

# Python - die Alternative zu Matlab?

Jochen Schulz

Georg-August Universität Göttingen 

- 1 **Einleitung**
- 2 Grundlegende Bedienung Python (Spyder)
- 3 3D - Grafik
- 4 Numerische Mathematik
- 5 Zusammenfassung

# Programmieren für den Wissenschaftler

- Daten erzeugen oder erheben (Simulation, Experiment)
- Weiterverarbeitung von Daten
- Visualisierung und Validierung
- Ergebnisse veröffentlichen bzw. kommunizieren

Wir wollen: eine *High-Level* Sprache:

- Programmieren ist leicht
- Vorhandene Elemente nutzen
- geeignet für Prototyping und Debugging (Interaktion)
- Möglichst nur ein Werkzeug für alle Probleme

- MATLAB steht für **Matrix laboratory**; ursprünglich speziell Matrizenrechnung.
- Interaktives System für numerische Berechnungen und Visualisierungen (Skriptsprache).

## Vorteile

- Vielfältige Visualisierungsmöglichkeiten.
- Viele zusätzliche Toolboxes (Symb. Math T., PDE T., Wavelet T.)
- Ausgereifte und integrierte Oberfläche.

## Nachteile

- Kostenintensiv.
- Ein/Ausgabe von Dateien kann umständlich sein.
- Spezialisierter Funktionsumfang macht manche Programmierung schwer.

# Python: NumPy, SciPy, SymPy

- Modulare Skriptsprache.

## Vorteile

- Viele Module mit wissenschaftlichen Fokus.
- Klare Code-Struktur.
- Ebenso viele Module für den nicht-wissenschaftlichen Gebrauch (nützlich z.B. für Ein-/Ausgabe).
- Frei und open-source.

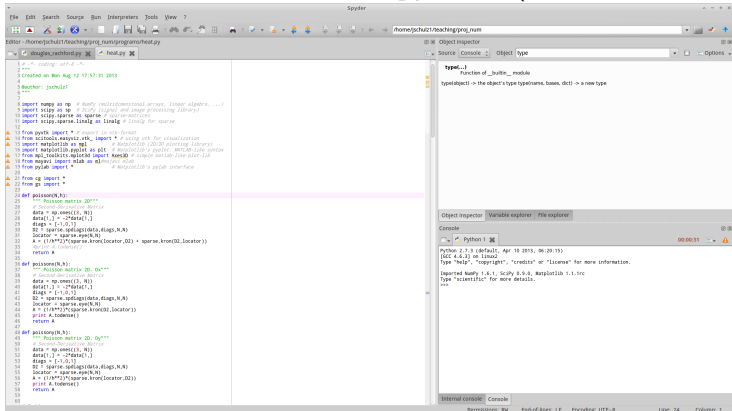
## Nachteile

- Entwicklungsumgebung etwas komplizierter (Spyder,ipython).
- Nicht alle spezialisierten Möglichkeiten anderer Software.

- 1 Einleitung
- 2 Grundlegende Bedienung Python (Spyder)**
- 3 3D - Grafik
- 4 Numerische Mathematik
- 5 Zusammenfassung

# Spyder Fenster-Aufbau

Starten von Spyder: Eingabe von `spyder &` (in einem Terminal).



- **Editor:** Dateimanipulation
- **Console:** Befehlseingabe und Standardausgabe

- **Object Inspector:** Hilfe und Variablenansicht
- **Grafik:** in separaten Fenstern

# Listen und Tuple

- Eine **Liste** ist in Python mit `[ .. , .. ]` gekennzeichnet (hat Ordnung)

```
liste = [21,22,24,23]
liste.sort(); liste
```

```
[21, 22, 23, 24]
```

- Ein **Tuple** ist in Python mit `( .. , .. )` gekennzeichnet (hat Struktur)

```
tuple = (liste[0], liste[2])
tuple, tuple[0]
```

```
(( 21, 24), 21)
```

- Liste von ganzen Zahlen von a bis b

```
range(a,b+1)
```



# Funktionen

```
def fun (arg1, arg2=<defaultvalue>, ... ,*args,**kwargs)  
    code-block  
    return returnvalue
```

```
lambda arg: codeline
```

- \*args: Tuple der Input- Argumente
- \*\*kwargs: Dictionary der benannten Input-Argumente
- \*: Entpackt Tuple in eine Liste von Argumenten
- \*\*: Entpackt Dictionary in eine Liste von benannten Argumenten
- Argumente mit Defaultwert sind optional
- Argumente mit Namen können in beliebiger Reihenfolge angegeben werden

