بسم الله الرحمن الرحيم

High Performance Multiprocessor Systems

Lecture 5:
Python
part2

Pandas

matplotlib

SciPy

https://matplotlib.org/

Getting started with Python for science

- Matplotlib: plotting
  1. Simple plot
  2. Other Types of Plots: examples and exercises

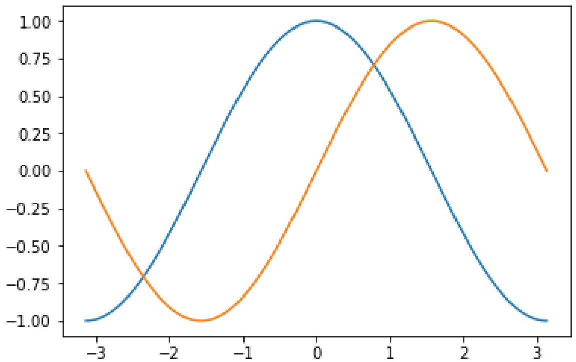
matplotlib https://matplotlib.org/ Self Study

Getting started with Python for science

- Matplotlib: plotting
  1. Simple plot

Plotting with default settings

```
In [13]: from matplotlib import pyplot as plt

In [14]: import numpy as np

In [15]: X = np.linspace(-np.pi, np.pi, 256, endpoint=True)

In [16]: C, S = np.cos(X), np.sin(X)

In [17]: plt.plot(X, C);plt.plot(X, S);plt.show()
```



https://matplotlib.org/

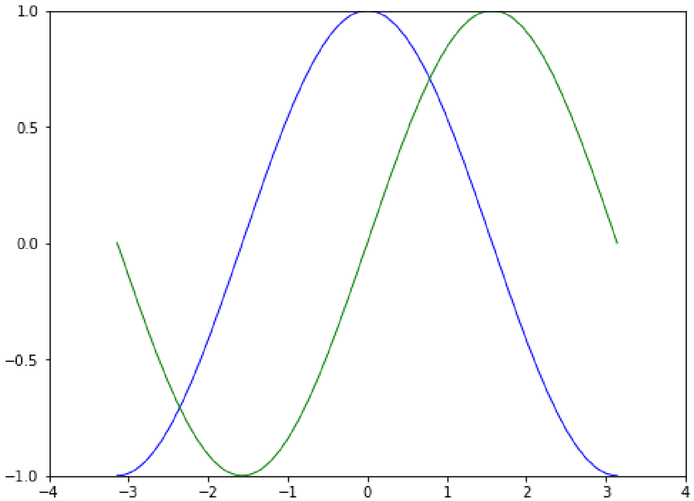
Getting started with Python for science

Matplotlib: plotting

1. Simple plot

Instantiating defaults

```python
import numpy as np
import matplotlib.pyplot as plt
# Create a figure of size 8x6 inches, 80 dots per inch
plt.figure(figsize=(8, 6), dpi=80)
# Create a new subplot from a grid of 1x1
plt.subplot(1, 1, 1)
X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
C, S = np.cos(X), np.sin(X)
# Plot cosine with a blue continuous line of width 1 (pixels)
plt.plot(X, C, color="blue", linewidth=1.0, linestyle="-")
# Plot sine with a green continuous line of width 1 (pixels)
plt.plot(X, S, color="green", linewidth=1.0, linestyle="-")
# Set x limits
plt.xlim(-4.0, 4.0)
# Set x ticks
plt.xticks(np.linspace(-4, 4, 9, endpoint=True))
# Set y limits
plt.ylim(-1.0, 1.0)
# Set y ticks
plt.yticks(np.linspace(-1, 1, 5, endpoint=True))
# Save figure using 72 dots per inch
# plt.savefig("exercice_2.png", dpi=72)
# Show result on screen
plt.show()
```



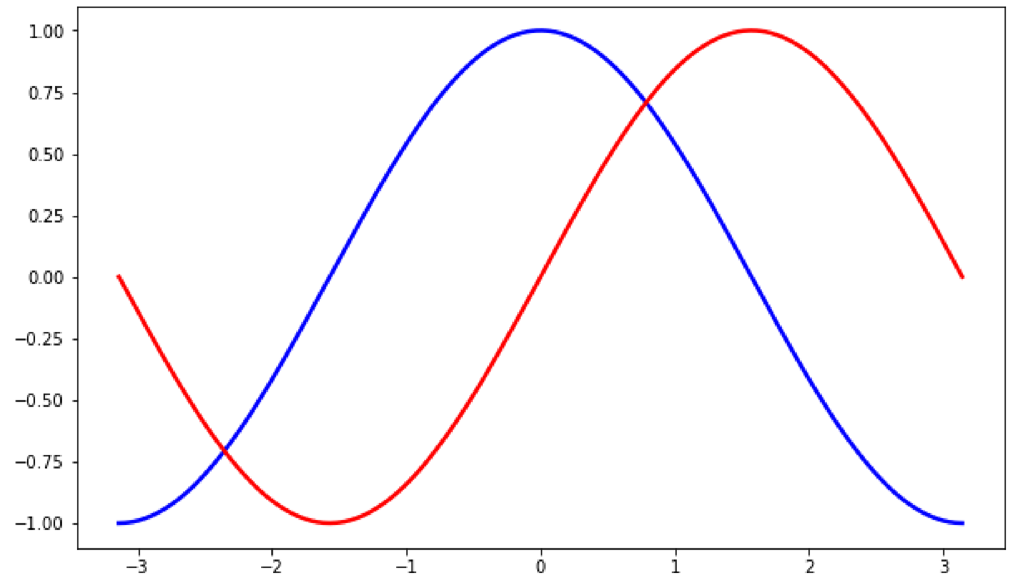
https://matplotlib.org/

Getting started with Python for science

- Matplotlib: plotting
  1. Simple plot

Changing colors and line widths

```
In [19]: plt.figure(figsize=(10, 6), dpi=80)
    ...: plt.plot(X, C, color="blue", linewidth=2.5, linestyle="-")
    ...: plt.plot(X, S, color="red", linewidth=2.5, linestyle="-")
Out[19]: [<matplotlib.lines.Line2D at 0xddda3f9588>]
```



matplotlib

https://matplotlib.org/

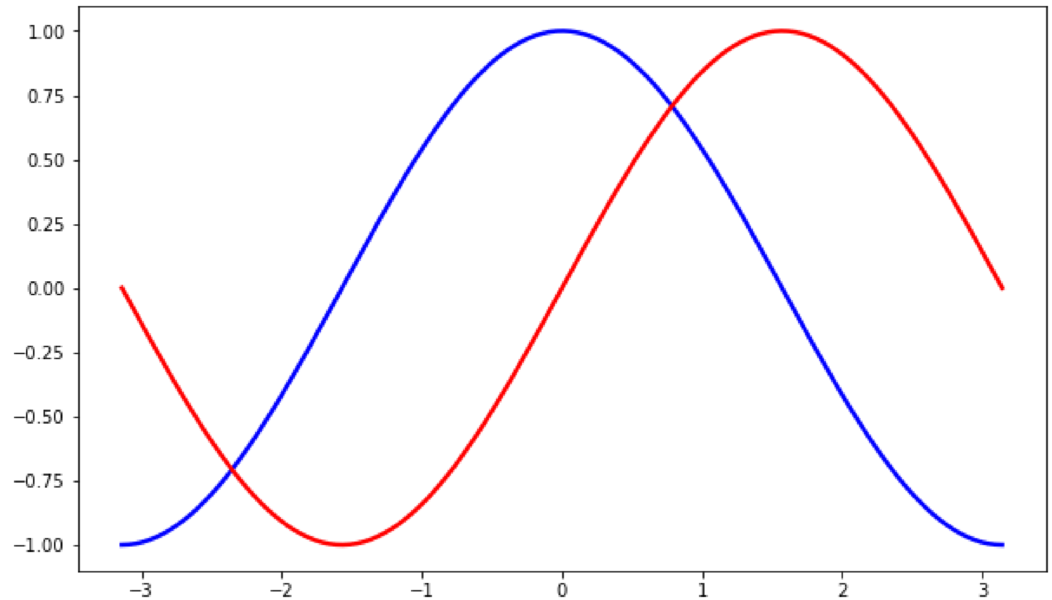
Self Study

Getting started with Python for science

- Matplotlib: plotting
  1. Simple plot

     Setting limits

```
In [23]: plt.figure(figsize=(10, 6), dpi=80)
    ...: plt.plot(X, C, color="blue", linewidth=2.5, linestyle="-")
    ...: plt.plot(X, S, color="red", linewidth=2.5, linestyle="-")
    ...:
    ...: plt.xlim(X.min() * 1.1, X.max() * 1.1)
    ...: plt.ylim(C.min() * 1.1, C.max() * 1.1)
Out[23]: (-1.1000000000000001, 1.0999165211263138)
```



