lektion4

November 8, 2018

```
In [1]: import numpy as np

In [2]: np.pi

Out[2]: 3.141592653589793

In [3]: A1 = np.array([1,2,3])
        A1

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

Ein eindimensionaler `array` ist wie ein Vektor

```
In [4]: 3*A1

Out[4]: array([3, 6, 9])

In [5]: A1**2

Out[5]: array([1, 4, 9])
```

Ein `array` braucht aber nicht eindimensional zu sein

```
In [6]: A2 = np.array([[1,2,3], [4,5,6]])
        A2

Out[6]: array([[1, 2, 3],
               [4, 5, 6]])

In [7]: A2**2

Out[7]: array([[ 1,  4,  9],
               [16, 25, 36]])
```

Indicierung

```
In [8]: A1

Out[8]: array([1, 2, 3])

In [9]: A1[1]
```

```
Out[9]: 2

In [10]: A2[1:, :]

Out[10]: array([[4, 5, 6]])

In [11]: A2[1:, :].shape

Out[11]: (1, 3)

In [12]: A2.flatten().reshape(2,3)

Out[12]: array([[1, 2, 3],
                [4, 5, 6]])

In [13]: A2[1:, :].flatten()

Out[13]: array([4, 5, 6])

In [14]: A2.reshape(3,2)

Out[14]: array([[1, 2],
                [3, 4],
                [5, 6]])
```

Vordefinierte Arrays

```
In [15]: np.arange(10)

Out[15]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [16]: x = np.linspace(0, 5*np.pi)
         x

Out[16]: array([ 0.        ,  0.32057068,  0.64114136,  0.96171204,  1.28228272,
                 1.60285339,  1.92342407,  2.24399475,  2.56456543,  2.88513611,
                 3.20570679,  3.52627747,  3.84684815,  4.16741883,  4.48798951,
                 4.80856018,  5.12913086,  5.44970154,  5.77027222,  6.0908429 ,
                 6.41141358,  6.73198426,  7.05255494,  7.37312562,  7.69369629,
                 8.01426697,  8.33483765,  8.65540833,  8.97597901,  9.29654969,
                 9.61712037,  9.93769105, 10.25826173, 10.5788324 , 10.89940308,
                11.21997376, 11.54054444, 11.86111512, 12.1816858 , 12.50225648,
                12.82282716, 13.14339784, 13.46396852, 13.78453919, 14.10510987,
                14.42568055, 14.74625123, 15.06682191, 15.38739259, 15.70796327])
```

Das Matrixprodukt erhält man mit '@'

```
In [17]: A3 = np.arange(1,5).reshape(2,2)
         M2 = A3 @ A2
         M2
```

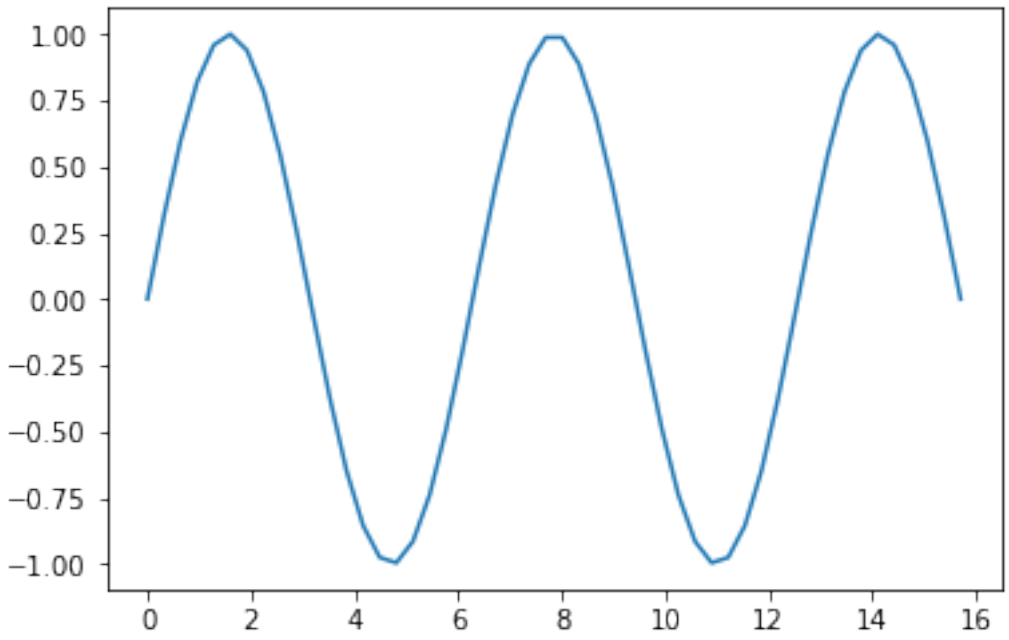
```
Out[17]: array([[ 9, 12, 15],
                [19, 26, 33]])