# Funktionales Programmieren

```
map(function, liste)
```

- führt `function(x)` für alle `x` aus `Liste` aus.

```
filter(function, liste)
```

- Gibt nur diejenigen Elemente aus der Liste zurück, für die `function` `True` zurückgegeben hat.

```
>>> filter(lambda x: mod(x,3) = 0, liste)
[21, 24]
```

# Wörterbücher (Dictionaries)

- Index kann nahezu beliebige Objekte enthalten.
- Sind gut geeignet für das Speichern großer Datenmengen, da der indizierte Zugriff sehr schnell ist.
- der Index ist eindeutig
- Iterieren:

```
d = {'a': 1, 'b':1.2, 'c':1j}
for key, val in d.iteritems():
    print key, val
```

```
a 1
c 1j
b 1.2
```

# Vektoren und Matrizen - NumPy arrays

## Vektoren

```
np.array([1,2,4])
```

## Matrizen

```
np.array([[1,2,4],[2,3,4]])
```

Viele Befehle ähnlich wie bei Matlab:

- `zeros(n,m)`(Matlab) `zeros((n,m))`(Python)  
( $n \times m$ )- Matrix mit 0 als Einträge.
- `ones(n,m)`(Matlab) `ones((n,m))`(Python)  
( $n \times m$ )- Matrix mit 1 als Einträge.
- `repmat(A,n,m)`(Matlab) `tile(A,(n,m))`(Python)  
Blockmatrix mit ( $n \times m$ ) aus A bestehenden Blöcken zusammenhängen

# Slicing

```
A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
```

A =

1	2	3
4	5	6
7	8	9

Abfragen eines Eintrags

```
>>> A[1, 0]  
4
```

Abfrage einer Zeile

```
>>> A[:, :]  
4      5      6
```

Slicing-Operator: start :ende:step

Abfrage von Blöcken

```
>>> A[1:3, 0:2]  
4      5  
7      8
```

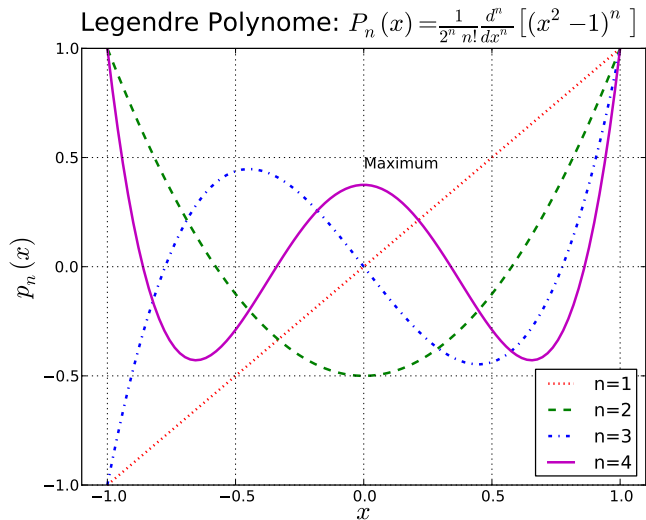
Abfrage mehrerer Zeilen

```
>>> A[(0, 2), :]  
1      2      3  
7      8      9
```

# Beispiel: Legendre Polynome

```
import numpy as np # NumPy
import scipy as sp # SciPy
import matplotlib as mpl # Matplotlib (2D/3D)
import matplotlib.pyplot as plt # Matplotlib's pyplot
from pylab import * # Matplotlib's pylab
x = linspace(-1,1,100)
p1 = x
p2 = (3./2)*x**2-1./2
p3 = (5./2)*x**3-(3./2)*x
p4 = (35./8)*x**4 -(15./4)*x**2 + 3./8
plot(x,p1,'r:',x,p2,'g--',x,p3,'b-.',x,p4,'m-',linewidth
     =2)
title('Legendre Polynome:  $P_n(x) = \frac{1}{2^{nn!}} \frac{d^n}{dx^n} \left[ (x^2-1)^n \right]$ ', fontsize= 15)
xlabel( '$x$', fontsize= 20)
ylabel( '$p_n(x)$', fontsize=20)
text( 0, 0.45 , 'Maximum' )
legend (('n=1', 'n=2', 'n=3', 'n=4'), loc='lower right')
grid('on'), box('on'), xlim( (-1.1 , 1.1) )
```

# Legendre Polynome



# loadtxt - gnuplot vergessen

```
array = np.loadtxt(fname, delimiter=None, comments='#')
```

- fname: Dateiname.
- delimiter: Trennzeichen. Z.B. ',' bei kommaseparierten Tabellen. Default-Einstellung sind Leerzeichen.
- comments: Kommentarzeichen. In Python-Dateien z.B. '#'
- array: Rückgabewert als (multidimensionaler) array.

