

Teil XII

Wissenschaftliches Rechnen in Python

Nochmal Modul math

- Konstanten `pi` und `e`
- Funktionen für `int` und `float`
- Alle Rückgabewerte sind `float`

```
ceil(x)
floor(x)
exp(x)
fabs(x)           # wie abs(), nur immer float
ldexp(x, i)       # x * 2**i
log(x [,base])
log10(x)          # == log(x, 10)
modf(x)           # (Nachkommateil, Integerteil)
pow(x, y)         # x**y
sqrt(x)
```

Trigonometrische Funktionen (Bogenmaß)

```
cos(x); cosh(x); acos(x)
sin(x); ...
tan(x); ...

degrees(x) # rad -> deg
radians(x) # deg -> rad
```

inf/nan

```
float("inf")
float("-inf")
float("nan")
isinf(x) # Überprüfung auf Unendlichkeit
isnan(x) # Überprüfung auf NaN
```

Complex Zahlen: cmath

Und RICHTIGE Mathematik?

Bisherige Sequenztypen (`list`, `tuple`, ...)

- Können als Arrays verwendet werden
- Können verschiedene beliebige Typen enthalten
 - Sehr flexibel, aber auch langsam
 - Schleifen sind nicht sehr effizient
- Vektoren/Matrizen und Operationen auf diesen sind extrem umständlich
- Für effizientes wissenschaftliches Rechnen sind andere Datentypen und Methoden nötig

Und RICHTIGE Mathematik?

Bisherige Sequenztypen (`list`, `tuple`, ...)

- Können als Arrays verwendet werden
- Können verschiedene beliebige Typen enthalten
 - Sehr flexibel, aber auch langsam
 - Schleifen sind nicht sehr effizient
- Vektoren/Matrizen und Operationen auf diesen sind extrem umständlich
- Für effizientes wissenschaftliches Rechnen sind andere Datentypen und Methoden nötig

Module

- NumPy
- Matplotlib
- SciPy

Modul numpy

Homogene Arrays

- NumPy stellt beliebig-dimensionale Arrays bereit
- Alle Elemente des Arrays haben den gleichen Typ
- Beispiel

```
from numpy import *
a = array([[1,2,3],[4,5,6]])
print a
type(a)
a.shape
print a[0,2]
a[0,2] = -1
b = a*2
print b
```

Homogene Arrays (2)

- Arrays können aus (verschachtelten) Sequenztypen erstellt werden
- Direkter Zugriff mit []

```
a = array([1,2,3,4,5,6,7,8])
a[1]
a = array([[1,2,3,4],[5,6,7,8]])
a[1,1]
a = array([[[1,2],[3,4]],[[5,6],[7,8]]])
a[1,1,1]
```

Homogene Arrays (2)

- Arrays können aus (verschachtelten) Sequenztypen erstellt werden
- Direkter Zugriff mit []

```
a = array([1,2,3,4,5,6,7,8])
a[1]
a = array([[1,2,3,4],[5,6,7,8]])
a[1,1]
a = array([[[1,2],[3,4]],[[5,6],[7,8]]])
a[1,1,1]
```

- Eigenschaften von Arrays

```
a.ndim          # Anzahl Dimensionen
a.shape         # Anzahl Einträge in jeder Dimension
a.size         # Anzahl Elemente
a.dtype        # Datentyp der Elemente
a.itemsize     # Nötige Anzahl Bytes für den Datentyp
```


Datentypen

- Exakte Typen (ähnlich C/C++) können angegeben werden
- Ohne Angabe werden Standardtypen verwendet
- Einige Typen (v.a. unterschiedlicher Speicherbedarf):
 - `int`, `int8`, `int16`, `int32`, `int64`,
 - `float`, (`float8`, `float16`,) `float32`, `float64`,
 - `complex`, `complex64`,
 - `bool`, `character`, `object`

```
array([[1,2,3],[4,5,6]], dtype=int)
array([[1,2,3],[4,5,6]], dtype=complex)
array([[1,2,3],[4,5,6]], dtype=int8)
array([[1,2,3],[4,5,1000]], dtype=int8) # falsch
array([[1,2,3],[4,5,"hi"]], dtype=object)
```

Numerische Typen - ganze Zahlen (int)

