Einführungskurs Python

Tom Hanika

25. April 2018

Wissenschaftliches Arbeiten mit Python

Ziele des Abschnitts

Ziele

Ziele des Abschnitts

Ziele

- Vorstellung von „Standard-Bibliotheken“

Ziele des Abschnitts

Ziele

- Vorstellung von „Standard-Bibliotheken“
- Vorstellung des „Python Scientific Computing Environments“ (SciPy)

Ziele des Abschnitts

Ziele

- Vorstellung von „Standard-Bibliotheken“
- Vorstellung des „Python Scientific Computing Environments“ (SciPy)

SciPy, was ist das? Ein großes Konglomerat von Python-Bibliotheken zum:

Ziele des Abschnitts

Ziele

- Vorstellung von „Standard-Bibliotheken“
- Vorstellung des „Python Scientific Computing Environments“ (SciPy)

SciPy, was ist das? Ein großes Konglomerat von Python-Bibliotheken zum:

- **Visualisieren** von verschiedensten Zusammhängen (2D Plot, 3D Plot, Animationen, bis hin zu fotorealistischen Darstellungen)

Ziele des Abschnitts

Ziele

- Vorstellung von „Standard-Bibliotheken“
- Vorstellung des „Python Scientific Computing Environments“ (SciPy)

SciPy, was ist das? Ein großes Konglomerat von Python-Bibliotheken zum:

- **Visualisieren** von verschiedensten Zusammhängen (2D Plot, 3D Plot, Animationen, bis hin zu fotorealistischen Darstellungen)
- Lösen von **Optimierungsproblemen**.

Ziele des Abschnitts

Ziele

- Vorstellung von „Standard-Bibliotheken“
- Vorstellung des „Python Scientific Computing Environments“ (SciPy)

SciPy, was ist das? Ein großes Konglomerat von Python-Bibliotheken zum:

- **Visualisieren** von verschiedensten Zusammhängen (2D Plot, 3D Plot, Animationen, bis hin zu fotorealistischen Darstellungen)
- Lösen von **Optimierungsproblemen**.
- **Datenanalyse**, zum Beispiel mittels importierten Paketen von `R`.

Ziele des Abschnitts

Ziele

- Vorstellung von „Standard-Bibliotheken“
- Vorstellung des „Python Scientific Computing Environments“ (SciPy)

SciPy, was ist das? Ein großes Konglomerat von Python-Bibliotheken zum:

- **Visualisieren** von verschiedensten Zusammhängen (2D Plot, 3D Plot, Animationen, bis hin zu fotorealistischen Darstellungen)
- Lösen von **Optimierungsproblemen**.
- **Datenanalyse**, zum Beispiel mittels importierten Paketen von `R`.
- Nutzen von **symbolischer Mathematik**

Ziele des Abschnitts

Ziele

- Vorstellung von „Standard-Bibliotheken“
- Vorstellung des „Python Scientific Computing Environments“ (SciPy)

SciPy, was ist das? Ein großes Konglomerat von Python-Bibliotheken zum:

- **Visualisieren** von verschiedensten Zusammhängen (2D Plot, 3D Plot, Animationen, bis hin zu fotorealistischen Darstellungen)
- Lösen von **Optimierungsproblemen**.
- **Datenanalyse**, zum Beispiel mittels importierten Paketen von `R`.
- Nutzen von **symbolischer Mathematik**
- Bereitstellen von besseren, an die zu lösenden Probleme angepasste, **interaktiven Interpretern**.

Ziele des Abschnitts

Ziele

- Vorstellung von „Standard-Bibliotheken“
- Vorstellung des „Python Scientific Computing Environments“ (SciPy)

SciPy, was ist das? Ein großes Konglomerat von Python-Bibliotheken zum:

- **Visualisieren** von verschiedensten Zusammhängen (2D Plot, 3D Plot, Animationen, bis hin zu fotorealistischen Darstellungen)
- Lösen von **Optimierungsproblemen**.
- **Datenanalyse**, zum Beispiel mittels importierten Paketen von `R`.
- Nutzen von **symbolischer Mathematik**
- Bereitstellen von besseren, an die zu lösenden Probleme angepasste, **interaktiven Interpretern**.
- Bereitstellen von Tools die dem jeweiligen **Wissenschaftsfeld angepasst sind**.

