Vortrag_Folien

June 7, 2017

```
In [1]: from sympy import *
        import numpy as np
        import scipy as sp
        import matplotlib.pyplot as plt
        from mpl_toolkits.mplot3d import Axes3D
        from IPython.display import display

        # Oft benutze Sympy-Variablen
        x, y, z = symbols("x,y,z", real=True)
        k, m, n = symbols("k,m,n", integer=True)

        # Funktionen und Variablen für Vektorgeometrie
        s,t = symbols("s,t", real=True)
        Vec = lambda *args: Matrix([[x] for x in args])

        # LaTeX-Formeln anzeigen
        init_printing()

        # Plots direkt im Dokument anzeigen
        # (Nur für Jupyter-Notebook, nicht für iPython)
        %matplotlib inline
        %config InlineBackend.figure_format = 'svg'
```

1 Mathematik mit SciPy und Python

1.1 Beni Keller

Kantonsschule Zug

2 Inhalt

- Warum Python?
- Umgebung für interaktives Python
- Überblick über die Pakete in `scipy`
- Beispiele aus dem Unterricht
- Wie weiter?

3 Inhalt

- Warum Python?
- Interaktives Python
  - Installation
  - Tools
  - Anwendungsbereiche im Unterricht.
- Kurzer Überblick über die Pakete
  - Computeralgebra mit `sympy`
  - Numerisches Rechnen mit `numpy` und `scipy`
  - Graphische Darstellungen `matplotlib`
- Beispiel-Aufgaben in Jupyter-Notebook
  - Vektorgeometrie
  - Analysis
  - Stochastik (?) / Regression
  - Lineare Algebra
- Wie weiter? Materialien und Quellen

4 Warum Python?

Python

```
In [2]: print("Hello World")
```

```
Hello World
```

```
In [3]: class Number:
            def __init__(self, value):
```

```
            self.value = value

        # Klärender Kommentar
        def sum(self):
            if self.value <= 0:
                return 0
            else:
                s = 0
                for i in range(self.value):
                    s += i
                return s

    fifty = Number(50)
    fifty.sum()
```

Out[3]:

$$1225$$

```
In [4]: zeile = [0,0]
        matrix = [zeile, zeile]
        matrix
```

Out[4]:

$$[[0,\ \ 0],\ \ [0,\ \ 0]]$$

```
In [5]: matrix[0][0] = 1
        matrix
```

Out[5]:

$$[[1,\ \ 0],\ \ [1,\ \ 0]]$$

5 Umgebung für interaktives Python

5.1 Installation von Python: Anaconda

- https://continuum.io/downloads

5.2 Interaktive Tools

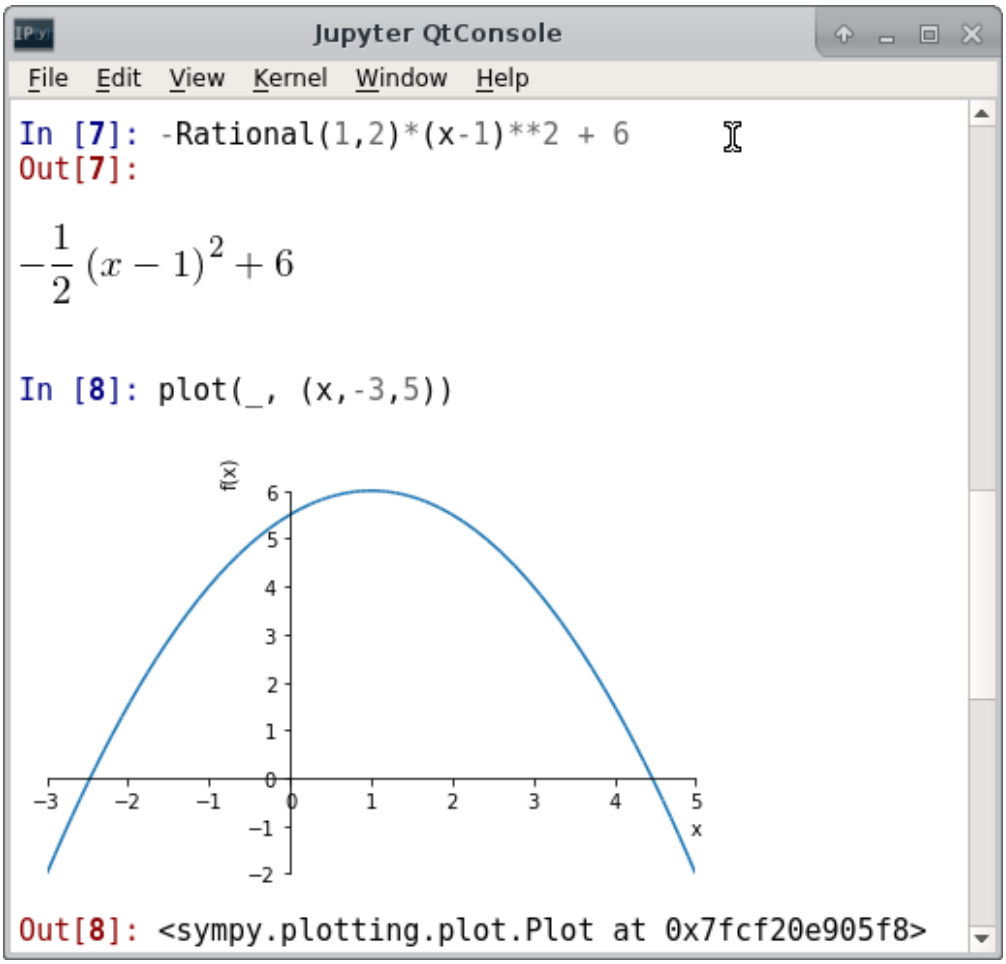
5.2.1 REPL: `qtconsole`

5.2.2 Im Web-Browser: `notebook`

- Interaktive Zellen-Basierte Oberfläche im Browser
  - Gleicher Ursprung wie die *Sage*-Oberfläche
  - Ähnliche Funktionsweise wie *wxmaxima*