In [18]: A1

Out[18]: array([1, 2, 3])

In [19]: A2 @ A1

Out[19]: array([14, 32])
```

universal functions

```
In [20]: np.sin(x)

Out[20]: array([ 0.00000000e+00,  3.15108218e-01,  5.98110530e-01,  8.20172255e-01,
                 9.58667853e-01,  9.99486216e-01,  9.38468422e-01,  7.81831482e-01,
                 5.45534901e-01,  2.53654584e-01, -6.40702200e-02, -3.75267005e-01,
                -6.48228395e-01, -8.55142763e-01, -9.74927912e-01, -9.95379113e-01,
                -9.14412623e-01, -7.40277997e-01, -4.90717552e-01, -1.91158629e-01,
                 1.27877162e-01,  4.33883739e-01,  6.95682551e-01,  8.86599306e-01,
                 9.87181783e-01,  9.87181783e-01,  8.86599306e-01,  6.95682551e-01,
                 4.33883739e-01,  1.27877162e-01, -1.91158629e-01, -4.90717552e-01,
                -7.40277997e-01, -9.14412623e-01, -9.95379113e-01, -9.74927912e-01,
                -8.55142763e-01, -6.48228395e-01, -3.75267005e-01, -6.40702200e-02,
                 2.53654584e-01,  5.45534901e-01,  7.81831482e-01,  9.38468422e-01,
                 9.99486216e-01,  9.58667853e-01,  8.20172255e-01,  5.98110530e-01,
                 3.15108218e-01,  6.12323400e-16])

In [21]: %%timeit
         y = np.sin(x)

1.18 ţs ś 5.32 ns per loop (mean ś std. dev. of 7 runs, 1000000 loops each)


In [22]: %%timeit
         l = []
         for xx in x:
             l.append(np.sin(xx))
         y = np.array(l)

49.7 ţs ś 1.29 ţs per loop (mean ś std. dev. of 7 runs, 10000 loops each)
```

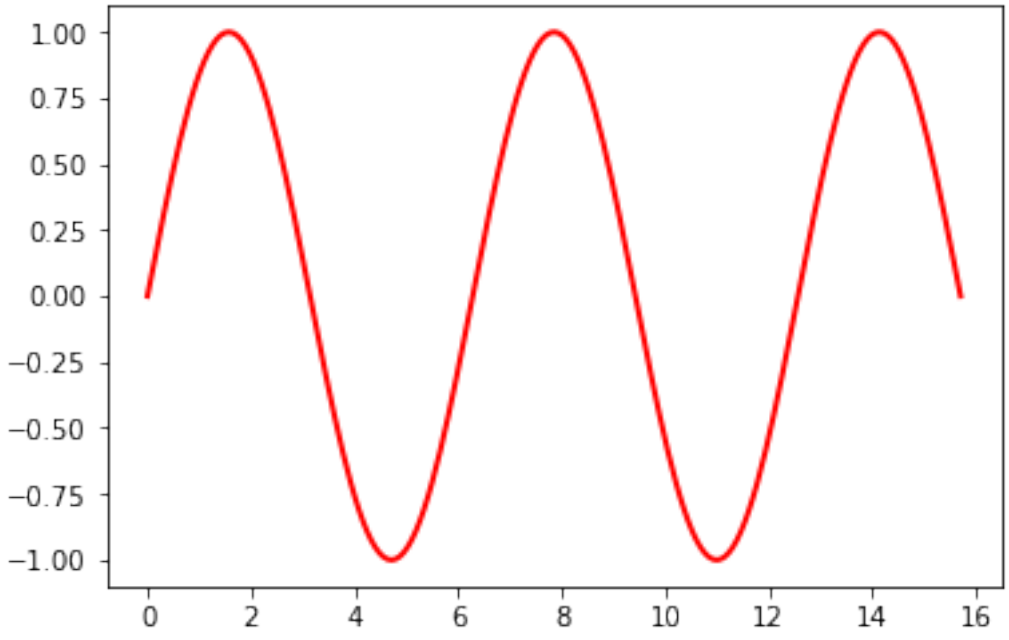
Plots