Ziele des Abschnitts

Ziele

- Vorstellung von „Standard-Bibliotheken“
- Vorstellung des „Python Scientific Computing Environments“ (SciPy)

SciPy, was ist das? Ein großes Konglomerat von Python-Bibliotheken zum:

- **Visualisieren** von verschiedensten Zusammhängen (2D Plot, 3D Plot, Animationen, bis hin zu fotorealistischen Darstellungen)
- Lösen von **Optimierungsproblemen**.
- **Datenanalyse**, zum Beispiel mittels importierten Paketen von `R`.
- Nutzen von **symbolischer Mathematik**
- Bereitstellen von besseren, an die zu lösenden Probleme angepasste, **interaktiven Interpretern**.
- Bereitstellen von Tools die dem jeweiligen **Wissenschaftsfeld angepasst sind**.
- Anbindung an **schnelle** Programmiersprachen wie Fortran 95 oder C++.

Wissenschaftliches Arbeiten mit Python

Numerisches Rechnen (NumPy)

Was ist NumPy?

[1] Basic Linear Algebra Subprograms
[2] Linear Algebra Package

Was ist NumPy?

- Spracherweiterung zu Python

[1] Basic Linear Algebra Subprograms
[2] Linear Algebra Package

Was ist NumPy?

- Spracherweiterung zu Python
- Anwendungsfeld ist (schnelles) numerisches Rechnen

[1] Basic Linear Algebra Subprograms
[2] Linear Algebra Package

Was ist NumPy?

- Spracherweiterung zu Python
- Anwendungsfeld ist (schnelles) numerisches Rechnen
- stellt mehrdimensionale Arrays und Operatoren dafür zur Verfügung

[1] Basic Linear Algebra Subprograms
[2] Linear Algebra Package

Was ist NumPy?

- Spracherweiterung zu Python
- Anwendungsfeld ist (schnelles) numerisches Rechnen
- stellt mehrdimensionale Arrays und Operatoren dafür zur Verfügung
- lagert Rechnungen an schnelle Bibliotheken BLAS[1] und LAPACK[2] aus.

[1] Basic Linear Algebra Subprograms
[2] Linear Algebra Package

Was ist NumPy?

- Spracherweiterung zu Python
- Anwendungsfeld ist (schnelles) numerisches Rechnen
- stellt mehrdimensionale Arrays und Operatoren dafür zur Verfügung
- lagert Rechnungen an schnelle Bibliotheken BLAS[1] und LAPACK[2] aus.

Woher bekomme ich es?

[1] Basic Linear Algebra Subprograms
[2] Linear Algebra Package

Was ist NumPy?

- Spracherweiterung zu Python
- Anwendungsfeld ist (schnelles) numerisches Rechnen
- stellt mehrdimensionale Arrays und Operatoren dafür zur Verfügung
- lagert Rechnungen an schnelle Bibliotheken BLAS[1] und LAPACK[2] aus.

Woher bekomme ich es?

- In Debian-ähnlichen OS: `aptitude install python3-numpy`

[1] Basic Linear Algebra Subprograms
[2] Linear Algebra Package

Was ist NumPy?

- Spracherweiterung zu Python
- Anwendungsfeld ist (schnelles) numerisches Rechnen
- stellt mehrdimensionale Arrays und Operatoren dafür zur Verfügung
- lagert Rechnungen an schnelle Bibliotheken BLAS[1] und LAPACK[2] aus.

Woher bekomme ich es?

- In Debian-ähnlichen OS: `aptitude install python3-numpy`
- Die neueste Version ist erhältlich unter:
  `git clone git://github.com/numpy/numpy.git numpy`

[1] Basic Linear Algebra Subprograms
[2] Linear Algebra Package

Was ist NumPy?

- Spracherweiterung zu Python
- Anwendungsfeld ist (schnelles) numerisches Rechnen
- stellt mehrdimensionale Arrays und Operatoren dafür zur Verfügung
- lagert Rechnungen an schnelle Bibliotheken BLAS[1] und LAPACK[2] aus.

Woher bekomme ich es?

- In Debian-ähnlichen OS: `aptitude install python3-numpy`
- Die neueste Version ist erhältlich unter:
  `git clone git://github.com/numpy/numpy.git numpy`
- Am besten via `pip3 install --user numpy`

[1] Basic Linear Algebra Subprograms
[2] Linear Algebra Package

Was ist NumPy?

- Spracherweiterung zu Python
- Anwendungsfeld ist (schnelles) numerisches Rechnen
- stellt mehrdimensionale Arrays und Operatoren dafür zur Verfügung
- lagert Rechnungen an schnelle Bibliotheken BLAS[1] und LAPACK[2] aus.

Woher bekomme ich es?

