

# Python im Schuleinsatz

**Lehrerarbeitstag des Fachkreises Mathematik, Freitag, 6. Juli 2012**

Kantonsschule Zürcher Oberland

Martin Guggisberg

Departement Mathematik und Informatik  
High Performance and Web Computing Group

kontakt: [martin.guggisberg@unibas.ch](mailto:martin.guggisberg@unibas.ch)

## Überblick

### Teil 1 : Einstieg in die Programmierung mit Python

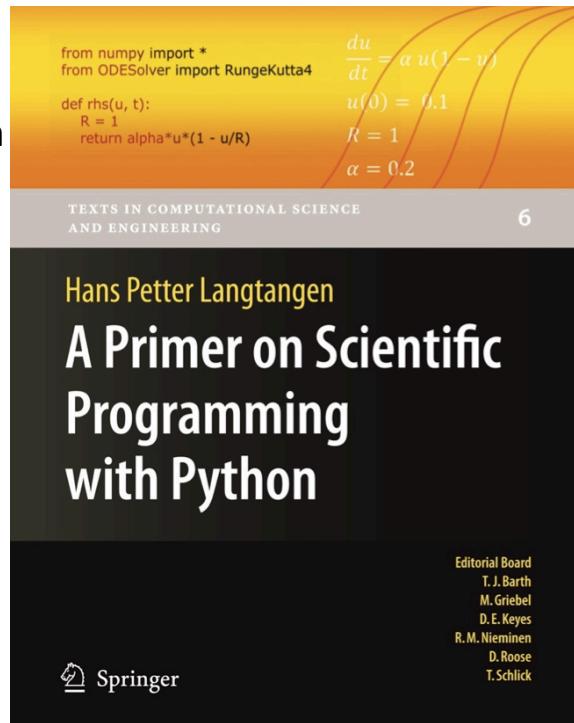
- Einführung in Python
- Programmieraufgaben mit Bezug zur Mathematik
- Laborteil – Lösen einzelner Aufgaben

### Teil 2 : Anwendungen

- Populationsdynamik
- Optimierung
- Rundreise (Traveling Salesman)
- Arbeiten mit aktuellen Datensätzen (Wetter)
- Laborteil – Experimentieren mit Python

## Einführung in Python

- Das Skript „Wissenschaftliches Rechnen mit Python“ soll einen schrittweisen Einstieg in die Programmierung mit Python ermöglichen.
- Es basiert auf den ersten Kapitel des Buchs „*A Primer on Scientific Programming with Python*“



## Unterlagen und Quellen

- Skript und alle Beispiele zum Workshop können von der Webseite <http://mgje.github.com/Python-Mathematik-Beispiele/> bezogen und frei verwendet werden.
- Falls Sie weitere Informationen, Ideen, Inhalte oder Quellen besitzen werde ich diese gerne integrieren.
- Nehmen Sie per eMail oder Twitter Kontakt auf.  
[Martin.Guggisberg@unibas.ch](mailto:Martin.Guggisberg@unibas.ch) oder @mgje

## Guido v. Rossum wollte ...

- eine neue Programmiersprache, die ..
  - einsteigerfreundlich und leicht zu lernen ist,
  - viele Möglichkeiten bietet ohne unübersichtlich zu werden,
  - mehr als ein Programmierparadigma unterstützt,
  - mit wenigen **Keywords** auskommt.

*Everyone a programmer!*

*Guido v. Rossum*

## Python ist ...

- eine moderne Sprache
  - objektorientiert
  - skalierbar
  - OS unabhängig
  - reich an Bibliotheken
  - erweiterbar
  - kein Muss – aber konsequent
  - von kleinen Skripten bis zu grossen Systemen
  - Win , \*nix , OSX, BeOS, usw.
  - Python Package Index od. Enthought
  - z.B. C/C++ Code

## Python Code ausführen

- Kommandozeile  
z.B auf Win, „Start“ ->  
„Ausführen“ -> cmd  
python myProg.py

```
root@nano-world:/home/guggisberg/backup - bash...
bash-3.2$ bash-3.2$ bash-3.2$ bash-3.2$ bash-3.2$ bash-3.2$ bash-3.2$ bash-3.2$ bash-3.2$ python scarve.py -n 22 landschaft.jpg
```

- IDLE – Python Shell

```
# sterne
for i in range(4):
    s = ''
    for j in range(3*i+1):
        s = s + '*'
    print s
```

- Interaktiver Modus  
    >>> <Anweisung>  
    >>> <Ergebnis>

```
$ python
>>> print 'Hello-World'
>>> Hello-World
```

## Grundlegende Sprachelemente

- Datentypen
- Operatoren
- Zeichenketten und deren Formatierung
- Tupel und Listen
- Schleifen
- Verzweigungen
- Funktionen

## Datentypen

Typ	False Wert
<code>NoneType</code>	<code>None</code>
<code>int, long</code>	<code>0</code>
<code>float</code>	<code>0.0</code>
<code>complex</code>	<code>0 + 0j</code>
<code>str</code>	<code>""</code>
<code>list</code>	<code>[]</code>
<code>tuple</code>	<code>()</code>
<code>dict</code>	<code>{}</code>

```
>>> type(1)
>>><type 'int'>

>>> type(1L)
>>><type 'long'>

>>> type(1.0)
>>><type 'float' >

>>> type(1)
>>><type 'complex' >
```

## Operationen

- Addition                `+`
- Subtraktion           `-`
- Division               `/`
- Integerdivision      `//`
- Multiplikation      `*`
- Exponentieren        `**`
- Modulo                `%`
  
- logisch (und)        `and`
- logisch (oder)       `or`
- logisch (nicht)      `not`

## Zeichenketten - Strings

```
s1 = 'Hallo Welt'
```

```
s2 = "Hallo Welt"
```

```
s3 = """ Hallo
Welt auf
mehr Zeilen """
```

Escape-Sequenz	Erklärung
\a	erzeugt Signalton
\b	Backspace
\f	Seitenvorschub
\n	Linefeed
\r	Carriage Return
\t	horizontal Tab
\v	vertikal Tab
\"	Escaping "
\'	Escaping '
\\	Escaping \

## Formatierung von Strings

### Syntax

```
'...%n...%m...' % (Wert1,Wert2)
```

### Beispiel

```
>>> '%10.2f' % 3.1415
      3.14
```

Format	Erklärung
d, i	Integer mit Vorzeichen
f	Float (Dezimaldarstellung)
g, G	Float (wiss. mit Exponent)
u	Integer ohne Vorzeichen
x	Hexzahl ohne Vorzeichen
o	Oktalzahl ohne Vorzeichen
e, E	Float (Exponentendarst.)
c	Zeichen (Länge 1)
s, r	String

