**Signal Processing : scipy.signal**

Analyzing, altering, and synthesising signals such as music, pictures, and so on is what signal processing is all about. SciPy includes routines for designing, filtering, and interpolating one-dimensional and two-dimensional data.

Filtering:

Filtering a signal essentially removes any undesirable components. The order\_filter function can be used to conduct ordered filtering. This function is used to filter an array in an ordered manner. This function's syntax is as follows:

***order\_filter(a, domain, rank)***

a = N-dimensional input array

domain = mask array where number of dimensions is same as `a`

rank = Non-negative number that selects elements from the list after it has been sorted (0 is the smallest followed by 1…)

**Example 1 :**

*from scipy import signal*

*import numpy as np*

*x = np.arange(15).reshape(3, 5)*

*domain = np.identity(3)*

*print("Input: " )*

*print(x)*

*print("Signal: ")*

*print(signal.order\_filter(x, domain, 1))*

*Output :*

*Input:*

*[[ 0 1 2 3 4]*

*[ 5 6 7 8 9]*

*[10 11 12 13 14]]*

*Signal:*

*[[0. 1. 2. 3. 0.]*

*[5. 6. 7. 8. 3.]*

*[0. 5. 6. 7. 8.]]*

Waveforms:

The scipy.signal subpackage also includes a number of routines for generating waveforms. Chirp is one of these functions. The syntax for this function, which is a frequency-swept cosine generator, is as follows:

SYNTAX:

chirp(t, f0, t1, f1, method=’linear’, phi=0, vertex\_zero=True)

where:

* **t** : array\_like - Times at which to evaluate the waveform.
* **fo** : float - Frequency (e.g. Hz) at time t=0.
* **t1** : float - Time at which “f1° is specified.
* **f1** : float - Frequency (e.g. Hz) of the waveform at time “ti”.
* **method** : {'linear’, ‘quadratic’, ‘logarithmic’, ‘hyperbolic'}, optional - Kind of frequency sweep. If not given, “linear” is assumed.
* **phi** : float, optional - Phase offset, in degrees. Default is 0.
* **vertex\_zero** : bool, optional - This parameter is only used when “method” is ‘quadratic’. It determines whether the vertex of the parabola that is the graph of the frequency is at t=0 or t=t1.

**Example 2 :**

*from scipy.signal import chirp, spectrogram*

*import matplotlib.pyplot as plt*

*t = np.linspace(4, 8, 300)*

*w = chirp(t, f0=4, f1=2, t1=5, method='linear')*

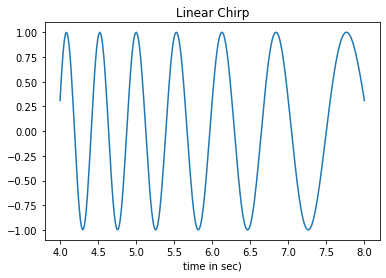
*plt.plot(t, w)*

*plt.title("Linear Chirp")*

*plt.xlabel('time in sec)')*

*plt.show()*

*Output:*

**

**Example 3 : Gaussian Filtering for Blurring Images**

*import numpy as np*

*from scipy import signal, misc*

*import matplotlib.pyplot as plt*

*image = misc.ascent()*

*w = signal.windows.gaussian(51, 10.0)*

*#Apply convolution*

*image\_new = signal.sepfir2d(image, w, w)*

*image = misc.ascent()*

*w = signal.windows.gaussian(51, 10.0)*

*image\_new = signal.sepfir2d(image, w, w)*

*#Original Image*

*plt.figure()*

*plt.imshow(image)*

*plt.gray()*

*plt.title('Original image')*

*plt.show()*

*#Filtered Image*

*plt.figure()*

*plt.imshow(image\_new)*

*plt.gray()*

*plt.title('Filtered image')*

*plt.show()*

*Output:*

**

**

**Example 4: Removal of constant and linear trend of a second-order polynomial time series**

Detrend is a SciPy function that removes a constant or linear trend from a data series so that higher-order effects can be seen.

The following example illustrates the residual signal components after eliminating the constant and linear trend of a second-order polynomial time series.

*import numpy as np*

*import scipy.signal as signal*

*import matplotlib.pyplot as plt*

*t = np.linspace(-10, 10, 20)*

*y = 1 + t + 0.01\*t\*\*2*

*yconst = signal.detrend(y, type='constant')*

*ylin = signal.detrend(y, type='linear')*

*plt.plot(t, y, '-rx')*

*plt.plot(t, yconst, '-bo')*

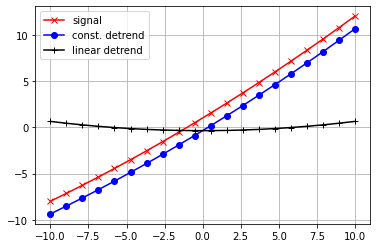
*plt.plot(t, ylin, '-k+')*

*plt.grid()*

*plt.legend(['signal', 'const. detrend', 'linear detrend'])*

*plt.show()*

*Output:*

**