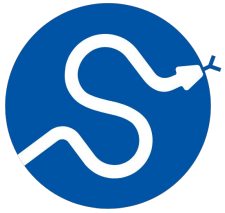
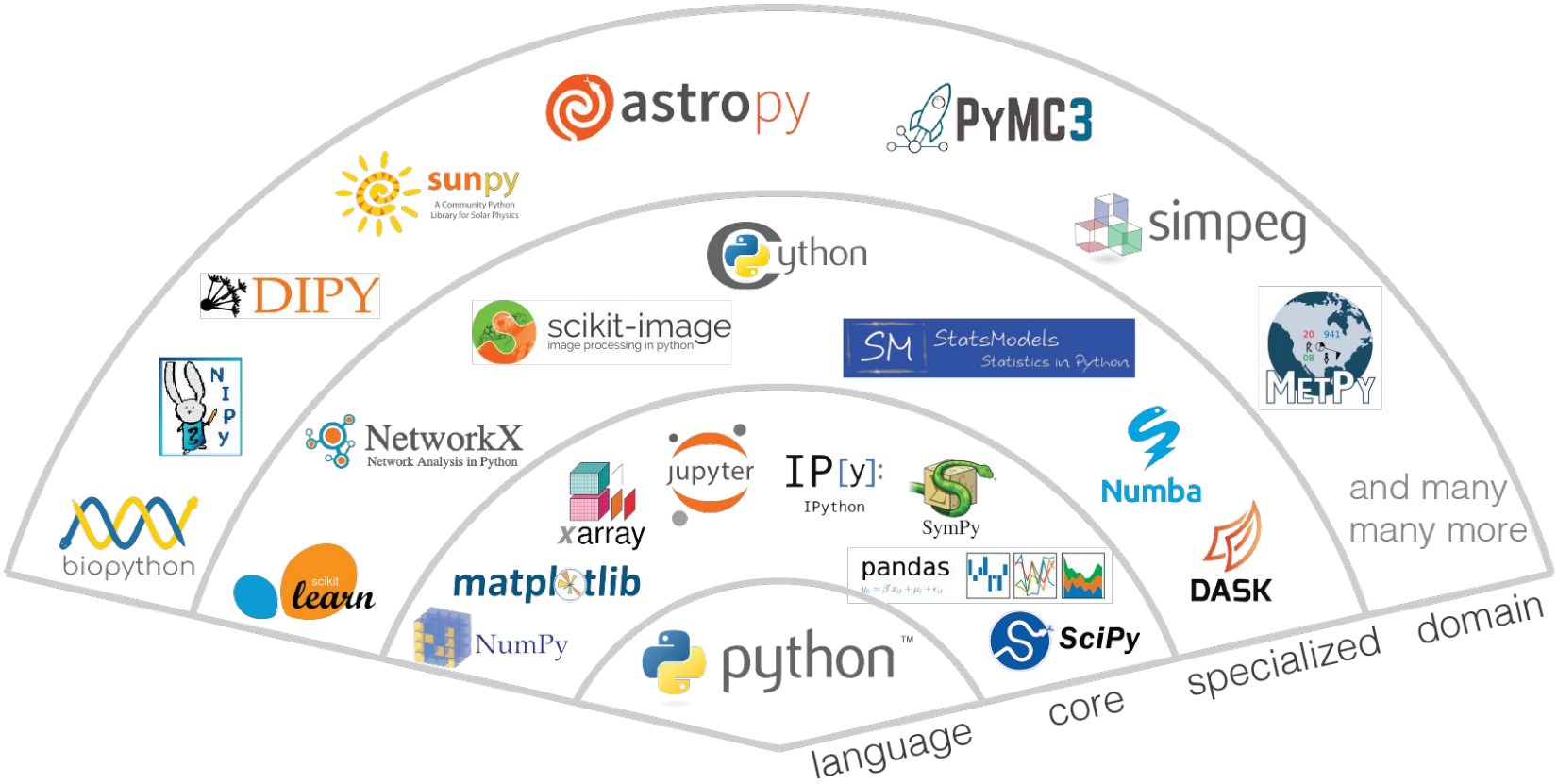


SciPy - Scientific Computing with Python

Ashutosh Gupta

Overview


astropy
PyMC3
sunpy
A Community Python Library for Solar Physics
simpeg
ython
DIPY
scikit-image
image processing in python
SM StatsModels
Statistics in Python
MetPy
NIPY
NetworkX
Network Analysis in Python
jupyter
IP[y]:
IPython
SymPy
Numba
and many
many more
biopython
xarray
pandas
DASK
scikit
learn
matplotlib
NumPy
python
SciPy
language
core
specialized
domain

Scipy - components

Scipy

The `scipy` package contains various toolboxes dedicated to common issues in scientific computing.