## Tupel

### Syntax

(Wert\_1, ..., Wert\_n)

kann

- nicht verändert werden
- bel. Elemente

### Funktionen

- Index
- count

### Beispiel

```
>>> t = (1,2,3)
>>> z = ('a','z',1)
>>> t.index(2)
1
>>> z.index('a')
0
```

## Listen

### Syntax

[Wert\_1, ..., Wert\_n]

kann

- verändert werden
- bel. Elemente
- sortiert werden

### Funktionen

- index, count, append,
- insert, remove,
- reverse, sort

### Beispiel

```
>>> l = [1,2,3]
>>> z = ['a', 'z', 1]
>>> l.append(4)
>>> del l[0]
>>> print l
[2,3,4]
>>> l.reverse()
>>> print l
[4,3,2]
```

## Operationen zu Tupel und Listen

### Operation Erklärung

s in x	prüft, ob s in x ist
s not in x	prüft, ob s nicht in x ist
x+y	Verkettung von x und y
x*n	Verkettung, so das n Kopien von x existieren
x[n]	liefert das n-te Element von x
x[n:m]	liefert eine Teilsequenz von n bis m
x[n:m:k]	liefert eine Teilsequenz von n bis m, aber nur jedes k-te Element wird berücksichtigt
<b>len(x)</b>	liefert die Anzahl von Elementen
<b>min(x)</b>	liefert das kleinste Element
<b>max(n)</b>	liefert das größte Element

## if-elif-else - Verzweigung

### Syntax

```
if <Bedingung>:  
    ...  
elif <Bedingung>:  
    ...  
else:  
    ...
```

### Abkürzung

```
y = ( 1 if x == 'a' else 2)
```

### Beispiel

```
>>> x = 1  
>>> if x == 1:  
...     print ("x=1")  
x=1
```

## for - Schleife

Syntax

```
for <Variable> in <object>:
```

...

**range()** Funktion

```
range(start, stop, step)
```

```
>>> range(1,10,2)
```

```
[1,3,5,7,9]
```

Beispiel

```
>>> x = ['a', 'b', 'c']
```

```
>>> count = 0
```

```
>>> for a in x:
```

```
... count += 1
```

```
... print a
```

...

a

b

c

```
>>> print count
```

3

## while - Schleife

Syntax

```
while <Bedingung>:
```

...

**else:**

...

break Anweisung

```
while <Bedingung1>:
```

```
  if <Bedingung2>:
```

```
    break
```

Beispiel

```
>>> x = 4
```

```
>>> while x > 1:
```

```
... print x
```

```
... x = x - 1
```

```
... else:
```

```
... print 'x = 1'
```

4

3

2

x = 1

## Funktionen

Syntax

```
def <Name> (P1, ..., Pn):
    ...
    return <Resultat>
    ...
```

Optionale Parameter

```
def test (par='Hallo'):
    print param
```

Beispiel

```
>>> def printSum(a,b):
...     print(a+b)
>>> printSum(1,2)
3
```

```
>>> def mult(a,b,c=3)
```

```
...     return (a*b*c)
>>> mult(2,4,3)
24
>>> mult(2,4)
24
```

## Programmieraufgaben mit Bezug zur Mathematik

- Polarkoordinatendarstellung einer komplexen Zahl      Ü-1.10
- Wie lange kocht ein perfektes Ei?      Ü-1.7
- Kettenbruchnäherung für Pi      Ü-2.15
- Länge eines Pfades      Ü-3.4
- Numerisch Ableiten      Ü-3.6
  
- Dichte der Primzahlen      [Dichte\\_Primzahlen.py](#)
- Regression und Interpolation      [Regression\\_Interpolation.py](#)

## Grosse Zahlen

z.B. Falkultät 49

```
>>>import math  
>>>math.factorial(49)  
6082818640342675608722521633212953768875528313792102400000000000L  
>>> float(math.factorial(49))  
6.082818640342675e+62  
  
>>> format(math.pi, '.49g')  
'3.141592653589793115997963468544185161590576171875'
```

## Polarkoordinatendarstellung einer komplexen Zahl      Ü-1.10

```
>>> z1 = 1.0 + 1.0j  
>>> abs(z1)  
1.4142135623730951  
>>> from cmath import phase,pi  
>>> ph = phase(z1)  
>>> print ph/pi*180  
45.0
```

## Wie lange kocht ein perfektes Ei?

Ü-1.7

$$t = \frac{M^{2/3} c \rho^{1/3}}{K \pi^2 (4\pi/3)^{2/3}} \ln \left[ 0.76 \frac{T_0 - T_w}{T_y - T_w} \right]$$

```
from math import log,sqrt,exp,pi

M=67.0      # Masse
rho=1.038   # Dichte
capa=3.7    # spez. Wärmekapazität
K=5.4E-3    # Termische Leitfähigkeit

TW=100.0    # Temp. kochendes Wasser
T0=4.0      # Anfangstemperatur
Ty=70       # Endtemperatur

t = (M** (2.0/3.0) * capa * rho** (1.0/3.0)) / (K*pi**2*\n        (4*pi/3.0)** (2.0/3.0)) * log(0.76* (T0-TW) / (Ty-TW))
```

## Kettenbruchnäherung für Pi

Ü-2.15

```
from math import pi
```

```
N = 9
res = float(N**2)
for n in range(N-1,0,-1):
    res = n**2/res+2*n-1
```

$$\frac{4}{\pi} = 1 + \frac{1^2}{3 + \frac{2^2}{5 + \frac{3^2}{7 + \frac{4^2}{9 + \frac{5^2}{\vdots}}}}}$$

```
print 'Pi berechnet ist %1.6f' %(4/res)
print 'Pi aus Math Bibliothek ist %1.6f' % pi
```

## Länge eines Pfades

Ü-3.4

```
from math import sqrt

def pfadlaenge(P):
    L = 0.0
    for i in range(1, len(P)):
        L=L+sqrt((P[i][0]-P[i-1][0])**2+\n                  (P[i][1]-P[i-1][1])**2)
    return L

pfad = [[0,0], [1,0], [1,2], [0,2]]
print pfadlaenge(pfad)
```

