

Ipython, Numeric, scipy, matplotlib — Wissenschaftliches Rechnen mit Python

Workshop — LinuxTag Dresden, 30. Oktober 2004

Arnd Bäcker, www.physik.tu-dresden.de/~baecker

1 Einleitende Bemerkungen

Python ist eine interpretierte objektorientierte Programmiersprache, deren Anwendungsbereich durch die in der Standardbibliothek vorhandenen Pakete ergänzt wird. Durch die Installation zusätzlicher Pakete kann Python in extrem vielseitiger Weise eingesetzt werden.

Mittels der Pakete

- **IPython**: verbesserte interaktive Shell
- **Numeric**: Arrays
- **matplotlib**: Plot-Paket
- **scipy**: numerische Toolbox

kann man Python für numerische Rechnungen einsetzen und erhält eine in vielen Bereichen ähnliche Funktionalität wie beispielsweise bei `matlab`.

Eine Liste der verfügbaren Module erhält durch Aufruf des Hilfe-Systems durch `help` gefolgt von `modules`. Alternativ kann eine Liste der Module der Standard-Bibliothek über den [Global Module Index](#) auf [python documentation page](#) erhalten werden. Wenn die Python Dokumentation installiert ist, so kann man die gleiche Information auch lokal erhalten, beispielsweise unter Linux `/usr/share/doc/python2.3`.

Installation von Python

Unter den meisten aktuellen Linux-Distributionen ist Python bereits installiert. Verwendet man `debian`, `sarge` sollten folgende Befehle genügen:

```
apt-get install python-numeric
apt-get install ipython
```

Für `scipy` und `matplotlib` fügt man folgende Zeilen zu `/etc/apt/sources.list` hinzu

```
deb http://deb-scipy.alioth.debian.org/apt/ ./
deb http://anakonda.altervista.org/debian packages/
```

Nach `update` (z.B. `dselect`) kann man die Pakete `python2.3-scipy` und `python-matplot` und deren Abhängigkeiten installieren.

Für Windows Benutzer ist die [enthought edition](#) (90 MB download!) eine Möglichkeit (jedoch fehlen hier noch `matplotlib` und `ipython`).

2 Numeric

Durch

```
from Numeric import *
```

werden alle Befehle, die im Modul Numeric enthalten sind, eingebunden.

Arrays

```
from Numeric import *
a=array([1,2,3,4])
b=array([2,4,6,8])
print a+b
```

Erzeugen von Arrays

```
# Array mit Werten 0,1,2,3,4
x1=arange(5)
print x1
# Array mit den Werten 7,8,9,10,11
x2=arange(7,12)
print x2
# Array mit Werten 0.0,1.0,2.0,3.0,4.0
x3=arange(5.0)
print x3
```

Mathematische Operationen

```
# Multiplikation mit Skalar ...
y1=2.0*pi*x1
print y1
# und Division:
y3=x3/3.0
print y3

# Addition
print y1+y3
# Division
print y1/(y3+1.0)
```

Anwendung von Funktionen auf Arrays

```
def f(x):
    return 4.0*x**2-5.0*x

print f(x1)
print f(x2)
```

Beispiele zum Ausprobieren

Probieren Sie folgende Befehle aus

```
x=arange(10)
y=zeros(10)
z=2*ones(10)
print x
print y
print z
```

```
z2=2.0*ones(10)
print z2
```

Was ist der Unterschied zwischen `z` und `z2`?
Addition und Division

```
print 4*x
print x+1
print x/10.0
print x-pi
```

Addition, Subtraktion, Multiplikation und Exponentiation von Arrays

```
w=arange(1,20,2)
print x+w
print x-w
print x*w
print x/w
print 1.0*x/w
print x**2
print x**(w/10.0)
```

Probieren Sie

```
v1=arange(10)
v2=zeros(9)
print v1+v2
```

Die Summe der Elemente eines Arrays erhält man durch den Befehl `sum`,

```
print sum(x)
```

Eine Vielzahl weiterer Befehle, die auf Arrays operieren, sind im `Numeric manual` aufgeführt.