When to use Scipy?

- **File input/output:** `scipy.io` *(very specific)*
- **Special functions:** `scipy.special`
- **Linear algebra operations:** `scipy.linalg`
- **Interpolation:** `scipy.interpolate`
- **Optimization and fit:** `scipy.optimize`
- **Statistics and random numbers:** `scipy.stats`
- **Numerical integration:** `scipy.integrate`
- **Fast Fourier transforms:** `scipy.fftpack`
- **Signal processing:** `scipy.signal`
- **Image manipulation:** `scipy.ndimage`

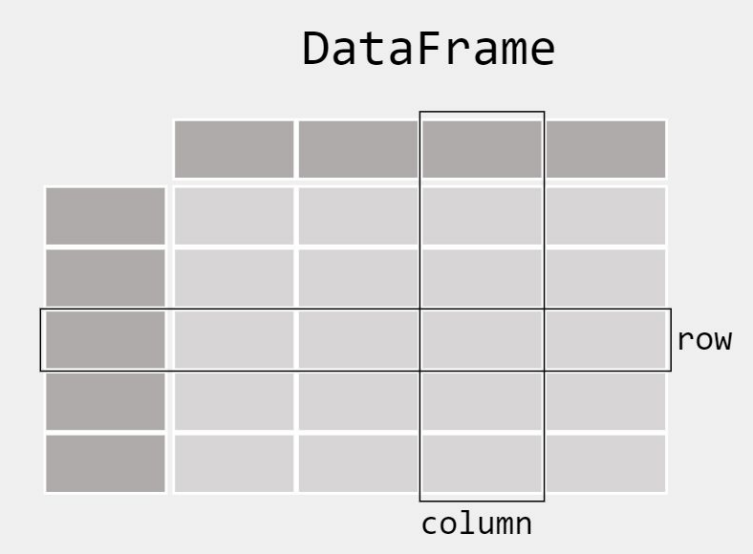
Rest of the Scipy session on Jupyter notebook…

Manipulating Tabular Data with Python

Ashutosh Gupta

When should you use Pandas?

- **Pa**nel **Da**ta
- When your data is *Tabular/Relational*
- MS Excel user



Series

	apples
0	3
1	2
2	0
3	1

+

Series

	oranges
0	0
1	3
2	7
3	2

=

DataFrame

	apples	oranges
0	3	0
1	2	3
2	0	7
3	1	2

Rest of the Pandas session on Jupyter notebook…

Scipy.io

Read/Write ‘.mat’ files

- Very important if you want to work with both MATLAB and Python.

Matlab files: Loading and saving:

```
>>> from scipy import io as spio
>>> a = np.ones((3, 3))
>>> spio.savemat('file.mat', {'a': a}) # savemat expects a dictionary
>>> data = spio.loadmat('file.mat')
>>> data['a']
array([[1.,  1.,  1.],
       [1.,  1.,  1.],
       [1.,  1.,  1.]])
```

- For txt files, use Numpy ‘loadtxt’ and ‘savetxt’ routines.

Scipy.linalg

Linear Algebra routines

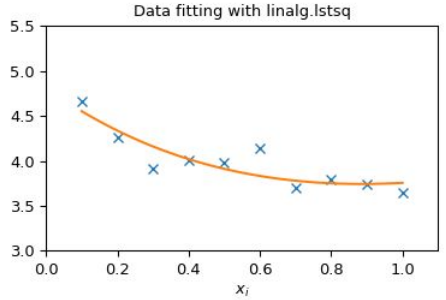
- Solve linear equations, compute norms, inverses, Eigen, SVD, LU, Cholesky, QR or other decompositions of a matrix:
- Scipy.linalg v/s numpy.linalg?