Anaconda

Jupyter


Jupyter QtConsole
File Edit View Kernel Window Help
In [7]: -Rational(1,2)*(x-1)**2 + 6
Out[7]:
$-\frac{1}{2}(x-1)^2+6$
In [8]: plot(_, (x,-3,5))
f(x)
x
Out[8]: <sympy.plotting.plot.Plot at 0x7fcf20e905f8>

qtconsole

- Unterstützt LaTeX-Formeln
- HTML und Markdown Formatierung von Text
- Dateiformat geeignet für Arbeitsblätter
- https://try.jupyter.org/

5.2.3 Konvertieren von Notebooks: `nbconvert`

- Konvertieren nach *HTML*, *PDF*, *LaTeX*, *Markdown*, ...
- Beispiel: Diese Präsentation...

6 Überblick über die Pakete in `scipy`

SciPy

6.1 Computer-Algebra mit `sympy`

- Term-Manipulation mit Python Syntax

```
In [6]: from sympy import *
        x = symbols('x', real=True)

        solve(x**2 + 4*x - 5, x)
```

Out[6]:

$$[-5, \quad 1]$$

6.1.1 Zahlen in sympy

```
In [7]: 42              # Beliebig grosse Integer in Python
```

Out[7]:

$$42$$

```
In [8]: (1/42)**(1/2)  # Eingeschränkte 64-bit Fliesskommazahlen
```

Out[8]:

$$0.1543033499620919$$

```
In [9]: Rational(1,42)**Rational(0.5)
```

Out[9]:

$$\frac{\sqrt{42}}{42}$$

```
In [10]: S('(1/42)^(1/2)') # sympyify...
```

Out[10]:

$$\frac{\sqrt{42}}{42}$$

6.1.2 Terme und Gleichungen

```
In [11]: (x**3 + 4*x**2 + sin(x)**2*x + cos(x)**2*x) / x - 1
```

Out[11]:

$$-1 + \frac{1}{x}\left(x^3 + 4x^2 + x\sin^2(x) + x\cos^2(x)\right)$$

```
In [12]: expr = _.simplify()
         expr
```

Out[12]:

$$x\,(x+4)$$

```
In [13]: expr.expand()
```

Out[13]:

$$x^2 + 4x$$

```
In [14]: expr
```

Out[14]:

$$x\left(x+4\right)$$

```
In [15]: expr.subs(x, 4)
```

Out[15]:

$$32$$

```
In [16]: Eq(expr, 5)
```

Out[16]:

$$x\left(x+4\right)=5$$

```
In [17]: solve(Eq(expr, 5), x)
```

Out[17]:

$$\left[-5,\quad 1\right]$$

6.1.3 Integral- und Differenzialrechnung

```
In [18]: f = x**3 + 2*x**2 + exp(2*x) - sin(x)
         f
```

Out[18]:

$$x^3+2x^2+e^{2x}-\sin\left(x\right)$$

```
In [19]: f.diff(x)
```

Out[19]:

$$3x^2+4x+2e^{2x}-\cos\left(x\right)$$

```
In [20]: f.integrate(x)
```

Out[20]:

$$\frac{x^4}{4}+\frac{2x^3}{3}+\frac{e^{2x}}{2}+\cos\left(x\right)$$

```
In [21]: f.integrate((x,-1, 3))
```

Out[21]:

$$\cos\left(3\right)-\cos\left(1\right)-\frac{1}{2e^2}+\frac{116}{3}+\frac{e^6}{2}$$

```
In [22]: N(_)
```

Out[22]:

$$238.783100968947$$

```
In [23]: f = symbols('f', cls=Function)
         f(x).integrate((x,3,5))
```

Out[23]:

$$\int_3^5 f(x)\,dx$$

6.2 Numerisches Rechnen

6.2.1 `numpy`

- Freie MatLab-Alternative für numerisches Matrizenrechnen

6.2.2 `scipy`

- Erweiterung mit diversen Algorithmen
  - Optimierung
  - Statistik
  - Wahrscheinlichkeitsrechnen

6.2.3 `numpy` Arrays

```
In [24]: import numpy as np
         data = np.array([[1, 2, 3, 4], [2, 3, -2, 3], [-4, 3, 2, -1]])
         data
```

```
Out[24]: array([[ 1,  2,  3,  4],
                [ 2,  3, -2,  3],
                [-4,  3,  2, -1]])
```

```
In [25]: np.arange(1, 10)
```

```
Out[25]: array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In [26]: np.linspace(0.5, 6.5, 9) # Bereiche mit nicht-ganzzahligen Schritten
```

```
Out[26]: array([ 0.5 ,  1.25,  2.  ,  2.75,  3.5 ,  4.25,  5.  ,  5.75,  6.5 ])
```

```
In [27]: # Rechnen mit Arrays
         data / 4 + 1
```

```
Out[27]: array([[ 1.25,  1.5 ,  1.75,  2.  ],
                [ 1.5 ,  1.75,  0.5 ,  1.75],
                [ 0.  ,  1.75,  1.5 ,  0.75]])
```

```
In [28]: data + np.array([5, 4, 3, 5])

Out[28]: array([[6, 6, 6, 9],
                [7, 7, 1, 8],
                [1, 7, 5, 4]])

In [58]: np.abs(data)

Out[58]: array([[1, 2, 3, 4],
                [2, 3, 2, 3],
                [4, 3, 2, 1]])
```