## Numerisch Ableiten

Ü-3.6

```
from math import pi,exp,sqrt,sin,cos,tan,log

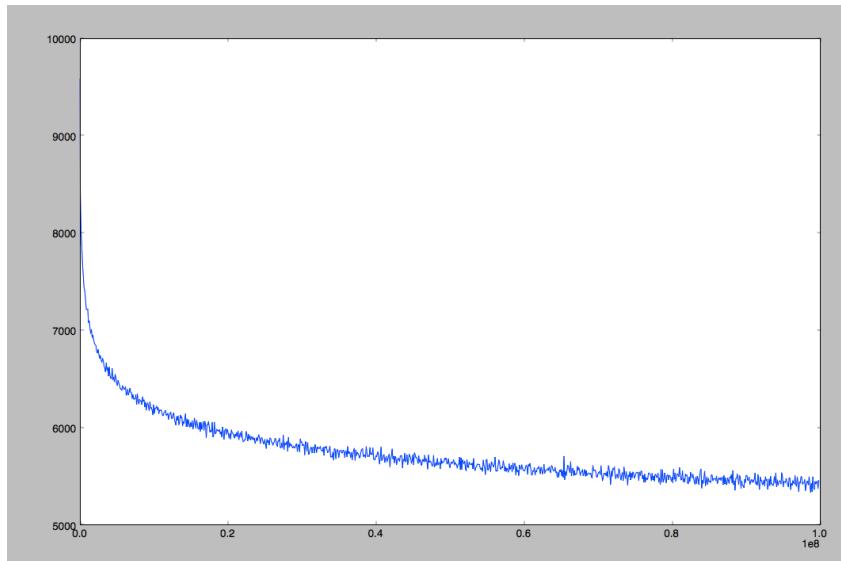
def ableitung(x,f,h=0.001):
    return (f(x+h)-f(x-h))/(2*h)

def f1(x):
    return exp(-2*x*x)

print "Die Ableitung von y=exp(-2x^2) an der \
Stelle %g beträgt %g" % (0,ableitung(0,f1))
```

## Dichte der Primzahlen

- Bereich  $0 – 10^8$
- Intervall 100'000



## Dichte\_Primzahlen.py

```
N = 100000000 # Obere Grenze
be = 100000 # Intervall / Bereich
gestrichen = [False]*(N+1)

# Sieb des Eratosthenes
for i in range(2,N+1):
    if not gestrichen[i]:
        for j in range(2*i,N+1,i):
            gestrichen[j] = True

gestrichen[0]=True
gestrichen[1]=True

# Liste Anzahl Primzahlen pro Intervall
d=[]
```

## Dichte\_Primzahlen.py

```
# Zu jedem Intervall
for j in range(N/be):
    sum =0
    # Summe der Primzahlen in diesem Intervall
    for k in range(j*be, (j+1)*be):
        if not gestrichen[k]:
            sum=sum+1
    d.append(sum)

# x-Skala gleichmaessig von 0 bis N
x = np.linspace(0, N, N/be)
y = np.array(d)
plot(x,y)
show()
```

## Laborteil – Lösen einzelner Aufgaben



## Fragen

- Matrizen invertieren
- Algebraisch ableiten
- Sage oder SymPy?

<https://github.com/sympy/sympy/wiki/SymPy-vs.-Sage>

## matrix.py

```
from numpy import matrix
from numpy import linalg
A = matrix( [[1,2,3],[11,12,13],[21,22,23]]) # Creates a matrix.
x = matrix( [[1],[2],[3]] )                      # Creates a matrix
# like a column vector.
y = matrix( [[1,2,3]] )                          # Creates a matrix
# like a row vector.
A_inv = linalg.inv(A)                            # invert matrix
print A_inv
print A.T                                         # Transpose of A.
print A*x                                         # Matrix
multiplication of A and x.
print A.I                                         # Inverse of A.
print linalg.solve(A, x)                         # Solve the linear equation system.
```

## Algebraisch ableiten, Symbolic Mathematics in Python

- <http://scipy-lectures.github.com/advanced/sympy.html#differentiation>

```
>>> from sympy import *
>>> from math import *
>>> diff(sin(x), x)
cos(x)
>>> diff(sin(2*x), x)
2*cos(2*x)
```

```
>>> diff(tan(x), x)
1 + tan(x)**2
```

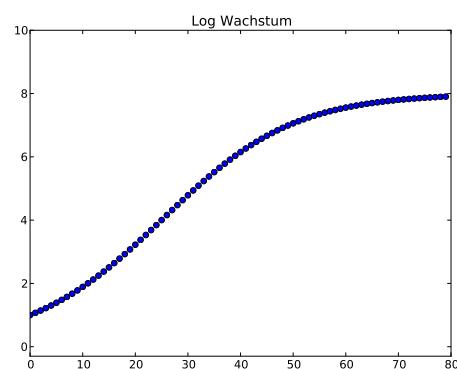
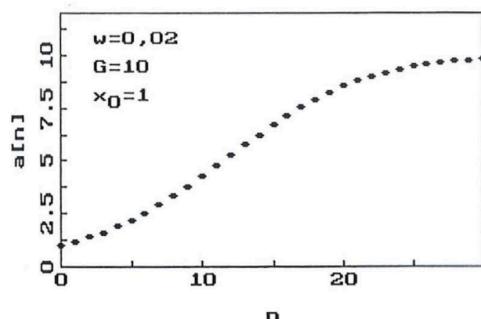
## Populationsdynamik

von Henning Körner, Oldenburg

$$(3) \quad x_{n+1} - x_n = w \cdot x_n \cdot (G - x_n)$$

(Logistisches Wachstum)

Es ergibt sich der typische s-förmige Verlauf:



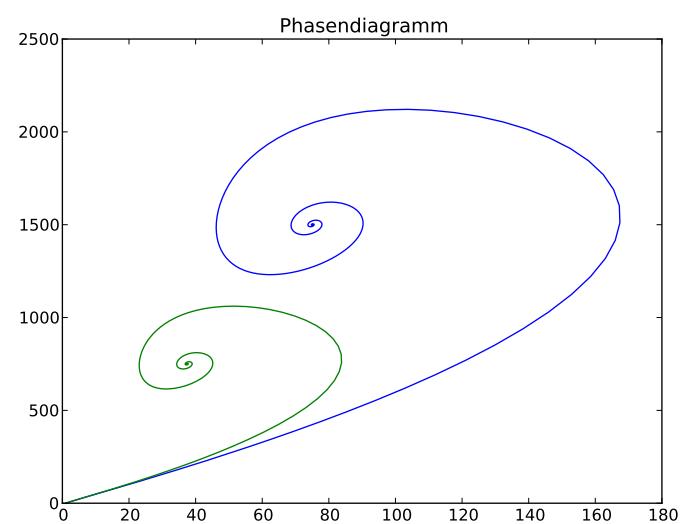
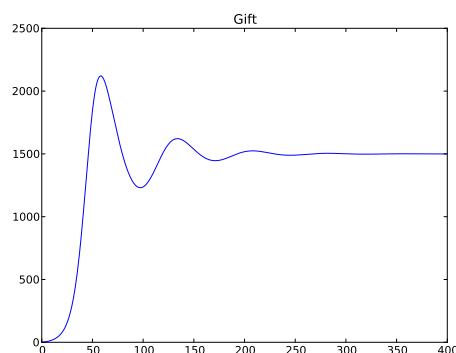
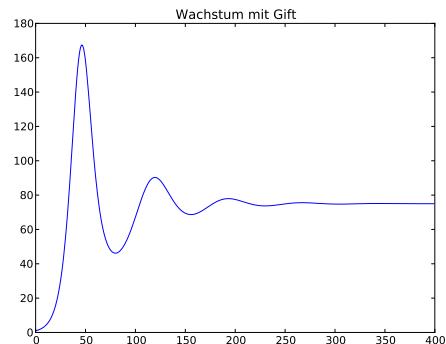
## Logistisches\_Wachstum.py

```
w=0.0099
G=8.0
def f(x):
    return x+w*x*(G-x)

def genPopulation(x0,n):
    x=[x0]
    # calculate 1..n-1 values
    for i in range(1,n):
        tmp = f(x[i-1])
        x.append(tmp)
    return x

popul=genPopulation(x0,80)
pp.plot(range(len(popul)),popul,'o')
```