Solve Ax = b

```
>>> import numpy as np
>>> from scipy import linalg
>>> A = np.array([[1, 2], [3, 4]])
>>> A
array([[1, 2],
       [3, 4]])
>>> b = np.array([[5], [6]])
>>> b
array([[5],
       [6]])
>>> linalg.inv(A).dot(b)  # slow
array([[-4. ],
       [ 4.5]])
>>> A.dot(linalg.inv(A).dot(b)) - b  # check
array([[  8.88178420e-16],
       [  2.66453526e-15]])
>>> np.linalg.solve(A, b)  # fast
array([[-4. ],
       [ 4.5]])
>>> A.dot(np.linalg.solve(A, b)) - b  # check
array([[ 0.],
       [ 0.]])
```

SVD

```
>>> import numpy as np
>>> from scipy import linalg
>>> A = np.array([[1,2,3],[4,5,6]])
>>> A
array([[1, 2, 3],
       [4, 5, 6]])
>>> M,N = A.shape
>>> U,s,Vh = linalg.svd(A)
>>> Sig = linalg.diagsvd(s,M,N)
>>> U, Vh = U, Vh
>>> U
array([[-0.3863177 , -0.92236578],
       [-0.92236578,  0.3863177 ]])
>>> Sig
array([[ 9.508032  ,  0.        ,  0.        ],
       [ 0.        ,  0.77286964,  0.        ]])
>>> Vh
array([[-0.42866713, -0.56630692, -0.7039467 ],
       [ 0.80596391,  0.11238241, -0.58119908],
       [ 0.40824829, -0.81649658,  0.40824829]])
>>> U.dot(Sig.dot(Vh)) #check computation
array([[ 1.,  2.,  3.],
       [ 4.,  5.,  6.]])
```



Data Fitting

```
>>> import numpy as np
>>> from scipy import linalg
>>> import matplotlib.pyplot as plt
>>> rng = np.random.default_rng()
```

$$y_i = c_1 e^{-x_i} + c_2 x_i,$$

```
>>> c1, c2 = 5.0, 2.0
>>> i = np.r_[1:11]
>>> xi = 0.1*i
>>> yi = c1*np.exp(-xi) + c2*xi
>>> zi = yi + 0.05 * np.max(yi) * rng.standard_normal(len(yi))
```

```
>>> A = np.c_[np.exp(-xi)[:, np.newaxis], xi[:, np.newaxis]]
>>> c, resid, rank, sigma = linalg.lstsq(A, zi)
```

```
>>> xi2 = np.r_[0.1:1.0:100j]
>>> yi2 = c[0]*np.exp(-xi2) + c[1]*xi2
```

```
>>> plt.plot(xi,zi,'x',xi2,yi2)
>>> plt.axis([0,1.1,3.0,5.5])
>>> plt.xlabel('$x_i$')
>>> plt.title('Data fitting with linalg.lstsq')
>>> plt.show()
```

Scipy.interpolate

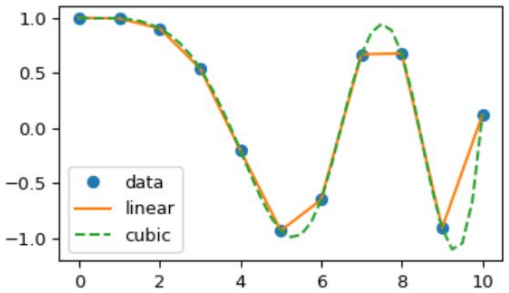
Data interpolation using pre-defined functions, splines

- Interpolation of N-dimensional points is possible

```
>>> from scipy.interpolate import interp1d
```

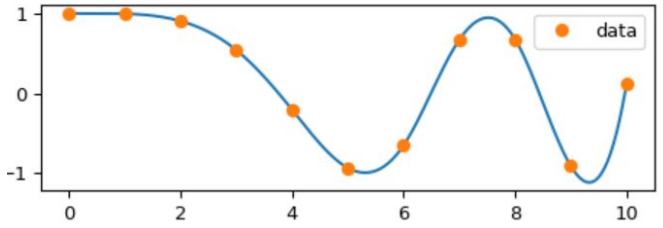
```
>>> x = np.linspace(0, 10, num=11, endpoint=True)
>>> y = np.cos(-x**2/9.0)
>>> f = interp1d(x, y)
>>> f2 = interp1d(x, y, kind='cubic')
```

```
>>> xnew = np.linspace(0, 10, num=41, endpoint=True)
>>> import matplotlib.pyplot as plt
>>> plt.plot(x, y, 'o', xnew, f(xnew), '-', xnew, f2(xnew), '--')
>>> plt.legend(['data', 'linear', 'cubic'], loc='best')
>>> plt.show()
```




```
>>> from scipy.interpolate import CubicSpline
>>> spl = CubicSpline([1, 2, 3, 4, 5, 6], [1, 4, 8, 16, 25, 36])
>>> spl(2.5)
5.57
```

```
>>> from scipy.interpolate import CubicSpline
>>> x = np.linspace(0, 10, num=11)
>>> y = np.cos(-x**2 / 9.)
>>> spl = CubicSpline(x, y)
```