- In Debian-ähnlichen OS: `aptitude install python3-numpy`
- Die neueste Version ist erhältlich unter:
  `git clone git://github.com/numpy/numpy.git numpy`
- Am besten via `pip3 install --user numpy`

Wie aktiviert man es?

```
1 import numpy as np
```

[1] Basic Linear Algebra Subprograms
[2] Linear Algebra Package

Einschub Package-installer (pip)

Einschub Package-installer (pip)

- Python besitzt seinen eigenen Package-installer pip.

Einschub Package-installer (pip)

- Python besitzt seinen eigenen Package-installer pip.
- Dieser ermöglicht unabhängig vom Betriebssystem Python-Pakete zu installieren aus dem PyPI (Python Package Index).

Einschub Package-installer (pip)

- Python besitzt seinen eigenen Package-installer pip.
- Dieser ermöglicht unabhängig vom Betriebssystem Python-Pakete zu installieren aus dem PyPI (Python Package Index).

Howto PyPi

Einschub Package-installer (pip)

- Python besitzt seinen eigenen Package-installer pip.
- Dieser ermöglicht unabhängig vom Betriebssystem Python-Pakete zu installieren aus dem PyPI (Python Package Index).

Howto PyPi

- `sudo apt-get install python3-pip` (einmalig)

Einschub Package-installer (pip)

- Python besitzt seinen eigenen Package-installer pip.
- Dieser ermöglicht unabhängig vom Betriebssystem Python-Pakete zu installieren aus dem PyPI (Python Package Index).

Howto PyPi

- `sudo apt-get install python3-pip` (einmalig)
- danach steht das Kommando `pip3` zur Verfügung

Einschub Package-installer (pip)

- Python besitzt seinen eigenen Package-installer pip.
- Dieser ermöglicht unabhängig vom Betriebssystem Python-Pakete zu installieren aus dem PyPI (Python Package Index).

Howto PyPi

- `sudo apt-get install python3-pip` (einmalig)
- danach steht das Kommando `pip3` zur Verfügung
- `pip3 install --user 'SomePackage'`, zum Beispiel:
  `pip3 install 'numpy'`

Einschub Package-installer (pip)

- Python besitzt seinen eigenen Package-installer pip.
- Dieser ermöglicht unabhängig vom Betriebssystem Python-Pakete zu installieren aus dem PyPI (Python Package Index).

Howto PyPi

- `sudo apt-get install python3-pip` (einmalig)
- danach steht das Kommando `pip3` zur Verfügung
- `pip3 install --user 'SomePackage'`, zum Beispiel:
  `pip3 install 'numpy'`

Für viele python-packages wird das Betriebssystempaket `python3-dev` vorausgesetzt, also `sudo aptitude install python3-dev`.

Los geht's NumPy: Arrays („To the n-th dimension“)

```
import numpy as np; import math as ma
A = np.arange(12).reshape(3, 4)
print(A); type(A)
A.dtype

B = A * ma.pi
print(B); type(B)
B.dtype

SomeList = [1.2, 3.4, 7.8, 9.6, 10.4, 7.4, 9.9, 3.3, 2.2]
C = np.array(SomeList).reshape(3,3); v = np.array([1,2,3])
C * v #?
C.dot(v)
```

NumPy: Matrices („To the 2nd dimension“)

```
import numpy as np;
A = np.arange(9).reshape(3, 3)
M = np.matrix(A)
type(M)
print(A*A)
[[ 0  1  4]
 [ 9 16 25]
 [36 49 64]]
print(M*M)
[[ 15  18  21]
 [ 42  54  66]
 [ 69  90 111]]
print(M.I)  # matrices have inverse by default
v= np.matrix([[1],[2],[3]])
print(M*v)  # does what one expects
```

NumPy: Matrices („To the 2nd dimension“)

```
import numpy as np;
A = np.arange(9).reshape(3, 3)
M = np.matrix(A)
type(M)
print(A*A)
[[ 0  1  4]
 [ 9 16 25]
 [36 49 64]]
print(M*M)
[[ 15  18  21]
 [ 42  54  66]
 [ 69  90 111]]
print(M.I)  # matrices have inverse by default
v= np.matrix([[1],[2],[3]])
print(M*v)  # does what one expects
```

NumPy: Lineare Algebra

```
import numpy as np; import numpy.linalg as npalg
M = np.matrix([1,2,0,2,0,0,0,3,3]).reshape(3, 3)
v = np.matrix([[1],[2],[3]])

print(npalg.det(M))

print(npalg.eigvals(M))

print(npalg.eig(M)) # Eigenvalues and (right) eigenvectors
print(npalg.solve(M,v))
```