3 matplotlib

Ein wesentlicher Aspekt in vielen Anwendungen ist die graphische Darstellung von Daten. Hier soll das Paket `matplotlib` vorgestellt werden, das über `ipython` auch interaktiv genutzt werden kann. Hierzu ruft man am Shell-Prompt

```
ipython -pylab
```

auf. Hierdurch werden sowohl die Plot-Befehle (intern via `from matplotlib.matlab import *`) als auch das Numeric Paket (intern via `from Numeric import *`) bereitgestellt.

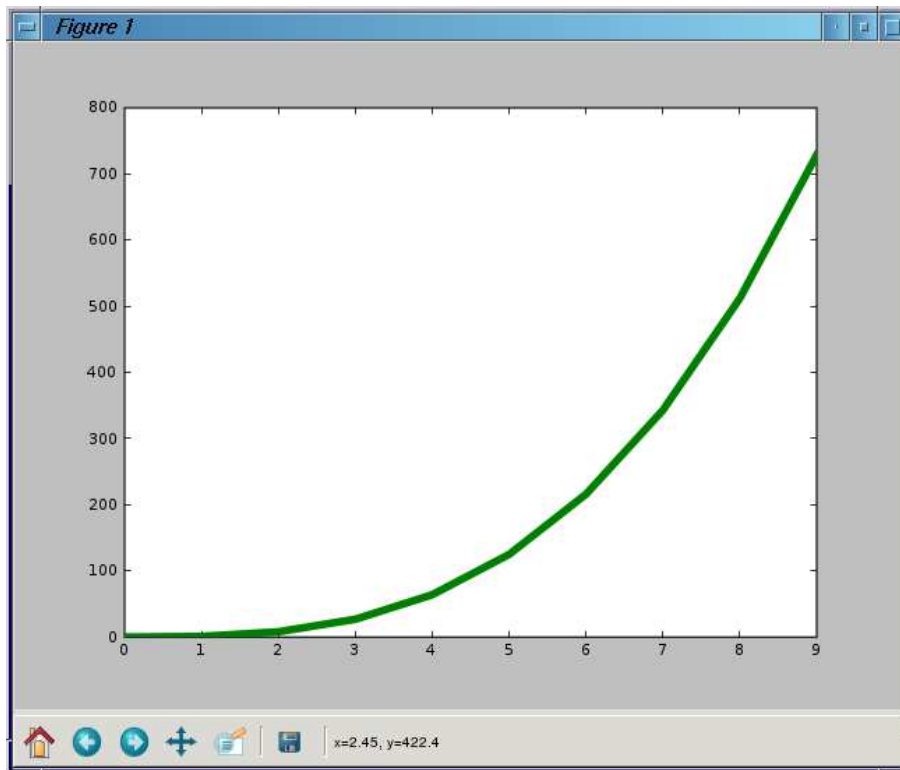
Beispiel:

```
#> ipython -pylab
Python 2.3.4 (#2, Sep 24 2004, 08:39:09)
Type "copyright", "credits" or "license" for more information.

IPython 0.6.3 -- An enhanced Interactive Python.
?      -> Introduction to IPython's features.
@magic -> Information about IPython's 'magic' @ functions.
help   -> Python's own help system.
object? -> Details about 'object'. ?object also works, ?? prints more.
```

```
Welcome to pylab, a matplotlib-based Python environment.
  help(matplotlib) -> generic matplotlib information.
  help(matlab)     -> matlab-compatible commands from matplotlib.
  help(plotting)  -> plotting commands.
In [1]: x=arange(10.0)
In [2]: plot(x,x**3,linewidth=5)
Out[2]: [<matplotlib.lines.Line2D instance at 0x4203bbac>]
In [3]:
```

Zu diesem Zeitpunkt öffnet sich ein Fenster mit der Grafik



und weiterhin ist die Eingabe von Kommandos am Ipython-Prompt möglich!
Ein weiterer Plot kann durch

```
In [3]: plot(x,5.0*x**2)
```

hinzugefügt werden.

Probieren Sie aus (Plotfenster vorher schließen)

```
x=arange(0.0,10.0,0.01)
plot(x,x*sin(2.0*pi*x))
```

Anmerkung: die graphische Ausgabe von matplotlib kann man auch GUIs (in Tk, GTK oder wxPython) einbetten.

