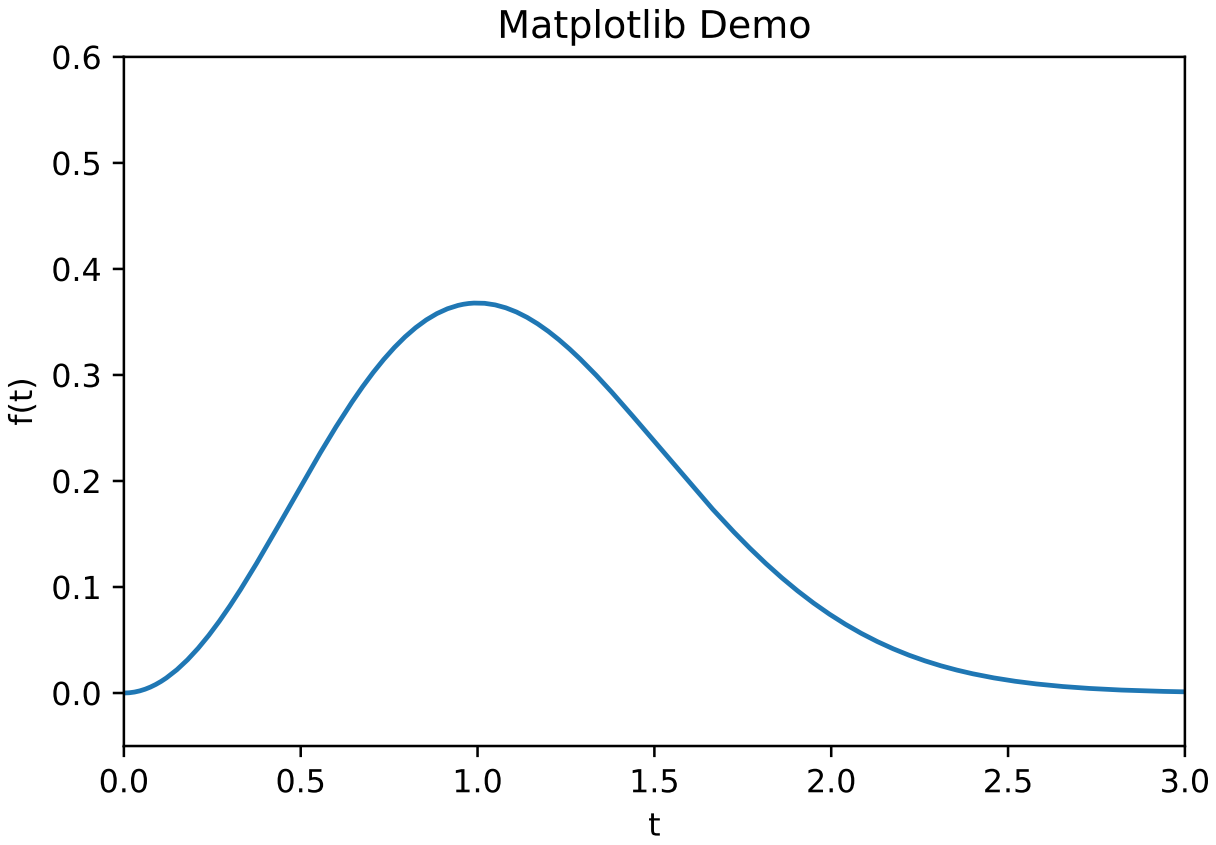
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
#Created on Tue Nov 14 14:22:01 2017
#
#@author: christianehelzel
```

1 6. Vorlesung

1.1 Matplotlib: Graphische Darstellung, Methode der kleinsten Quadrate

Ein kleines Beispiel

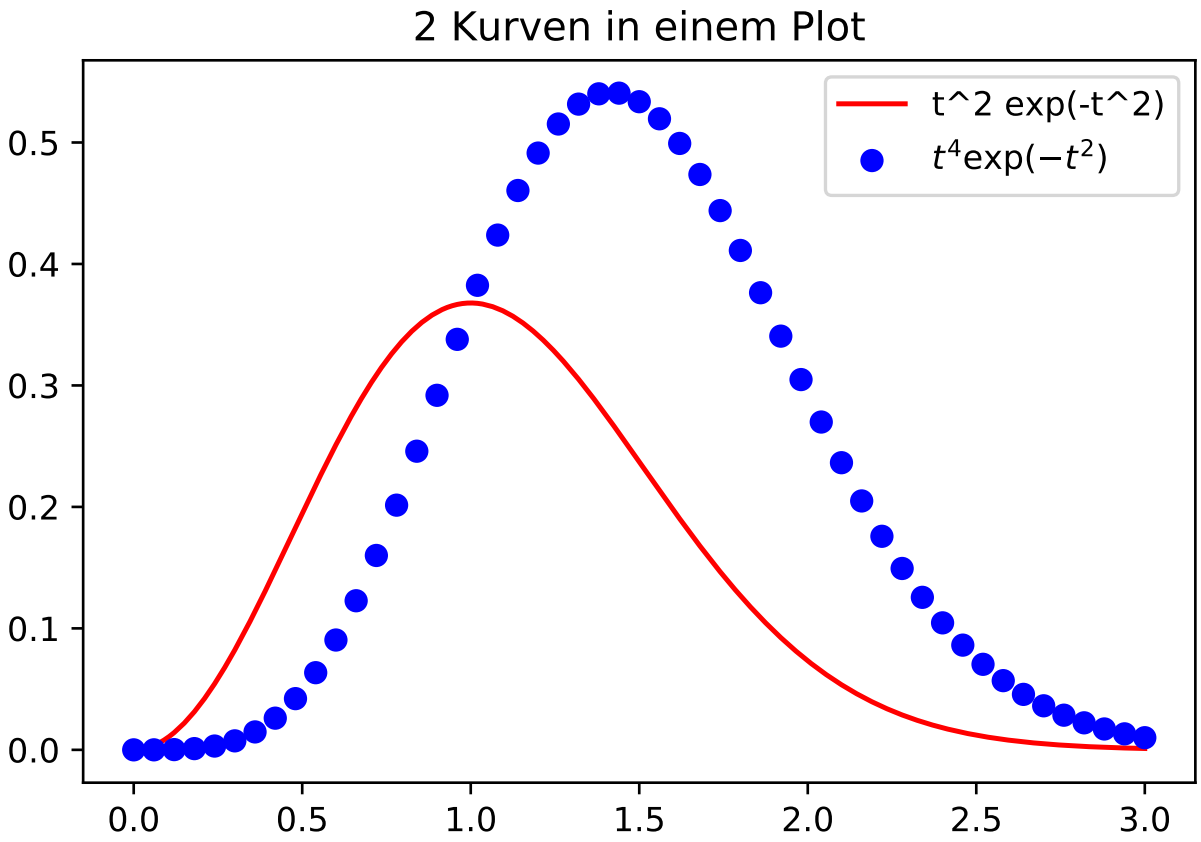
```
import numpy as np
import matplotlib.pyplot as plt


