한국항공대학교
KOREA AEROSPACE UNIVERSITY

AI 프로그래밍

Matplotlib 및 SciPy 소개

소프트웨어학과
문의현

Matplotlib 소개

matplotlib
Version 3.4.1

Fork me on GitHub

Installation | Documentation | Examples | Tutorials | Contributing | Search

home | contents » Matplotlib: Python plotting | modules | index

Matplotlib: Visualization with Python

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.

Matplotlib makes easy things easy and hard things possible.

Create

- Develop publication quality plots with just a few lines of code
- Use interactive figures that can zoom, pan, update...

Customize

- Take full control of line styles, font properties, axes properties...
- Export and embed to a number of file formats and interactive environments

Extend

- Explore tailored functionality provided by third party packages
- Learn more about Matplotlib through the many external learning resources

Documentation

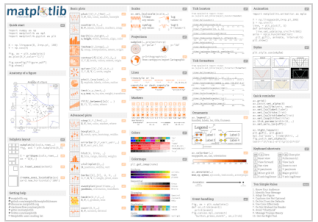
To get started, read the User's Guide.

Latest stable release
3.4.1: docs | changelog

Last release for Python 2
2.2.5: docs | changelog

Development version
docs

Matplotlib cheatsheets



Support Matplotlib

Source: https://matplotlib.org

Matplotlib 소개

- Matplotlib은 파이썬의 대표적인 과학 계산용 그래프 라이브러리
- 다양한 데이터를 시각화 할 수 있도록 도와주는 라이브러리
  - 간단한 데이터 분석에서부터 인공지능 모델의 시각화까지 활용도가 매우 높음
- 선 그래프, 히스토그램, 산점도 등을 지원
- 데이터와 분석 결과를 시각화함으로써 다양한 관점에서 관측하고 중요한 통찰을 얻을 수 있음
- NumPy 데이터 구조와 함께 많이 쓰임

Matplotlib 소개

- 간단한 직선 그래프 그리기

```
import matplotlib.pyplot as plt

x = [1, 2, 3]
y = [1, 2, 3]
plt.plot(x, y)
plt.title("My Plot")
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```

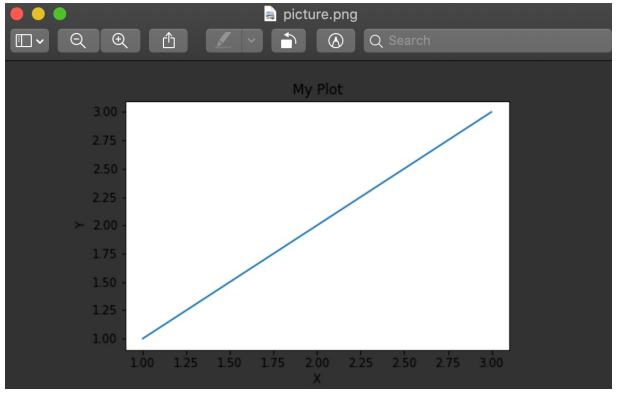
<실행 결과>

Matplotlib 소개

- 그래프 저장하기