- Darstellung im Rechner: zyklisch (neg,0,pos)

Numerische Typen - ganze Zahlen (int)

- Darstellung im Rechner: zyklisch (neg,0,pos)
- Beispiel overflow (C++, python):

```
#include <iostream>
#include <limits>
int main() {
    int d=std::numeric_limits<int>::max()-10;
    while(d > 0) std::cout << d++ << std::endl;
    return 1;
}
```

```
import sys
d = sys.maxint-10
while d>0:
    d += 1
    print d
```

Arrays initialisieren

- Standard-Matrizen (optionaler Parameter: dtype)

```
arange([a,] b [,stride]) # analog zu range, 1D
zeros( (3,4) )
ones( (1,3,4) )
empty( (3,4) )           # keine Init. (schnell)
linspace(a, b [, n])     # n äquid. Werte in [a,b]
logspace(a, b [, n])     # log zw. 10^a und 10^b
identity(n)              # Identität (Diagonalmat.)

fromfunction(lambda i,j: i+j, (3,4), dtype=int)
def f(i,j):
    return i+j
fromfunction(f, (3,4), dtype=int)
```

Arrays manipulieren

```
>>> a = arange(12); a
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
>>> b = a.reshape((3,4)); b; a
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
>>> a.resize((3,4)) # a wird verändert
>>> a.transpose() # a wird nicht verändert
>>> a.flatten() # a wird nicht verändert

# Beispiel:
a = arange(144)
a.resize((12,12))
```

Noch mehr Manipulations- und Initialisierungsmethoden

- Neue Matrizen aus existierenden Matrizen erstellen

```
a = arange(12); a.resize((3,4))
copy(a)
diag(a); tril(a); triu(a)

empty_like(a)           # Kopiert nur die Form
zeros_like(a)
ones_like(a)

a = loadtxt("matrix.txt") # fromfile() bei Binärdatei
```

- Matrizen ausgeben

```
a.tolist()

savetxt("matrix.txt", a) # tofile() bei Binärdatei
```

Lesen und Schreiben von Array-Elementen

- Die von Listen bekannte Indizierung kann verwendet werden
- Dimensionen werden mit Komma separiert

```
a = arange(20); a.resize((4,5))
a[1]
a[1:3,:]
a[:,::2]
a[:,:2]
a[:,:2,::2] = [[0, -2, -4],[-10, -12, -14]]
a[1::2,1::2] = -1*a[1::2,1::2]
```

- Selektiver Zugriff

```
a[a > 9]
a[a > 9] *= -1
```

Über Elemente von Matrizen iterieren

```
a = arange(20); a.resize((4,5))

for row in a:
    print row

b = arange(30); b.resize((2,3,4))
for row in b:
    for col in row:
        print col

for entry in a.flat: # Iterator
    print entry
```


Rechnen mit Arrays

- Schnelle built-in Methoden für Arrays

```
a = arange(12); a.resize((3,4))
3*a
a**2
a+a**2
sin(a)
sqrt(a)
prod(a)
sum(a)

it = transpose(a)

x = array([1,2,3])
y = array([10,20,30])
inner(x, y)
dot(it, x)
cross(x,y)
```

Rechnen mit Arrays

- Es gibt noch viel mehr...

```
var()           cov()           std()
mean()         median()
min()          max()
tensordot()
...
```

- Matrizen (mit `mat`) sind Unterklassen von `ndarray`, aber strikt zweidimensional, mit zusätzlichen Attributen

```
m = mat(a)
m.T      # Transponierte
m.A      # Als 2d Array
m.H      # Konjugiert Transponierte
```

Untermodule

Modul `numpy.random`

- Zufallszahlen aus vielen unterschiedlichen Verteilungen
- Mächtiger als das Standard-Modul `random`
- Arbeitet auf Arrays und gibt Arrays zurück

```
from numpy.random import *
binomial(10, 0.5)           # 10 Versuche, Erfolg 50%
binomial(10, 0.5, 15)      # 15 Mal voriges
binomial(10, 0.5, (3,4))   # (3x4) Array
randint(0, 10, 15)         # [0,10), int

rand()                      # [0,1), float
rand(3,4)                   #          (3x4) Array
```