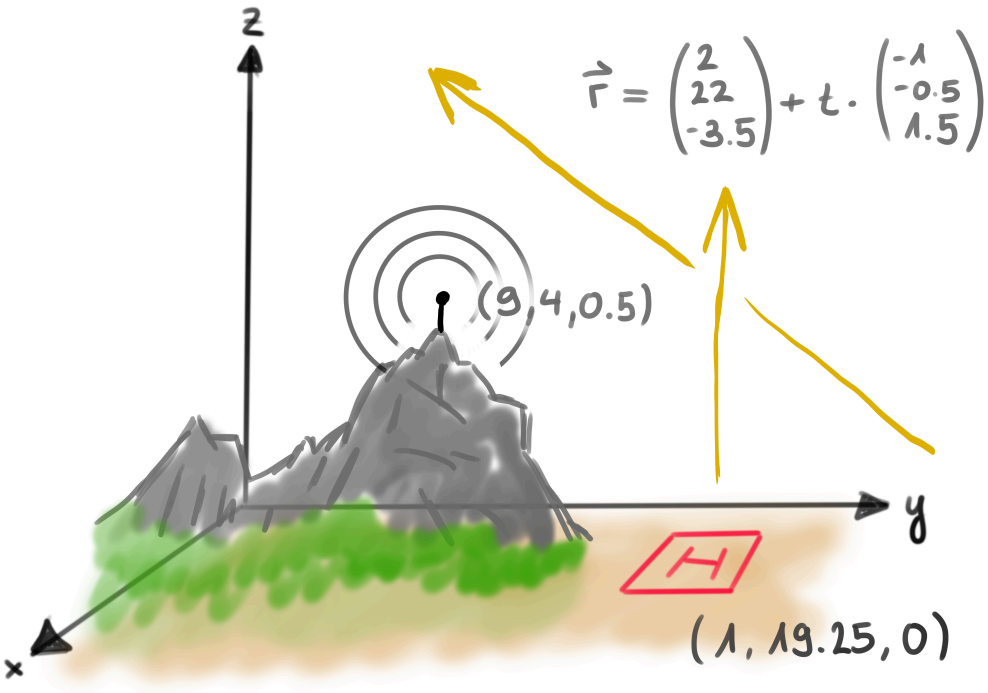
6.2.4 Graphische Darstellungen mit `matplotlib`

- 2D und 3D Darstellungen von Daten
  - Punkt- und Liniengraphen
  - Balkendiagramm
  - Animationen
  - Interaktive Plots
- http://matplotlib.org/examples/
  - `simple_3danim.py`
  - `slider_demo.py`

6.3 Initialisieren eines Notebooks

```
from sympy import *
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from IPython.display import display

# Oft benutze Sympy-Variablen
x, y, z = symbols("x,y,z", real=True)
k, m, n = symbols("k,m,n", integer=True)

# Funktionen und Variablen für Vektorgeometrie
s,t = symbols("s,t", real=True)
Vec = lambda *args: Matrix([[x] for x in args])

# LaTeX-Formeln anzeigen
init_printing()

# Plots direkt im Dokument anzeigen
# (Nur für Jupyter-Notebook, nicht für iPython)
%matplotlib inline
%config InlineBackend.figure_format = 'svg'
```

7 Vektorgeometrie: Flugbahnen



Heli

In [30]:
```
# P_1 = Matrix([[1], [19.25], [0]])
P_1 = Vec(1, 19.25, 0)
P_2 = Vec(2, 22, -3.5)
r_1 = Vec(0, 0, 1)
r_2 = Vec(-1, -0.5, 1.5)

Eq(Vec(x,y,z), P_2 + s*r_2)
```

Out[30]:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} -s + 2 \\ -0.5s + 22 \\ 1.5s - 3.5 \end{bmatrix}$$

In [31]:
```
n = r_1.cross(r_2)
n
```

Out[31]:

$$\begin{bmatrix} 0.5 \\ -1 \\ 0 \end{bmatrix}$$

In [32]:
```
Eq(n.dot(Vec(x,y,z)), n.dot(P_1))
```

Out[32]:

$$0.5x - y = -18.75$$

In [33]:
```
(n.dot(P_2) - n.dot(P_1))/n.norm()
```

Out[33]:

$$-2.01246117974981$$

In [34]:
```
abs(_)
```

Out[34]:

$$2.01246117974981$$

7.1 Numerischer Ansatz

- Algorithmisch das Minimum in zwei Variablen finden
  - Default: BFGS-Verfahren

In [35]:
```
g_1 = P_1 + s*r_1
g_2 = P_2 + t*r_2
d = (g_2 - g_1).norm()
d
```

Out[35]:

$$\sqrt{(0.5t - 2.75)^2 + (t - 1)^2 + (s - 1.5t + 3.5)^2}$$

In [36]:
```
from scipy.optimize import minimize

def dist(vals):
    return d.subs(s, vals[0]).subs(t, vals[1])

minimize(dist, (0,0))
```