NumPy: Lineare Algebra

```
import numpy as np; import numpy.linalg as npalg
M = np.matrix([1,2,0,2,0,0,0,3,3]).reshape(3, 3)
v = np.matrix([[1],[2],[3]])

print(npalg.det(M))

print(npalg.eigvals(M))

print(npalg.eig(M)) # Eigenvalues and (right) eigenvectors
print(npalg.solve(M,v))
```

Für alles was man brauchen könnte gibt es eine methode.

http://docs.scipy.org/doc/numpy/reference/routines.linalg.html

Common mistake!

```
>>> a = np.array(1,2,3,4)     # WRONG
>>> a = np.array([1,2,3,4])   # RIGHT
```

Reshape!

```
>>> b = np.arange(12).reshape(4,3); print(b) # 2d array
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
>>> c = np.arange(24).reshape(2,3,4); print(c) # 3d array
[[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]
 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]
```

Wissenschaftliches Arbeiten mit Python

It's all about that (drawing) space: *Matplotlib*

matplotlib, pyplot and pylab?!?

- `matplotlib` ist das „whole package“

matplotlib, pyplot and pylab?!?

- `matplotlib` ist das „whole package“
- `pyplotb` ist ein Modul in `matplotlib`, vorzugsweise zu verwenden für nicht-interaktive Plots.

matplotlib, pyplot and pylab?!?

- `matplotlib` ist das „whole package“
- `pyplotb` ist ein Modul in `matplotlib`, vorzugsweise zu verwenden für nicht-interaktive Plots.
- `pylab` ist vorzugsweise in interaktiven Berechnungen/Plots zu verwenden.

matplotlib, pyplot and pylab?!?

- `matplotlib` ist das „whole package“
- `pyplotb` ist ein Modul in `matplotlib`, vorzugsweise zu verwenden für nicht-interaktive Plots.
- `pylab` ist vorzugsweise in interaktiven Berechnungen/Plots zu verwenden.

Viele Beispiele zu Plots auf dieser und auf den nächsten Folien wurden `http://matplotlib.org/users/screenshots.html` entnommen.

matplotlib, pyplot and pylab?!?

- `matplotlib` ist das „whole package“
- `pyplotb` ist ein Modul in `matplotlib`, vorzugsweise zu verwenden für nicht-interaktive Plots.
- `pylab` ist vorzugsweise in interaktiven Berechnungen/Plots zu verwenden.

Viele Beispiele zu Plots auf dieser und auf den nächsten Folien wurden `http://matplotlib.org/users/screenshots.html` entnommen.

Nutzung von matplotlib und Python3 unter Debian

`sudo aptitude install tk-dev`,
`sudo aptitude install python-tk`, und danach
`pip3 install --user matplotlib`.

Simple plot (code)

```
import matplotlib.pyplot as plt
plt.plot([1,2,3,4])
plt.show()
```

Little less simple plot (code)

```
from pylab import *

t = arange(0.0, 2.0, 0.01)
s = sin(2*pi*t)
plot(t, s)

xlabel('time␣(s)')
ylabel('voltage␣(mV)')
title('About␣as␣simple␣as␣it␣gets,␣folks')
grid(True)
savefig("test.png")
show()
```

Little less simple plot (Bild)

3D Plot

```
from mpl_toolkits.mplot3d import Axes3D;  import numpy as np
from matplotlib import cm; import matplotlib.pyplot as plt;
from matplotlib.ticker import LinearLocator,FormatStrFormatter

fig = plt.figure(); ax = fig.gca(projection='3d')
X = np.arange(-5, 5, 0.25); Y = np.arange(-5, 5, 0.25)
X, Y = np.meshgrid(X, Y)
R = np.sqrt(X**2 + Y**2); Z = np.sin(R)
surf = ax.plot_surface(X, Y, Z, rstride=1, cstride=1,
        cmap=cm.coolwarm,linewidth=0, antialiased=False)
ax.set_zlim(-1.01, 1.01)
ax.zaxis.set_major_locator(LinearLocator(10))
ax.zaxis.set_major_formatter(FormatStrFormatter('%.02f'))

fig.colorbar(surf, shrink=0.5, aspect=5)
plt.show()
```

3D plot (Bild)

1.01
0.79
0.56
0.34
0.11
−0.11
−0.34
−0.56
−0.79
−1.01
−6
−4
−2
0
2
4
6
−6
−4
−2
0
2
4
6
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8

