Numpy Tutorium

January 11, 2010

Numpy ist eine Python Bibliothek die einen mehrdimensionalen Arraydatentyp bereitstellt. Damit bekommt man in Python eine sehr ähnliche Funktionalität wie in Matlab.
Um Numpy verwenden zu können müssen wir zuerst das entsprechende Modul laden.

Sage code

```
import numpy
```

Alle Numpy Funktionen beginnen mit **numpy.**
Mit <TAB> können wir eine Liste von allen Numpy Funktionen anzeigen.

Sage code

```
numpy.a
```

Wir erstellen ein eindimensionales Array aus einer Liste von Integern.

Sage code

```
a = numpy.array([0,..,14])
a
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
```

Sage code

```
type(a)
```

```
<type 'numpy.ndarray'>
```

Den Typ der Arrayelemente erhält man mit **dtype**

Sage code

```
a.dtype
```

```
dtype('int32')
```

Die Anzahl der Elemente mittels **shape**

Sage code

```
a.shape
```

```
(15,)
```

Zugriff auf die Elemente erfolgt ähnlich wie bei Listen.

Sage code

```
a[2]
```

```
2
```

Sage code

```
a[0:5]
```

```
array([0, 1, 2, 3, 4])
```

Sage code
```
a[0:10:3]
```

```
array([0, 3, 6, 9])
```

Mit **reshape** können wir a in ein 2-dimensionales Array umwandeln.

Sage code
```
b = a.reshape((5,3))
b
```

```
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11],
       [12, 13, 14]])
```

Sage code
```
b.shape
```

```
(5, 3)
```

Wir benötigen jetzt zwei Indices um auf Elemente zugreifen zu können.

Sage code
```
print b[0,0]
print b[2,1]
```

```
0
7
```

Wir können Slices auch auf mehrdimensionale Arrays anwenden.

Sage code
```
print b[0:2,0:2]
```

```
[[0 1]
 [3 4]]
```

Die zweite Zeile des Arrays

Sage code
```
b[1]
```

```
array([3, 4, 5])
```

Die erste Spalte

Sage code
```
b[:,0]
```

```
array([ 0,  3,  6,  9, 12])
```

0.1 Rechnen mit Arrays

Mathematische Operationen werden auf Arrays Elementweise durchgeführt.

Sage code
```
3*b
```

```
array([[ 0,  3,  6],
       [ 9, 12, 15],
       [18, 21, 24],
       [27, 30, 33],
       [36, 39, 42]])
```

Sage code

```
b^2
```

```
array([[  0,   1,   4],
       [  9,  16,  25],
       [ 36,  49,  64],
       [ 81, 100, 121],
       [144, 169, 196]])
```

Sage code

```
b+b^2
```

```
array([[  0,   2,   6],
       [ 12,  20,  30],
       [ 42,  56,  72],
       [ 90, 110, 132],
       [156, 182, 210]])
```

Sage code

```
sin(b)
```

```
array([[ 0.        ,  0.84147098,  0.90929743],
       [ 0.14112001, -0.7568025 , -0.95892427],
       [-0.2794155 ,  0.6569866 ,  0.98935825],
       [ 0.41211849, -0.54402111, -0.99999021],
       [-0.53657292,  0.42016704,  0.99060736]])
```

Nicht alle Sage Funktionen können auf Numpy Matrizen aufgerufen werden, z.B. **sqrt()**
In diesen Fällen sollte man die in äquivalente Funktion aus dem Numpy Modul verwenden: **numpy.sqrt()**

Sage code

```
sqrt(b)
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "_sage_input_23.py", line 4, in <module>
    exec compile(ur'sqrt(b)' + '\n', '', 'single')
  File "", line 1, in <module>

  File "/local/data/huss/software/sage/local/lib/python2.6/site-packages/sage/functio\
ns/other.py", line 836, in __call__
    return self._do_sqrt(x, *args, **kwds)
  File "/local/data/huss/software/sage/local/lib/python2.6/site-packages/sage/functio\
ns/other.py", line 783, in _do_sqrt
    if x == -1:
ValueError: The truth value of an array with more than one element is ambiguous. Use \
a.any() or a.all()
```

Sage code

```
numpy.sqrt(b)
```

```
array([[ 0.        ,  1.        ,  1.41421356],
       [ 1.73205081,  2.        ,  2.23606798],
       [ 2.44948974,  2.64575131,  2.82842712],
       [ 3.        ,  3.16227766,  3.31662479],
       [ 3.46410162,  3.60555128,  3.74165739]])
```

0.2 Komplexere Indizierungen

Sage code

```
e = numpy.arange(1,11)
e[3] = 7
e[8] = 1
e.resize(2,5)
e
```

```
array([[ 1,  2,  3,  7,  5],
       [ 6,  7,  8,  1, 10]])
```

Wir geben alle Elemente zurück die größer oder gleich 5 sind.