```
In [23]: import matplotlib.pyplot as plt
         %matplotlib inline

In [24]: y = np.sin(x)
         plt.plot(x, y) ;
```



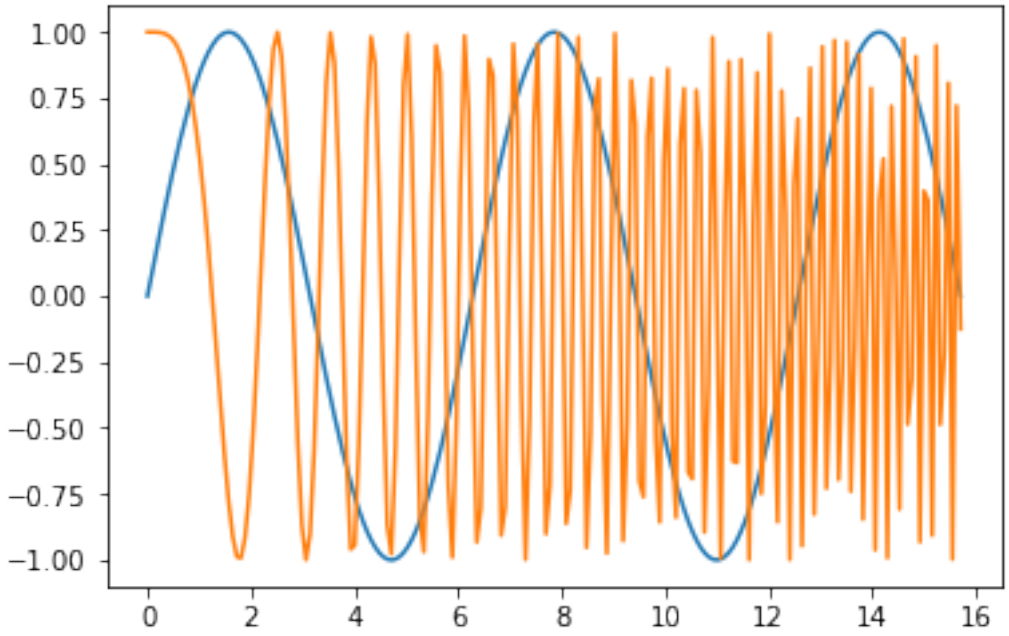
Das ist zu zackelig und die Striche sind zu dünn

```
In [25]: x = np.linspace(0, 5*np.pi, 201)
         y = np.sin(x)
         plt.plot(x, y, 'r', linewidth=2);
```



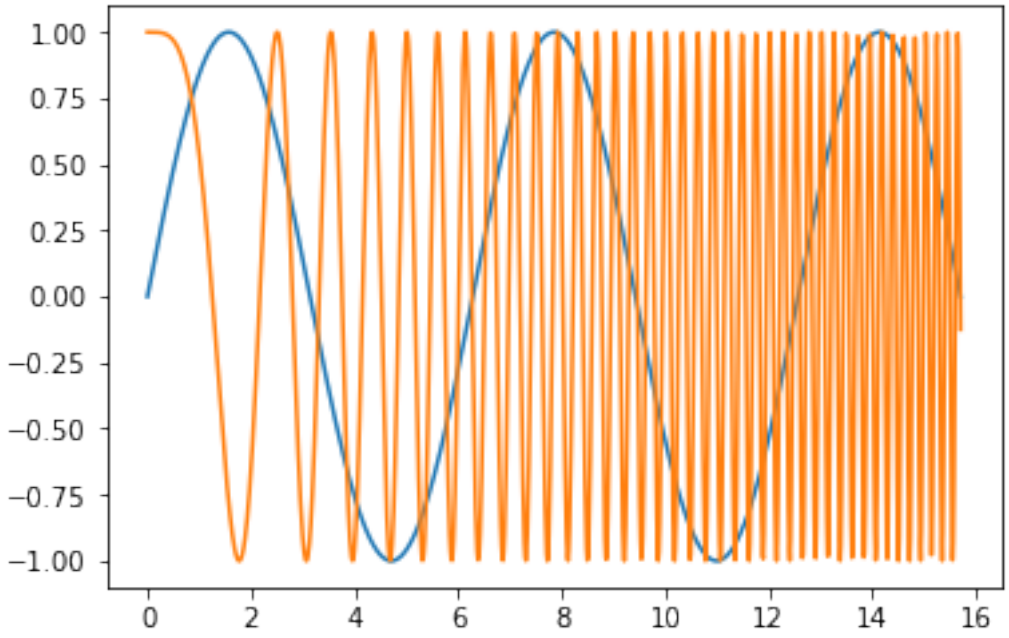
Mehrere Graphen in einem Bild