Polar Plot (Code)

```
import numpy as np
import matplotlib.pyplot as plt


r = np.arange(0, 3.0, 0.01)
theta = 2 * np.pi * r

ax = plt.subplot(111, polar=True)
ax.plot(theta, r, color='r', linewidth=3)
ax.set_rmax(2.0)
ax.grid(True)

ax.set_title("A␣line␣plot␣on␣a␣polar␣axis", va='bottom')
plt.show()
```

Polar Plot (Bild)

Of course you can use TEX (code)

```
import numpy as np; import matplotlib.pyplot as plt
t = np.arange(0.0, 1.0 + 0.01, 0.01)
s = np.cos(4 * np.pi * t) + 2

plt.rc('text', usetex=True);  plt.rc('font', family='serif')
plt.plot(t, s)

plt.xlabel(r'\textbf{time}␣(s)')
plt.ylabel(r'\textit{voltage}␣(mV)',fontsize=16)
plt.title(r"\TeX\␣is␣Number␣"
       r"␣$\displaystyle\sum_{n=1}^\infty\frac{-e^{i\pi}}{2^n}$
       fontsize=16, color='gray')
# Make room for the ridiculously large title.
plt.subplots_adjust(top=0.8)
plt.savefig('tex_demo')
plt.show()
```

Of course you can use TEX (code)

Scientific and beyond: XKCD (code)

```
import matplotlib.pyplot as plt; import numpy as np

plt.xkcd()
 # Based on 'Stove Ownership' from XKCD by Randall Monroe
fig = plt.figure();  ax = fig.add_axes((0.1, 0.2, 0.8, 0.7))
plt.xticks([]);  plt.yticks([]); ax.set_ylim([-30, 10])

data = np.ones(100); data[70:] -= np.arange(30)
plt.annotate(
 'THE␣DAY␣I␣REALIZED\nI␣COULD␣COOK␣BACON\nWHENEVER␣I␣WANTED',
  xy=(70, 1), arrowprops=dict(arrowstyle='->'),
  xytext=(15, -10))

plt.plot(data);
plt.xlabel('time'); plt.ylabel('my␣overall␣health')
plt.show()
```

Scientific and beyond: XKCD (Bild)

In Debian muss das Paket: `fonts-humor-sans` installiert sein. Weiterhin muss der möglicherweise schon erstellte font-cache mittels `rm ./.cache/matplotlib/fontList.py3k.cache` gelöscht werden.

Wissenschaftliches Arbeiten mit Python

Symbolisches Rechnen: SymPy

Es geht eine Menge...

- Einfache Terme umformen via einfachen arithmetischen Operationen.

Es geht eine Menge...

- Einfache Terme umformen via einfachen arithmetischen Operationen.
- Rechnen mit Polynomen (bis hin zu Gröbnerbasen).

Es geht eine Menge...

- Einfache Terme umformen via einfachen arithmetischen Operationen.
- Rechnen mit Polynomen (bis hin zu Gröbnerbasen).
- Berechnen von Abeleitungen, Integralen und Taylorreihen.

Es geht eine Menge...

- Einfache Terme umformen via einfachen arithmetischen Operationen.
- Rechnen mit Polynomen (bis hin zu Gröbnerbasen).
- Berechnen von Abeleitungen, Integralen und Taylorreihen.
- Symbolisches lösen von Gleichungen und Differentialgleichungen.

Es geht eine Menge...

- Einfache Terme umformen via einfachen arithmetischen Operationen.
- Rechnen mit Polynomen (bis hin zu Gröbnerbasen).
- Berechnen von Abeleitungen, Integralen und Taylorreihen.
- Symbolisches lösen von Gleichungen und Differentialgleichungen.
- Symbolisches Rechnen mit Matrizen.

Es geht eine Menge...

- Einfache Terme umformen via einfachen arithmetischen Operationen.
- Rechnen mit Polynomen (bis hin zu Gröbnerbasen).
- Berechnen von Abeleitungen, Integralen und Taylorreihen.
- Symbolisches lösen von Gleichungen und Differentialgleichungen.
- Symbolisches Rechnen mit Matrizen.
- Rechnen mit Punkten, Schnitten und Tangenten

Es geht eine Menge...

- Einfache Terme umformen via einfachen arithmetischen Operationen.
- Rechnen mit Polynomen (bis hin zu Gröbnerbasen).
- Berechnen von Abeleitungen, Integralen und Taylorreihen.
- Symbolisches lösen von Gleichungen und Differentialgleichungen.
- Symbolisches Rechnen mit Matrizen.
- Rechnen mit Punkten, Schnitten und Tangenten
- Nutzen von physikalische Einheiten.