Out[36]:
```
      fun: 2.012461179750537
 hess_inv: array([[ 5.49142869,  2.31123865],
       [ 2.31123865,  1.53496148]])
      jac: array([ -8.94069672e-08,   1.07288361e-06])
  message: 'Optimization terminated successfully.'
     nfev: 48
      nit: 10
     njev: 12
   status: 0
  success: True
        x: array([-0.64999788,  1.90000152])
```

```
In [37]: r = g_1.subs(s, _["x"][0]) - g_2.subs(t, _["x"][1])
         r.norm()
```

Out[37]:

$$2.01246117975054$$

8 Analysis: Optimale Geschwindigkeit

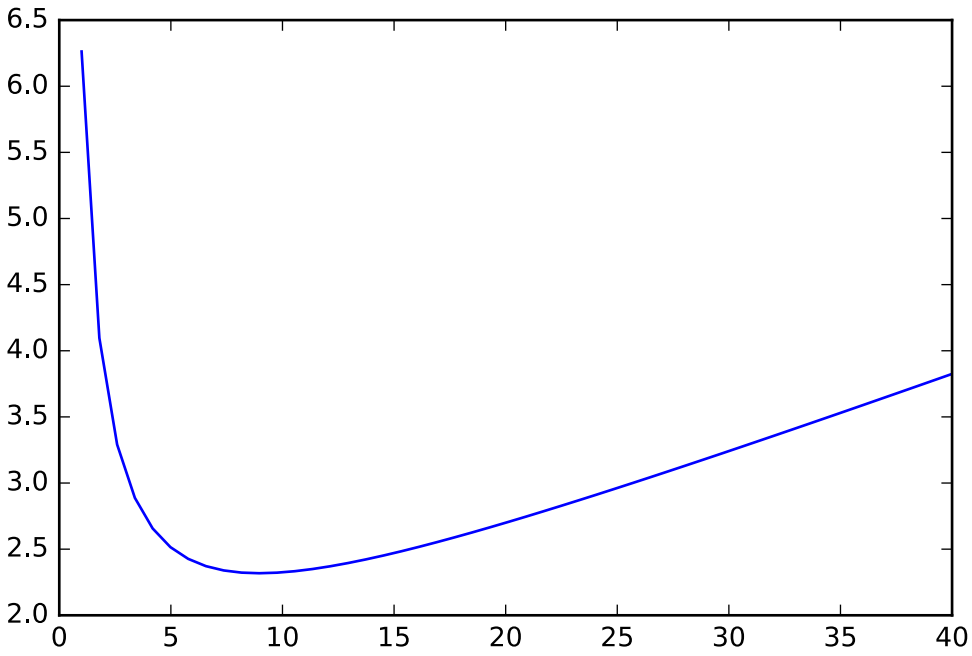
Bei welcher Geschwindigkeit passen am meisten Autos durch eine Strasse?

- Reaktionszeit
$$t_r = 1.2\,\mathrm{s}$$
- Bremsverzögerung
$$a = 8\,\frac{\mathrm{m}}{\mathrm{s}^2}$$
- Länge der Fahrzeuge
$$l = 5\,\mathrm{m}$$
- Bremsweg
$$s_b = \frac{v^2}{2a}$$

```
In [38]: def bremsweg(v, a):
             return v**2 / (2*a)
```

```
In [39]: # Zeit pro Auto:
         def fahrzeug_zeit(v, a = 8, t_r = 1.2, l = 5):
             platz_pro_auto = l + t_r*v + bremsweg(v, a)
             return platz_pro_auto / v

         v = np.linspace(1, 40)
         plt.plot(v, fahrzeug_zeit(v))
```

```
Out[39]: [<matplotlib.lines.Line2D at 0x7f1ef23e4320>]
```



In [40]:
```
# Numerische Lösung
from scipy.optimize import minimize
result = minimize(fahrzeug_zeit, 5)
if result["success"]:
    opt_speed = result["x"][0]

# In Kilometer pro Stunde
opt_speed * 3.6
```

Out[40]:

$$32.1993271375$$

In [41]:
```
# Algebraische Lösung mit Sympy
v, a, t_r, l = symbols("v, a, t_r, l", real=True)
fahrzeug_zeit(v, a, t_r, l)
```

Out[41]:

$$\frac{1}{v}\left(l + t_r v + \frac{v^2}{2a}\right)$$

In [42]:
```
ziel_func = _.simplify()
ziel_func
```

Out[42]:

$$\frac{l}{v} + t_r + \frac{v}{2a}$$

In [43]:
```
ziel_func.diff(v)
```

Out[43]:

$$-\frac{l}{v^2} + \frac{1}{2a}$$

In [44]:
```
solve(_, v)
```

Out[44]:

$$\left[-\sqrt{2}\sqrt{al},\quad \sqrt{2}\sqrt{al}\right]$$

In [45]:
```
_[1]
```

Out[45]:

$$\sqrt{2}\sqrt{al}$$

In [46]:
```
_.subs(a,8).subs(l,5)
```

Out[46]:

$$4\sqrt{5}$$

In [47]:
```
_.n() * 3.6
```

Out[47]:

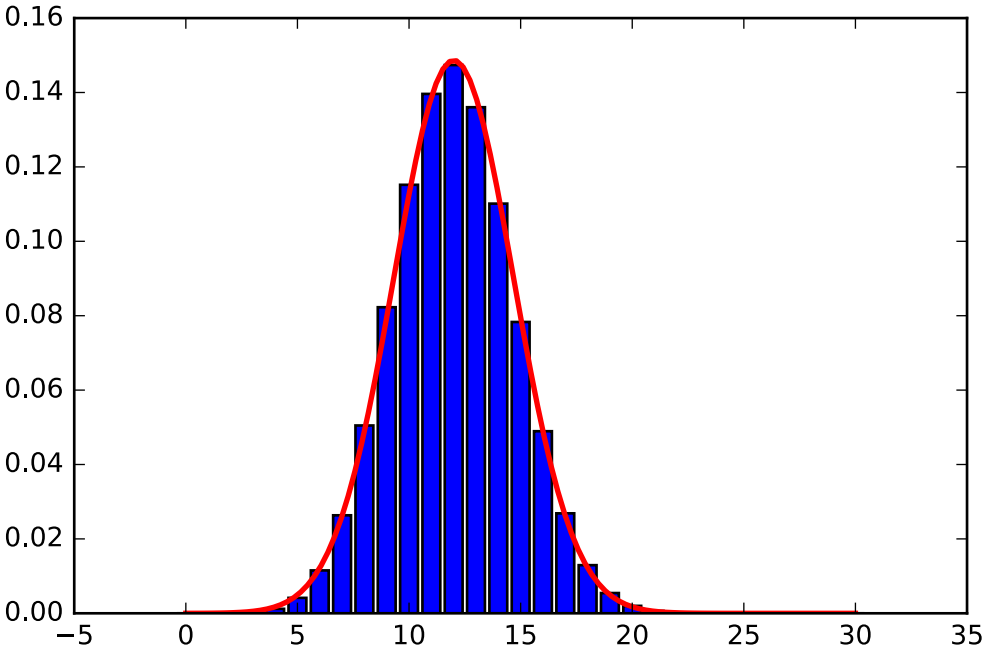
$$32.199378875997$$

9 Stochastik: Binomial und Normalverteilung

In [48]:
```
import scipy.stats as st
n = 30
p = 0.4
x = np.arange(0, n + 1)
xn = np.linspace(0,n,100)
```

In [49]:
```
y = st.binom(n, p).pmf(x)
yn = st.norm.pdf(xn, loc=n*p, scale=(n*p*(1-p))**0.5)
plt.bar(x, y, align="center", width=0.8)
plt.plot(xn, yn, color="red", linewidth=2)
```

Out[49]:
```
[<matplotlib.lines.Line2D at 0x7f1ef1c77208>]
```



10 Schwerpunktfach: Regressionskurven

In einer Klasse wurden bei einer Semesterprüfung die Vorbereitungszeiten und die erreichten Punktzahlen der einzelnen Schülerinnen und Schüler festgehalten (Punktemaximum 100).

Zeit	3.0	4.2	2.0	9.3	5.2	5.5	0.5	6.2	7.6	1.0	1.0	3.5	8.4	1.4	3.5
Punkte	82	90	53	89	74	78	40	87	87	47	60	88	100	48	87

Zeit	4.9	7.0	2.6	0.5	8.2	5.1	1.5	3.7	6.5	3.0	2.5	2.4	5.4	2.3	5.4
Punkte	93	96	75	56	94	86	73	85	85	86	71	82	70	61	92

1. Stelle die Daten graphisch dar.
2. Bestimme die Gleichung der besten Regressionskurve und füge sie in den Plot ein.

Wir betrachten die folgenden Modelle:
Logarithmische x- und y-Achse:

$$f(x) = a \cdot x^b$$

Logarithmische x-Achse:

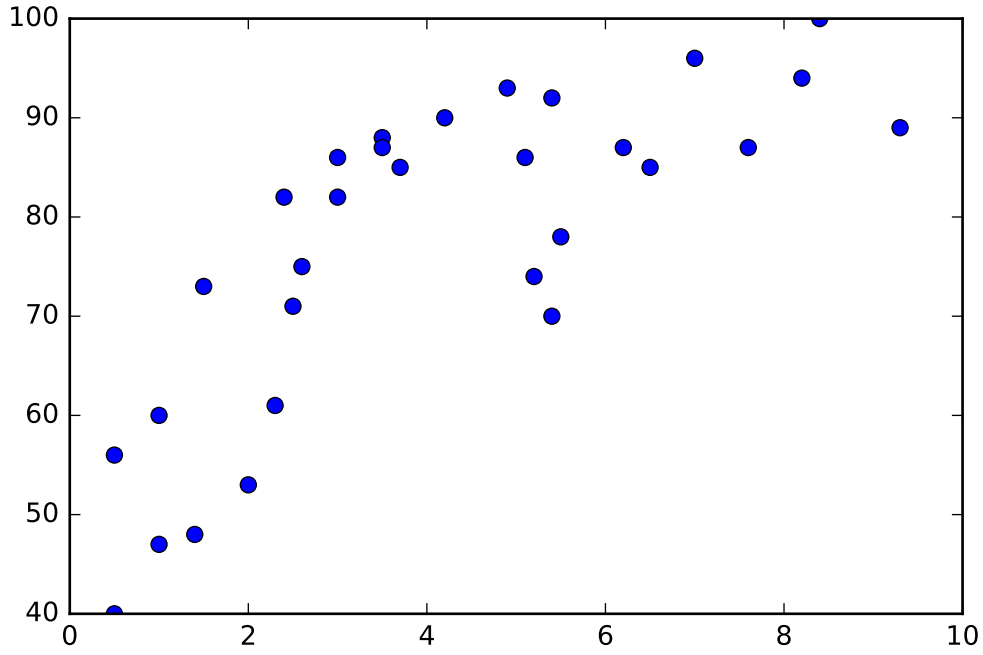
$$f(x) = a \cdot \ln(x) + b$$

Logarithmische y-Achse:

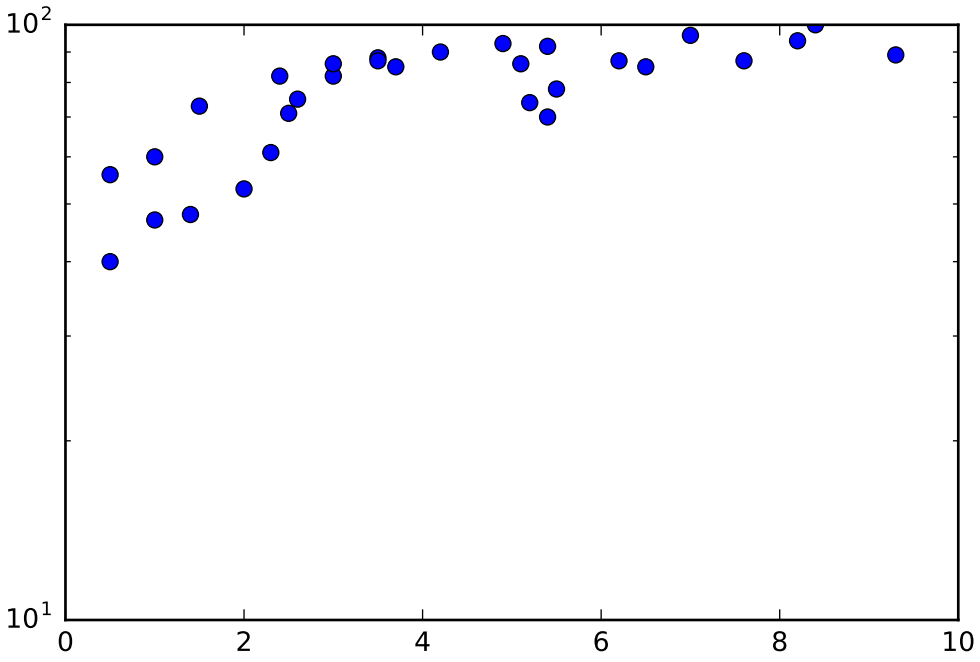
$$f(x) = a \cdot b^x$$

In [50]:
```
time = np.array([3.0,4.2,2.0,9.3,5.2,5.5,0.5,6.2,7.6,1.0,1.0,3.5,8.4,1.4,3.5,4.9,7.0,2.
points = np.array([82,90,53,89,74,78,40,87,87,47,60,88,100,48,87,93,96,75,56,94,86,73,8
plt.plot(time, points, 'o')
```

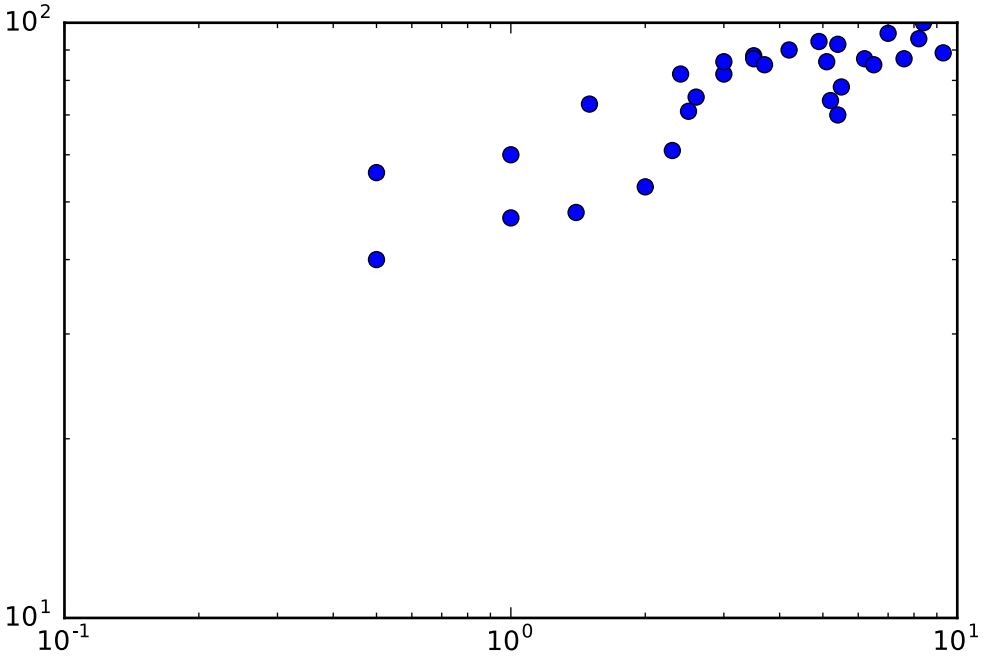
Out[50]:
```
[<matplotlib.lines.Line2D at 0x7f1ef1820518>]
```