Sage code

```
e[e >= 5]
```

```
array([ 7,  5,  6,  7,  8, 10])
```

Alle Elemente ≥ 5 werden auf 0 gesetzt.

Sage code

```
e[e >= 5] = 0

e
```

```
array([[1, 2, 3, 0, 0],
       [0, 0, 0, 1, 0]])
```

0.3 Erzeugen eines Arrays mit Einträgen der Form $f(x, y)$

Sage code

```
def f(x,y):
    return (x^2+y^2)

c = numpy.fromfunction(f, (50,50), dtype=int)
```

Große arrays werden standardmäßig abgekürzt gedruckt.

Sage code

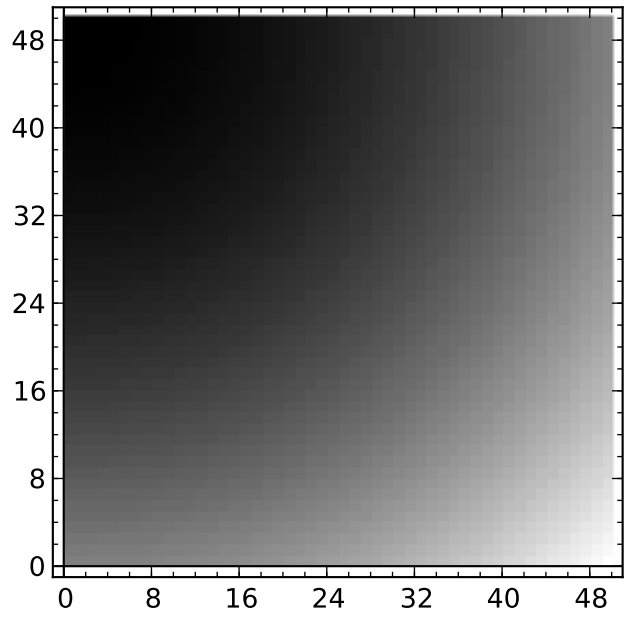
```
c
```

```
array([[   0,    1,    4, ..., 2209, 2304, 2401],
       [   1,    2,    5, ..., 2210, 2305, 2402],
       [   4,    5,    8, ..., 2213, 2308, 2405],
       ...,
       [2209, 2210, 2213, ..., 4418, 4513, 4610],
       [2304, 2305, 2308, ..., 4513, 4608, 4705],
       [2401, 2402, 2405, ..., 4610, 4705, 4802]])
```

Zweidimensionale Arrays kann man bequem mit der Funktion **matrix_plot()** visualisieren.

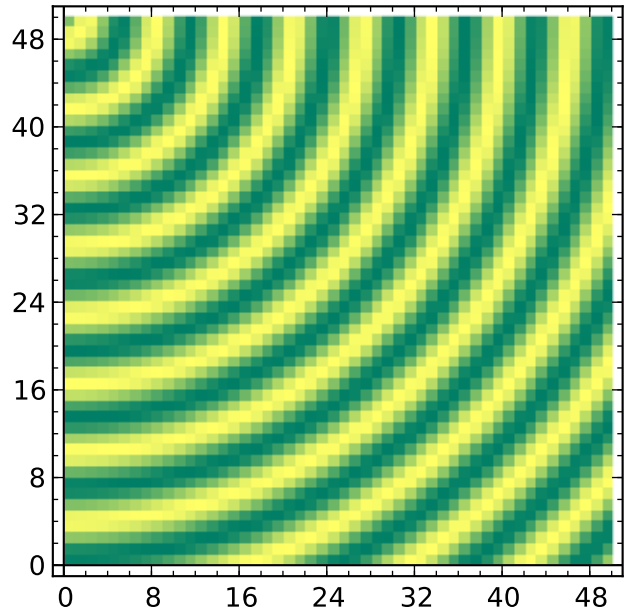
Sage code

```
matrix_plot(c)
```



Sage code

```
matrix_plot(sin(c^(1/2)), cmap = 'summer')
```



0.4 Ein Array mit Einträgen aus den komplexen Zahlen.

Sage code

```
def g(x,y):
    return (x+CDF(I)*y)
```

Sage code

```
d = numpy.fromfunction(g, (4, 4))
d
```

```
array([[ 0.+0.j,  0.+1.j,  0.+2.j,  0.+3.j],
       [ 1.+0.j,  1.+1.j,  1.+2.j,  1.+3.j],
       [ 2.+0.j,  2.+1.j,  2.+2.j,  2.+3.j],
       [ 3.+0.j,  3.+1.j,  3.+2.j,  3.+3.j]])
```

Das Array hat Werte vom Typ Komplexe Zahlen mit 128 Bits Genauigkeit.

Sage code

```
d.dtype
```

```
dtype('complex128')
```