Es geht eine Menge...

- Einfache Terme umformen via einfachen arithmetischen Operationen.
- Rechnen mit Polynomen (bis hin zu Gröbnerbasen).
- Berechnen von Abeleitungen, Integralen und Taylorreihen.
- Symbolisches lösen von Gleichungen und Differentialgleichungen.
- Symbolisches Rechnen mit Matrizen.
- Rechnen mit Punkten, Schnitten und Tangenten
- Nutzen von physikalische Einheiten.

Woher bekommt man es?

- `sudo aptitude install python-sympy` (!Python2 only!), oder
- `pip3 install --user sympy`

Los geht's SymPy: Klassifizieren und in chic

```
import sympy as sy
x = sy.symbols('x') # Klassifiziert x zum ein Symbol
y, z = sy.symbols('y z')
print(x+y*z)
sy.pprint(x+y*z)  # PrettyPrint-Funktion von sympy

f = sy.Function('f')
sy.pprint(sy.Integral(f(x), x))

f = 1/sy.cos(x)
sy.pprint(f.series(x, 0, 10))

sy.pprint(sy.expand((x+y)**5))
```

SymPy: Funktionen, Grenzwerte und Integrale

```
from sympy import *
init_printing()  #Schaltet pretty printing generell an
x = Symbol('x')
f = Function('f'); f = sqrt(x**3 - 5*x + 2)
limit(f, x, oo); limit(f, x, -oo); limit(f, x, 0)

g = Function('g'); g = (x+1) / (x-1)
limit(g, x, oo); limit(g, x, -oo); limit(g, x, 0)

limit(f/g,x,0); limit(f/g,x,1)
```

SymPy: Simplify and calculus

```
from sympy import *
init_printing()
x, y, z = symbols('x y z')

simplify(sin(x)**2 + cos(x)**2)
simplify((x**3 + x**2 - x - 1)/(x**2 + 2*x + 1))
simplify(gamma(x)/gamma(x - 2))
```

```
diff(x**4, x, 2);  diff(x**4, x, 3)
diff(exp((x*y*z +y)**2),x)
integrate(cos(x), x)
integrate(exp(-x), (x, 0, oo))
integrate(exp(-x**2 - y**2), (x, -oo, oo), (y, -oo, oo))
```

SymPy: Solve my Equation!

Gleichungen werden mit Eq(*linker Term*,*rechter Term*) definiert.

```
from sympy import *
init_printing()
x, y, z = symbols('x y z')
f = Function('f')

solve(Eq(4, x*2),x)

solve(Eq(y*x-4*x*z, z**2-4*x),x)
```

```
f = Function('f')
k = symbols('k')
g = f(x).diff(x, x) + k**2*f(x)
diffeq = Eq(g,0)
dsolve(diffeq,f(x))
```

Wissenschaftliches Arbeiten mit Python

SciPy und Freunde, was geht noch?

Konvexen Hülle einer Punktmenge (scipy.spatial)

```
from scipy.spatial import ConvexHull
import numpy as np
import matplotlib.pyplot as plt

points = np.random.randn(40, 2)
hull = ConvexHull(points)
plt.plot(points[:,0], points[:,1], 'o')
for simplex in hull.simplices:
  plt.plot(points[simplex,0], points[simplex,1], 'k-')

plt.savefig('con.pdf')
plt.show()
```

Konvexen Hülle einer Punktmenge (scipy.spatial) (bild)

Graphs Graphs Graphs (networkx)

```
import networkx as nx
import matplotlib.pyplot as plt; import pylab

G = nx.complete_graph(5)
nx.draw_circular(G,node_size=1500)
pylab.show()
```

Strongly connected components (networkx)

`pip3 install --user networkx`

```
import networkx as nx
import matplotlib.pyplot as plt; import pylab
G = nx.DiGraph()
for e in [(1,2),(2,3),(3,1),(3,4),(4,5),(5,6),(6,7),(7,4)]:
    G.add_edge(*e)

SCC  = nx.strongly_connected_components(G)
SCCG = nx.strongly_connected_component_subgraphs(G)
for c in SCC:
    print(c)

for g in SCCG:
    nx.draw_circular(g,node_size=1500)
    pylab.show()
```

Strongly connected components (networkx)