```
In [26]: z = np.cos(x**2)
         plt.plot(x, y)
         plt.plot(x, z);
```



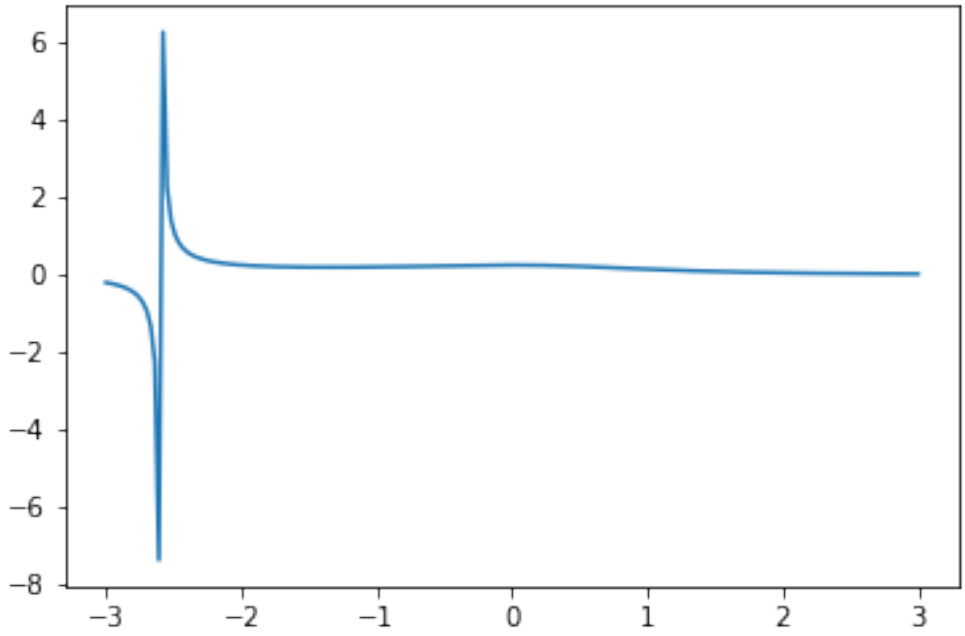
Hier gibt es nicht genug Punkte

```
In [27]: x = np.linspace(0, 5*np.pi, 1101)
         y = np.sin(x)
         z = np.cos(x**2)
         plt.plot(x, y)
         plt.plot(x, z);
```



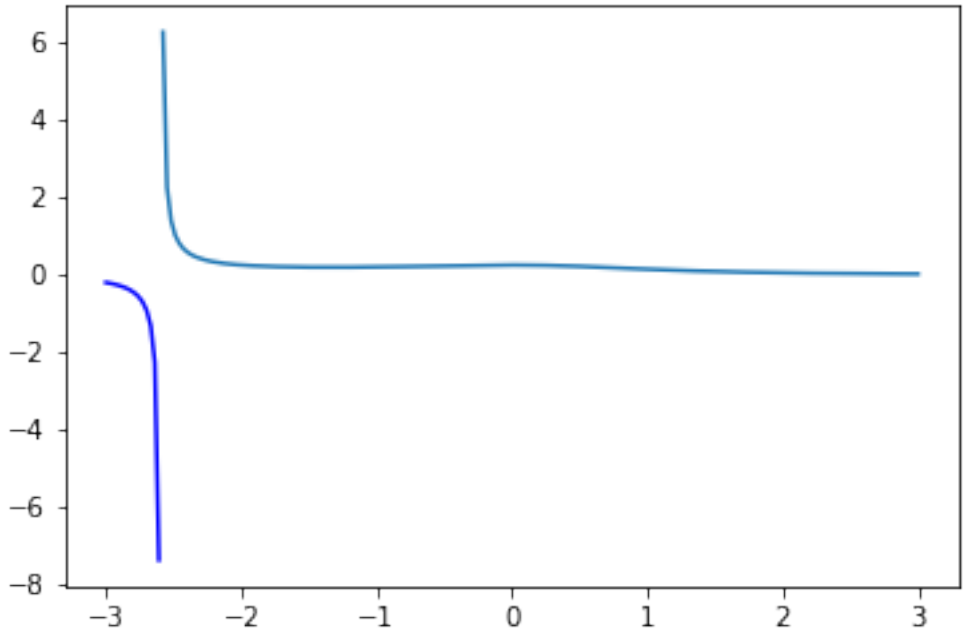
Slicing