def f(t):
    return t**2*np.exp(-t**2)

t = np.linspace(0,3,201)
y = f(t)

# mit plt.figure() neues Plotfenster öffnen
fig = plt.figure()
plt.plot(t,y)

plt.xlabel('t')    #Achsenbeschriftung
plt.ylabel('f(t)')

plt.axis([0, 3, -0.05,0.6])    # Achsenangabe
plt.title('Matplotlib Demo')   # Bildüberschrift


fig.savefig('bild1.pdf')  # speichere Bild als pdf-File
# fig.savefig('bild1.png')  # speichere Bild als png-File
```



2 Kurven in einem Plot

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

def f1(t):
    return t**2*np.exp(-t**2)
def f2(t):
    return t**2*f1(t)

t1 = np.linspace(0,3,201)
t2 = np.linspace(0,3,51)
y1 = f1(t1)
y2 = f2(t2)

plt.figure()
plt.plot(t1,y1,'r-')
plt.plot(t2,y2,'bo')

plt.title('2 Kurven in einem Plot')
#LaTeX geht auch
plt.legend(['t^2 exp(-t^2)','$t^4 \exp(-t^2)$'])
```

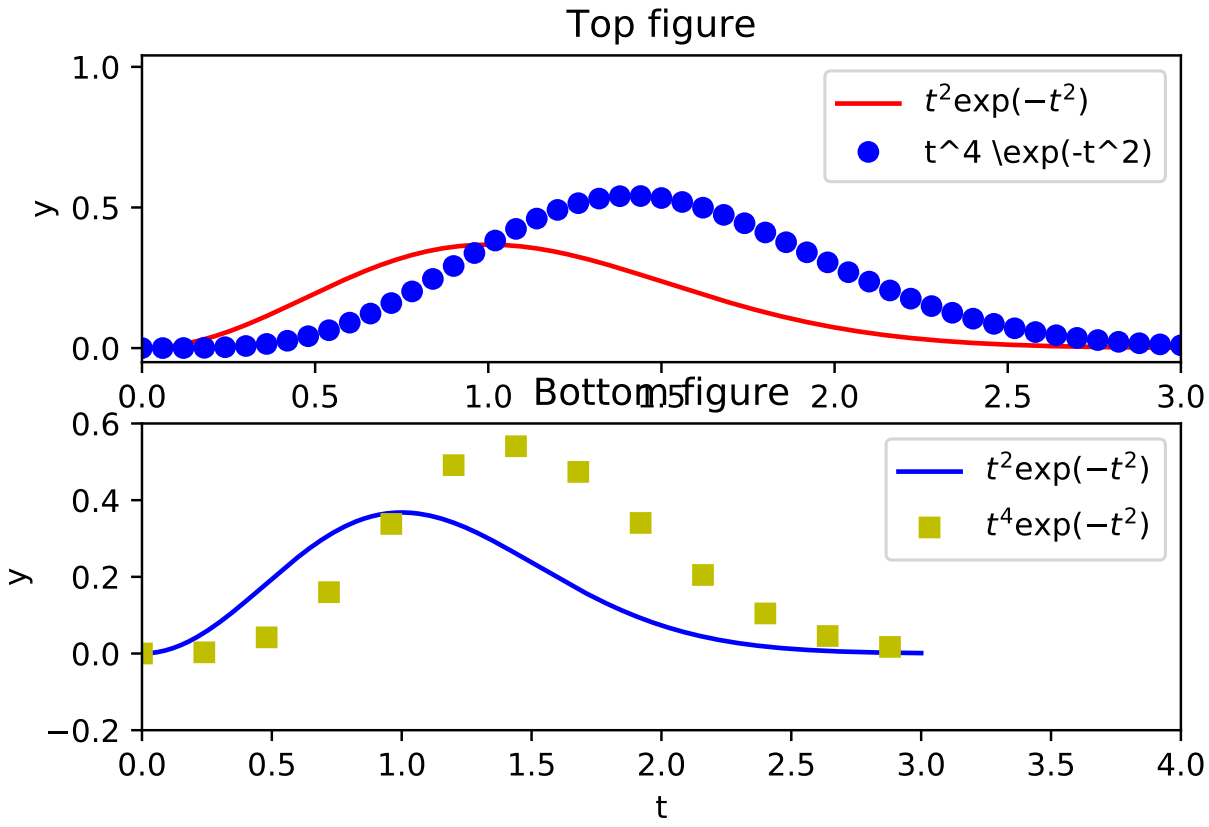
```
<matplotlib.legend.Legend at 0x7fdd945bf908>
```



2 separate Plots

```
plt.figure() # neues Plotfenster (sonst wäre das im selben wie oben)

#subplot: 2 Reihen, eine Spalte, erster Plot
plt.subplot(2, 1, 1)
t1 = np.linspace(0, 3, 201)
t2 = np.linspace(0, 3, 51)
y1 = f1(t1)
y2 = f2(t2)

plt.plot(t1, y1, 'r-', t2, y2, 'bo')
plt.xlabel('t')
plt.ylabel('y')
plt.axis([t[0], t[-1], min(y2)-0.05, max(y2)+0.5])
plt.legend(['$t^2 \exp(-t^2)$', 't^4 \exp(-t^2)'])
plt.title('Top figure')

#subplot: 2 Reihen, eine Spalte, zweiter Plot
plt.subplot(2, 1, 2)
t3 = t2[::4]
y3 = f2(t3)
plt.plot(t1, y1, 'b-', t3, y3, 'ys')
plt.xlabel('t')
plt.ylabel('y')
plt.axis([0, 4, -0.2, 0.6])
plt.legend(['$t^2 \exp(-t^2)$', '$t^4 \exp(-t^2)$'])
plt.title('Bottom figure')
```

```
Text(0.5,1,'Bottom figure')
```