```
import matplotlib.pyplot as plt

x = [1, 2, 3]
y = [1, 2, 3]
plt.plot(x, y)
plt.title("My Plot")
plt.xlabel("X")
plt.ylabel("Y")
plt.savefig('picture.png')
```

<실행 결과>

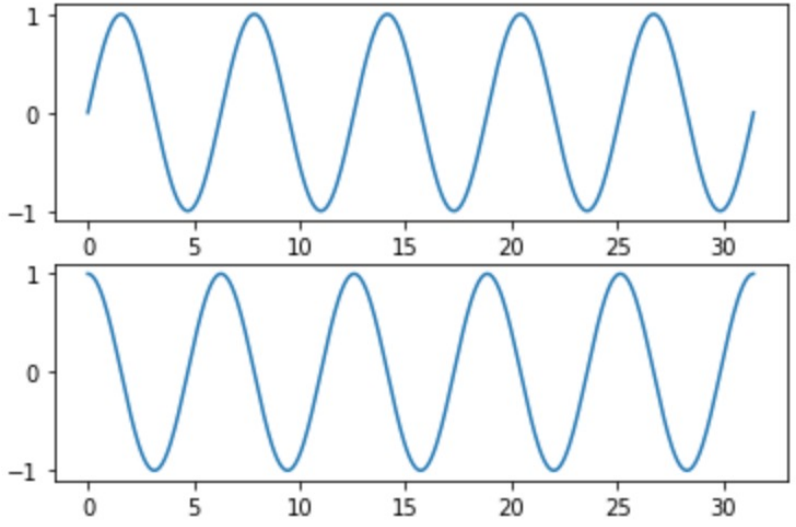


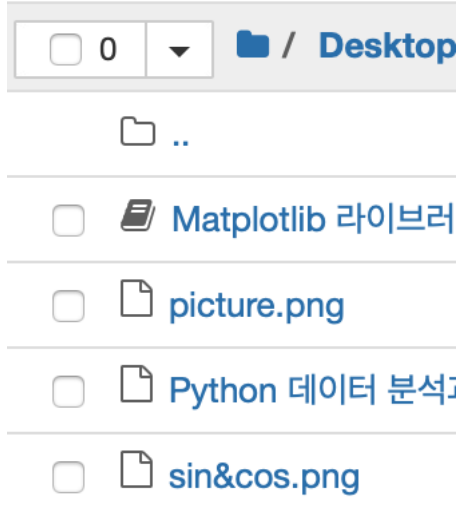
Matplotlib 소개

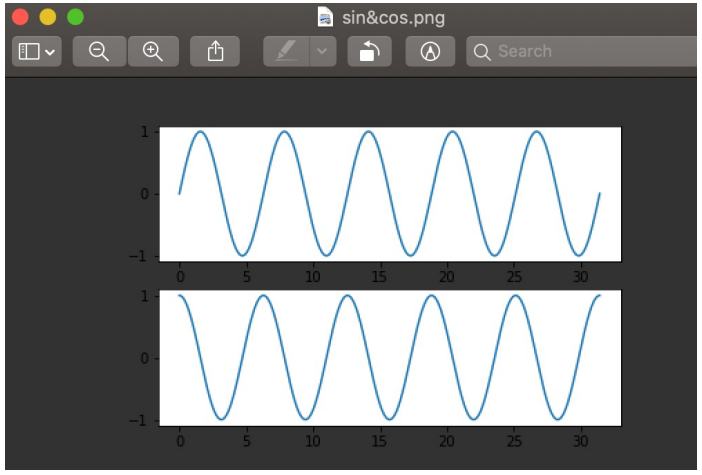
- 그래프 저장하기

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, np.pi * 10, 500) # PI * 10 너비에, 500개의 점을 균일하게 찍기
fig, axes = plt.subplots(2, 1) # 2개의 그래프가 들어가는 Figure 생성
axes[0].plot(x, np.sin(x)) # 첫 번째 그래프는 사인(Sin) 그래프
axes[1].plot(x, np.cos(x)) # 두 번째 그래프는 코사인(Cos) 그래프
fig.savefig("sin&cos.png")
```

<실행 결과>





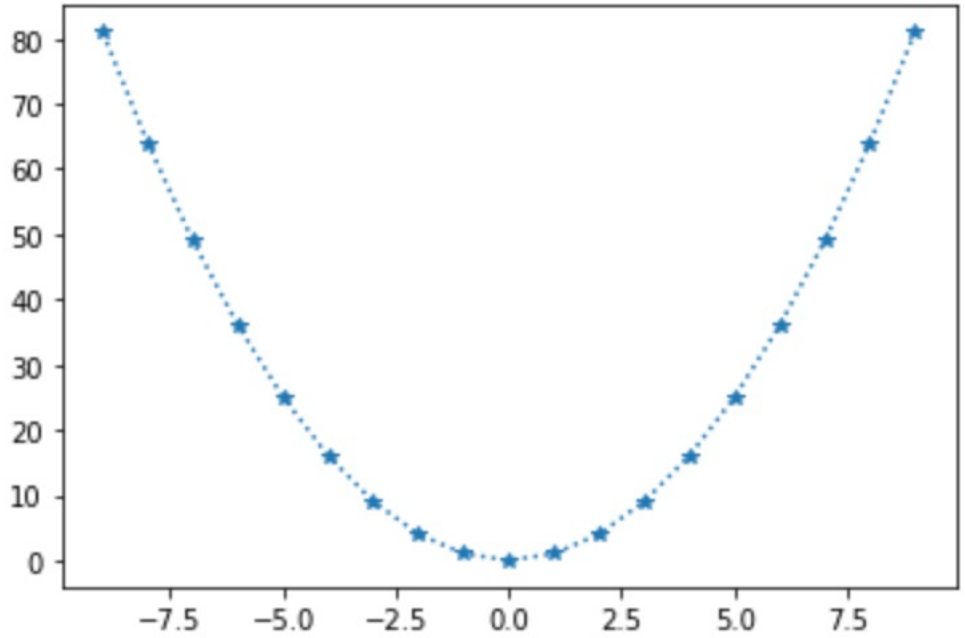


Matplotlib 활용

- 선 그래프 그리기

```
import matplotlib.pyplot as plt
import numpy as np

x = np.arange(-9, 10)
y = x ** 2
# 라인 스타일로는 '-', ':', '-.', '--' 등이 사용될 수 있습니다.
plt.plot(x, y, linestyle=":", marker="*")
# X축 및 Y축에서 특정 범위를 자를 수도 있습니다.
plt.show()
```