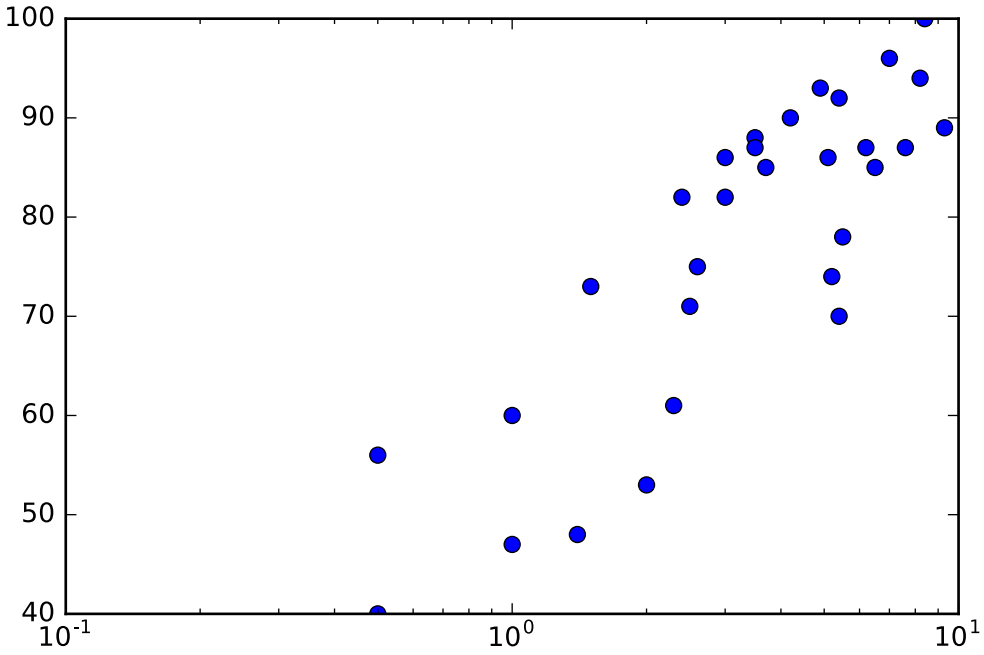
In [51]:
```
plt.plot(time, points, 'o')
plt.yscale('log')
plt.show()
```



In [52]:
```
plt.plot(time, points, 'o')
plt.yscale('log')
plt.xscale('log')
plt.show()
```



In [53]:
```
plt.plot(time, points, 'o')
plt.xscale('log')
plt.show()
```



In [54]:
```
from scipy.stats import linregress

# Exponentialfunktion
slope_e, intercept_e, r_e = linregress(time, np.log(points))[:3]

# Potenzfunktion
slope_p, intercept_p, r_p = linregress(np.log(time), np.log(points))[:3]

# Logarithmusfunktion
slope_l, intercept_l, r_l = linregress(np.log(time), points)[:3]

r_e, r_p, r_l
```

Out[54]:

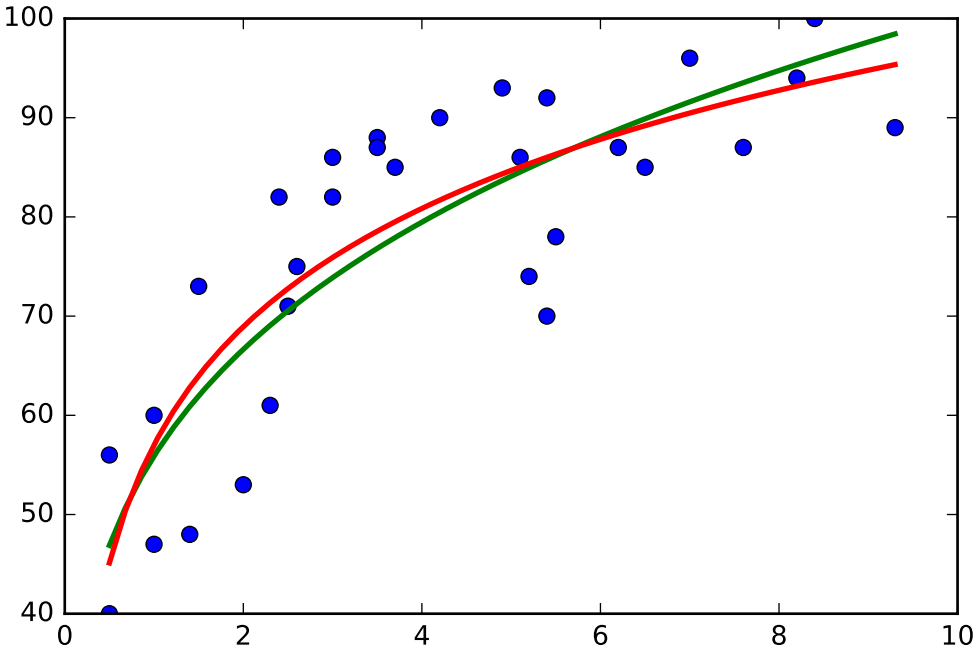
$$(0.741757806423, \quad 0.845077371888, \quad 0.84579004091)$$

In [55]:
```
def potenzfkt(x):
    return np.exp(intercept_p)*x**slope_p

def logfkt(x):
    return slope_l*np.log(x) + intercept_l
```

```
time_vals = np.linspace(min(time), max(time))
```

```
In [56]: plt.plot(time, points, 'o')
         plt.plot(time_vals, potenzfkt(time_vals), linewidth=2)
         plt.plot(time_vals, logfkt(time_vals), linewidth=2)
```

```
Out[56]: [<matplotlib.lines.Line2D at 0x7f1ef1637898>]
```