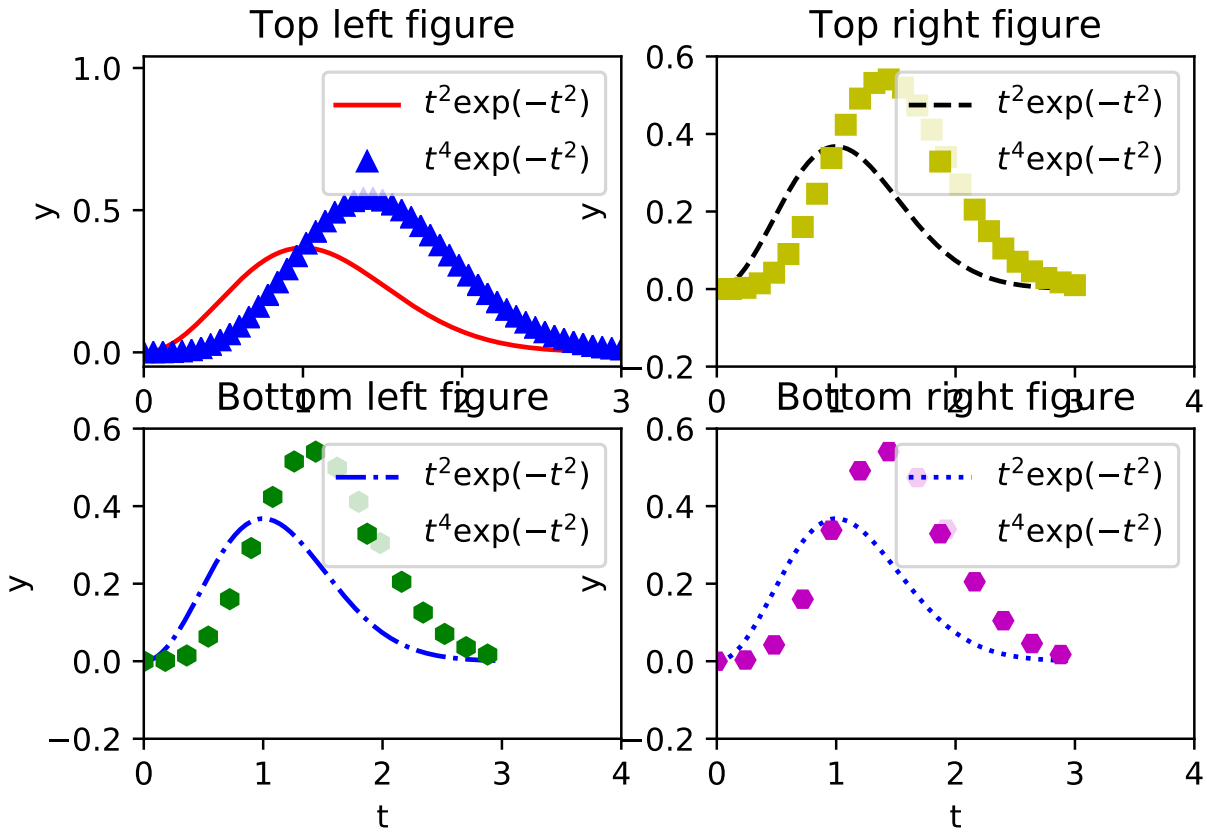


4 plots in one figure

```
plt.figure() # make separate figure

plt.subplot(2, 2, 1)
t1 = np.linspace(0, 3, 201)
t2 = np.linspace(0, 3, 51)
y1 = f1(t1)
y2 = f2(t2)

plt.plot(t1, y1, 'r-', t2, y2, 'b^')
plt.xlabel('t')
plt.ylabel('y')
plt.axis([t[0], t[-1], min(y2)-0.05, max(y2)+0.5])
plt.legend(['$t^2 \exp(-t^2)$', '$t^4 \exp(-t^2)$'])
plt.title('Top left figure')


plt.subplot(2, 2, 2)
t3 = t2[::2]
y3 = f2(t3)
plt.plot(t1, y1, 'k--', t3, y3, 'ys')
plt.xlabel('t')
plt.ylabel('y')
plt.axis([0, 4, -0.2, 0.6])
plt.legend(['$t^2 \exp(-t^2)$', '$t^4 \exp(-t^2)$'])
plt.title('Top right figure')

plt.subplot(2, 2, 3)
t3 = t2[::3]
y3 = f2(t3)
```

```
plt.plot(t1, y1, 'b-.', t3, y3, 'gh')
plt.xlabel('t')
plt.ylabel('y')
plt.axis([0, 4, -0.2, 0.6])
plt.legend(['$t^2 \exp(-t^2)$', '$t^4 \exp(-t^2)$'])
plt.title('Bottom left figure')

plt.subplot(2, 2, 4)
t3 = t2[::4]
y3 = f2(t3)
plt.plot(t1, y1, 'b:', t3, y3, 'mH')
plt.xlabel('t')
plt.ylabel('y')
plt.axis([0, 4, -0.2, 0.6])
plt.legend(['$t^2 \exp(-t^2)$', '$t^4 \exp(-t^2)$'])
plt.title('Bottom right figure')
```