```
In [28]: x = np.linspace(-3, 3, 200)
         y = 1/(x**3 + 2*x**2 + 4)
         plt.plot(x, y);
```



Der Sprung von $-\infty$ nach ∞ muss weg:

```
In [29]: b = y>0
         c = y<0
         plt.plot(x[b], y[b])
         plt.plot(x[c], y[c], 'b');
```

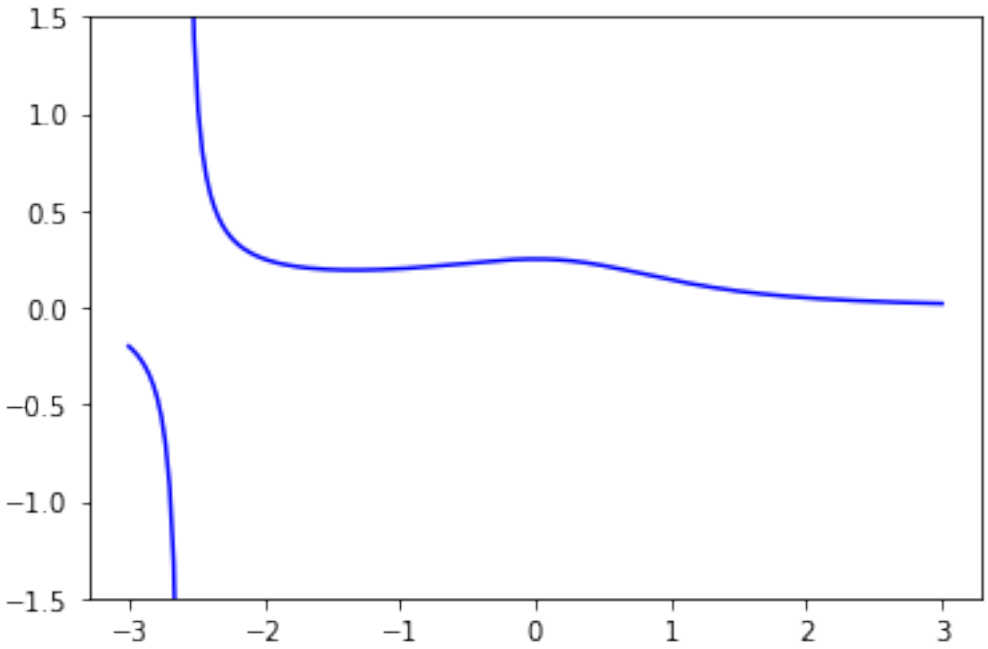


Farben als Kürzel (Alternativen später)

b	g	r	c	m	y	k	w
blue	green	red	cyan	magenta	yellow	black	white

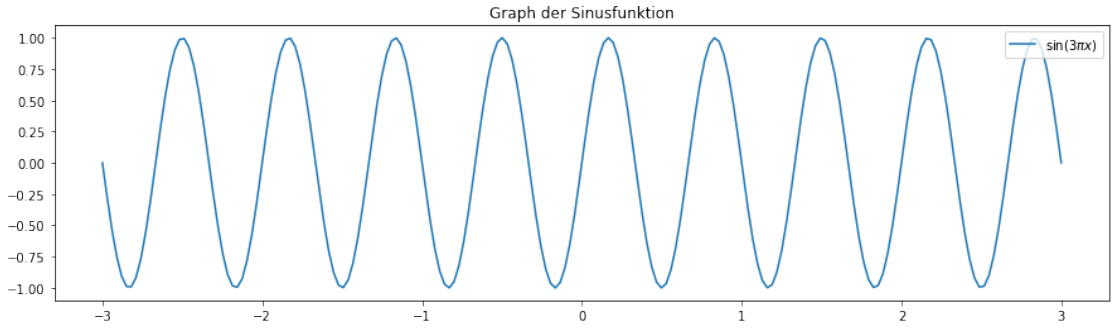
Oben und unten abschneiden