## Wachstum mit Gift (optimistisch)



## Gift\_Wachstum\_optimistisch.py

```

k,m,g=1.15,0.95,0.0001      (5)     $x_{n+1} = (k - g \cdot y_n) \cdot x_n$ 
x0 = 1.0                       $y_{n+1} = m \cdot y_n + x_n$ 

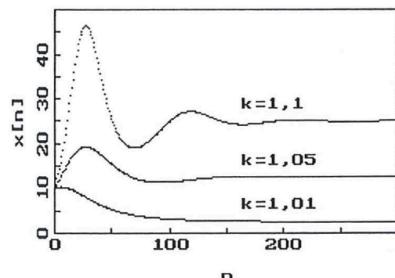
def f(x,y):
    return (k-g*y)*x

def genPopulation(x0,n):
    x,y=[x0],[0]
    for i in range(1,n):
        tmp = f(x[i-1],y[i-1])
        x.append(tmp)
        tmp2 = m*y[i-1]+x[i-1]
        y.append(tmp2)
    return x,y

popul,gift=genPopulation(x0,400)
pp.plot(range(len(gift)),gift)

```

(a) Variation von k:  
 $g = 0,0002; m = 0,95; x_0 = 10; y_0 = 0$



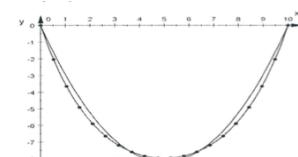
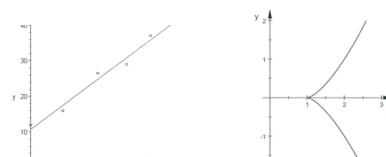
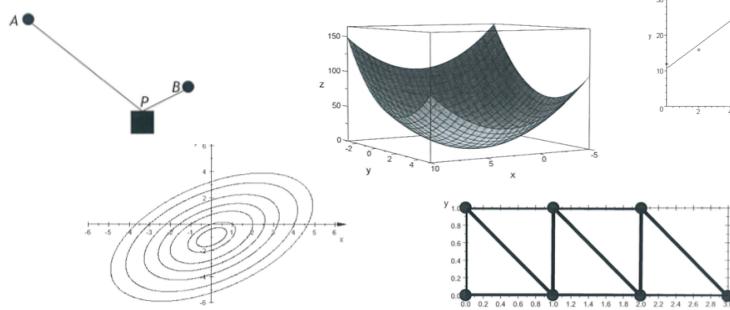
## Numerische Optimierung - ein schneller Weg zu komplexer Modellbildung

von Reinhard OLDENBURG, Heidelberg

### Einführung

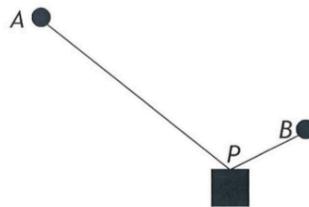
Optimierung ist ein sehr flexibles Wort. Was verbirgt sich eigentlich dahinter? Kurz gesagt: Fast alles. Einige Beispiele:

- Evolution als Optimierungsprozess
- Extremalprinzipien der Physik
- Betriebswirtschaft und Politik



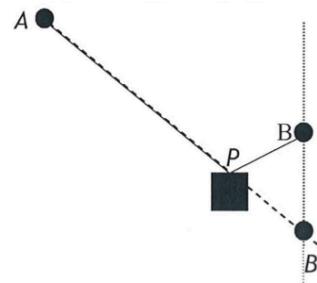
# Gewicht an einem Faden

von Reinhard OLDENBURG



Wenn man nun Koordinaten für die Aufhängepunkte (z.B.  $A(0, 5)$ ,  $B(3, 2)$ ) und einen Wert für die Fadenlänge (z.B.  $L = 8$ ) vorgibt, wohin rutscht dann  $P$ ? Natürlich wird die potentielle Energie minimiert, d.h. die y-Koordinate von  $P$  wird minimal unter der Nebenbedingung, dass die Strecken  $AP$  und  $PB$  zusammen die vorgegebene Länge  $L$  besitzen. Damit ist die Modellbildung abgeschlossen und wir können im Modell nach einer Lösung suchen.

- Geometrische Lösung:
- $B' (3.0, -2.416)$
- $P (2.106, -0.208)$



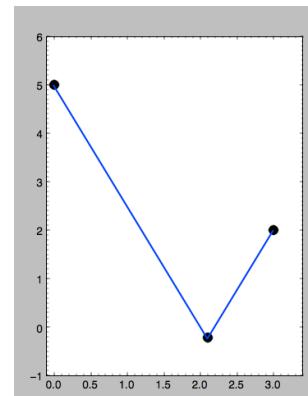
## Energieminimierung.py

```
import scipy.optimize as optimize
from math import sqrt

P1=[0.0,5.0]           # P1 (0 | 5)
P2=[3.0,2.0]           # P2 (3 | 2)
L=8                   # Laenge

def potEnergie(X):
    e=X[1]+(L-sqrt((P1[0]-X[0])**2+(P1[1]-X[1])**2) \
    -sqrt((P2[0]-X[0])**2+(P2[1]-X[1])**2))**2*1000
    return e

x0=[2.0,2.5]
xopt = optimize.fmin(potEnergie,x0, xtol=1e-9,
disp=True)
print(xopt)
Optimization terminated successfully.
Current function value: -0.208172
Iterations: 99
Function evaluations: 191
[ 2.1067561 -0.2082447]
```