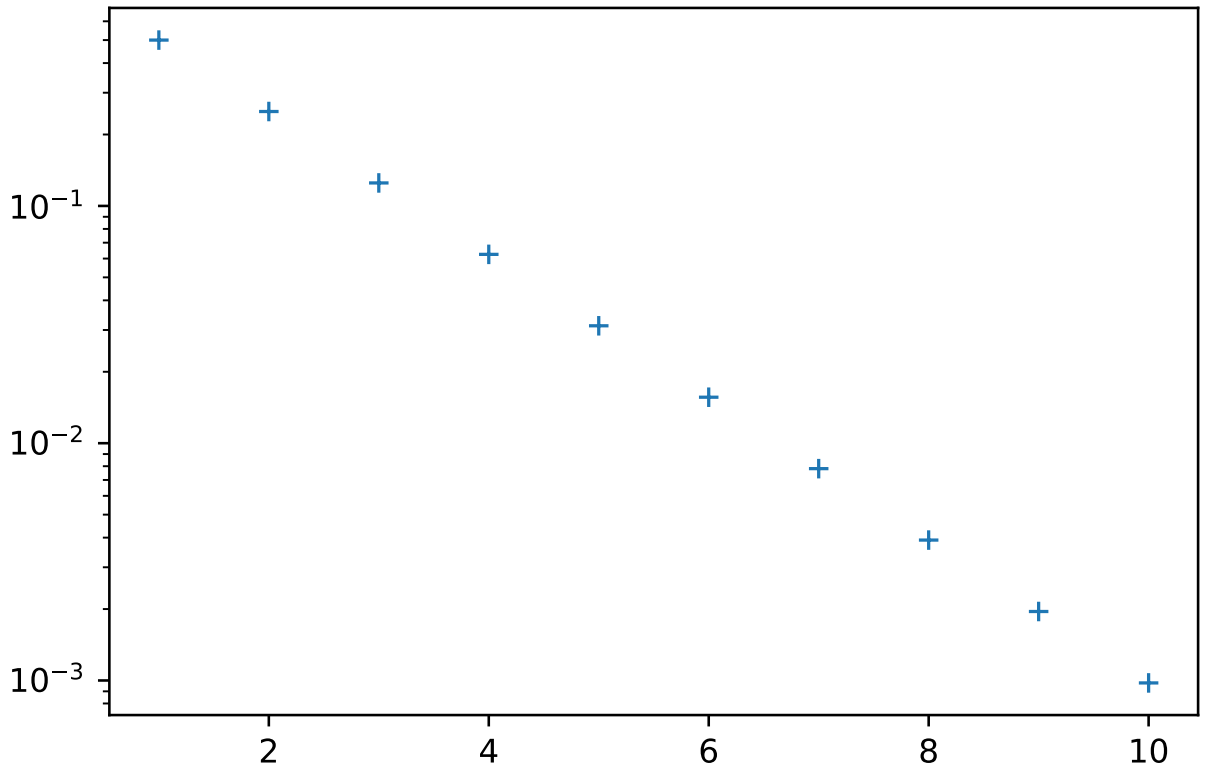
```
Text(0.5,1,'Bottom right figure')
```



Mehr Infos (wie zu jedem Befehl): help(plt.plot)

logarithmische Achsen

```
x = np.linspace(1,10,10)
y = 2**(-x)
plt.figure()
#plt.plot(x,y,'r+')
plt.semilogy(x,y,'+')
plt.show()
```



1.2 stückweise stetige Funktionen

Heaviside Funktion

```
def H(x):
    return (0 if x < 0 else 1)

import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-10,10,5)
#y=H(x) # liefert Fehlermeldung
#plt.plot(x,y)
```

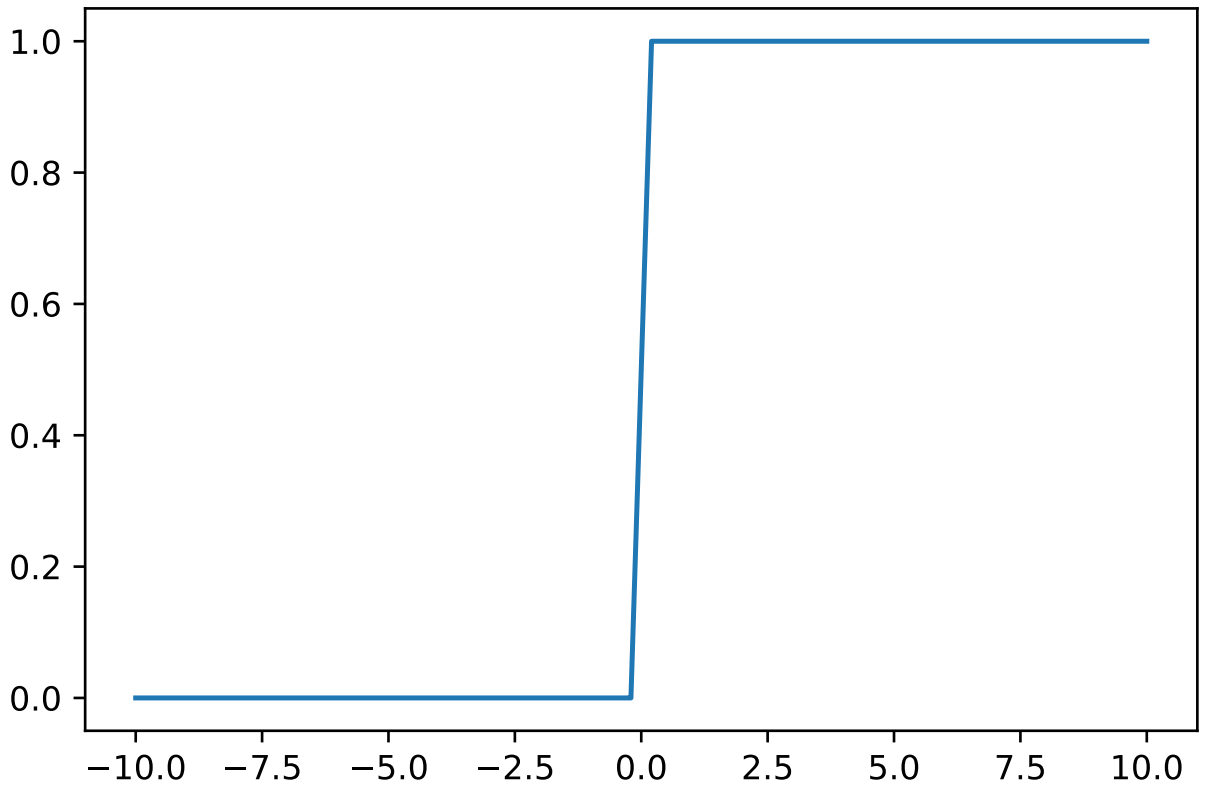
Grund der Fehlermeldung: x<0 liefert ein Array mit logischen Werten, if-Test benötigt skalaren boolischen Wert