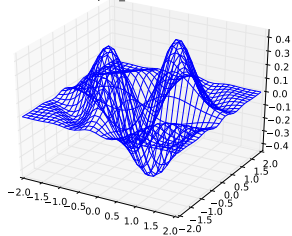
Flexibleres Einlesen: `np.genfromtxt()`



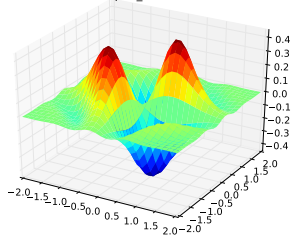
- 1 Einleitung
- 2 Grundlegende Bedienung Python (Spyder)
- 3 3D - Grafik**
- 4 Numerische Mathematik
- 5 Zusammenfassung

# 3D: Funktionenplot

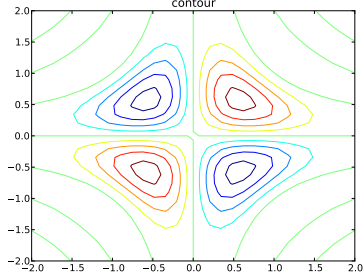
plot\_wireframe



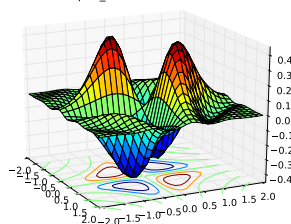
plot\_surf



contour



plot\_surface+contour



## 3D: Funktionenplot - Implementation

```
x = linspace(-2,2,30)
y = linspace(-2,2,30)
[X,Y] = meshgrid(x,y)
Z = exp(-X**2-Y**2)*sin(pi*X*Y)
fig=figure()
ax = fig.add_subplot(2, 2, 1, projection='3d')
ax.plot_wireframe(X,Y,Z),title('plot_wireframe')

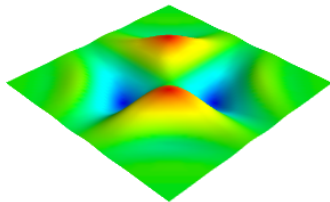
ax = fig.add_subplot(2, 2, 2, projection='3d')
ax.plot_surface(X,Y,Z,rstride=1,cstride=1,cmap=cm.jet,
               linewidth=0),title('plot_surface')

subplot(2, 2, 3)
contour(X,Y,Z,10), title('contour')

ax = fig.add_subplot(2, 2, 4, projection='3d')
ax.plot_surface(X,Y,Z,rstride=1,cstride=1,cmap=cm.jet)
ax.contour(X, Y, Z, zdir='z', offset=-0.5)
ax.view_init(20,-26),title('plot_surface + contour')
```

# Mayavi mlab!

```
from mayavi import mlab as ml #majavi mlab
ml.surf(X.T,Y.T,Z)
title('plot_surf (mayavi)')
```



## 4D: Mayavi mlab

### Slices

```
ml.pipeline.image_plane_widget(ml.pipeline.scalar_field(V
    ), plane_orientation=<'x_axes'|'y_axes'|'z_axes'>,
    slice_index=<idx>)
```

- $V$ : Funktionswerte  $V(i)$  zu  $(X(i), Y(i), Z(i))$ .
- `plane_orientation`: Schnitte durch x-/y-/z-Achse
- `slice_index`: Index in den Matrizen (keine direkte Koordinaten-Angabe)

### Volume rendering

```
ml.pipeline.volume(ml.pipeline.scalar_field(V))
```

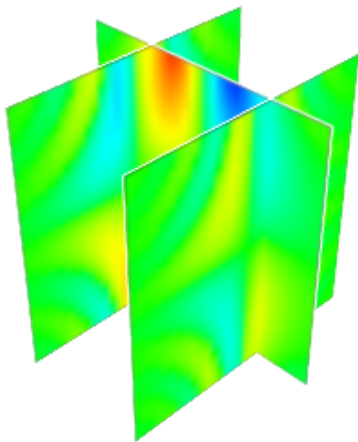
### isosurfaces

```
ml.contour3d(V)
```