Scipy.optimize

Fit to custom functions, find roots, minima/maxima

- Constrained and unconstrained optimization
- Techniques such as gradient descent, conjugate gradient descent are implemented (large number of variants)
- Linear programming problems (e.g. Simplex)

Custom function fitting

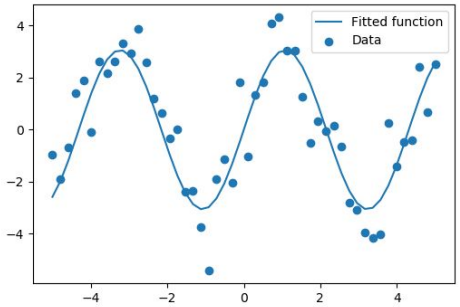
Suppose we have data on a sine wave, with some noise:

```
>>> x_data = np.linspace(-5, 5, num=50)
>>> y_data = 2.9 * np.sin(1.5 * x_data)
      + np.random.normal(size=50)
```

```
>>> def test_func(x, a, b):
...     return a * np.sin(b * x)
```

We then use `scipy.optimize.curve_fit()` to find a and b:

```
>>> params, params_covariance = opti-
    mize.curve_fit(test_func, x_data,
    y_data, p0=[2, 2])
>>> print(params)
[3.05931973  1.45754553]
```



Minimization

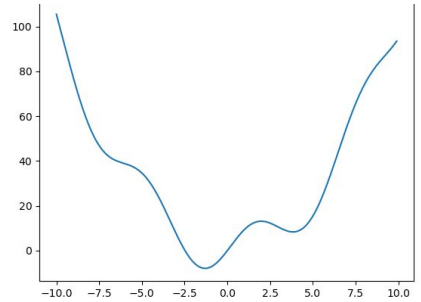
Let's define the following function:

```
>>> def f(x):
...     return x**2 + 10*np.sin(x)
```

and plot it:

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

```
>>> result = optimize.minimize(f, x0=0)
>>> result
      fun: -7.9458233756...
 hess_inv: array([[0.0858...]])
      jac: array([-1.19209...e-06])
  message: 'Optimization terminated suc-
     cessfully.'
     nfev: 18
      nit: 5
     njev: 6
   status: 0
  success: True
        x: array([-1.30644...])
>>> result.x # The coordinate of the mini-
    mum
array([-1.30644...])
```



Scipy.signal

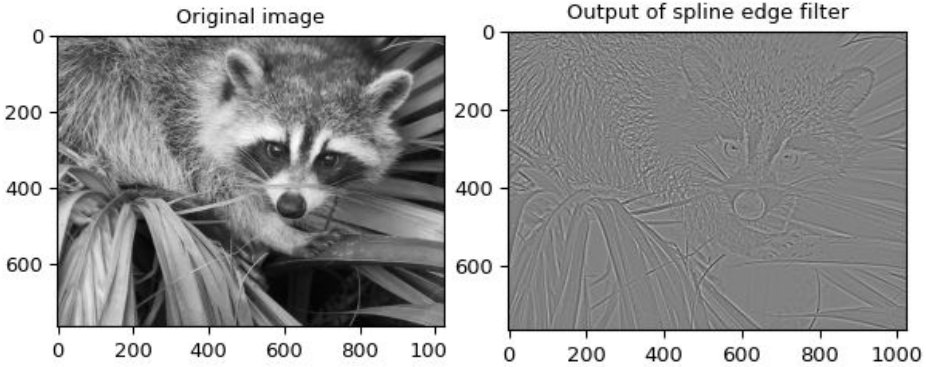
Convolution, Filtering, Filter design, Spectral Analysis, Common filters