1. Lösungsmöglichkeit: verwende loop (geht jetzt aber nicht mehr mit "floats" oä, da len(x) nicht geht))

```
def H_loop(x):
    r = np.zeros(len(x))
    for i in np.arange(len(x)):
        r[i] = H(x[i])
    return r

x = np.linspace(-10,10,50)
y = H_loop(x)
```

```
plt.figure()
plt.plot(x,y)
```

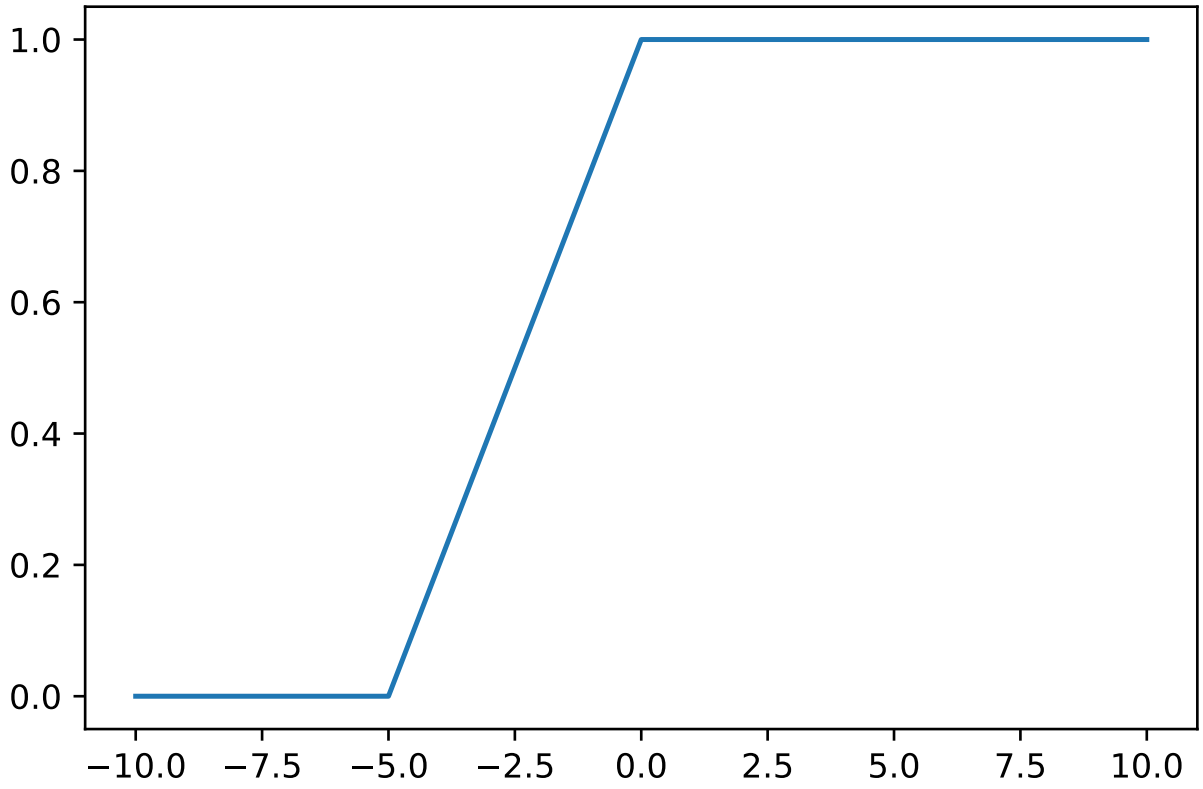
```
[<matplotlib.lines.Line2D at 0x7fdd94666710>]
```



2. Lösungsansatz: automatische Vektorisierung

```
H_vec = np.vectorize(H)

x = np.linspace(-10,10,5)
y = H_vec(x)

plt.figure()
plt.plot(x,y)
```

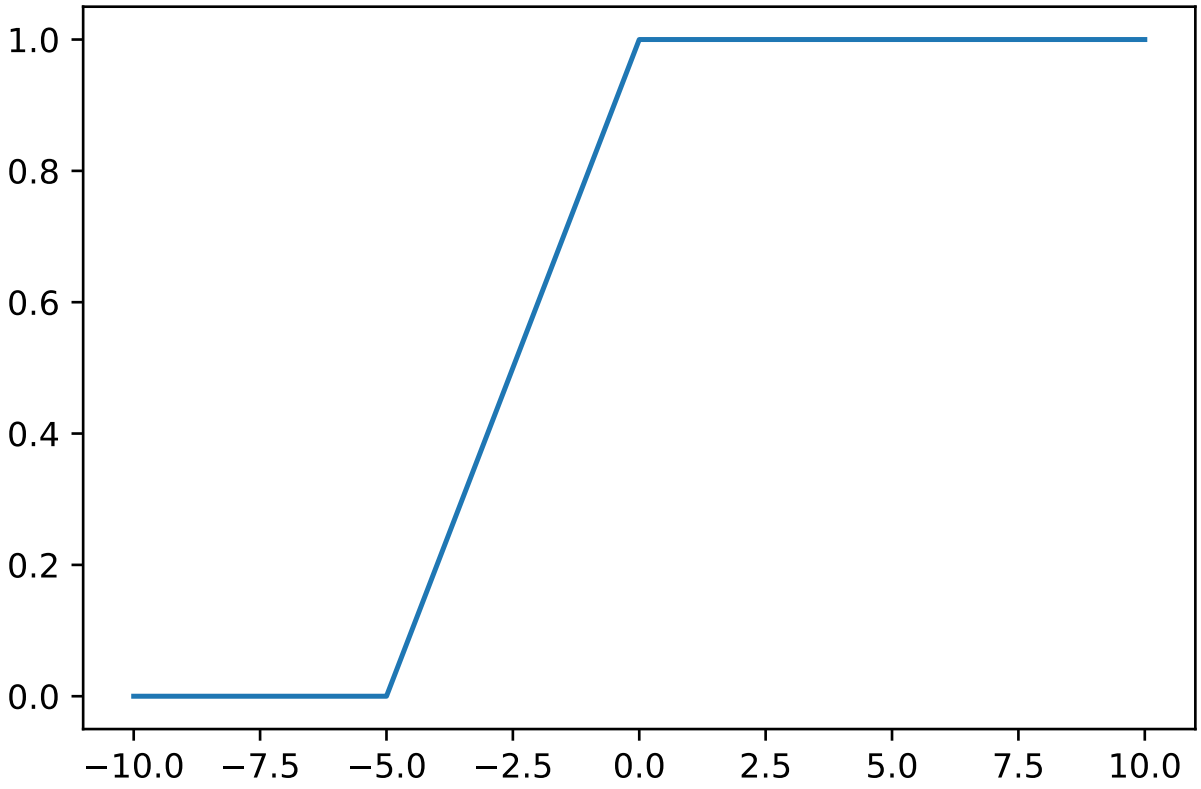
```
[<matplotlib.lines.Line2D at 0x7fdd93e84c88>]
```