```
In [30]: b = y>0
         c = y<0
         plt.plot(x[b], y[b], 'b')
         plt.plot(x[c], y[c], 'b')
         plt.axis(ymin=-1.5, ymax=1.5);
```



GröSSe ändern, Überschrift

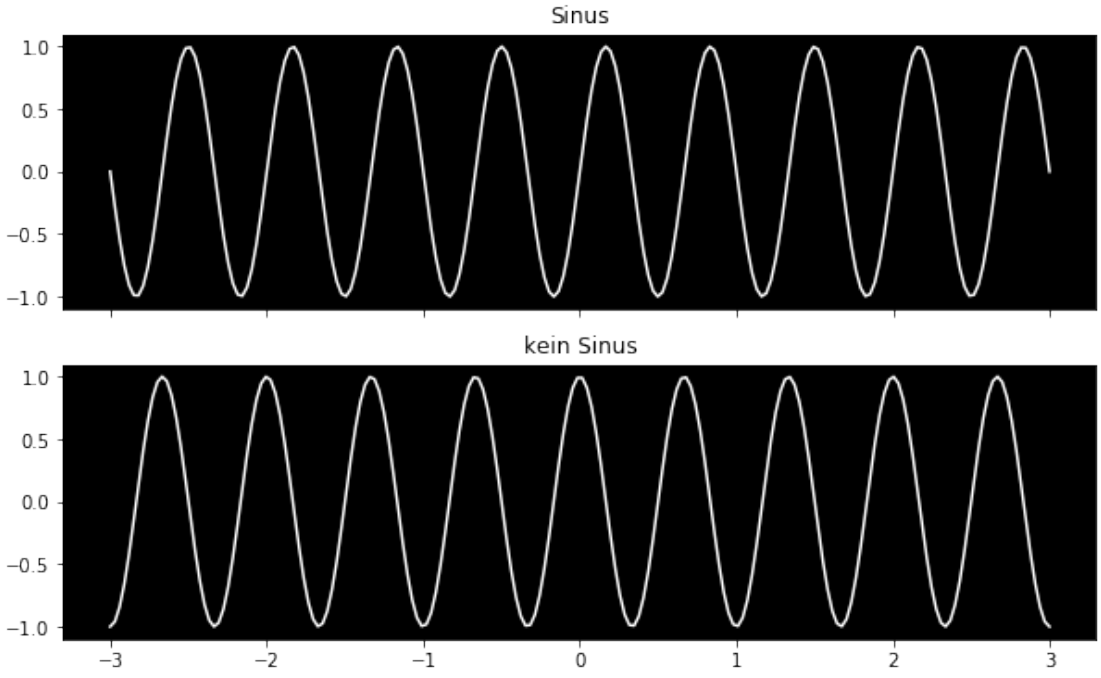
```
In [31]: plt.figure(None, (15, 4))
         plt.plot(x, np.sin(3*np.pi*x), label='$\sin(3\pi x)$')
         plt.title("Graph der Sinusfunktion")
         plt.legend(loc="upper right");
```



Dasselbe objektorientiert

```
In [32]: fig = plt.figure(None, (10,6))
         ax1, ax2 = fig.subplots(2,1, sharex=True)
         ax1.plot(x, np.sin(3*np.pi*x), 'w')
         ax2.plot(x, np.cos(3*np.pi*x), 'w')
         ax1.set_title("Sinus")
```

```
    ax2.set_title("kein Sinus")
    for a in [ax1, ax2]:
        a.set_facecolor('k')
```



In [ ]: