Author: Sanjoy Biswas

Topic : SciPy Practice for Machine Learning and Data Science

Email : sanjoy.eee32@gmail.com

SciPy contains varieties of sub packages which help to solve the most common issue related to Scientific Computation. SciPy is the most used Scientific library only second to GNU Scientific Library for C/C++ or Matlab's. Easy to use and understand as well as fast computational power. It can operate on an array of NumPy library.

SciPy contains varieties of sub packages which help to solve the most common issue related to Scientific Computation. SciPy is the most used Scientific library only second to GNU Scientific Library for C/C++ or Matlab's. Easy to use and understand as well as fast computational power. It can operate on an array of NumPy library.

In [29]:

```
from scipy import special
a = special.exp10(3)
print(a)

b = special.exp2(3)
print(b)

c = special.sindg(90)
print(c)

d = special.cosdg(45)
print(d)
```

```
1000.0
8.0
1.0
0.7071067811865475
```

In [33]:

```
from scipy import integrate
a = lambda y, x: x*y**2
b = lambda x: 1
c = lambda x: -1
integrate.dblquad(a, 0, 2, b, c)
```

Out[33]:

```
(-1.3333333333333335, 1.4802973661668755e-14)
```

In [6]:

```
import numpy as np
from scipy.optimize import rosen
a = 1.2 * np.arange(5)
rosen(a)
```

Out[6]:

```
7371.0399999999945
```

Nelder-Mead

In [7]:

```
from scipy import optimize
a = [2.4, 1.7, 3.1, 2.9, 0.2]
b = optimize.minimize(optimize.rosen, a, method='Nelder-Mead')
```

```
b.x
```

Out[7]:

```
array([0.96570182, 0.93255069, 0.86939478, 0.75497872, 0.56793357])
```

Fourier Transform Functions:

In [13]:

```
from scipy.fftpack import fft, ifft
x = np.array([0,1,2,3])
y = fft(x)
print(y)
```

```
[ 6.-0.j -2.+2.j -2.-0.j -2.-2.j]
```

Signal Processing Functions:

In [16]:

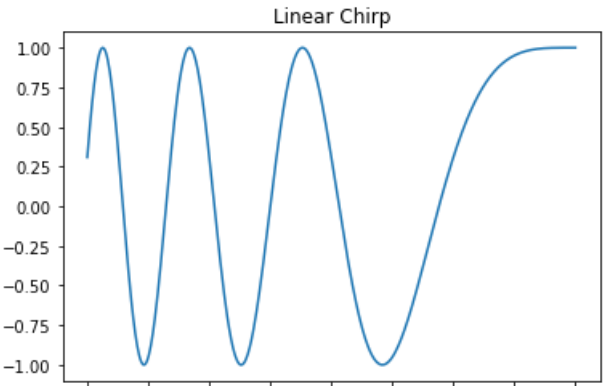
```
from scipy import signal
x = np.arange(35).reshape(7, 5)
domain = np.identity(3)
print(x,end='nn')
print(signal.order_filter(x, domain, 1))
```

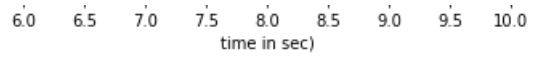
```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]
 [20 21 22 23 24]
 [25 26 27 28 29]
 [30 31 32 33 34]]nn[[ 0.  1.  2.  3.  0.]
 [ 5.  6.  7.  8.  3.]
 [10. 11. 12. 13.  8.]
 [15. 16. 17. 18. 13.]
 [20. 21. 22. 23. 18.]
 [25. 26. 27. 28. 23.]
 [ 0. 25. 26. 27. 28.]]
```

Waveforms:

In [17]:

```
from scipy.signal import chirp, spectrogram
import matplotlib.pyplot as plt
t = np.linspace(6, 10, 500)
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()
```





Finding the Inverse of a Matrix:

In [18]:

```
import numpy as np
from scipy import linalg
A = np.array([[1,2], [4,3]])
B = linalg.inv(A)
print(B)
```

```
[[-0.6  0.4]
 [ 0.8 -0.2]]
```

Finding the Determinants:

In [19]:

```
import numpy as np
from scipy import linalg
A = np.array([[1,2], [4,3]])
B = linalg.det(A)
print(B)
```

```
-5.0
```

Sparse Eigenvalues:

In [20]:

```
from scipy.linalg import eigh
import numpy as np
A = np.array([[1, 2, 3, 4], [4, 3, 2, 1], [1, 4, 6, 3], [2, 3, 2, 5]])
a, b = eigh(A)
print("Selected eigenvalues :", a)
print("Complex ndarray :", b)
```

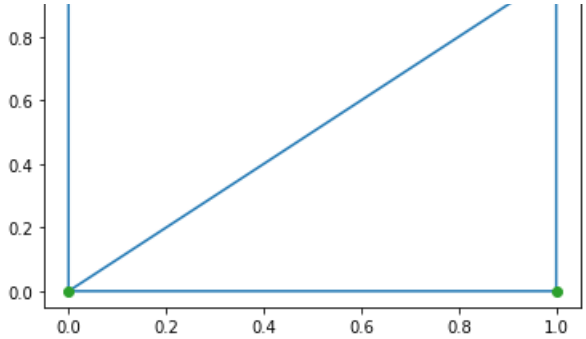
```
Selected eigenvalues : [-2.53382695  1.66735639  3.69488657 12.17158399]
Complex ndarray : [[ 0.69205614  0.5829305   0.25682823 -0.33954321]
 [-0.68277875  0.46838936  0.03700454 -0.5595134 ]
 [ 0.23275694 -0.29164622 -0.72710245 -0.57627139]
 [ 0.02637572 -0.59644441  0.63560361 -0.48945525]]
```

Spatial Data Structures and Algorithms:

In [21]:

```
import matplotlib.pyplot as plt
from scipy.spatial import Delaunay
points = np.array([[0, 1], [1, 1], [1, 0],[0, 0]])
a = Delaunay(points)        #Delaunay object
print(a)
print(a.simplices)
plt.triplot(points[:,0], points[:,1], a.simplices)
plt.plot(points[:,1], points[:,0], 'o')
plt.show()
```

```
<scipy.spatial.qhull.Delaunay object at 0x000001D516C39608>
[[3 2 1]
 [3 1 0]]
```



Working with Polynomials

In [22]:

```
from numpy import poly1d

# We'll use some functions from numpy remember!!
# Creating a simple polynomial object using coefficients
somePolynomial = poly1d([1,2,3])

# Printing the result
# Notice how easy it is to read the polynomial this way
print(somePolynomial)

# Let's perform some manipulations
print("\nSquaring the polynomial: \n")
print(somePolynomial* somePolynomial)

#How about integration, we just have to call a function
# We just have to pass a constant say 3
print("\nIntegrating the polynomial: \n")
print(somePolynomial.integ(k=3))

#We can also find derivatives in similar way
print("\nFinding derivative of the polynomial: \n")
print(somePolynomial.deriv())

# We can also solve the polynomial for some value,
# let's try to solve it for 2
print("\nSolving the polynomial for 2: \n")
print(somePolynomial(2))
```

```
   2
1 x + 2 x + 3

Squaring the polynomial:

   4     3      2
1 x + 4 x + 10 x + 12 x + 9

Integrating the polynomial:

        3     2
0.3333 x + 1 x + 3 x + 3

Finding derivative of the polynomial:


2 x + 2

Solving the polynomial for 2:

11
```

SciPy Example – Linear Algebra

In [23]:

```
# Import required modules/ libraries
```

```
# Import required modules/ libraries
import numpy as np
from scipy import linalg

# We are trying to solve a linear algebra system which can be given as:
#               1x + 2y =5
#               3x + 4y =6

# Create input array
A= np.array([[1,2],[3,4]])

# Solution Array
B= np.array([[5],[6]])

# Solve the linear algebra
X= linalg.solve(A,B)

# Print results
print(X)

# Checking Results
print("\n Checking results, following vector should be all zeros")
print(A.dot(X)-B)
```

```
[[-4. ]
 [ 4.5]]

 Checking results, following vector should be all zeros
[[0.]
 [0.]]
```