### Beispiel:

```
X, Y, Z = np.ogrid[-2:2:20j, -2:2:20j, -2:2:20j]
V = exp(-X**2-Y**2) * sin(pi*X*Y*Z)
```

## 4D: Beispiel-slice



- 1 Einleitung
- 2 Grundlegende Bedienung Python (Spyder)
- 3 3D - Grafik
- 4 Numerische Mathematik**
- 5 Zusammenfassung

Seien  $A \in \mathbb{C}^{n \times n}$  und  $b \in \mathbb{C}^n$ . Das lineare Gleichungssystem

$$Ax = b$$

wird gelöst durch  $x = A \setminus b$  | `solve(A,b)`.

Iterative Verfahren:

- `gmres` (generalized minimum residual)
- `pcg` | `cg` (preconditioned conjugate gradient)
- `bicgstab` (biconjugate gradients stabilized)
- ...



## Aufgabe

Es sei  $A$  eine  $20.000 \times 20.000$  Matrix, deren Einträge Null sind bis auf die Primzahlen  $2, 3, 5, 7, \dots, 224737$  auf der Diagonalen und der Ziffer 1 in allen Einträgen  $a_{ij}$  mit  $|i - j| = 1, 2, 4, 8, \dots, 16384$ .

Was ist der  $(1, 1)$  Eintrag von  $A^{-1}$ ?

# Wärmeleitungsgleichung

Gegeben sei ein rechteckiges Gebiet  $\Omega \subset \mathbb{R}^2$  und eine zeitabhängige Funktion  $u(x, t)$ ,  $x \in \Omega$ ,  $t \in \mathbb{R}^+$ . Dann ist die Wärmeleitungsgleichung gegeben durch

$$\frac{\partial u}{\partial t} - \alpha \Delta u = 0 \text{ in } \Omega$$

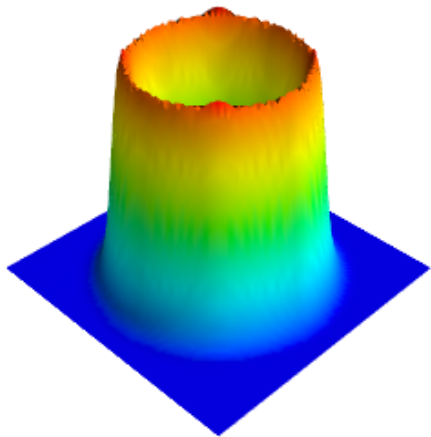
mit einer Konstante  $\alpha \in \mathbb{R}$ . Gegeben seien Dirichlet Randbedingungen

$$u = R, \text{ auf } \partial\Omega$$

mit einer Funktion  $R : \partial\Omega \mapsto C(\partial\Omega)$ . Zum Zeitpunkt  $t = 0$  sei

$$u(x, 0) = f(x), \forall x \in \Omega.$$

mit einer beliebigen, aber fest gewählten initialen Funktion  $f : \mathbb{R} \mapsto \mathbb{R}$ .



# Gewöhnliche Differentialgleichungen

```
r = scipy.integrate.ode(f[, jac])
```

- $f$ : Rechte Seite der DGL:  $y'(t) = f(t, y)$
- $jac$ : (optionale) Jacobi-Matrix
- `r.set_integrator (<name>[, <params>])`: Setzt den zu nutzenden Löser `<name>` mit den Parametern `<params>`.
- `r.set_initial_value (y[, t])`: Setzt den Anfangswert.
- `r.integrate (t)`: Findet  $y(t)$  und setzt den neuen Anfangswert im Löser.
- `r.successful ()`: Wahrheitswert über Erfolg der Integration.

