    ```
  - Lösung 1: escapen

    ```
    s = '$t/\\tau$'
    ```
  - Lösung 2: raw strings

    ```
    s = r'$t/\tau$'
    ```

`scipy` – scientific computing package

SciPy – umfangreiches Packet fürs wissenschaftliche Rechnen

- Spezielle Funktionen

```
# Beispiel Besselfunktionen jn
from scipy.special import jn
```

- ODE Solver haben wir bereits kennengelernt

```
from scipy.integrate import odeint
```

- Statistik

```
from scipy import stats
```

- Interpolation

```
from scipy import interpolate
```

Beispiel Nullstelle einer 2D-Fkt. suchen

- Datei `roots.py`

```
from math import *
from scipy.optimize import fsolve

def f(pair):      # 2D Funktion
    x = pair[0]; y = pair[1]
    return [ 2.1* x * y**2 + x**3*y,\
             4.0*(x**2)*(y-1.0) + 3*x + 2]

erg = fsolve(f, x0=[1.1, -2.0])
print "nullstelle:\n   x = %s" % erg
print "Funktionswert bei Nullstelle:\n   f(x) = %s" % f
```

- Output:

```
nullstelle:
   x = [ 0.92095498 -0.4038848 ]
Funktionswert bei Nullstelle:
   f(x) = [1.1102230246251565e-16, 0.0]
```

Performance analysieren

- Datei `tooslow.py`

```
import numpy

def mysum(ar):
    N = len(ar)
    erg = 0.0
    for x in ar:
        erg += x
    return erg

x=numpy.random.rand(10000000)

print mysum(x)
```

- in ipython (performance profile):

```
>>> %run -p tooslow.py
```

Ergebnis

```
In [1]: %run -p tooslow.py
-6666613.10949
         7 function calls in 24.457 CPU seconds

Ordered by: internal time

ncalls  tottime  percall  cumtime  percall filename:lineno(func
     1   24.210   24.210   24.210   24.210 tooslow.py:4(myfunc)
     1    0.247    0.247    0.247    0.247 {method 'rand' of 'm
     1    0.000    0.000   24.457   24.457 {execfile}
     1    0.000    0.000   24.457   24.457 tooslow.py:2(<module
     1    0.000    0.000   24.457   24.457 <string>:1(<module>)
     1    0.000    0.000    0.000    0.000 {len}
     1    0.000    0.000    0.000    0.000 {method 'disable' of
```

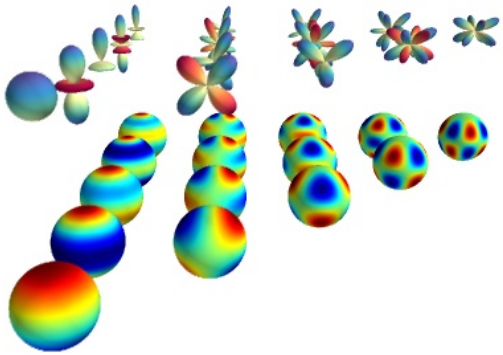
- myfunc ist zu langsam!

Funktion in C schreiben

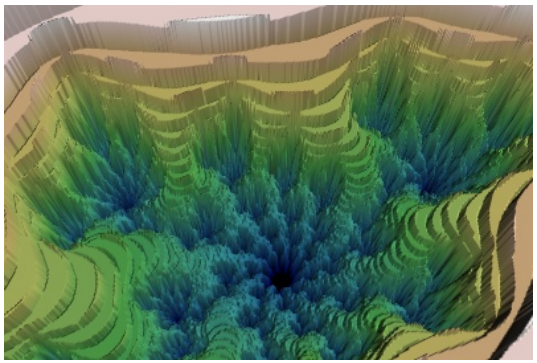
```
import numpy
from scipy import weave

def csum(ar):
    N = len(ar)
    code="""
    double summe = 0.0;

    for(int i = 0; i < N; i++)
    {
      summe += ar[i];
    }
    return_val = summe;
    """
    erg = weave.inline(code,
                       ['ar','N'],
                       compiler = 'gcc')
    return erg

x=numpy.random.rand(100)
print
```

Mayavi – 3D Visualisierung

Beispiele

- 

  `spherical_harmonics.py`

- 

  `julia.py`

sympy

Symbolische Rechnungen (ähnlich wie in Mathematica)

- Projekt noch relativ jung (etwas in Bewegung)
- Beispiel `ableitung.py`

```
from sympy import *

x,y,z = symbols('xyz')

f = sqrt(2*x*y + 2*sin(x*y))
df = f.diff(x)

print "\npython:\n"
print df
print "\n\npretty printing:\n"
pretty_print(df)
print "\n\nlatex:\n"
print latex(df)
print " "
```