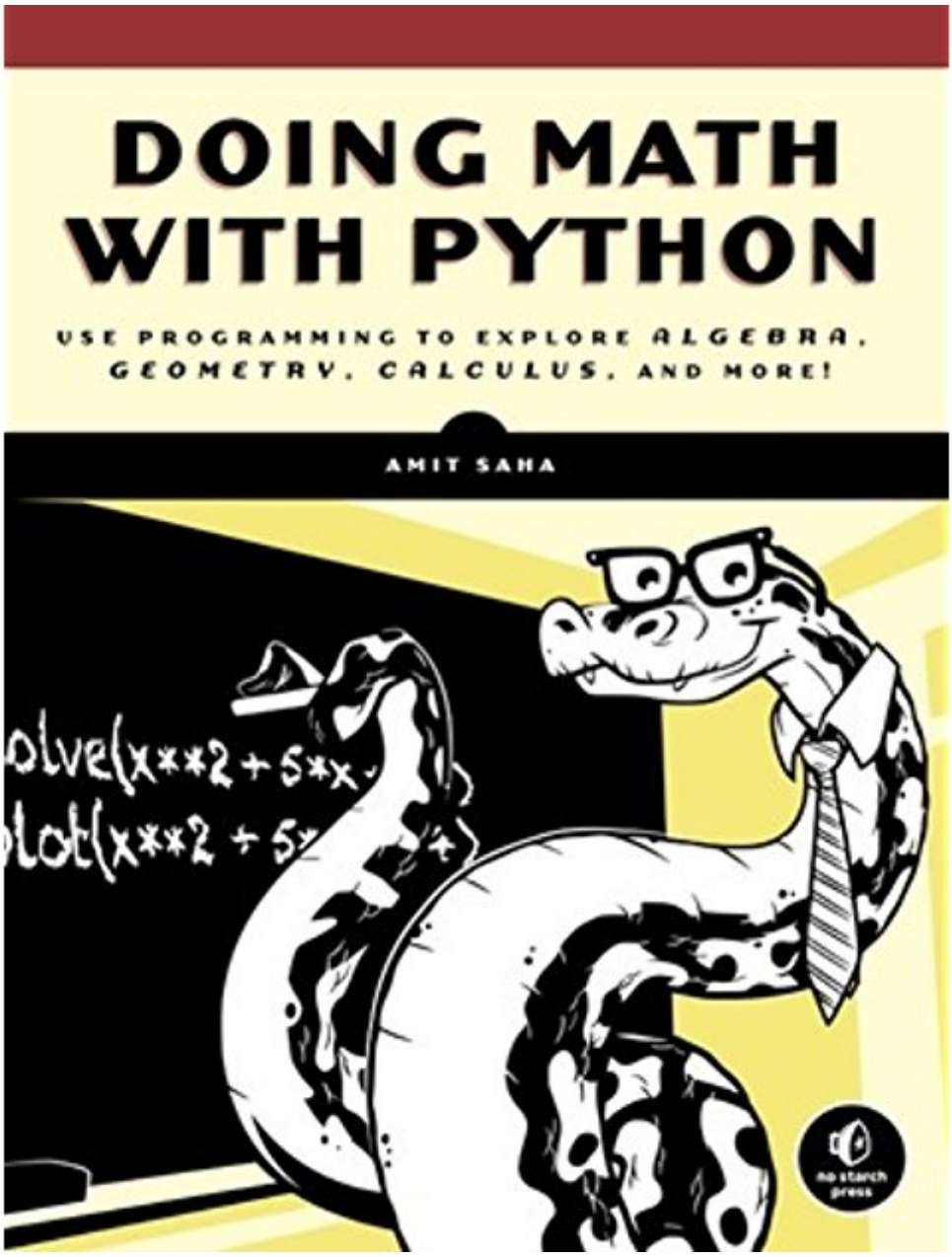
11 Wie weiter?

11.1 Quellen

- Anaconda: https://www.continuum.io/downloads
- Jupyter Tools: http://jupyter.org/
- Scipy: https://www.scipy.org/
- Sympy: http://www.sympy.org/
  - Sympy Live: http://live.sympy.org/
- Matplotlib: http://matplotlib.org/

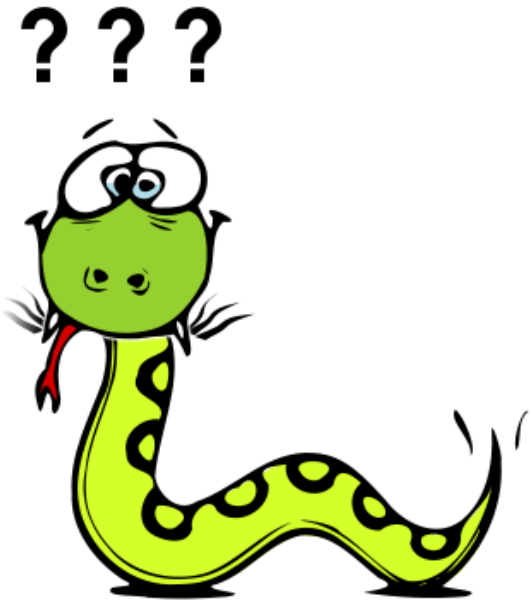
11.2 Lesen

- https://minireference.com/static/tutorials/sympy_tutorial.pdf
- https://www.nostarch.com/doingmathwithpython


DOING MATH
WITH PYTHON
USE PROGRAMMING TO EXPLORE ALGEBRA, GEOMETRY, CALCULUS, AND MORE!
AMIT SAHA
olve(x**2 + 5*x
lot(x**2 + 5x
no starch press

Doing Math

12 Fragen



Fragen