1D Convolve

```
>>> x = np.array([1.0, 2.0, 3.0])
>>> h = np.array([0.0, 1.0, 0.0, 0.0, 0.0])
>>> signal.convolve(x, h)
array([ 0.,  1.,  2.,  3.,  0.,  0.,  0.])
>>> signal.convolve(x, h, 'same')
array([ 2.,  3.,  0.])
```

2D filtering

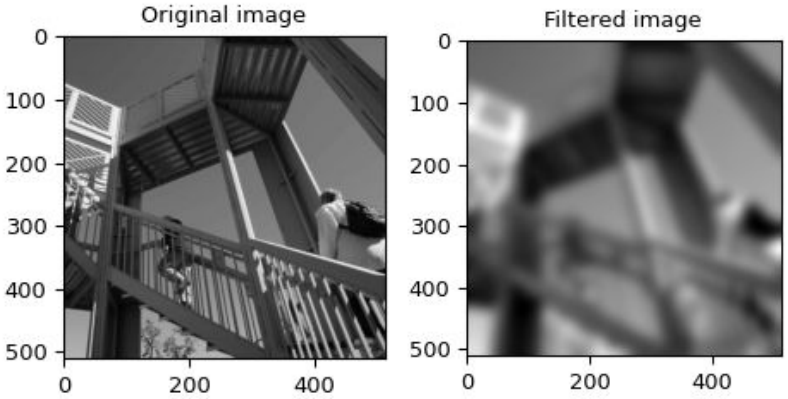
```
laplacian = np.array([[0,1,0], [1,-4,1], [0,1,0]], dtype=np.float32)
deriv2 = signal.convolve2d(ck,laplacian,mode='same',boundary='symm')
```



2D Separable filtering (Gaussian)

```
>>> import numpy as np
>>> from scipy import signal, datasets
>>> import matplotlib.pyplot as plt
```

```
>>> image = np.asarray(datasets.ascent(), np.float64)
>>> w = signal.windows.gaussian(51, 10.0)
>>> image_new = signal.sepfir2d(image, w, w)
```



Scipy.signal

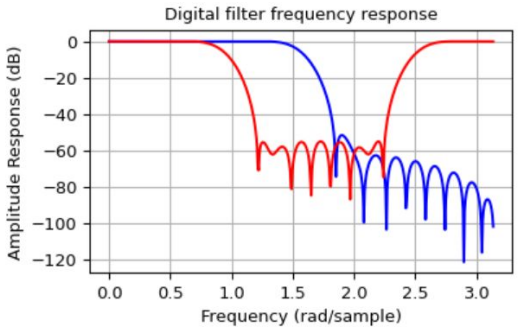
Filter design, Spectral Analysis, Common filters

FIR Filter design (Window method)

```
>>> import numpy as np
>>> import scipy.signal as signal
>>> import matplotlib.pyplot as plt
```

```
>>> b1 = signal.firwin(40, 0.5)
>>> b2 = signal.firwin(41, [0.3, 0.8])
>>> w1, h1 = signal.freqz(b1)
>>> w2, h2 = signal.freqz(b2)
```

```
>>> plt.title('Digital filter frequency response')
>>> plt.plot(w1, 20*np.log10(np.abs(h1)), 'b')
>>> plt.plot(w2, 20*np.log10(np.abs(h2)), 'r')
>>> plt.ylabel('Amplitude Response (dB)')
>>> plt.xlabel('Frequency (rad/sample)')
>>> plt.grid()
>>> plt.show()
```

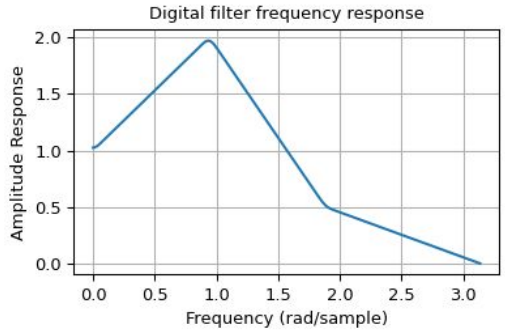


IIR Filter design

```
>>> import numpy as np
>>> import scipy.signal as signal
>>> import matplotlib.pyplot as plt
```

```
>>> b = signal.firwin2(150, [0.0, 0.3, 0.6, 1.0], [1.0, 2.0, 0.5, 0.0])
>>> w, h = signal.freqz(b)
```

```
>>> plt.title('Digital filter frequency response')
>>> plt.plot(w, np.abs(h))
>>> plt.title('Digital filter frequency response')
>>> plt.ylabel('Amplitude Response')
>>> plt.xlabel('Frequency (rad/sample)')
>>> plt.grid()
>>> plt.show()
```