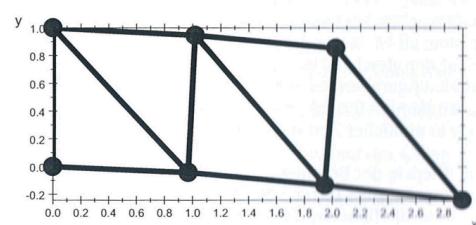
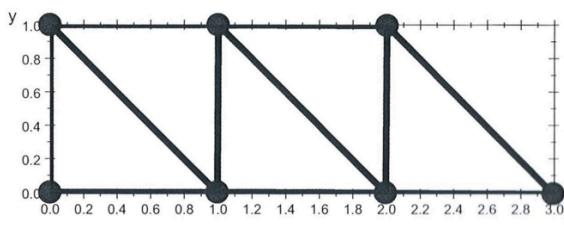
## Kranausleger

von Reinhard OLDENBURG

Stahlträger verhalten sich nahezu wie elastische Federn - zumindest solange sie nicht übermäßig belastet werden. Das bedeutet, dass sie eine Spannenergie besitzen, die proportional ist zum Quadrat der Abweichung der aktuellen Länge von der

$$\text{Ruhelänge: } E = \frac{1}{2} \cdot k \cdot (l - l_0)^2 \text{ (dabei ist } k \text{ die Hooke'sche Federkonstante).}$$

Interessant ist die Frage, wie sich der Ausleger verformt, wenn man ein Gewicht anhängt. Als Minimierungsproblem ist das leicht formuliert: Gesucht sind die Koordinaten  $x_i, y_i$  der Federendpunkte, die zum Minimum der Gesamtenergie (bestehend aus der Summe der Spannenergien aller Federn und der potentiellen Energie der angehängten Last) führen.



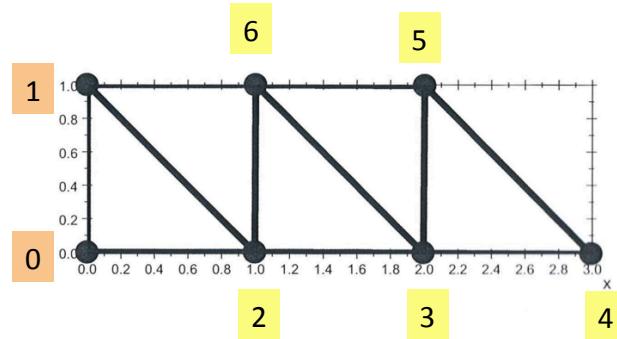
## Kranausleger.py

...

```
def feder(i,j,K,L,X) :
    """Input Knoten i,j ; Federkonstante K ;
Laenge L
        Rueckgabe Energie in der Feder"""
    xi = 2*i
    xj = 2*j
    return 0.5*K*(L-sqrt((X[xi]-X[xj])**2\
    + (X[xi+1]-X[xj+1])**2))**2
```

## Kranausleger.py II

```
def gesEnergie(x,x_fix,m):
    ...
    #Horizontal unten
    e=feder(0,2,100.0,1.0,x)
    e+=feder(2,3,100.0,1.0,x)
    e+=feder(3,4,100.0,1.0,x)
    #Horizontal oben
    e+=feder(1,6,100.0,1.0,x)
    e+=feder(6,5,100.0,1.0,x)
    #Vertikal
    e+=feder(6,2,100.0,1.0,x)
    e+=feder(5,3,100.0,1.0,x)
    #Schraeg
    e+=feder(1,2,100.0,1.414,x)
    e+=feder(6,3,100.0,1.414,x)
    e+=feder(5,4,100.0,1.414,x)
    e+=9.81*m*x[7]
    return e
```

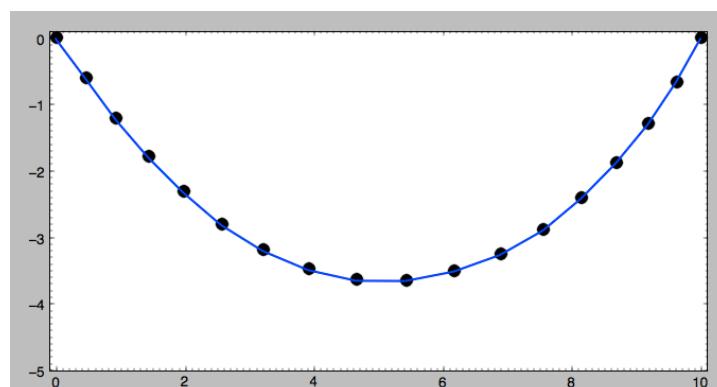


## Die hängende Kette

von Reinhard OLDENBURG

Dies ist ein weiteres klassisches Problem und experimentell ebenfalls leicht zugänglich (möglichst feingliedrige, nicht zu leichte Kette verwenden). Nachdem bereits bei der Brachistochrone diskretisiert wurde, ist es nicht überraschend, dass das hier nach dem gleichen Schema läuft. Man stellt sich die Kette vor als Polygonzug mit Stützstellen  $(x_i, y_i)$ ,  $i = 0, \dots, n$ . Minimiert werden soll die potentielle Energie

$$\tilde{f} = \sum_{i=0}^{n-1} \sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2} \cdot \frac{y_i + y_{i+1}}{2} + \mu \cdot \left( L - \sum_{i=0}^{n-1} \sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2} \right)$$



## Kette.py

```
N=17
lg = 12.0/N
...
def potEnergie(X):
    ...
    #Sum pot energy of every element
    sum=0.0
    for i in range(N):
        x=2*i
        y=2*i+1
        # ll calculated length, lg correct length
        ll =sqrt((w[x]-w[x+2])**2+(w[y]-w[y+2])**2)
        sum+=ll*(w[y]+w[y+2])/2.0+70*(lg-ll)**2

    return sum
```

## Optimierungsprobleme

- Häufig in Alltagssituationen anzutreffen (z.B. Kauf eines Gerätes)
- Optimierungsprobleme (OPs) sind Probleme, die i.A. viele **zulässige** Lösungen besitzen
- Jeder Lösung ist ein bestimmter Wert (**Zielfunktionswert, Kosten**) zugeordnet
- **Optimierungsalgorithmen** suchen in der Menge aller zulässigen Lösungen, diejenige mit dem **optimalen** Wert
- **Heuristiken** sind Algorithmen, die irgendeine Lösung berechnen

## Rundreiseproblem (TSP)

- Planung einer kürzesten Rundreise durch Europa (50 Städte):
  - jeder Ort muss einmal besucht werden
  - am Ende muss ich wieder zum Startort zurück

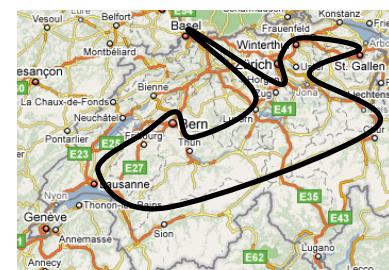


## Rundreiseproblem: Mathematisches Modell

- **Gegeben:** Graph  $G = (V, E)$  mit  $c : E \rightarrow \mathbb{R}^+$ 
  - Jeder Ort  $v_i$  wird zu einem Knoten
  - Verbindungsstrecke zwischen zwei Orten zu einer Kante
- **Gesucht:**
  - Kostenminimale Rundreise: z.B.  $v_1, \dots, v_n, v_1$

$\mathbb{R}^+ \ni c_{ij} :=$  Reisekosten zwischen Ort  $i$  und Ort  $j$

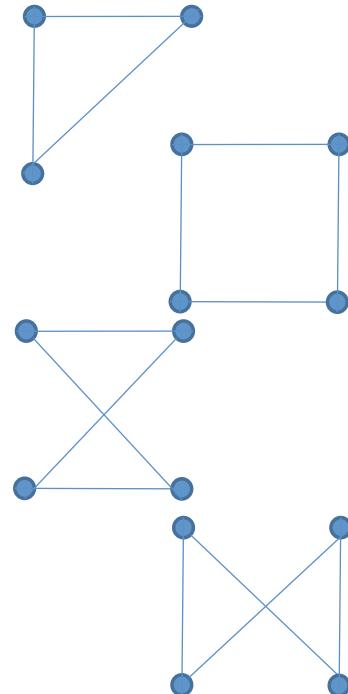
$$C = \begin{pmatrix} c_{11} & \dots & c_{1n} \\ \vdots & \ddots & \vdots \\ c_{n1} & \dots & c_{nn} \end{pmatrix}$$



## Was macht diese Probleme NP - schwierig ?

- Bsp.: Rundreiseproblem

Anzahl Orte	Anzahl mögliche Touren
3	1
4	3
5	12
6	60
7	360
8	2.520
9	20.160
10	181.440
11	1.814.400
12	19.958.400
n	$(n-1)! / 2$



## Rundreiseproblem: Rechenzeit

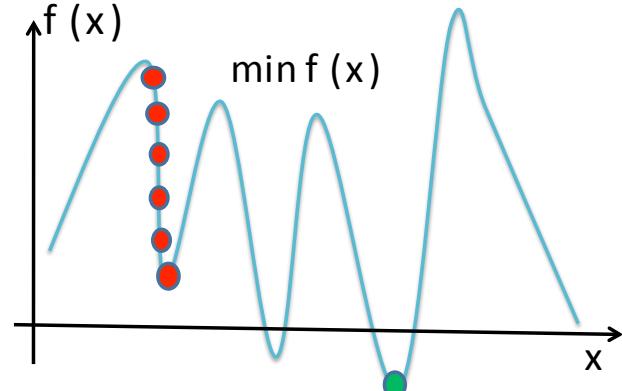
- Annahme: Rechner schafft 40 Mio. Touren in einer Sekunde

n	Anzahl Touren	Zeit
10	181.440	0.045 Sek.
17	ca. $10^{13}$	3 Tage
19	ca. $10^{15}$	2.5 Jahre
20	ca. $10^{17}$	48 Jahre
25	ca. $10^{23}$	$10^8$ Jahre
60	ca. $10^{81}$	$10^{66}$ Jahre

Anzahl Atome im Weltall: ca.  $10^{80}$  Atome

## Einfache Lokale Suche: Diskussion

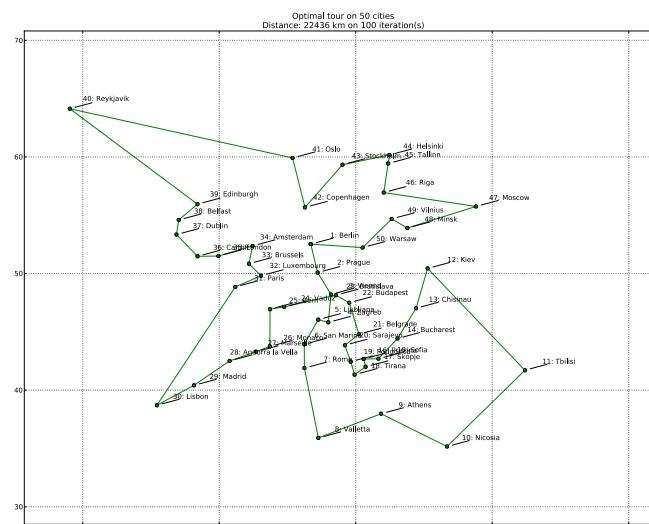
- Einfache lokale Suche ist in der Praxis sehr beliebt: deutliche Verbesserung innerhalb kurzer Zeit
- Problem: man bleibt relativ schnell in lokale Optima hängen, die relativ weit von Optimallösung entfernt sind, da nur bessere Lösungen akzeptiert werden
- Daher z.B. Metaheuristiken (naturinspirierte Algorithmen):
  - Evolutionäre Algorithmen
  - Simulated Annealing
  - Ameisenalgorithmen
  - Tabu Suche
  - ...



## Rundreise Vorschlag 22436 km

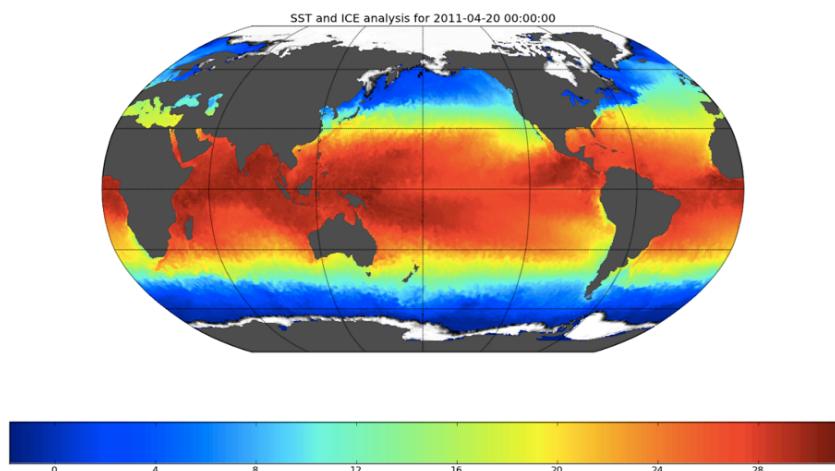
```
python Rundreise.py input [plot] [iterations] [nb_cities] [c_factor]
[t_start] [t_end] [r_seed]
```

```
$ python Rundreise.py cities.csv save 100 -1 0.93 1600 .5 -1
```



## Arbeiten mit aktuellen Datesätzen

- Messdaten auf Karte darstellen
- Datensätze aus dem Internet beziehen
- Wissenschaftliche selbstbeschreibende Daten Modelle
  - NETCDF 4
  - OPeNDAP

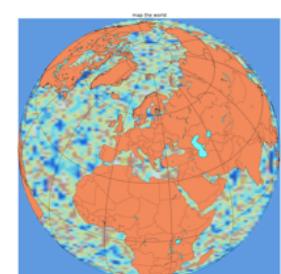


## Visualize a calculated measurement I

```
from netCDF4 import Dataset Load_Show_NETCDF_Data_Map.py
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap, cm
import numpy as np

root_grp = Dataset('test.nc')
temp = root_grp.variables['temp']
lons = root_grp.variables['longitude'][:,:]
lats = root_grp.variables['latitude'][:,:]
nx=len(lats)
ny=len(lons)

fig=plt.figure(figsize=(16,20))
```



## Visualize the calculated measurement II

```

fig=plt.figure(figsize=(16,20))
map =
Basemap(projection='ortho',lat_0=45,lon_0=20,resolution='l')
map.drawcoastlines(linewidth=0.25)
map.drawcountries(linewidth=0.25)
map.fillcontinents(color='coral',lake_color='aqua')
map.drawmeridians(np.arange(0,360,30))
map.drawparallels(np.arange(-90,90,30))

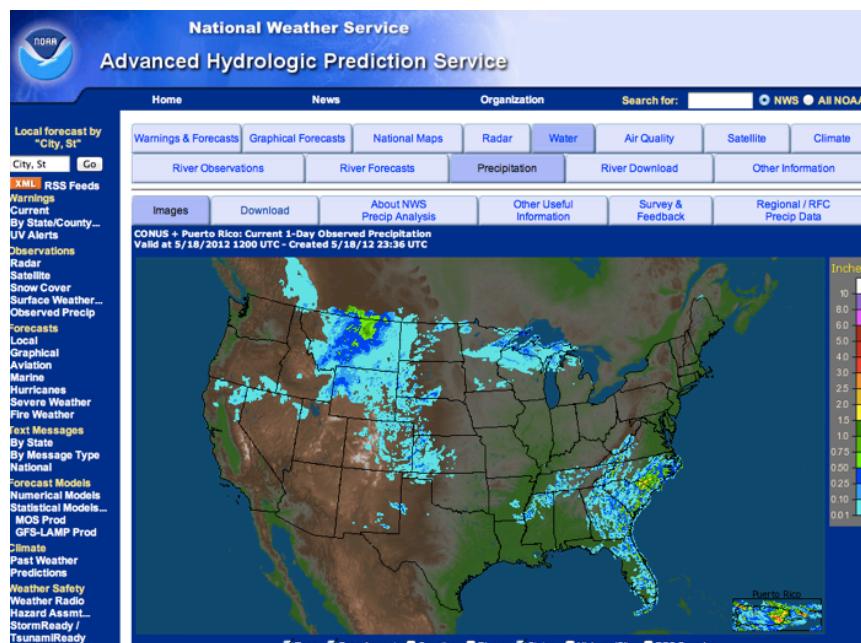
topodat = map.transform_scalar(temp[0],lons,lats,nx,ny)
im = map.imshow(topodat,cm.GMT_haxby,alpha=0.85)

c = plt.colorbar(im,orientation='horizontal')
plt.title('map the world')
plt.savefig('contour1.png')

```

## Rain Data from NOAA

- <http://water.weather.gov/precip/>



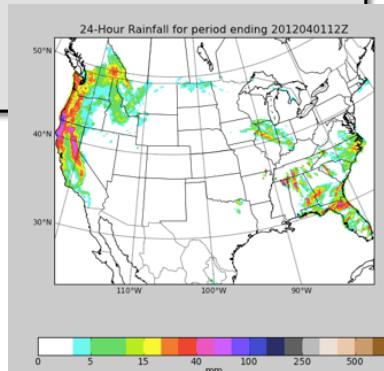
## Rain Data USA

```

• ...
• nc = NetCDFFile('nws_precip_conus_20120401.nc')
• # data from http://water.weather.gov/precip/
• prcpvar = nc.variables['amountofprecip']
• data = 0.01*prcpvar[:]
• latcorners = nc.variables['lat'][:]
• loncorners = -nc.variables['lon'][:]
• ...

```

Show\_RainData\_USA.py



## Acronyms for different data repositories

- Live Access Server (LAS)
- OPeNDAP Open-Source Project for a Network Data Access Protocol (DAP)
- The Global Change Master Directory (GCMD)
- <http://gcmd.gsfc.nasa.gov/>
- Grid Analysis and Display System (GrADS)
- GrADS Data Server (GDS)
- Global Forecast System (GFS)
- Global Data Assimilation System (GDAS)
- Thematic Realtime Environmental Distributed Data Services (THREDDS)
- Dataset Descriptor Structure (DDS)

## OPeNDAP

- OPeNDAP is a data server architecture that allows users to use data files that are stored on remote computers with their favorite analysis and visualization client software.
- OPeNDAP URL :  
<http://test.opendap.org/dap/data/nc/sst.mnmean.nc.gz>
- Dataset Descriptor Structure:  
<http://test.opendap.org/dap/data/nc/sst.mnmean.nc.gz.dds>
- OPeNDAP provides an info service:  
<http://test.opendap.org/dap/data/nc/sst.mnmean.nc.gz.info>
- Peeking at Data:  
[http://test.opendap.org/dap/data/nc/sst.mnmean.nc.gz.asc?time\[0:6\]](http://test.opendap.org/dap/data/nc/sst.mnmean.nc.gz.asc?time[0:6])  
[http://test.opendap.org/dap/data/nc/sst.mnmean.nc.gz.asc?sst\[0:1\]\[13:16\]\[103:105\]](http://test.opendap.org/dap/data/nc/sst.mnmean.nc.gz.asc?sst[0:1][13:16][103:105])
- Data Sources  
<http://opendap.nmdc.eu/> (Climate )  
<http://nomad2.ncep.noaa.gov> (Meteo / Climate)  
<http://opendap.deltares.nl/> (EarthScience)  
[http://docs.opendap.org/index.php/Dataset\\_List](http://docs.opendap.org/index.php/Dataset_List)

## GrADS Data Server

```

• ...
Aktuelle_Temp_2M.py
• data=Dataset ("http://nomad2.ncep.noaa.gov:9090/dods/
gdas/rotating/gdas"+YYYYMMDDHH+".grib2")

• # read lats,lons.
• lats = data.variables['lat'][:]
• lons1 = data.variables['lon'][:]
• nlats = len(lats)
• nlons = len(lons1)
• # read tmax 2m (K)
• tmax2m = data.variables['tmax2m'][0]-273.15
• # shift data so lons go from -180 to 180
• prmsl,lons1 = shiftgrid(180.,prmsl,lons1,start=False)
• ...
Tmax2m ** 2 m above ground maximum temperature [k]

```

## GrADS Data

<http://nomad2.ncep.noaa.gov:9090/dods/gdas/rotating/gdas2012051900.grib2>

OPeNDAP/DODS Data URL: <http://nomad2.ncep.noaa.gov:9090/dods/gdas/rotating/gdas2012051900.grib2>

**Description:** GDAS 00Z19may2012 downloaded May 19 07:04 UTC  
**Documentation:** (none provided)  
**Longitude:** 0.00000000000°E to 359.00000000000°E (360 points, avg. res. 1.0°)  
**Latitude:** -90.0000000000°N to 90.0000000000°N (181 points, avg. res. 1.0°)  
**Altitude:** 1000.0000000000 to 10.0000000000 (26 points, avg. res. 39.6)  
**Time:** 00Z19MAY2012 to 06Z19MAY2012 (2 points, avg. res. 0.25 days)  
**Variables:** (total of 26 variables listed)  
 absvprs \*\* (1000 975 950 925 900.. 70 50 30 20 10) absolute vorticity [1/s]  
 no4lftxsfc \*\* (1000 975 950 925 900.. 70 50 30 20 10) lowest (4 layer) lifted index [k]  
 no5wava500m \*\* (1000 975 950 925 900.. 70 50 30 20 10) 5th limb 5-wave geopotential height anomaly [gpm]  
 tedccell convective cloud layer total cloud cover [%]  
 tmax2m \*\* 2 m above ground maximum temperature [k]  
 tmin2m \*\* 2 m above ground minimum temperature [k]  
 tmplelt \*\* low cloud top level temperature [k]  
 tmpmelt \*\* middle cloud top level temperature [k]  
 tmphelt \*\* high cloud top level temperature [k]  
 tempsfc \*\* surface temperature [k]  
 tmpprs \*\* (1000 975 950 925 900.. 70 50 30 20 10) temperature [k]  
 tmp\_1829m \*\* 1829 m above mean sea level temperature [k]  
 tmp\_2743m \*\* 2743 m above mean sea level temperature [k]  
 tmp\_3658m \*\* 3658 m above mean sea level temperature [k]

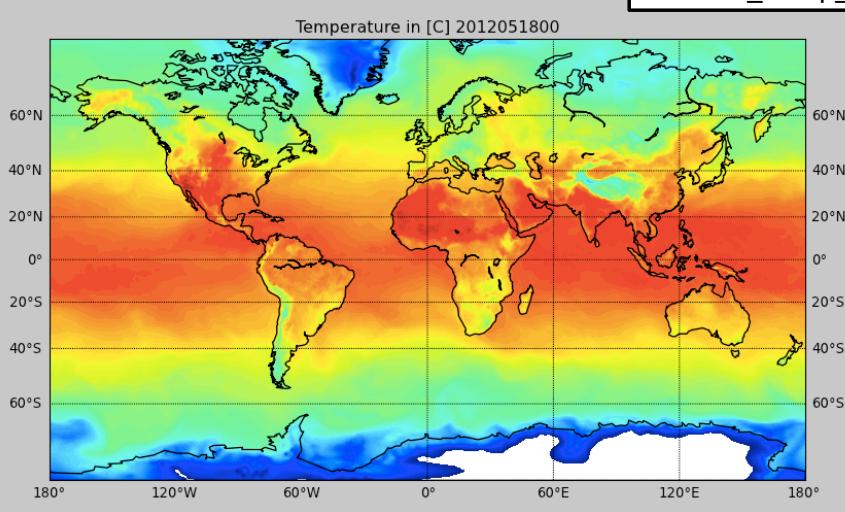
© Martin Guggisberg

<http://mgje.github.com/Python-Mathematik-Beispiele/>

61

## 2 m above ground maximum temperature

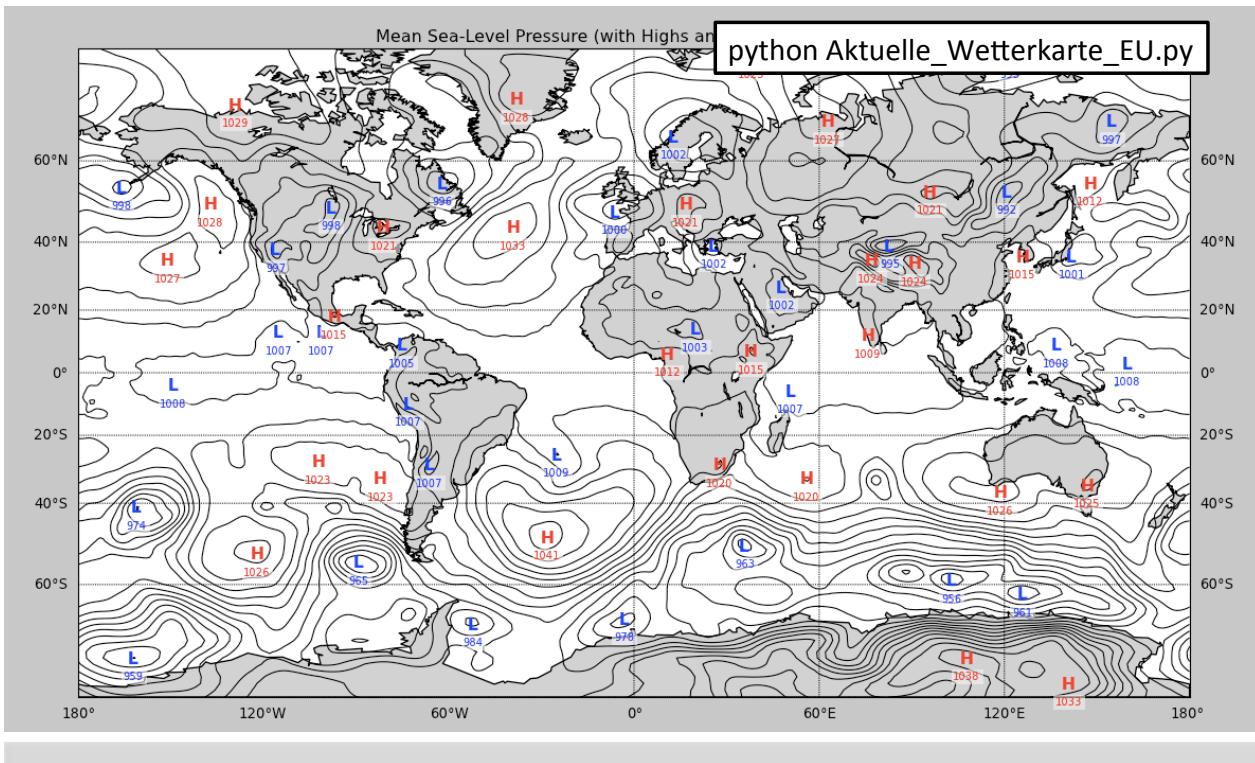
Aktuelle\_Temp\_2M.py



© Martin Guggisberg <http://mgje.github.com/Python-Mathematik-Beispiele/>

62

## mean sea level pressure reduced to msl [pa]



## Laborteil – Experimentieren mit Python