<실행 결과>

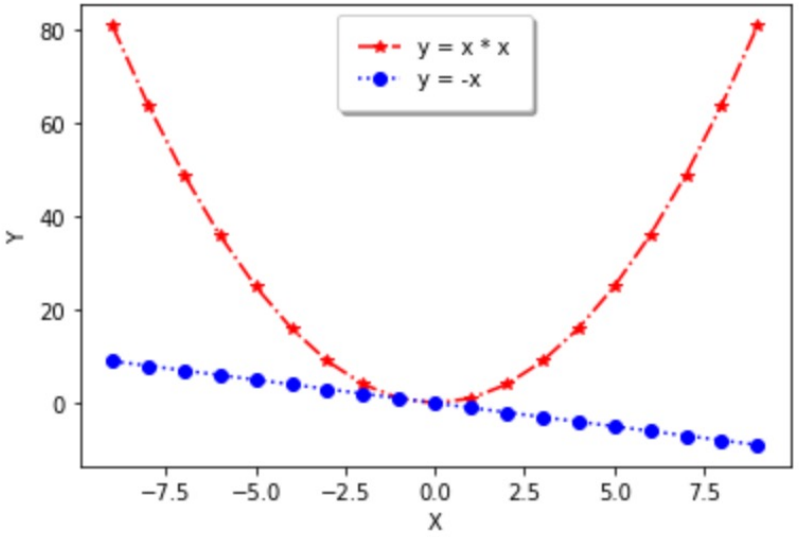


Matplotlib 활용

- 선 그래프 그리기

```
import matplotlib.pyplot as plt
import numpy as np

x = np.arange(-9, 10)
y1 = x ** 2
y2 = -x
plt.plot(x, y1, linestyle="-.", marker="*", color="red", label="y = x * x")
plt.plot(x, y2, linestyle=":", marker="o", color="blue", label="y = -x")
plt.xlabel("X")
plt.ylabel("Y")
plt.legend(
  shadow=True,
  borderpad=1
)
plt.show()
```