4 scipy

Das `scipy` Paket bietet eine Vielzahl von Routinen aus verschiedenen Bereichen wissenschaftlichen Rechnens. Mittels `import scipy; help(scipy)` erhält man eine Aufstellung der Unter-Pakete, u.a.

```
scipy.cluster    --- Vector Quantization / Kmeans
scipy.cow        --- Cluster Of Workstations
scipy.fftpack    --- Discrete Fourier Transform algorithms
scipy.ga         --- Genetic Algorithms
scipy.gplt       --- Plotting with gnuplot
scipy.integrate  --- Integration routines
scipy.interpolate --- Interpolation Tools
scipy.io         --- Data input and output
scipy.linalg     --- Linear algebra routines
scipy.optimize   --- Optimization Tools
scipy.plt        --- Plotting with wxPython
scipy.signal     --- Signal Processing Tools
scipy.sparse     --- Sparse matrix
scipy.special    --- Special Functions
scipy.stats      --- Statistical Functions
scipy.xplt       --- Plotting routines based on Gist
weave           --- N/A
```

Hilfe zu den weiteren Unterpunkten kann man beispielsweise mit `help(scipy.optimize)` erhalten.

Es sollen nun einige einfache Beispiele zur Verwendung von `scipy` vorgestellt werden.

Finden der Nullstellen einer Funktion

```
ipython -pylab
from scipy.optimize import brentq

def f(x):
    return -0.3*x**3 + x**2 + 10.0*x - 5.0

x0=brentq(f,-3.0,3.0)
```

```

print x0
x=arange(-10.0,10.,0.4)
plot(x,f(x),linewidth=2)
plot(x,0*x,linewidth=0.5)
x1=brentq(f,-10.0,-1.0)
print x1
x2=brentq(f,1.0,10.0)
print x2
# Plot the zeros
plot([x0,x1,x2],[0.0,0.0,0.0], 'rs',markersize=10)

```

Bestimmung von Minima and Maxima

```

from scipy.optimize import fmin
xmin=fmin(f,-1.0)
print xmin

def minus_f(x): return -f(x)

xmax=fmin(minus_f,1.0)
print xmax
# Plot der Extremwerte
plot([xmin,xmax],[f(xmin),f(xmax)], 'go',markersize=10)

```

Beispiel für eine spezielle Funktion

```

ipython -pylab
from scipy import special
x=arange(0.0,30.0,0.05)
y=special.j0(x)
plot(x,y,linewidth=2)

```

Interpolation mit Splines (Beispiel aus dem [scipy tutorial](#) von Travis Oliphant)

```

ipython -pylab
from scipy import *
x=arange(0.0,2.0*pi+0.1,2.0*pi/8.0)
y=sin(x)
spline_coefs=interpolate.splrep(x,y,s=0)
xfine=arange(0.0,2.0*pi,2.0*pi/100.0)
yfine=interpolate.splev(xfine,spline_coefs,der=0)
plot(x,y,'ro',markersize=10)
plot(xfine,yfine,linewidth=4)
plot(xfine,sin(xfine),linewidth=1)

Bestimmung der Ableitung

yder=interpolate.splev(xfine,spline_coefs,der=1)
clf() # Bild loeschen
plot(xfine,yder,'ro',markersize=10)
plot(xfine,cos(xfine),linewidth=1)

```

Was passiert, wenn man statt

```
x=arange(0.0,2.0*pi+0.1,2.0*pi/8.0)
```

folgendes verwendet?

```
x=arange(0.0,2.0*pi,2.0*pi/8.0)
```

Wie kommt der Unterschied zustande?

Integration von Funktionen (Beispiel aus dem [scipy tutorial](#) von Travis Oliphant)

Beispiel: Integration der Besselfunktion $j_v(2.5, x)$ auf dem Intervall $[0, 4.5]$,

$$I = \int_0^{4.5} J_{2.5}(x) dx. \quad (1)$$

Hierzu kann man die Routine `quad` aus `scipy.integrate` verwenden:

```
from scipy import *
```

```
def fkt(x):  
    return special.jv(2.5,x)
```

```
result = integrate.quad(fkt , 0, 4.5)  
print result
```

```
# Analytische Integration ergibt
```

```
I = sqrt(2/pi)*(18.0/27*sqrt(2)*cos(4.5)-4.0/27*sqrt(2)*sin(4.5)+  
    sqrt(2*pi)*special.fresnl(3/sqrt(pi))[0])
```

```
print I
```

```
# Differenz:
```

```
print abs(result[0]-I)
```