# Die Lorenz-Gleichungen

- Chaostheorie / Schmetterlingseffekt.

$$\frac{d}{dt}y_1(t) = 10(y_2(t) - y_1(t))$$

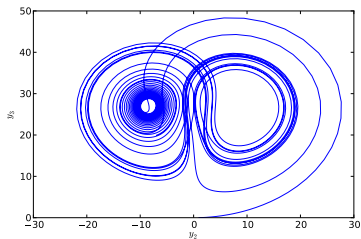
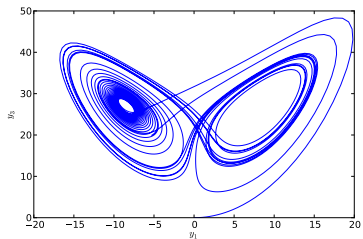
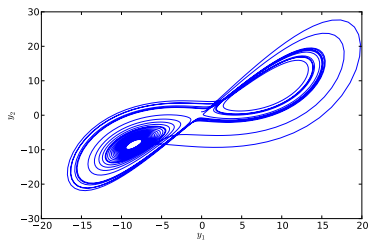
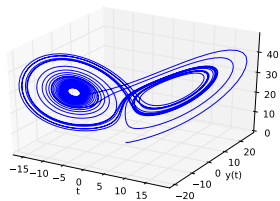
$$\frac{d}{dt}y_2(t) = 28y_1(t) - y_2(t) - y_1(t)y_3(t)$$

$$\frac{d}{dt}y_3(t) = y_1(t)y_2(t) - 8y_3(t)/3$$

# Die Lorenz-Gleichungen

```
def lorenz_rhs(t,y):
    return array([[10*(y[1]-y[0])], [28*y[0]-y[1]-y[0]*y
        [2]], [y[0]*y[1]-8*y[2]/3]])
y = array([0,1,0])
r = ode(lorenz_rhs)
r.set_initial_value(y, 0)
r.set_integrator('dopri5',atol=1e-7,rtol=1e-4)
tmax = 30,dt = 0.01,t=[]
while r.successful() and r.t < tmax:
    r.integrate(r.t+dt)
    t.append(r.t)
    y = vstack( (y, r.y) )
fig = figure(figsize=(16,10))
ax = fig.add_subplot(2, 2, 1, projection='3d')
ax.plot(y[:,0],y[:,1],y[:,2]),xlabel('t'), ylabel('y(t)')
subplot(2,2,2),plot(y[:,0],y[:,1]), xlabel('y_1')
subplot(2,2,3),plot(y[:,0],y[:,2]), xlabel('y_1')
subplot(2,2,4),plot(y[:,1],y[:,2]), xlabel('y_2')
```

# Die Lorenz-Gleichungen



# Nichtlineare Löser

```
from scipy import optimize
x0 = -5 # Startwert
f = lambda x: 2*x - exp(-x)
res,info,i,mesg = optimize.fsolve(f,x0,xtol=1e-5,
    full_output=True)
print ("res: {} \nnfev: {} \nfvec: {}".format(res,info['
    nfev'],info['fvec'])) )
```

```
res: [ 0.35173371]
```

```
nfev: 13
```

```
fvec: [ -1.50035540e-12]
```



# Ableitungsfrei / Nebenbedingungen

Finde Minimum ...

ohne Nebenbedingungen, multidimensional (Nelder-Mead-Simplex):

```
fmin(func, x0, args=(), xtol=0.0001, ftol=0.0001, maxiter  
=None, maxfun=None, full_output=0, disp=1, retall=0,  
callback=None)
```

- func: Function handle
- x0: Start-wert/-vektor
- xtol, ftol: Abbruch-Toleranz in  $x$  und  $func$ .

mit Nebenbedingungen, multidimensional:

```
fminbound(func, x1, x2, args=(), xtol=1e-05, maxfun=500,  
full_output=0, disp=1)
```

- Bezier
- Splines
- Mandelbrot
- Debugger
- profiler

- 1 Einleitung
- 2 Grundlegende Bedienung Python (Spyder)
- 3 3D - Grafik
- 4 Numerische Mathematik
- 5 Zusammenfassung**

## Python ist...

- Flexibel
- Mächtig
- Klare Codestruktur
- Schnell (NumPy, Cython)
- fast alles worin Matlab stark ist, hat eine (gute) Entsprechung in Python
- Frei und Open Source!
- Symbolisches Rechnen mit SymPy
- Basis von Sage (<http://sagemath.org>)

*Haupt-Nachteil:* Mit Flexibilität kommt Komplexität (Lernkurve, Modul-handling)

-  NumPy, SciPy SciPy developers (<http://scipy.org/>),
-  SciPy-lectures, F. Perez, E. Guillard, G. Varoquaux, V. Haenel (<http://scipy-lectures.github.io>),
-  Matplotlib (<http://matplotlib.org>)
-  scitools (<https://code.google.com/p/scitools/>)
-  mayavi (<http://docs.enthought.com/mayavi/mayavi/mlab.html>)