<실행 결과>

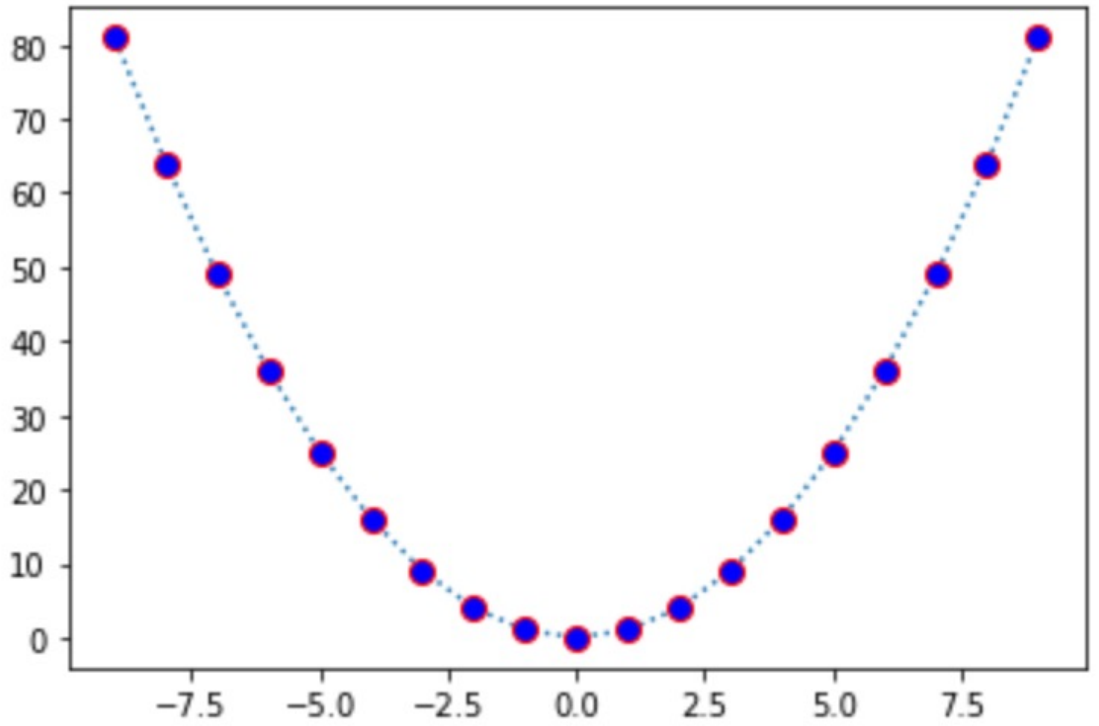


Matplotlib 활용

- 선 그래프 그리기

```
import matplotlib.pyplot as plt
import numpy as np

x = np.arange(-9, 10)
y1 = x ** 2
plt.plot(
    x, y1,
    linestyle=":",
    marker="o",
    markersize=8,
    markerfacecolor="blue",
    markeredgecolor="red"
)
plt.show()
```