Scipy.fftpack

FFT, FFT2, Different types of DCT/DCT2 and inverses

Generate the signal

```
# Seed the random number generator
np.random.seed(1234)
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))
plt.figure(figsize=(6, 5))
plt.plot(time_vec, sig, label='Original signal')
```

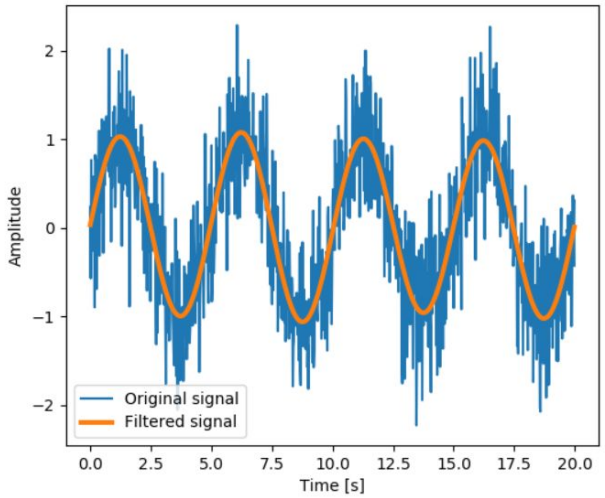
Compute and plot the power

```
# The FFT of the signal
sig_fft = fftpack.fft(sig)
# And the power (sig_fft is of complex dtype)
power = np.abs(sig_fft)**2
# The corresponding frequencies
sample_freq = fftpack.fftfreq(sig.size, d=time_step)
# Plot the FFT power
plt.figure(figsize=(6, 5))
plt.plot(sample_freq, power)
plt.xlabel('Frequency [Hz]')
plt.ylabel('plower')
# Find the peak frequency: we can focus on only the positive frequencies
pos_mask = np.where(sample_freq > 0)
freqs = sample_freq[pos_mask]
peak_freq = freqs[power[pos_mask].argmax()]
# Check that it does indeed correspond to the frequency that we generate
# the signal with
np.allclose(peak_freq, 1./period)
# An inner plot to show the peak frequency
axes = plt.axes([0.55, 0.3, 0.3, 0.5])
plt.title('Peak frequency')
plt.plot(freqs[:8], power[pos_mask][:8])
plt.setp(axes, yticks=[])
# scipy.signal.find_peaks_cwt can also be used for more advanced
# peak detection
```

Remove all the high frequencies

We now remove all the high frequencies and transform back from frequencies to signal.

```
high_freq_fft = sig_fft.copy()
high_freq_fft[np.abs(sample_freq) > peak_freq] = 0
filtered_sig = fftpack.ifft(high_freq_fft)
plt.figure(figsize=(6, 5))
plt.plot(time_vec, sig, label='Original signal')
plt.plot(time_vec, filtered_sig, linewidth=3, label='Filtered signal')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')
plt.legend(loc='best')
```



Workout: Repeat this example for a 2D signal / image !

Scipy.ndimage

histogram, geometric and radiometric manipulations

Changing orientation, resolution, ..

```
>>> from scipy import misc  # Load an image
>>> face = misc.face(gray=True)
>>> from scipy import ndimage # Shift, roate and zoom it
>>> shifted_face = ndimage.shift(face, (50, 50))
>>> shifted_face2 = ndimage.shift(face, (50, 50), mode='nearest')
>>> rotated_face = ndimage.rotate(face, 30)
>>> cropped_face = face[50:-50, 50:-50]
>>> zoomed_face = ndimage.zoom(face, 2)
>>> zoomed_face.shape
(1536, 2048)
```

Note: Matplotlib.pyplot.imread can be used for reading writing standard png/jpg/bmp/tif images

Scipy.ndimage

histogram, geometric and radiometric manipulations

Generate a noisy face:

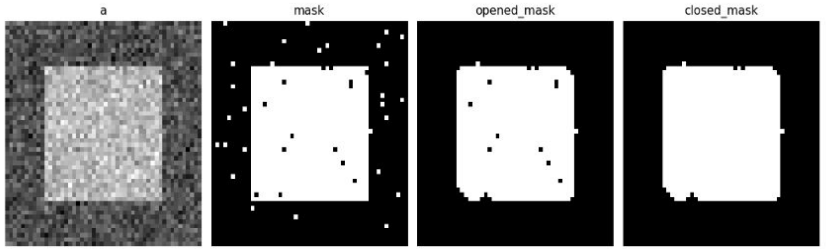
```
>>> from scipy import misc
>>> face = misc.face(gray=True)
>>> face = face[:512, -512:]  # crop out square on right
>>> import numpy as np
>>> noisy_face = np.copy(face).astype(np.float)
>>> noisy_face += face.std() * 0.5 * np.random.standard_normal(face.shape)
```

Apply a variety of filters on it:

```
>>> blurred_face = ndimage.gaussian_filter(noisy_face, sigma=3)
>>> median_face = ndimage.median_filter(noisy_face, size=5)
>>> from scipy import signal
>>> wiener_face = signal.wiener(noisy_face, (5, 5))
```

Morphological operations

```
>>> a = np.zeros((50, 50))
>>> a[10:-10, 10:-10] = 1
>>> a += 0.25 * np.random.standard_normal(a.shape)
>>> mask = a>=0.5
>>> opened_mask = ndimage.binary_opening(mask)
>>> closed_mask = ndimage.binary_closing(opened_mask)
```



Note: Matplotlib.pyplot.imread can be used for reading writing standard png/jpg/bmp/tif images

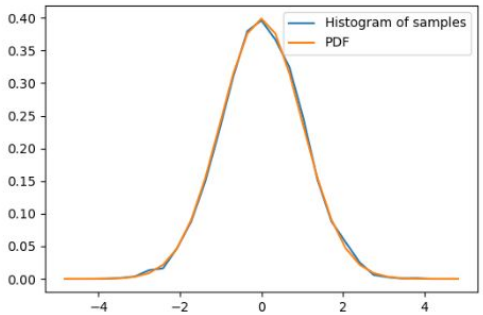
Scipy.stats, Scipy.special, Scipy.Integrate

Stats is for histograms, probability distributions, mean/mode/percentiles

Samples and distribution

Given observations of a random process, their histogram is an estimator of the random process's PDF (probability density function):

```
>>> samples = np.random.normal(size=1000)
>>> bins = np.arange(-4, 5)
>>> bins
array([-4, -3, -2, -1,  0,  1,  2,  3,  4])
>>> histogram = np.histogram(samples, bins=bins, density=True)[0]
>>> bins = 0.5*(bins[1:] + bins[:-1])
>>> bins
array([-3.5, -2.5, -1.5, -0.5,  0.5,  1.5,  2.5,  3.5])
>>> from scipy import stats
>>> pdf = stats.norm.pdf(bins)  # norm is a distribution object
>>> plt.plot(bins, histogram)
[<matplotlib.lines.Line2D object at ...>]
>>> plt.plot(bins, pdf)
[<matplotlib.lines.Line2D object at ...>]
```



Scipy.Special for special functions

- Bessel function, such as **`scipy.special.jn()`** (nth integer order Bessel function)
- Elliptic function (**`scipy.special.ellipj()`** for the Jacobian elliptic function, …)
- Gamma function: **`scipy.special.gamma()`**, also note **`scipy.special.gammaln()`** which will give the log of Gamma to a higher numerical precision.
- Erf, the area under a Gaussian curve: **`scipy.special.erf()`**

Scipy.Integrate

Methods for Integrating Functions given function object.

quad -- General purpose integration.
dblquad -- General purpose double integration.
tplquad -- General purpose triple integration.
fixed_quad -- Integrate func(x) using Gaussian quadrature of order n.

Methods for Integrating Functions given fixed samples.

trapezoid -- Use trapezoidal rule to compute integral.
cumulative_trapezoid -- Use trapezoidal rule to cumulatively compute integral.
simpson -- Use Simpson's rule to compute integral from samples.
*romb -- Use Romberg Integration to compute integral from -- (2**k + 1) evenly-spaced samples.*