https://matplotlib.org/

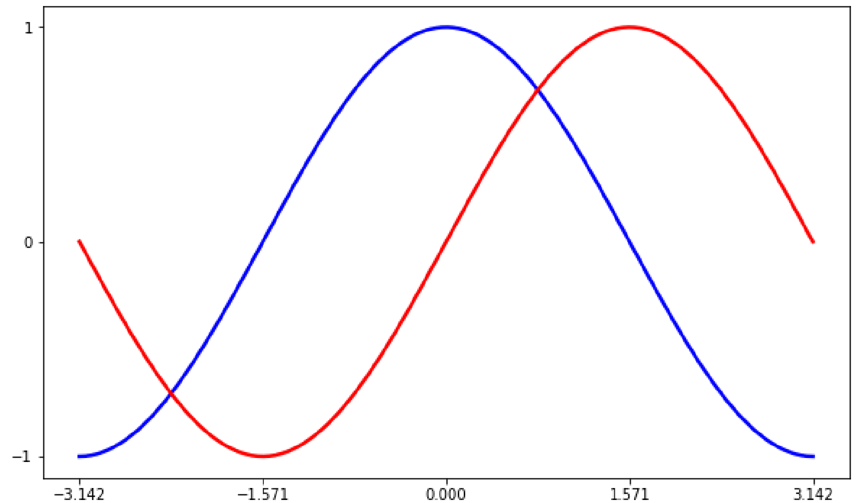
Getting started with Python for science

- Matplotlib: plotting
  1. Simple plot

Setting ticks

```
plt.figure(figsize=(10, 6), dpi=80)
plt.plot(X, C, color="blue", linewidth=2.5, linestyle="-")
plt.plot(X, S, color="red", linewidth=2.5, linestyle="-")
plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi])
plt.yticks([-1, 0, +1])
```

```
Out[24]:
([<matplotlib.axis.YTick at 0xddda953c50>,
  <matplotlib.axis.YTick at 0xddda86b240>,
  <matplotlib.axis.YTick at 0xddda98fb00>],
 <a list of 3 Text yticklabel objects>)
```



matplotlib https://matplotlib.org/ Self Study

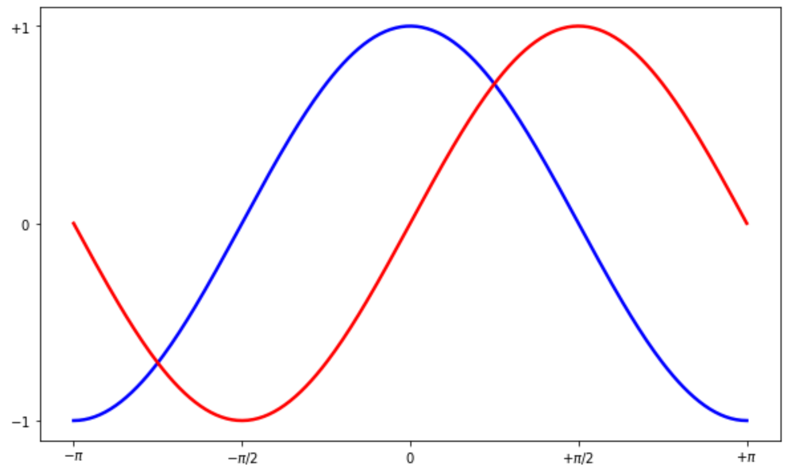
Getting started with Python for science

- Matplotlib: plotting
  1. Simple plot

Setting tick labels

```
In [25]: plt.figure(figsize=(10, 6), dpi=80)
    ...: plt.plot(X, C, color="blue", linewidth=2.5, linestyle="-")
    ...: plt.plot(X, S, color="red", linewidth=2.5, linestyle="-")
    ...: plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi],
    ...: [r'$-\pi$', r'$-\pi/2$', r'$0$', r'$+\pi/2$', r'$+\pi$'])
    ...: plt.yticks([-1, 0, +1],
    ...: [r'$-1$', r'$0$', r'$+1$'])
```

```
Out[25]:
([<matplotlib.axis.YTick at 0xddda9ad2b0>,
  <matplotlib.axis.YTick at 0xddda9a05f8>,
  <matplotlib.axis.YTick at 0xddda9a6b70>],
 <a list of 3 Text yticklabel objects>)
```



https://matplotlib.org/

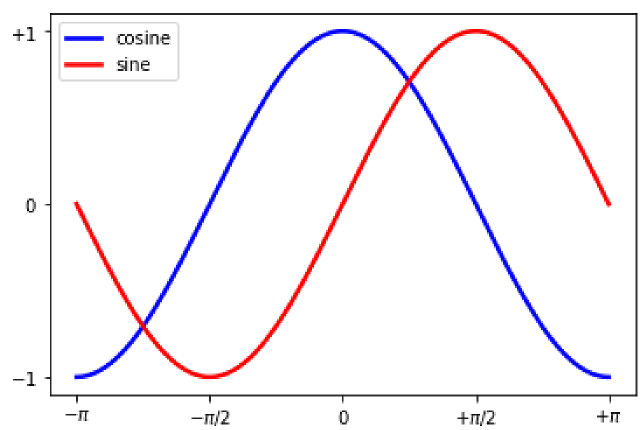
Getting started with Python for science

- Matplotlib: plotting
  1. Simple plot

Adding a legend

```
In [27]: plt.plot(X, C, color="blue", linewidth=2.5, linestyle="-", label="cosine")
    ...: plt.plot(X, S, color="red", linewidth=2.5, linestyle="-", label="sine")
    ...: plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi],
    ...: [r'$-\pi$', r'$-\pi/2$', r'$0$', r'$+\pi/2$', r'$+\pi$'])
    ...: plt.yticks([-1, 0, +1],
    ...: [r'$-1$', r'$0$', r'$+1$'])
    ...: plt.legend(loc='upper left')
```

```
Out[27]: <matplotlib.legend.Legend at 0xddda8a3f60>
```



matplotlib https://matplotlib.org/

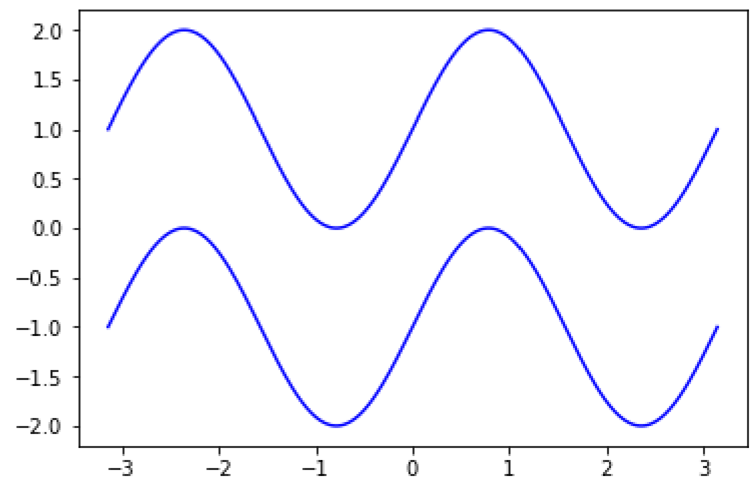
Self Study

Getting started with Python for science

- Matplotlib: plotting
  2. Other Types of Plots: examples and exercises

Regular Plots

```
In [32]: n = 256
    ...: X = np.linspace(-np.pi, np.pi, n, endpoint=True)
    ...: Y = np.sin(2 * X)
    ...: plt.plot(X, Y + 1, color='blue', alpha=1.00)
    ...: plt.plot(X, Y - 1, color='blue', alpha=1.00)
Out[32]: [<matplotlib.lines.Line2D at 0xdddbf0c5f8>]
```



Self Study

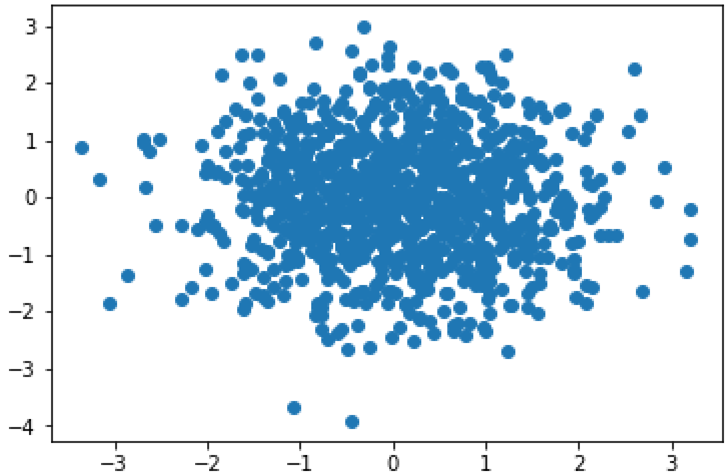
Getting started with Python for science

- Matplotlib: plotting

2. Other Types of Plots: examples and exercises

Scatter Plots

```
In [33]: n = 1024
    ...: X = np.random.normal(0,1,n)
    ...: Y = np.random.normal(0,1,n)
    ...: plt.scatter(X,Y)
    ...:
Out[33]: <matplotlib.collections.PathCollection at 0xdddab4b5c0>
```



https://matplotlib.org/

Self Study

Getting started with Python for science

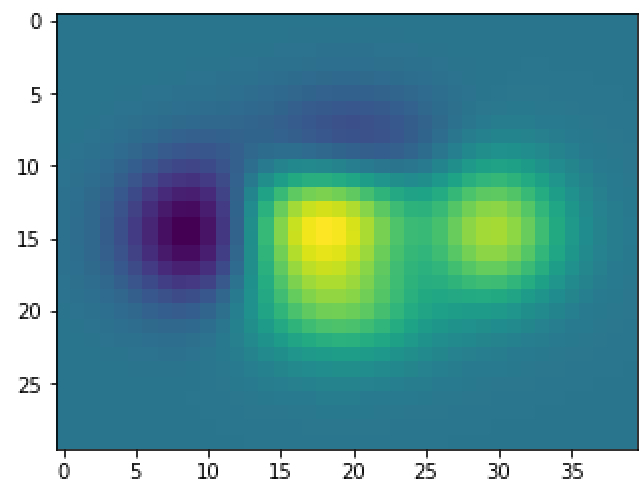
- Matplotlib: plotting

2. Other Types of Plots: examples and exercises

Imshow

```
In [36]: def f(x, y):
    ...:     return (1 - x / 2 + x ** 5 + y ** 3) * np.exp(-x ** 2 - y ** 2)
    ...:
    ...: n = 10
    ...: x = np.linspace(-3, 3, 4 * n)
    ...: y = np.linspace(-3, 3, 3 * n)
    ...: X, Y = np.meshgrid(x, y)
    ...: plt.imshow(f(X, Y))
```

```
Out[36]: <matplotlib.image.AxesImage at 0xdddbe4b358>
```



https://matplotlib.org/

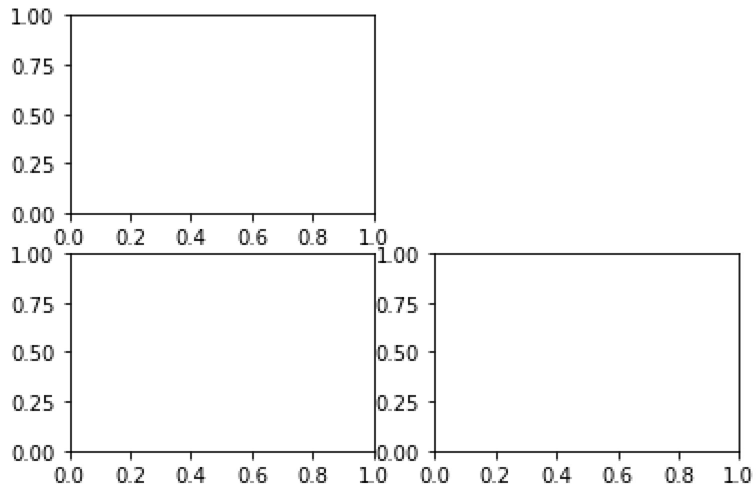
Self Study

Getting started with Python for science

- Matplotlib: plotting
  2. Other Types of Plots: examples and exercises

Multi Plots

```
In [38]: plt.subplot(2, 2, 1)
    ...: plt.subplot(2, 2, 3)
    ...: plt.subplot(2, 2, 4)
Out[38]: <matplotlib.axes._subplots.AxesSubplot at 0xdddd023a58>
```



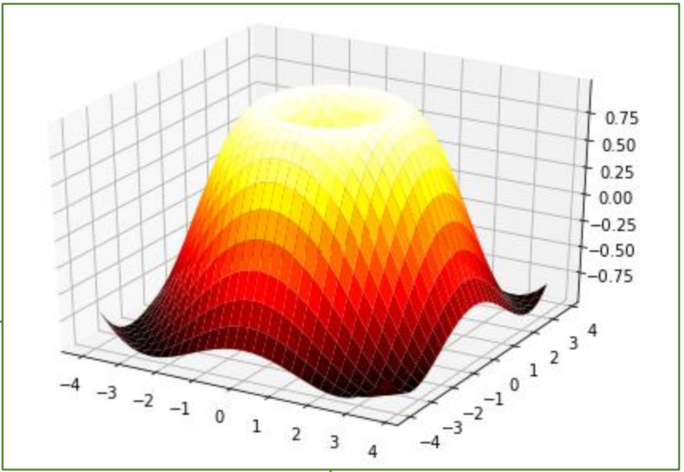
https://matplotlib.org/

Self Study

Getting started with Python for science

- Matplotlib: plotting
  2. Other Types of Plots: examples and exercises

3D Plots




```
In [41]: fig = plt.figure()
    ...: ax = Axes3D(fig)
    ...: X = np.arange(-4, 4, 0.25)
    ...: Y = np.arange(-4, 4, 0.25)
    ...: X, Y = np.meshgrid(X, Y)
    ...: R = np.sqrt(X**2 + Y**2)
    ...: Z = np.sin(R)
    ...: ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap='hot')
```

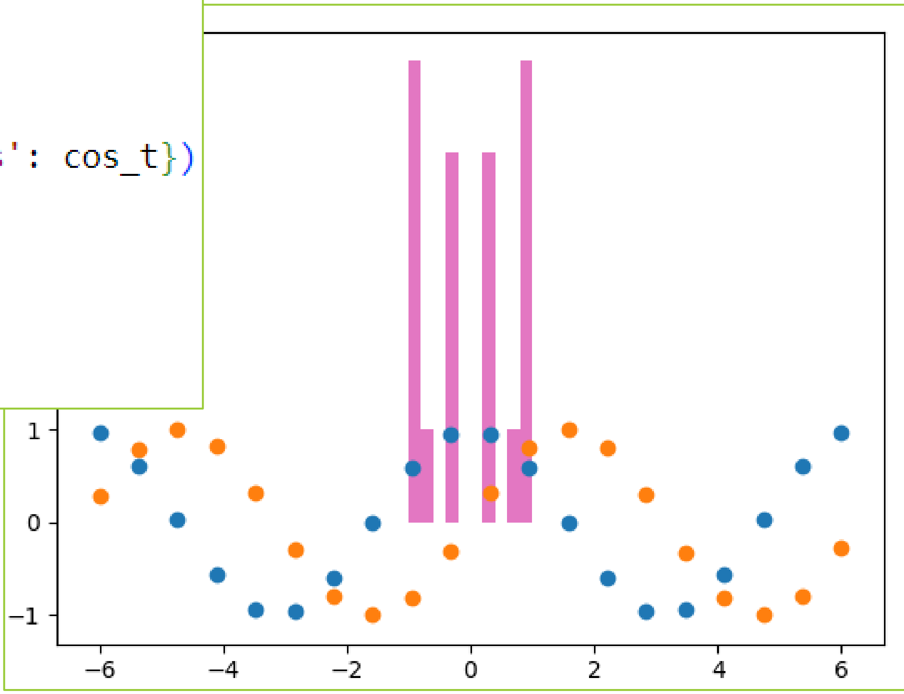
https://matplotlib.org/

Self Study

Getting started with Python for science

- Matplotlib: plotting

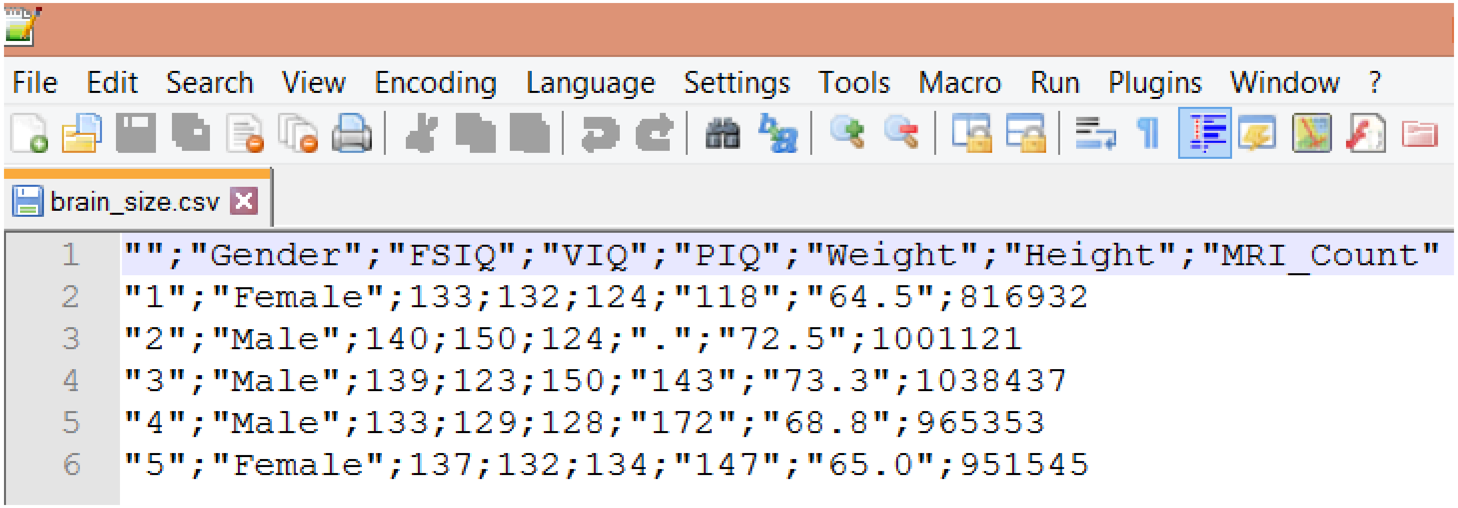
```
import numpy as np
import matplotlib.pyplot as plt
import pandas
t = np.linspace(-6, 6, 20)
sin_t = np.sin(t)
cos_t = np.cos(t)
data1 = pandas.DataFrame({'t': t, 'sin': sin_t, 'cos': cos_t})
plt.figure()
plt.scatter(data1['t'], data1['cos'], zorder=3)
plt.scatter(data1['t'], data1['sin'], zorder=2)
plt.hist(data1['sin'],color='tab:pink',zorder=1)
plt.show()
```



Pandas

Data representation and interaction

brain_size.csv




```
"";"Gender";"FSIQ";"VIQ";"PIQ";"Weight";"Height";"MRI_Count"
"1";"Female";133;132;124;"118";"64.5";816932
"2";"Male";140;150;124;".";"72.5";1001121
"3";"Male";139;123;150;"143";"73.3";1038437
"4";"Male";133;129;128;"172";"68.8";965353
"5";"Female";137;132;134;"147";"65.0";951545
```

Pandas

Data representation and interaction

- **The panda data-frame**

```
1    import pandas
2    data = pandas.read_csv('brain_size.csv')
```

1

```
>>> data
  ;"Gender";"FSIQ";"VIQ";"PIQ";"Weight";"Height";"MRI_Count"
0         1;"Female";133;132;124;"118";"64.5";816932
1            2;"Male";140;150;124;".";"72.5";1001121
2          3;"Male";139;123;150;"143";"73.3";1038437
3           4;"Male";133;129;128;"172";"68.8";965353
4         5;"Female";137;132;134;"147";"65.0";951545
```

2

```
>>> data = pandas.read_csv('brain_size.csv',sep=';')
>>> data
   Unnamed: 0  Gender  FSIQ  VIQ  PIQ Weight  Height  MRI_Count
0           1  Female   133  132  124    118    64.5     816932
1           2    Male   140  150  124      .    72.5    1001121
2           3    Male   139  123  150    143    73.3    1038437
3           4    Male   133  129  128    172    68.8     965353
4           5  Female   137  132  134    147    65.0     951545
```

Pandas

Data representation and interaction

➢ The panda data-frame

```
1    import pandas
2    data = pandas.read_csv('brain_size.csv')
```

3

```
>>> data = pandas.read_csv('brain_size.csv',sep=';',na_values='.')
>>> data
   Unnamed: 0  Gender  FSIQ  VIQ  PIQ  Weight  Height  MRI_Count
0           1  Female   133  132  124   118.0    64.5     816932
1           2    Male   140  150  124     NaN    72.5    1001121
2           3    Male   139  123  150   143.0    73.3    1038437
3           4    Male   133  129  128   172.0    68.8     965353
4           5  Female   137  132  134   147.0    65.0     951545
```

```
>>> type(data)
<class 'pandas.core.frame.DataFrame'>
```

Pandas

Data representation and interaction

➢ The panda data-frame (Creating from arrays)

- A **pandas.DataFrame** can also be seen as a dictionary of 1D arrays or lists.

code3.py > ...

```python
import pandas
import numpy as np
t = np.linspace(-6, 6, 20)
sin_t = np.sin(t)
cos_t = np.cos(t)
data1 = pandas.DataFrame({'t': t, 'sin': sin_t, 'cos': cos_t})
pandas.DataFrame.to_csv(data1,'data1_frame.csv', sep = ',')
data2 = pandas.read_csv('data1_frame.csv', sep = ',')
```

Pandas

Data representation and interaction

The panda data-frame (Creating from arrays)

- A **pandas.DataFrame** can also be seen as a dictionary of 1D arrays or lists.

```
>>> data1
         cos       sin         t
0   0.960170  0.279415 -6.000000
1   0.609977  0.792419 -5.368421
2   0.024451  0.999701 -4.736842
3  -0.570509  0.821291 -4.105263
4  -0.945363  0.326021 -3.473684
5  -0.955488 -0.295030 -2.842105
6  -0.596979 -0.802257 -2.210526
7  -0.008151 -0.999967 -1.578947
8   0.583822 -0.811882 -0.947368
9   0.950551 -0.310567 -0.315789
10  0.950551  0.310567  0.315789
11  0.583822  0.811882  0.947368
12 -0.008151  0.999967  1.578947
13 -0.596979  0.802257  2.210526
14 -0.955488  0.295030  2.842105
15 -0.945363 -0.326021  3.473684
16 -0.570509 -0.821291  4.105263
17  0.024451 -0.999701  4.736842
18  0.609977 -0.792419  5.368421
19  0.960170 -0.279415  6.000000
```

```
>>> data2
    Unnamed: 0       cos       sin         t
0            0  0.960170  0.279415 -6.000000
1            1  0.609977  0.792419 -5.368421
2            2  0.024451  0.999701 -4.736842
3            3 -0.570509  0.821291 -4.105263
4            4 -0.945363  0.326021 -3.473684
5            5 -0.955488 -0.295030 -2.842105
6            6 -0.596979 -0.802257 -2.210526
7            7 -0.008151 -0.999967 -1.578947
8            8  0.583822 -0.811882 -0.947368
9            9  0.950551 -0.310567 -0.315789
10          10  0.950551  0.310567  0.315789
11          11  0.583822  0.811882  0.947368
12          12 -0.008151  0.999967  1.578947
13          13 -0.596979  0.802257  2.210526
14          14 -0.955488  0.295030  2.842105
15          15 -0.945363 -0.326021  3.473684
16          16 -0.570509 -0.821291  4.105263
17          17  0.024451 -0.999701  4.736842
18          18  0.609977 -0.792419  5.368421
19          19  0.960170 -0.279415  6.000000
```

Pandas

Data representation and interaction

The panda data-frame (Creating from arrays)

- A **pandas.DataFrame** can also be seen as a dictionary of 1D arrays or lists.

1

```
>>> data1.shape
(20, 3)
>>> data1.columns
Index(['cos', 'sin', 't'], dtype='object')
```

2

```
>>> data1['t']
0    -6.000000
1    -5.368421
2    -4.736842
3    -4.105263
4    -3.473684
5    -2.842105
6    -2.210526
7    -1.578947
8    -0.947368
9    -0.315789
10    0.315789
11    0.947368
12    1.578947
13    2.210526
14    2.842105
15    3.473684
16    4.105263
17    4.736842
18    5.368421
19    6.000000
Name: t, dtype: float64
```

3

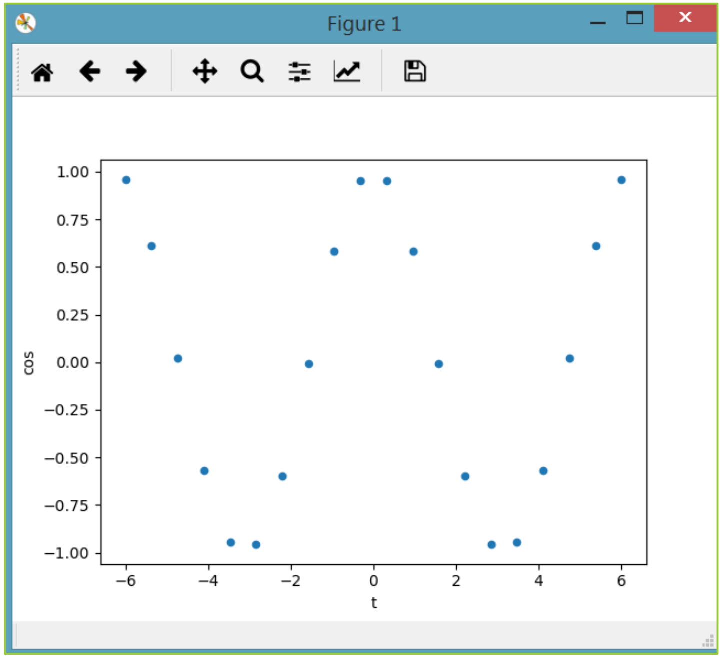
```
11    data1['t'].
              abs
              add
              add_prefix
              add_suffix
              agg
              aggregate
              align
              all
              any
              apply
              argmax
              argmin
```

Data representation and interaction

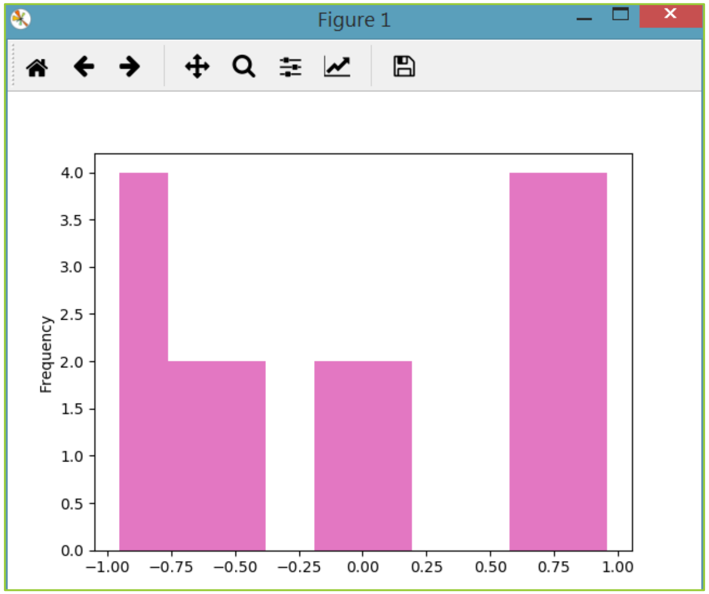
➢ The panda data-frame (Creating from arrays)

- A **pandas.DataFrame** can also be seen as a dictionary of 1D arrays or lists.

```
data1.plot(kind = 'scatter', x = 't', y = 'cos')
plt.show()
```




```
data1['cos'].plot(kind = 'hist',color='tab:pink')
plt.show()
```



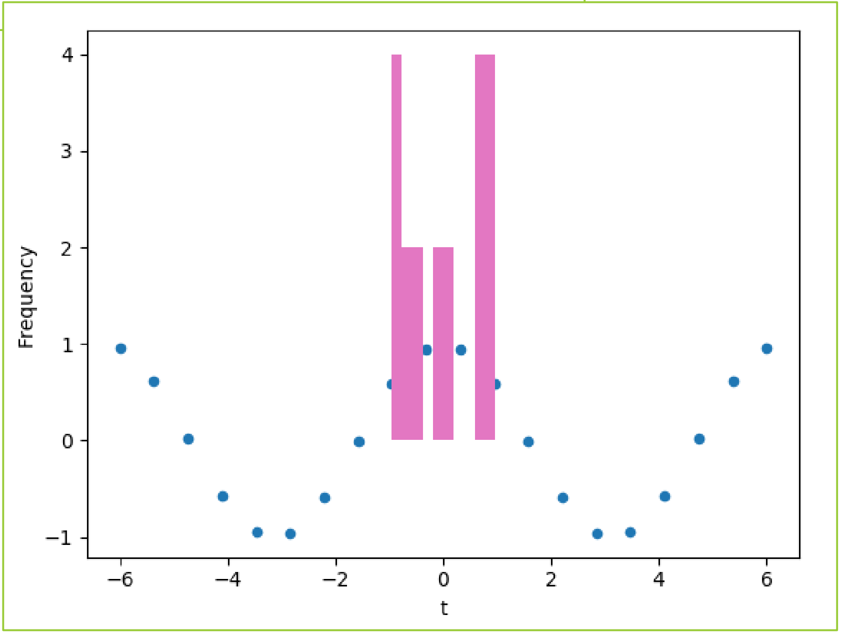
Pandas

Data representation and interaction

The panda data-frame (Creating from arrays)

- A **pandas.DataFrame** can also be seen as a dictionary of 1D arrays or lists.

```
data1.plot(kind = 'scatter', x = 't', y = 'cos')
data1['cos'].plot(kind = 'hist',color='tab:pink')
plt.show()
```

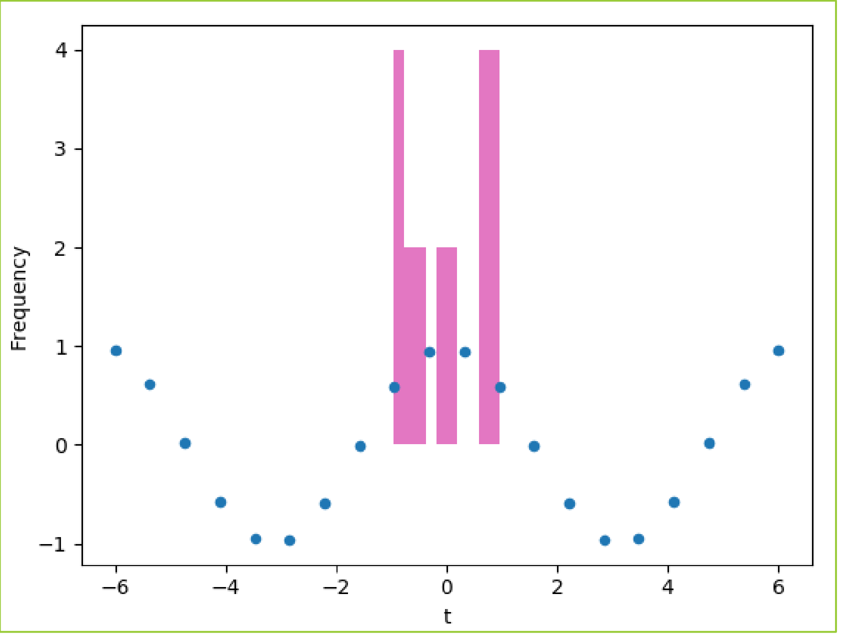


Data representation and interaction

The panda data-frame (Creating from arrays)

- A **pandas.DataFrame** can also be seen as a dictionary of 1D arrays or lists.

```
data1.plot(kind = 'scatter', x = 't', y = 'cos',zorder=2)
data1['cos'].plot(kind = 'hist',color='tab:pink',zorder=1)
plt.show()
```



Getting started with Python for science

- Scipy : high-level scientific computing
  1. File input/output: scipy.io
  2. Linear algebra operations: scipy.linalg
  3. Fast Fourier transforms: scipy.fftpack
  4. Optimization and fit: scipy.optimize
  5. Statistics and random numbers: scipy.stats
  6. Interpolation: scipy.interpolate
  7. Numerical integration: scipy.integrate
  8. Signal processing: scipy.signal
  9. Image processing: scipy.ndimage

Getting started with Python for science

- Scipy : high-level scientific computing
  1. File input/output: scipy.io

Loading and saving matlab files:

```
In [44]: from scipy import io as spio
    ...: a = np.ones((3, 3))
    ...: spio.savemat('E:/gpu/presentations/final/data/\
    ...: file.mat', {'a': a}) # savemat expects a dictionary
    ...: data = spio.loadmat('E:/gpu/presentations/final/\
    ...: data/file.mat', struct_as_record=True)
    ...: data['a']
    ...:
Out[44]:
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
```

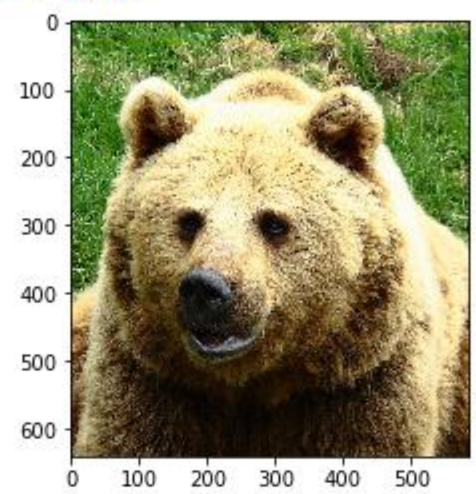
Getting started with Python for science

- Scipy : high-level scientific computing
  1. File input/output: scipy.io

Reading images:

```
In [49]: from scipy import misc
    ...: m1 = misc.imread('E:/gpu/presentations/final/data/images/img2.jpg')
    ...: type(m1)
Out[49]: numpy.ndarray

In [50]: m1.shape
Out[50]: (640, 586, 3)

In [51]: plt.imshow(m1)
Out[51]: <matplotlib.image.AxesImage at 0xdddd42ec18>
```

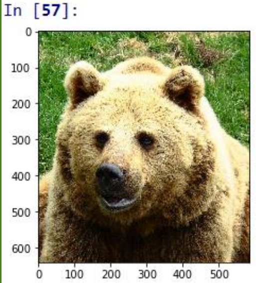
In [52]:



Matplotlib also has a similar function

```
In [54]: import matplotlib.pyplot as plt
    ...: m2 = plt.imread('E:/gpu/presentations/final/data/images/img2.jpg')
    ...: type(m2)
Out[54]: numpy.ndarray

In [55]: m2.shape
Out[55]: (640, 586, 3)

In [56]: plt.imshow(m2)
Out[56]: <matplotlib.image.AxesImage at 0xdddd495d68>
```

In [57]:



Getting started with Python for science

- Scipy : high-level scientific computing
  2. Linear algebra operations: scipy.linalg

The scipy.linalg.det() function computes the determinant of a square matrix:

```
from scipy import linalg
    ...: arr = np.array([[1, 2],
    ...:                 [3, 4]])

In [58]: linalg.det(arr)
Out[58]: -2.0

In [59]: np.linalg.det(arr)

In [59]: Out[59]: -2.0000000000000004

In [60]: arr = np.array([[3, 2],
    ...:                 [6, 4]])
    ...: linalg.det(arr)

In [60]: Out[60]: 6.661338147750939e-16
```

SciPy

Getting started with Python for science

- Scipy : high-level scientific computing

  2. Linear algebra operations: scipy.linalg

The scipy.linalg.inv() function computes the inverse of a squarematrix:

```
In [61]: arr = np.array([[1, 2],
    ...:                 [3, 4]])
    ...: linalg.inv(arr)
Out[61]:
array([[-2. ,  1. ],
       [ 1.5, -0.5]])
```

Other functions

```
In [62]: linalg.
         linalg.special_matrices
         linalg.sqrtm
         linalg.svd
         linalg.svdvals
         linalg.tanhm
         linalg.tanm
         linalg.test
         linalg.Tester
         linalg.toeplitz
```

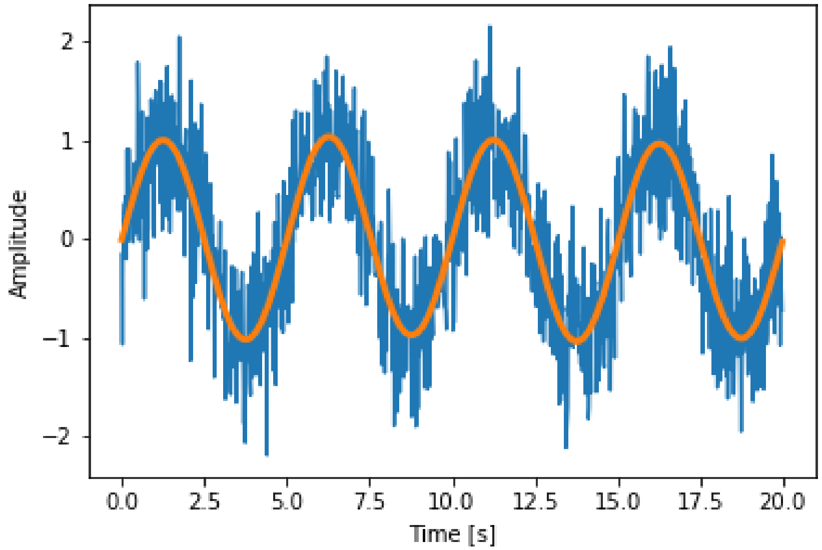
Getting started with Python for science

- Scipy : high-level scientific computing

3. Fast Fourier transforms: scipy.fftpack

The scipy.fftpack module allows to compute fast Fourier transforms. As an illustration, a (noisy) input signal may look like:

```
time_step = 0.02
period = 5.
time_vec = np.arange(0, 20, time_step)
sig = np.sin(2 * np.pi / period * time_vec) + \
       0.5 * np.random.randn(time_vec.size)
from scipy import fftpack
sample_freq = fftpack.fftfreq(sig.size, d=time_step)
sig_fft = fftpack.fft(sig)
pidxs = np.where(sample_freq > 0)
freqs = sample_freq[pidxs]
power = np.abs(sig_fft)[pidxs]
freq = freqs[power.argmax()]
sig_fft[np.abs(sample_freq) > freq] = 0
main_sig = fftpack.ifft(sig_fft)
plt.figure()
plt.plot(time_vec, sig)
plt.plot(time_vec, main_sig, linewidth=3)
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')
```



SciPy

Self Study

Getting started with Python for science

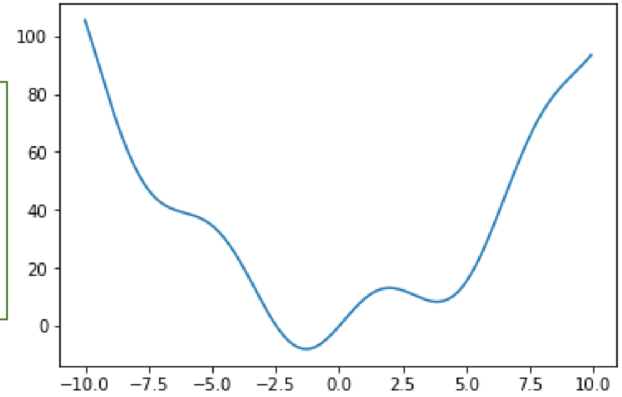
- Scipy : high-level scientific computing
  4. Optimization and fit: scipy.optimize

Optimization is the problem of finding a numerical solution to a minimization or equality. The scipy.optimize module provides useful algorithms for function minimization (scalar or multidimensional), curve fitting and root finding.

Finding the minimum of a scalar function

- This function has a global minimum around -1.3 and a local minimum around 3.8.

```
from scipy import optimize
def f(x):
    return x**2 + 10*np.sin(x)

x = np.arange(-10, 10, 0.1)
plt.plot(x, f(x))
plt.show()
```



Self Study

Getting started with Python for science

- Scipy : high-level scientific computing
  4. Optimization and fit: scipy.optimize

Finding the minimum of a scalar function

```
In [68]: optimize.fmin_bfgs(f, 0)
Optimization terminated successfully.
         Current function value: -7.945823
         Iterations: 5
         Function evaluations: 18
         Gradient evaluations: 6
Out[68]: array([-1.30644012])

In [69]: optimize.fmin_bfgs(f, 3)
Optimization terminated successfully.
         Current function value: 8.315586
         Iterations: 6
         Function evaluations: 21
         Gradient evaluations: 7
Out[69]: array([ 3.83746709])
```

- This function has a global minimum around -1.3 and a local minimumaround 3.8.
- The general and efficient way to find a minimum for this function is to conduct a gradient descent starting from a given initial point. The BFGS algorithmis a good way of doing this:

Getting started with Python for science

- Scipy : high-level scientific computing
  4. Optimization and fit: scipy.optimize

Finding the minimum of a scalar function

```
In [71]: optimize.fminbound(f, 0, 10)
Out[71]: 3.8374671194983834

In [72]: optimize.fminbound(f, -2, 0)
Out[72]: -1.3064400997882621
```

```
In [70]: optimize.basinhopping(f, 0)
Out[70]:
                        fun: -7.9458233756152845
 lowest_optimization_result:       fun: -7.9458233756152845
 hess_inv: array([[ 0.08583057]])
      jac: array([  1.19209290e-07])
  message: 'Optimization terminated successfully.'
     nfev: 15
      nit: 3
     njev: 5
   status: 0
  success: True
        x: array([-1.30644002])
                    message: ['requested number of basinhopping iterations completed successfully']
      minimization_failures: 0
                       nfev: 1542
                        nit: 100
                       njev: 514
                          x: array([-1.30644002])
```

Getting started with Python for science

- Scipy : high-level scientific computing
  4. Optimization and fit: scipy.optimize

Finding the roots of a scalar function

To find a root, i.e. a point where f(x) = 0, of the function f above we can use for example scipy.optimize.fsolve():

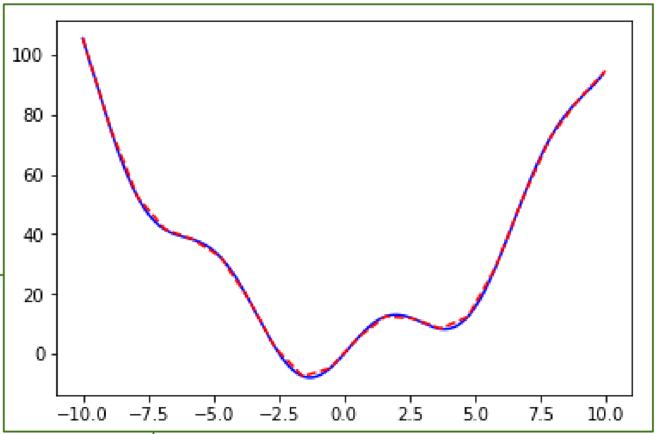
```
In [80]: optimize.fsolve(f, 1) # our initial guess is 1

In [80]: Out[80]: array([ 0.])

In [81]: optimize.fsolve(f, -2.5)
Out[81]: array([-2.47948183])

In [82]: optimize.fsolve(f, -5)

In [82]: Out[82]: array([ 0.])

In [83]: optimize.fsolve(f, 5)

IC:\Program Files\Anaconda3\lib\site-packages\scipy\optimize\minpack.py:161: RuntimeWarning: The
iteration is not making good progress, as measured by the n [84]:
  improvement from the last ten iterations.
  warnings.warn(msg, RuntimeWarning)
Out[83]: array([ 3.83742568])
```

Getting started with Python for science

- Scipy : high-level scientific computing

  4. Optimization and fit: scipy.optimize

Curve fitting




```
In [101]: xdata = np.linspace(-10, 10, num=20)
     ...: ydata = f(xdata) + np.random.randn(xdata.size)
     ...: def f2(x, a, b):
     ...:     return a*x**2 + b*np.sin(x)
     ...:
     ...: params, params_covariance = optimize.curve_fit(f2, xdata, ydata)
     ...: a = 1.00158773
     ...: b = 10.04166737
     ...: def f2(x, a, b):
     ...:     return a*x**2 + b*np.sin(x)
     ...:
     ...: y = f2(xdata, a, b)
     ...: plt.plot(x, f(x), color="blue")
     ...: plt.plot(xdata, y, color="red", linestyle="--")
     ...: plt.show()
```

Getting started with Python for science

- Scipy : high-level scientific computing

Other functions

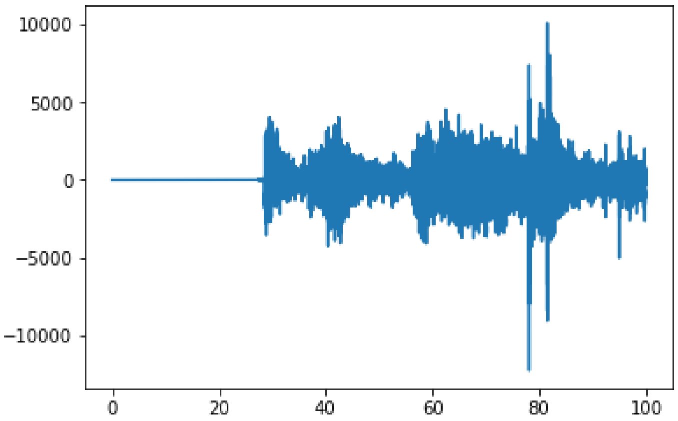
```
In [2]: scipy.mis
scipy.memmap
scipy.meshgrid
scipy.mgrid
scipy.min_scalar_type
scipy.minimum
scipy.mintypecode
scipy.mirr
scipy.misc
```

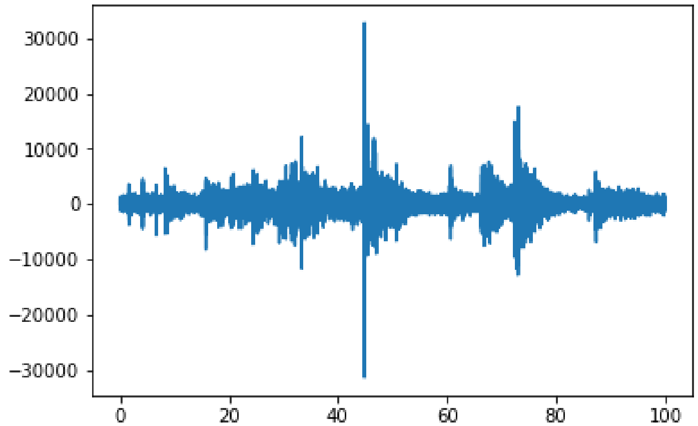
```
In [103]: scipy.
scipy.signbit
scipy.signedinteger
scipy.sin
scipy.sinc
scipy.single
scipy.singlecomplex
scipy.sinh
scipy.size
```

```
In [51]: scipy.ndi
scipy.nansum
scipy.nanvar
scipy.nbytes
scipy.ndarray
scipy.ndenumerate
scipy.ndfromtxt
scipy.ndim
scipy.ndimage
```

Getting started with Python for science

```
In [3]: from scipy.io import wavfile
   ...: import numpy as np
   ...: import matplotlib.pyplot as plt
   ...: import os
   ...: audio_dir = 'E:/gpu/presentations/final/data/wave'
   ...: for audio in os.listdir(audio_dir):
   ...:     audio = audio_dir + '/' + audio
   ...:     sample_rate, audio_info = wavfile.read(audio)
   ...:     audio_length = audio_info.shape[0]
   ...:     t = np.linspace(0,100,len(audio_info))
   ...:     plt.figure()
   ...:     plt.plot(t,audio_info)
```





https://docs.python.org/3/library/argparse.html

Getting started with Python for science

Command line arguments with argparse

```
# Import the library
import argparse
# Create the parser
parser = argparse.ArgumentParser()
# Add an argument
parser.add_argument('--name', type=str, required=True)
parser.add_argument('--rep_number', type=int, required=True)
# Parse the argument
args = parser.parse_args()
# Print "Hello" + the user input argument
for i in range(args.rep_number):
    print('Hello,', args.name)
```

```
C:\Users\MSN>python E:\gpu\presentations\args_test.py --name msh --rep_number 2
Hello, msh
Hello, msh
```

https://towardsdatascience.com/a-simple-guide-to-command-line-arguments-with-argparse-6824c30ab1c3