3. Lösungsansatz: Vermische logische Werte mit Gleitkommazahl in Rechnung

```
def H(x):
    return x>=0

x = np.linspace(-10,10,5)
y = 1.*H(x)
plt.plot(x,y)
```

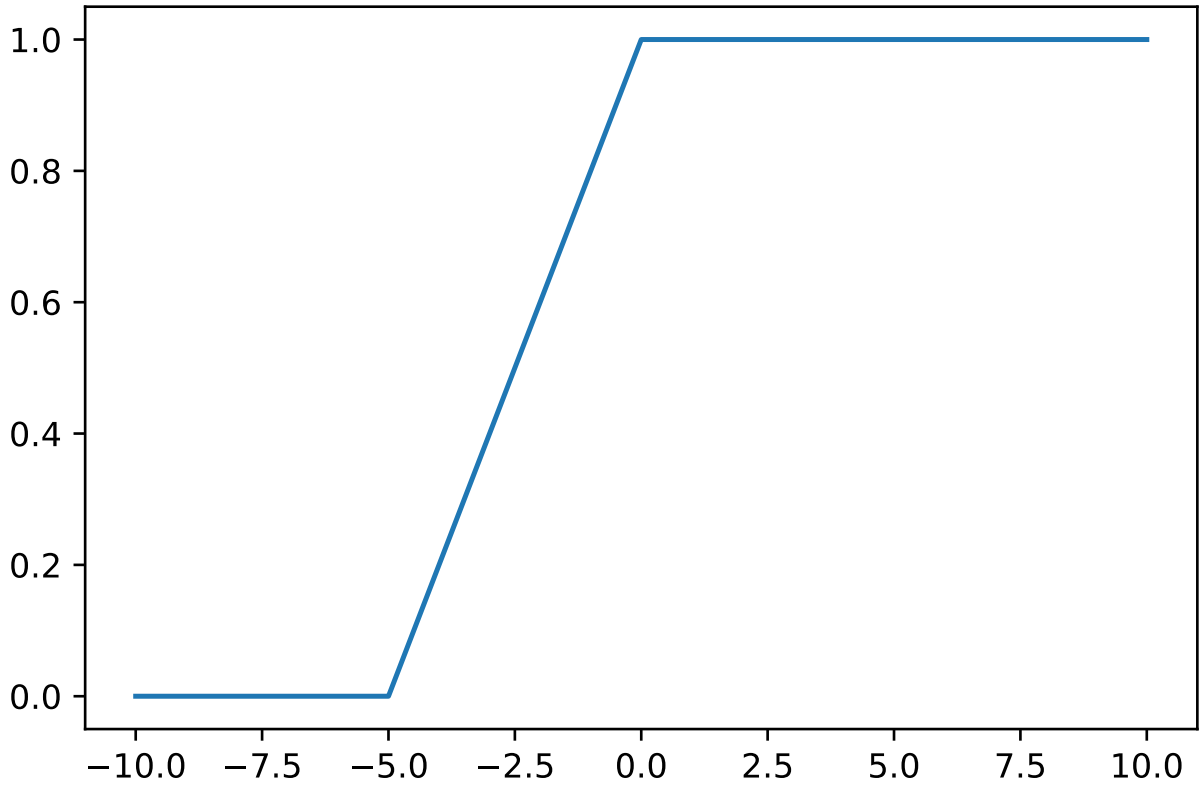
```
[<matplotlib.lines.Line2D at 0x7fdd93ce5780>]
```



4. Lösungsansatz: manuelle Vektorisierung

```
def Hv(x):
    r = np.zeros(len(x),dtype=np.int)
    r[x>=0]=1
    return r

x = np.linspace(-10,10,5)
y = Hv(x)

plt.plot(x,y)
```

```
[<matplotlib.lines.Line2D at 0x7fdd93c45d30>]
```



Siehe hierzu auch die “hat”-Datei in den Vorlesungsunterlagen

1.3 Methode der kleinsten Quadrate - least square fit

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

Erzeuge Datenpunkte

```
f = np.poly1d([5, 1])
x = np.linspace(0, 10, 5)
y = f(x) + 6*np.random.normal(size=len(x))


print(x,y)
```

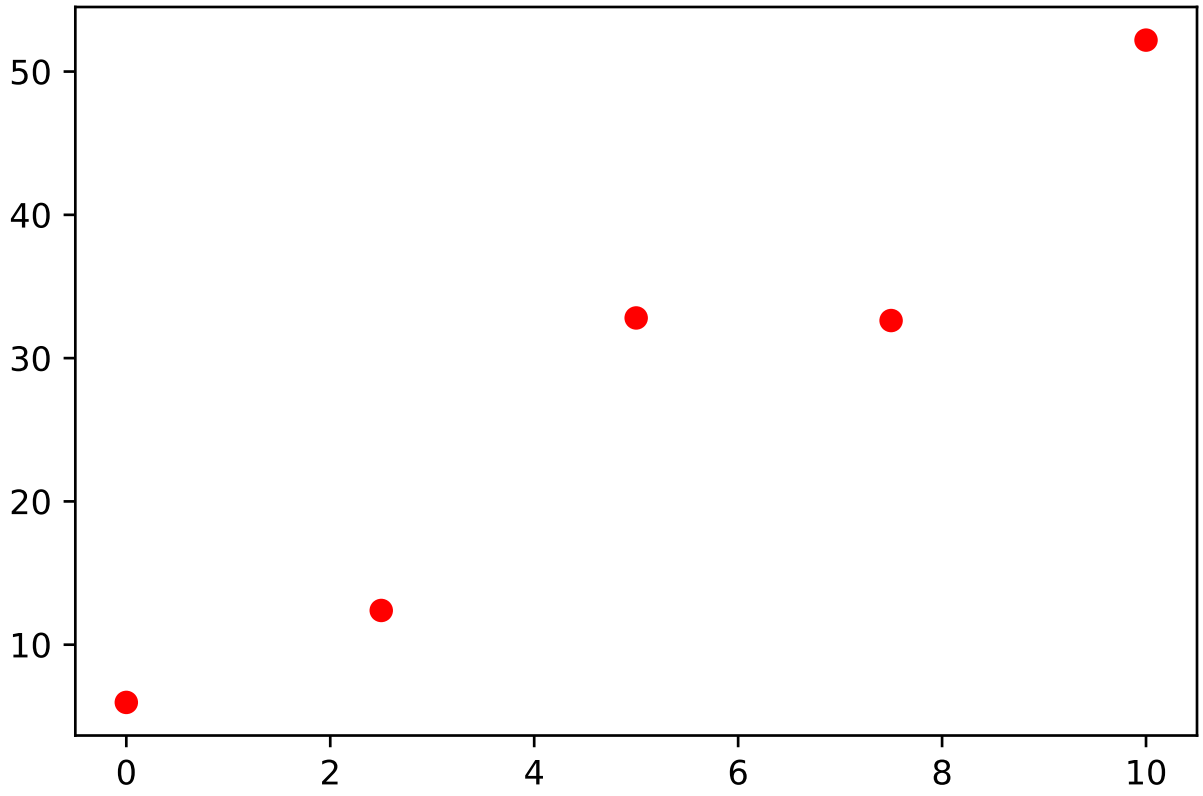
```
[  0.    2.5   5.    7.5  10. ] [  5.97168208  12.38857487
32.80384027  32.62275238  52.19467106]
```

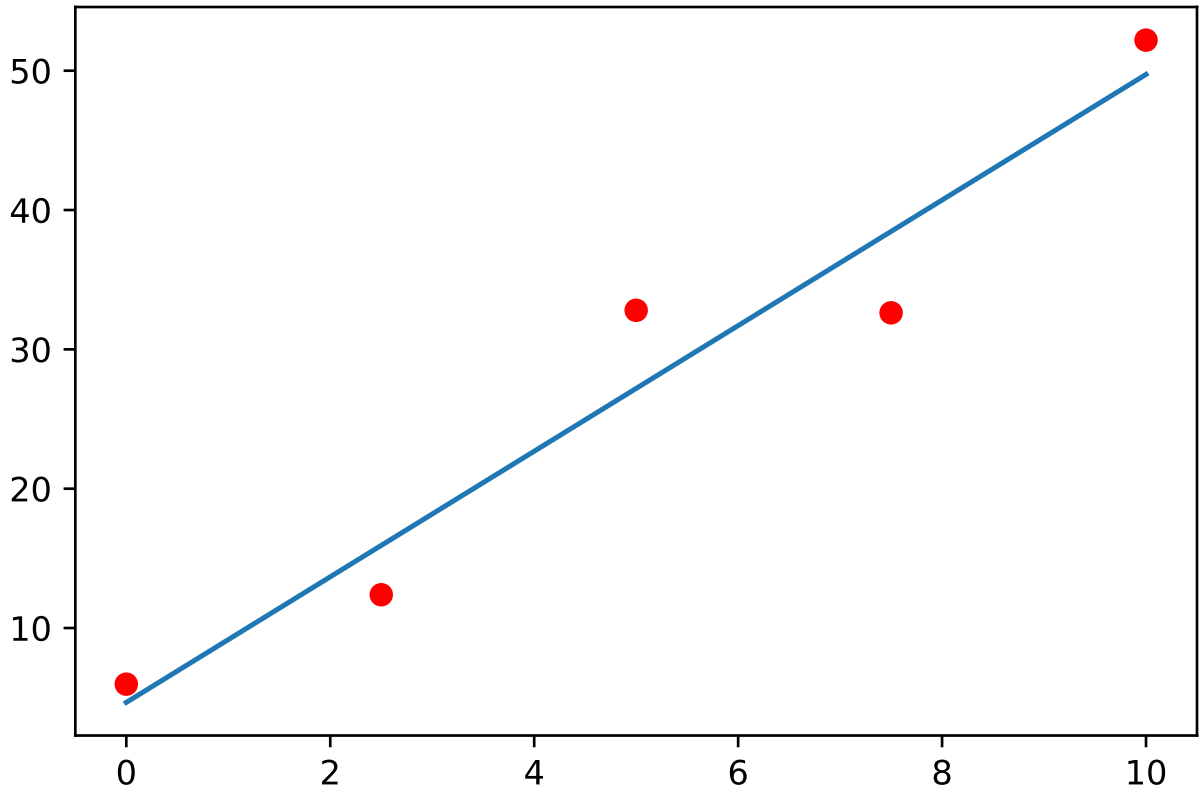
Zeichne Daten

```
plt.figure()
plt.plot(x, y, 'or')
plt.show()

A = np.vstack([x, np.ones(len(x))]).T
# np.dot(np.linalg.inv(np.dot(a.T, a)), np.dot(a.T, y))
```

```
coeff = npl.solve(A.T@A,A.T@y)

lsf = coeff[0]*x+coeff[1]

plt.plot(x, y, 'or',x, lsf)

# alternativ: Benutze npl.lstsq
```

```
[<matplotlib.lines.Line2D at 0x7fdd93b92128>,
 <matplotlib.lines.Line2D at 0x7fdd93b922b0>]
```





Wir lösen wieder least square Problem. Die Daten werden diesmal aus einer Datei eingelesen

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

1.4 Einlesen der Daten aus einer Datei

Variante 1: Standardpythonstyle

```
x = []
y = []

# Datei öffnen
f = open('data.txt')

for line in f.readlines():  #Line ist ein string
        #Teilt den string bei "whitespace", Rückgabe ist eine Liste mit strings
        numbers_str = line.split()
        #strings in floats umwandeln
        numbers_float = [float(x) for x in numbers_str]
        #die neuen Werte zu x und y hinzufügen
        x.append(numbers_float[0])
        y.append(numbers_float[1])
        #Alternativ:
        #x.extend([numbers_float[0]])
        #y.extend([numbers_float[1]])
#Am Ende die Datei wieder schließen
f.close()
```

Unterschied zwischen 'append' und 'extend':

x.append(y) hängt an die Liste 'x' das Objekt 'y' an. Egal was 'y' ist (float oder selbst eine Liste)

x.extend(y) hängt an die Liste 'x' die Elemente der Liste 'y' an. 'y' MUSS also eine Liste (bzw tupel) sein.

Die Daten sind jetzt in zwei Listen 'x' und 'y'

Variante 2: Numpystyle

Numpy hat Funktionen zum bequemen Ein- und Auslesen von Daten aus Textdateien

```
x2,y2=np.loadtxt('data.txt',unpack=True)
```

np.loadtxt() läd die Daten (durch whitespace getrennt) in ein(!) array. Die Option 'unpack' teilt dieses array direkt auf, so dass ohne Umwege in 'x' und 'y' gespeichert werden kann.

x2 und y2 sind in diesem Fall np.arrays (nicht wie oben Listen)

Der Inhalt von x und x2 (bzw. y und y2) ist gleich:

```
print((np.linalg.norm(np.array(x)-x2),np.linalg.norm(np.array(y)-y2)))
```

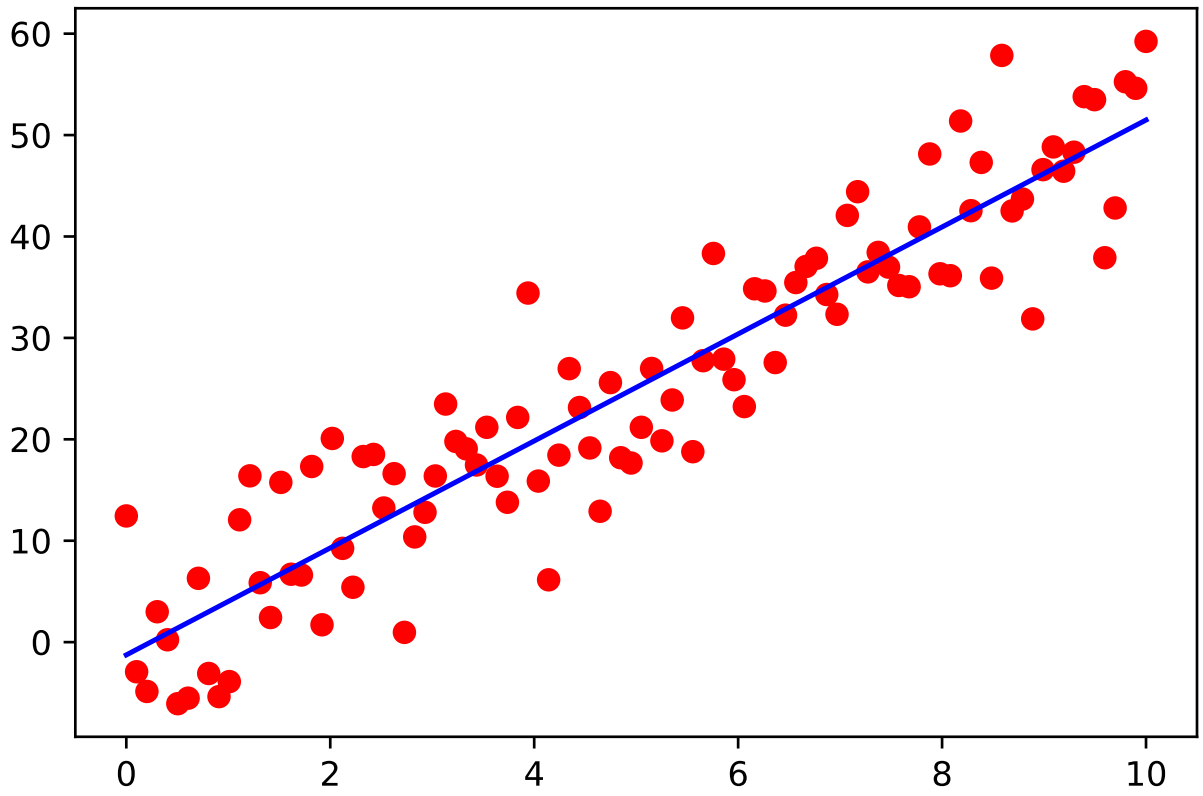
```
(0.0, 0.0)
```

Weitere Infos zum Datei Ein- und Auslesen unter

https://www.python-kurs.eu/numpy_dateien_lesen_schreiben.php

```
# Zeichne Daten
plt.figure()
```

```
plt.plot(x, y, 'or')

A = np.vstack([x, np.ones(len(x))]).T

coeff = npl.solve(A.T@A,A.T@y)

xp = np.linspace(0,10,100)

lsf = coeff[0]*xp+coeff[1]

plt.plot(xp, lsf,'b-')
plt.show()
```



Eine kleine Einführung zu Plots gibt es unter
https://www.python-kurs.eu/matplotlib.php
oder auf der offiziellen matplotlib Seite:
https://matplotlib.org/