`pip3 install --user networkx`

```
import networkx as nx
import matplotlib.pyplot as plt; import pylab
G = nx.DiGraph()
for e in [(1,2),(2,3),(3,1),(3,4),(4,5),(5,6),(6,7),(7,4)]:
    G.add_edge(*e)

SCC  = nx.strongly_connected_components(G)
SCCG = nx.strongly_connected_component_subgraphs(G)
for c in SCC:
    print(c)

for g in SCCG:
    nx.draw_circular(g,node_size=1500)
    pylab.show()
```

Strongly connected components (networkx)

`pip3 install --user networkx`

```
import networkx as nx
import matplotlib.pyplot as plt; import pylab
G = nx.DiGraph()
for e in [(1,2),(2,3),(3,1),(3,4),(4,5),(5,6),(6,7),(7,4)]:
    G.add_edge(*e)

SCC  = nx.strongly_connected_components(G)
SCCG = nx.strongly_connected_component_subgraphs(G)
for c in SCC:
    print(c)

for g in SCCG:
    nx.draw_circular(g,node_size=1500)
    pylab.show()
```

Strongly connected components (networkx)

`pip3 install --user networkx`

```
import networkx as nx
import matplotlib.pyplot as plt; import pylab
G = nx.DiGraph()
for e in [(1,2),(2,3),(3,1),(3,4),(4,5),(5,6),(6,7),(7,4)]:
    G.add_edge(*e)

SCC  = nx.strongly_connected_components(G)
SCCG = nx.strongly_connected_component_subgraphs(G)
for c in SCC:
    print(c)

for g in SCCG:
    nx.draw_circular(g,node_size=1500)
    pylab.show()
```

Lifeforms, (scikits.bio)

`pip3 install --user scikit-bio`

Ein Problem aus der Biologie: Finde die längste Folge von Purinen (Guanin oder Adenin) in einem gegebenen DNS-Strang.

```
from skbio import NucleotideSequence
nuc = NucleotideSequence.read(
        'single_sequence1.fasta', seq_num=1)

max(nuc.find_features('purine_run'), key=lambda e: len(e[2]))
```

Making things faster: Numba

„Numba works by generating optimized machine code using the LLVM compiler [..] and is designed to integrate with the Python scientific software stack.“

Beispiel aus Numba Docu

```
from numba import jit; from numpy import arange
# jit decorator tells Numba to compile this function.
@jit
def sum2d(arr):
    M, N = arr.shape; result = 0.0
    for i in range(M):
        for j in range(N):
            result += arr[i,j]
    return result

a = arange(9).reshape(3,3)
print(sum2d(a))
```

NLTK - Natural Language Toolkit

„NLTK is a leading platform for building Python programs to work with human language data.“

Beispiel

```
import nltk
s = """The lecture by Tom about Python was fantastic."""
tokens = nltk.word_tokenize(s)
['The', 'lecture', 'by', 'Tom', 'about',
  'Python', 'was', 'fantastic', '.']
tagged = nltk.pos_tag(tokens)
[('The', 'DT'), ('lecture', 'NN'), ('by', 'IN'),
 ('Tom', 'NNP'), ('about', 'IN'), ('Python', 'NNP'),
 ('was', 'VBD'), ('fantastic', 'JJ'), ('.', '.')]
```

Wobei: `DT - Determiner`, `NN - Noun, singular or mass`,
`IN - Preposition or subordinating conjunction`
`NNP - Proper noun, singular`, `VBD - Verb, past tense`

Pandas - data manipulation and analysis

Beispiel Pandas Docu

```
import pandas as pd #this is how I usually import pandas
names = ['Bob','Jessica','Mary','John','Mel']
births = [968, 155, 77, 578, 973]
BabyDataSet = list(zip(names,births))
[('Bob', 968), ('Jessica', 155), ('Mary', 77),
 ('John', 578), ('Mel', 973)]
pd.DataFrame(data = BabyDataSet, columns=['Names', 'Births'])

     Names  Births
0      Bob     968
1  Jessica     155
2     Mary      77
3     John     578
4      Mel     973
```

Tolle Pakete für die keine Zeit war.

- PsychoPy, cognitive science und neuroscience Experimente durchführen

Tolle Pakete für die keine Zeit war.

- PsychoPy, cognitive science und neuroscience Experimente durchführen
- ScientificPython, theoretische Biophysik Simulation und mehr

Tolle Pakete für die keine Zeit war.

- PsychoPy, cognitive science und neuroscience Experimente durchführen
- ScientificPython, theoretische Biophysik Simulation und mehr
- Thuban, georaphische Datenanalyse

Tolle Pakete für die keine Zeit war.