SciPy Integration

In [24]:

```
# Import required packages
from scipy import integrate

# Using quad as we can see in list quad is used for simple integration
# arg1: A lambda function which returns x squared for every x
# We'll be integrating this function
# arg2: lower limit
# arg3: upper limit
result= integrate.quad(lambda x: x**2, 0,3)
print(result)
```

```
(9.000000000000002, 9.992007221626411e-14)
```

SciPy Fourier Transforms

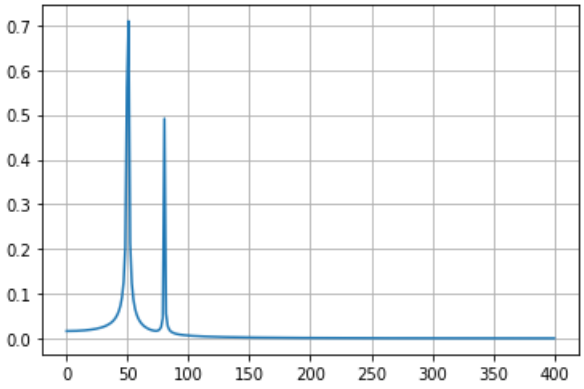
In [25]:

```
# Import Fast Fourier Transformation requirements
from scipy.fftpack import fft
import numpy as np

# Number of sample points
N = 600

# sample spacing
T = 1.0 / 800.0
x = np.linspace(0.0, N*T, N)
y = np.sin(50.0 * 2.0*np.pi*x) + 0.5*np.sin(80.0 * 2.0*np.pi*x)
yf = fft(y)
xf = np.linspace(0.0, 1.0/(2.0*T), N//2)

# matplotlib for plotting purposes
import matplotlib.pyplot as plt
plt.plot(xf, 2.0/N * np.abs(yf[0:N//2]))
plt.grid()
plt.show()
```

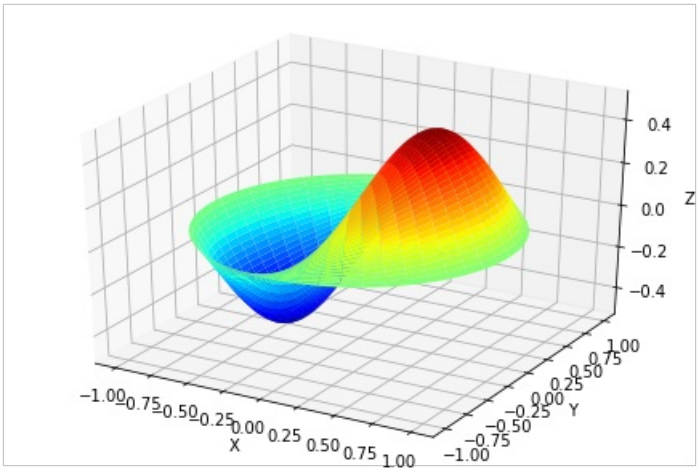


SciPy Special Functions

In [26]:

```
# Import special package
from scipy import special
import numpy as np
def drumhead_height(n, k, distance, angle, t):
    kth_zero = special.jn_zeros(n, k)[-1]
    return np.cos(t) * np.cos(n*angle) * special.jn(n, distance*kth_zero)
theta = np.r_[0:2*np.pi:50j]
radius = np.r_[0:1:50j]
x = np.array([r * np.cos(theta) for r in radius])
y = np.array([r * np.sin(theta) for r in radius])
z = np.array([drumhead_height(1, 1, r, theta, 0.5) for r in radius])

# Plot the results for visualization
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
fig = plt.figure()
ax = Axes3D(fig)
ax.plot_surface(x, y, z, rstride=1, cstride=1, cmap=cm.jet)
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
plt.show()
```



In [ ]:

```
```