Untermodule (2)

Modul `numpy.linalg`

- Die wichtigsten Werkzeuge für lineare Algebra

```
from numpy.linalg import *  
a = arange(16); a.resize((4,4))  
norm(a); norm(x) # Matrix- und Vektornorm  
inv(a)           # Inverse der Matrix  
solve(a, b)      # LAPACK LU Zerlegung  
det(a)           # Determinante  
eig(a)           # Eigenwerte und Eigenvektoren  
cholesky(a)      # Cholesky-Zerlegung
```

Untermodule (2)

Modul `numpy.linalg`

- Die wichtigsten Werkzeuge für lineare Algebra

```
from numpy.linalg import *
a = arange(16); a.resize((4,4))
norm(a); norm(x) # Matrix- und Vektornorm
inv(a)           # Inverse der Matrix
solve(a, b)     # LAPACK LU Zerlegung
det(a)          # Determinante
eig(a)          # Eigenwerte und Eigenvektoren
cholesky(a)     # Cholesky-Zerlegung
```

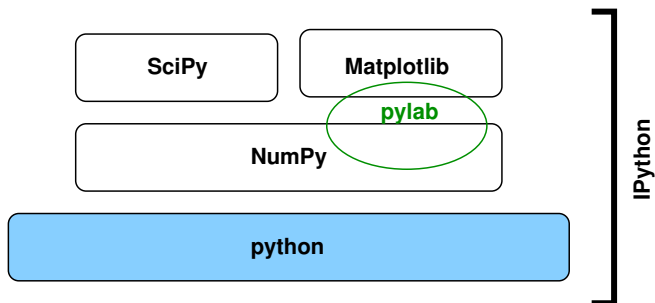
Modul `numpy.fft`

- Fourier-Transformation

Und weitere ...

Version Mania

Aktuelle Situation



Version Mania

Probleme:

- Numpy, scipy, pylab, ipython und matplotlib werden oft gleichzeitig benutzt
- Die Pakete hängen voneinander ab (matplotlib benutzt z.B. numpy-Arrays)
- Je nach Betriebssystem(sversion) müssen evtl. unterschiedliche Pakete installiert werden (D.h. Der Modulname im import-Befehl kann auch unterschiedlich sein).

Vision: Alles in einem (neuen) Modul PyLab vereinigen!

- gibt es als inoffizielles Paket
- Achtung Name: wieder pylab!

Matplotlib

Matplotlib

Wozu Matplotlib?

- Objektorientierte Bibliothek für zweidimensionale plots
- Entworfen mit dem Ziel einer ähnlichen Funktionalität wie Matlab plots
- Insbesondere zum Darstellen wissenschaftlicher Daten, baut auf numpy-Datenstrukturen auf

Möglichkeiten zum Importieren

ipython interaktiv

```
> ipython
import scipy, matplotlib.pyplot
x = scipy.randn(10000)
matplotlib.pyplot.hist(x, 100)
matplotlib.pyplot.show()
```

```
> ipython
import numpy.random, matplotlib.pyplot
x = numpy.random.randn(10000)
matplotlib.pyplot.hist(x, 100)
matplotlib.pyplot.show()
```

ipython mit pylab-Modus

```
> ipython -pylab
x = randn(10000)
hist(x, 100)
```

Beispiel - Erster Plot

<http://matplotlib.sourceforge.net/users/screenshots.html>

```
from pylab import *  
  
x = arange(0.0, 2*pi, 0.01)  
y = sin(x)  
plot(x, y)  
  
xlabel('Label fuer x-Achse')  
ylabel('Label fuer y-Achse')  
title('Einfacher sin plot')  
grid(True)  
show()
```

Beispiel - Subplots verwenden

```
from pylab import *
def f(t):
    s1 = cos(2*pi*t)
    e1 = exp(-t)
    return multiply(s1,e1)

t1 = arange(0.0, 5.0, 0.1)
t2 = arange(0.0, 5.0, 0.02)
t3 = arange(0.0, 2.0, 0.01)

show() # Gibt einen Fehler aber hilft ;-)

subplot(2,1,1) # Zeilen, Spalten, welche anzuzeigen
plot(t1, f(t1), 'go', t2, f(t2), 'k--')
subplot(2,1,2)
plot(t3, cos(2*pi*t3), 'r.')
```