Getting started with Python for science

- Iterators, generator expressions and generators

Iterators 1 iter

```
In [38]: nums = [1, 2, 3]

In [39]: it = iter(nums)

In [40]: next(it)
Out[40]: 1

In [41]: next(it)
Out[41]: 2

In [42]: next(it)
Out[42]: 3

In [43]: next(it)
Traceback (most recent call last):

  File "<ipython-input-43-2cdb14c0d4d6>", line 1, in <module>
    next(it)

StopIteration
```

```
In [44]: dict1 = {'a': 2, 'c': 4}

In [45]: d1 = iter(dict1)

In [46]: next(d1)
Out[46]: 'c'

In [47]: next(d1)
Out[47]: 'a'
```

```
In [49]: a1 = np.array([0,1,2,3])

In [50]: a11 = iter(a1)

In [51]: next(a11)
Out[51]: 0

In [52]: next(a11)
Out[52]: 1

In [53]: next(a11)
Out[53]: 2

In [54]: next(a11)
Out[54]: 3
```

Getting started with Python for science

- Iterators, generator expressions and generators

Generator expressions

2

A second way in which iterator objects are created is through generator expressions

```
In [80]: t = (i for i in nums)

In [81]: t
Out[81]: <generator object <genexpr> at 0x000000892F1CCAF0>

In [82]: next(t)
Out[82]: 1

In [83]: next(t)
Out[83]: 2

In [84]: next(t)
Out[84]: 3

In [85]: next(t)
Traceback (most recent call last):

  File "<ipython-input-85-9494367a8bed>", line 1, in <module>
    next(t)

StopIteration
```

Getting started with Python for science

- Iterators, generator expressions and generators

Generator expressions

2

A second way in which iterator objects are created is through generator expressions

```
In [86]: l1 = list(i for i in nums)

In [87]: l1
Out[87]: [1, 2, 3]

In [88]: l = [i for i in nums]

In [89]: l
Out[89]: [1, 2, 3]

In [90]: s = {i for i in range(3)}

In [91]: s
Out[91]: {0, 1, 2}

In [92]: d = {i:i**2 for i in range(3)}

In [93]: d
Out[93]: {0: 0, 1: 1, 2: 4}
```

- If rectangular parentheses are used, the process is short-circuited and we get a list.
- The list comprehension syntax also extends to dictionary and set comprehensions.

```
In [94]: type(l)
Out[94]: list

In [95]: type(l1)
Out[95]: list

In [96]: type(s)
Out[96]: set

In [97]: type(d)
Out[97]: dict
```

Getting started with Python for science

- Iterators, generator expressions and generators

Generators

3

A generator is a function that produces a sequence of results instead of a single value.
A third way to create iterator objects is to call a generator function.

```
In [106]: def f():
     ...:     print("-- start --")
     ...:     yield 3
     ...:     print("-- middle --")
     ...:     yield 4
     ...:     print("-- finished --")
     ...:
     ...:
```

```
In [111]: for value in f():
     ...:     print(value)
     ...:
-- start --
3
-- middle --
4
-- finished --
```

```
In [107]: gen = f()

In [108]: next(gen)
-- start --

In [108]: 3In [109]:

In [109]: next(gen)
-- middle --
Out[109]: 4

In [110]: next(gen)
-- finished --
Traceback (most recent call last):

  File "<ipython-input-110-8a6233884a6c>", line 1, in <module>
    next(gen)

StopIteration
```

Getting started with Python for science

- Iterators, generator expressions and generators

Generators

3 A third way to create iterator objects is to call a generator function.

```
def nextSquare():
    i = 1
    while True:
        yield i*i
        i += 1
```

```
In [113]: nextSquare()
Out[113]: <generator object nextSquare at 0x000000892F472048>

In [114]: next(nextSquare())
Out[114]: 1

In [115]: next(nextSquare())
Out[115]: 1
```

```
for num in nextSquare():
     ...:     if num > 100:
     ...:         break
     ...:     print(num)
     ...:
     ...:

In [119]: 1
4
9
16
25
36
49
64
81
100
```

Getting started with Python for science

- Iterators, generator expressions and generators

Generators

3 A third way to create iterator objects is to call a generator function.

```
In [119]: import itertools

In [120]: def g():
     ...:     for i in itertools.count():
     ...:         print('--yielding %i --' % i)
     ...:         ans = yield i
     ...:         print('--yield returned %s --' % ans)
```

```
In [121]: it = g()

In [122]: next(it)
--yielding 0 --
Out[122]: 0

In [123]: next(it)
--yield returned None --
--yielding 1 --
Out[123]: 1

In [124]: next(it)
--yield returned None --
--yielding 2 --
Out[124]: 2
```

Getting started with Python for science

- Decorators

1 Various different names can be bound to the same function object.

```
In [185]: def first(msg):
     ...:     print(msg)
     ...:

In [186]: first("Hello")
Hello

In [187]: second = first

In [188]: second("Hello")
Hello
```

2 Functions can be passed as arguments to another function.

```
In [189]: def inc(x):
     ...:     return x + 1
     ...:
     ...:

In [190]: def dec(x):
     ...:     return x - 1
     ...:
     ...:

In [191]: def operate(func, x):
     ...:     result = func(x)
     ...:     return result
     ...:
     ...:

In [192]: operate(inc,3)
Out[192]: 4

In [193]: operate(dec,3)

Out[193]: 2
```

3 A function can return another function.

```
In [194]: def is_called():
     ...:     def is_returned():
     ...:         print("Hello")
     ...:     return is_returned
     ...:
     ...:

In [195]: is_called()()
Hello

In [196]: new = is_called()
     ...: new()

In [197]: Hello
```

Self Study

Getting started with Python for science

Decorators

4

Basically, a decorator takes in a function, adds some functionality and returns it.

1

```
In [198]: def make_pretty(func):
     ...:     def inner():
     ...:         print("I got decorated")
     ...:         func()
     ...:     return inner
     ...:
     ...:
     ...: def ordinary():
     ...:     print("I am ordinary")
```

2

```
In [199]: ordinary()

In [199]: I am ordinary


In [200]: pretty = make_pretty(ordinary)

In [201]: pretty()

In [201]: I got decorated
I am ordinary
```

3

```
In [202]: ordinary = make_pretty(ordinary)

In [203]: ordinary()

In [203]: I got decorated
I am ordinary
```

4

```
In [204]: ordinary = make_pretty(ordinary)

In [205]: ordinary()

In [205]: I got decorated
I got decorated
I am ordinary
```

5

```
In [206]: @make_pretty
     ...: def ordinary():
     ...:     print("I am ordinary")
     ...:
     ...:

In [207]: ordinary()

In [207]: I got decorated
I am ordinary


In [208]: ordinary()
I got decorated
I am ordinary
```

Getting started with Python for science

➢ What is pass statement in Python?

Suppose we have a loop or a function that is not implemented yet, but we want to implement it in the future. They cannot have an empty body. The interpreter would give an error. So, we use the pass statement to construct a body that does nothing.

```python
'''pass is just a placeholder for
functionality to be added later.'''
sequence = {'p', 'a', 's', 's'}
```

loop

```python
for val in sequence:
    pass
```

function

```python
def function(args):
    pass
```

class

```python
class Example:
    pass
```

```
In [179]: for val in sequence:
     ...:
     ...:
  File "<ipython-input-179-0fa2b3f516ef>", line 2

    ^
SyntaxError: unexpected EOF while parsing


In [180]: def function(args):
     ...:
     ...:
  File "<ipython-input-180-8e750e7958bf>", line 2

    ^
SyntaxError: unexpected EOF while parsing


In [181]: class Example:
     ...:
     ...:
  File "<ipython-input-181-3dfce30f501a>", line 2

    ^
SyntaxError: unexpected EOF while parsing
```

Getting started with Python for science

- Lambda functions

Syntax:
lambda arguments : expression

```
In [211]: x = lambda a : a + 10

In [212]: print(x(5))
15

In [213]: x = lambda a, b : a * b

In [214]: print(x(5, 6))

In [215]: 30
```

A lambda function can take any number of arguments, but can only have one expression.

Self Study

Getting started with Python for science

- Lambda functions

https://www.machinelearningplus.com/python/lambda-function/

```
In [3]: mylist = [2,3,4,5,6,7,8,9,10]
   ...: list_new  = list(filter(lambda x : (x%2==0), mylist))
   ...: print(list_new)
[2, 4, 6, 8, 10]
```

```
In [4]: mylist = [2,3,4,5,6,7,8,9,10]
   ...: list_new  = list(map(lambda x : x%2, mylist))
   ...: print(list_new)
[0, 1, 0, 1, 0, 1, 0, 1, 0]
```

```
In [5]: from functools import reduce
   ...: list1 = [1,2,3,4,5,6,7,8,9]
   ...: sum1 = reduce((lambda x,y: x+y), list1)
   ...: print(sum1)
45
```