- PsychoPy, cognitive science und neuroscience Experimente durchführen
- ScientificPython, theoretische Biophysik Simulation und mehr
- Thuban, georaphische Datenanalyse
- TensorFlow

Tolle Pakete für die keine Zeit war.

- PsychoPy, cognitive science und neuroscience Experimente durchführen
- ScientificPython, theoretische Biophysik Simulation und mehr
- Thuban, georaphische Datenanalyse
- TensorFlow
- Plotly

Tolle Pakete für die keine Zeit war.

- PsychoPy, cognitive science und neuroscience Experimente durchführen
- ScientificPython, theoretische Biophysik Simulation und mehr
- Thuban, georaphische Datenanalyse
- TensorFlow
- Plotly
- SciKit-Learn

Tolle Pakete für die keine Zeit war.

- PsychoPy, cognitive science und neuroscience Experimente durchführen
- ScientificPython, theoretische Biophysik Simulation und mehr
- Thuban, georaphische Datenanalyse
- TensorFlow
- Plotly
- SciKit-Learn
- Theano

Tolle Pakete für die keine Zeit war.

- PsychoPy, cognitive science und neuroscience Experimente durchführen
- ScientificPython, theoretische Biophysik Simulation und mehr
- Thuban, georaphische Datenanalyse
- TensorFlow
- Plotly
- SciKit-Learn
- Theano
- Scrapy

Tolle Pakete für die keine Zeit war.

- PsychoPy, cognitive science und neuroscience Experimente durchführen
- ScientificPython, theoretische Biophysik Simulation und mehr
- Thuban, georaphische Datenanalyse
- TensorFlow
- Plotly
- SciKit-Learn
- Theano
- Scrapy
- ...

Wissenschaftliches Arbeiten mit Python

Sage

Let's have some sage tea

Sage ist ein Konglomerat von Programmen, die jeweils für sich für spezielle Problemstellungen entwickelt wurden, vereint in einem Interface.

Algebra	Singular, PolyBoRi
Analysis	Maxima, SymPy
Zahlentheorie	PARI/GP, NTL
Numerik	NumPy, SciPy
Statistik	R
Lineare Algebra	LinBox, LAPACK
Graphentheorie	NetworkX
Gruppentheorie	GAP

Wie kann man es benutzen?

Let's have some sage tea

Sage ist ein Konglomerat von Programmen, die jeweils für sich für spezielle Problemstellungen entwickelt wurden, vereint in einem Interface.

Algebra	Singular, PolyBoRi
Analysis	Maxima, SymPy
Zahlentheorie	PARI/GP, NTL
Numerik	NumPy, SciPy
Statistik	R
Lineare Algebra	LinBox, LAPACK
Graphentheorie	NetworkX
Gruppentheorie	GAP

Wie kann man es benutzen?

- Mit einem browser, z. B. via `https://cloud.sagemath.com/`

Let's have some sage tea

Sage ist ein Konglomerat von Programmen, die jeweils für sich für spezielle Problemstellungen entwickelt wurden, vereint in einem Interface.

Algebra	Singular, PolyBoRi
Analysis	Maxima, SymPy
Zahlentheorie	PARI/GP, NTL
Numerik	NumPy, SciPy
Statistik	R
Lineare Algebra	LinBox, LAPACK
Graphentheorie	NetworkX
Gruppentheorie	GAP

Wie kann man es benutzen?

- Mit einem browser, z. B. via `https://cloud.sagemath.com/`
- Lokal installiert und mittels `IPython` basierter Konsole.

Let's have some sage tea

Sage ist ein Konglomerat von Programmen, die jeweils für sich für spezielle Problemstellungen entwickelt wurden, vereint in einem Interface.

Algebra	Singular, PolyBoRi
Analysis	Maxima, SymPy
Zahlentheorie	PARI/GP, NTL
Numerik	NumPy, SciPy
Statistik	R
Lineare Algebra	LinBox, LAPACK
Graphentheorie	NetworkX
Gruppentheorie	GAP

Wie kann man es benutzen?

- Mit einem browser, z. B. via `https://cloud.sagemath.com/`
- Lokal installiert und mittels `IPython` basierter Konsole.
- Via Python durch Nutzung der Sage-Bibliotheken.

Wissenschaftliches Arbeiten mit Python

Final Chapter

Lift off to „Jupyter“

„ALL THESE WORLDS ARE YOURS - EXCEPT EUROPA.
ATTEMPT NO LANDINGS THERE.“

-- HAL

Now: Life demonstration of `Jupyter`.