Beispiel - Subplots verwenden

```
# Fortsetzung der vorigen Folie

subplot(2,1,1)
grid(True)
title('A tale of 2 subplots')
ylabel('Gedaempfte Oszillation')

subplot(2,1,2)
grid(True)
xlabel('Zeit (s)')
ylabel('Ungedaempft')
```

Beispiel - Histogramm

```
from numpy import *
from matplotlib.mlab import *
from matplotlib.pyplot import *
#alternativ: ipython -pylab und unten random. weglassen

mu, sigma = 100, 15
x = mu + sigma*random.randn(10000)
n, bins, patches = hist(x, 50, normed=1, \
    facecolor='green', alpha=0.75)

# Linie mit 'best fit' einzeichnen
y = normpdf( bins, mu, sigma)
plot(bins, y, 'r--', linewidth=1)

axis([40, 160, 0, 0.03])
show()
```

SciPy

Mehr als NumPy?

- SciPy hängt von NumPy ab
- SciPy arbeiten mit NumPy arrays
- Stellt Funktionalität für Mathematik, Naturwissenschaft und Ingenieuranwendungen zur Verfügung
- Noch in Entwicklung
- Bei NumPy geht es im Wesentlichen um (N-dimensionale) Arrays.
- SciPy stellt eine große Zahl an Werkzeugen zur Verfügung, die diese Arrays verwenden.
- SciPy beinhaltet die NumPy Funktionalität (nur ein import nötig).
- Es gibt noch einige weitere Bibliotheken für wissenschaftliches Rechnen (teilweise aufbauend auf NumPy und SciPy).
- Hier daher nur eine kurze Übersicht
- Weitere Informationen auf www.scipy.org

SciPy - Unterpakete

| | |
|-------------|--|
| cluster | Clustering Algorithmen |
| constants | Physikalische und mathematische Konstanten |
| fftpack | Fast Fourier Transformation |
| integrate | Integration und Löser für gewöhnliche DGLs |
| interpolate | Interpolation und Splines |
| io | Ein- und Ausgaberroutinen |
| linalg | Lineare Algebra |
| maxentropy | Maximale Entropie |
| ndimage | N-dimensionale Bildverarbeitung |
| odr | Orthogonal distance regression |
| optimize | Optimierung und Nullstellensuche |
| signal | Signalverarbeitung |
| sparse | Dünne Matrizen und zugehörige Routinen |
| spatial | Räumliche Datenstrukturen und Algorithmen |
| special | Spezielle Funktionen |
| stats | Statistische Verteilungen und Funktionen |
| weave | Integration von C/C++ Programmen |

Spezielle Funktionen

- Airy-Funktion
- Elliptische Integralfunktion
- Bessel-Funktionen (+ Nullstellen, Integrale, Ableitungen,...)
- Struve-Funktion
- Jede Menge statistische Funktionen
- Gamma-Funktion
- Hypergeometrische Funktionen
- Orthogonale Polynome (Legendre, Chebyshev, Jacobi,...)
- Parabolische Zylinderfunktion
- Mathieu-Funktion
- Kelvin-Funktion
- ...

Beispiel: Interpolation - Linear

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import interpolate

x = np.arange(0, 2.25*np.pi, np.pi/4)
y = np.sin(x)
f = interpolate.interp1d(x, y)

xnew = np.arange(0, 2.25*np.pi, np.pi/4)
ynew = f(xnew)
plt.plot(x, y, 'o', xnew, ynew, '-')
plt.title('Lineare Interpolation')
plt.show()
```

Beispiel: Interpolation - Kubische Splines

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import interpolate

x = np.arange(0, 2.25*np.pi, np.pi/4)
y = np.sin(x)
spline = interpolate.splrep(x,y,s=0)
xnew = np.arange(0,2.02*np.pi,np.pi/50)
ynew = interpolate.splev(xnew, spline)

plt.plot(x,y,'o',xnew,ynew)
plt.legend(['Interp-Punkte', 'Kubischer Spline'])
plt.axis([-0.05,6.33,-1.05,1.05])
plt.title('Interpolation mit kubischen Splines')
plt.show()
```