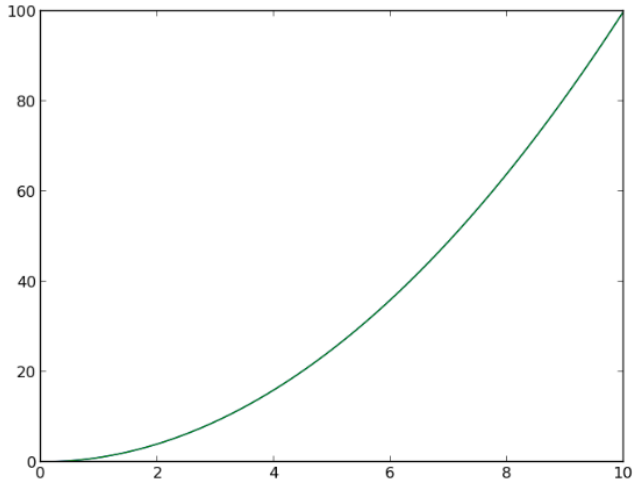
<실행 결과>



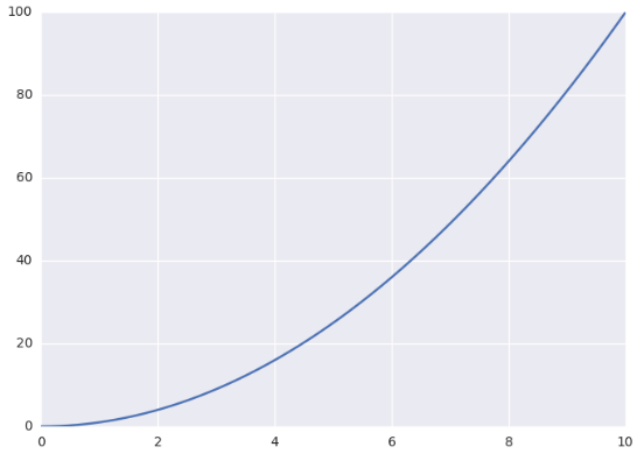
Matplotlib 활용

- 선 그래프 그리기
  - Seaborn 라이브러리를 활용

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 10, 1000)
y = np.power(x, 2)
plt.plot(x, y)
plt.show()
```



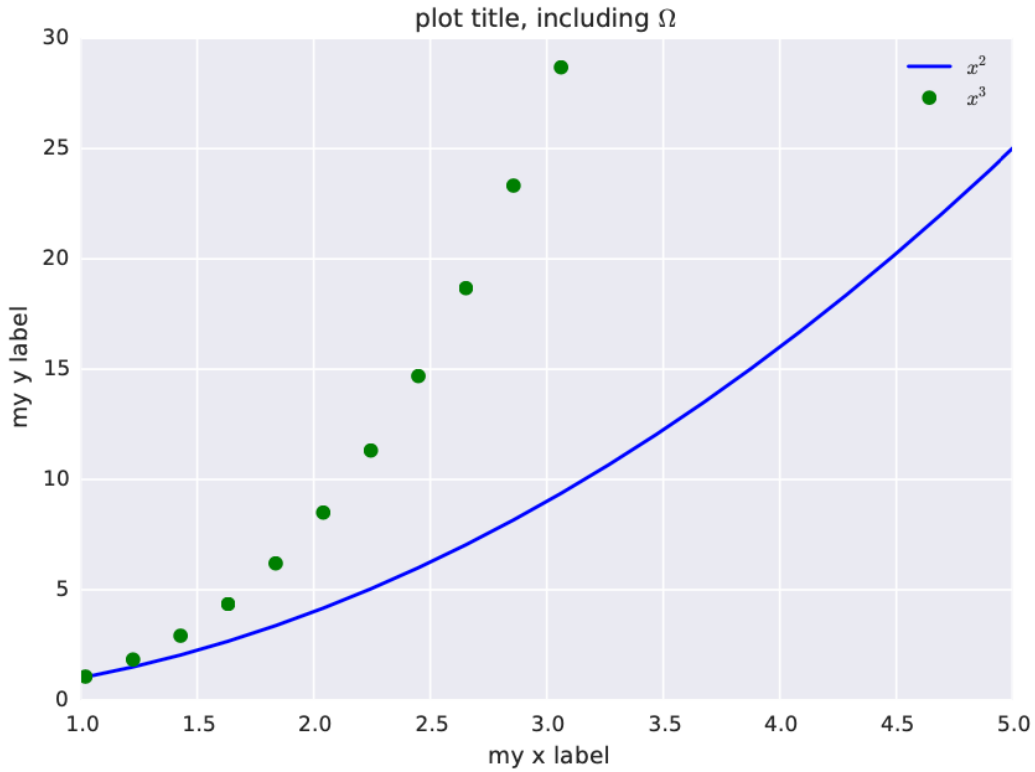

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

x = np.linspace(0, 10, 1000)
y = np.power(x, 2)
plt.plot(x, y)
plt.show()
```



Matplotlib 활용

- 선 그래프 그리기

```
x = np.linspace(0, 10, 50)
y1 = np.power(x, 2)
y2 = np.power(x, 3)

plt.plot(x, y1, 'b-', label='$x^2$')
plt.plot(x, y2, 'go', label='$x^3$')
plt.xlim((1, 5))
plt.ylim((0, 30))
plt.xlabel('my x label')
plt.ylabel('my y label')
plt.title('plot title, including $\Omega$')
plt.legend()

plt.savefig('line_plot_plus2.pdf')
```

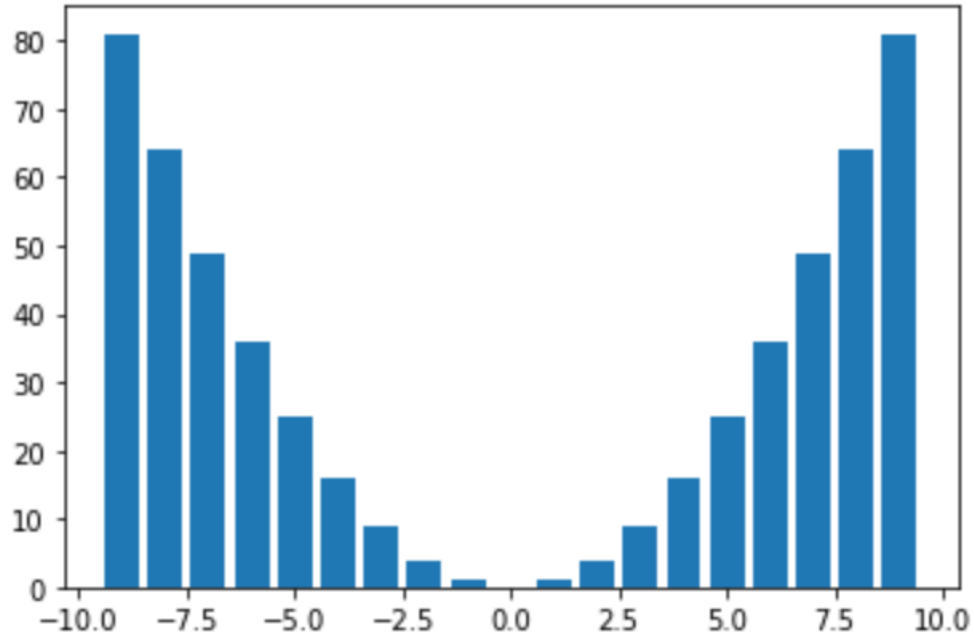


Matplotlib 활용

- 막대 그래프 그리기

```
import matplotlib.pyplot as plt
import numpy as np

x = np.arange(-9, 10)
plt.bar(x, x ** 2)
plt.show()
```

<실행 결과>

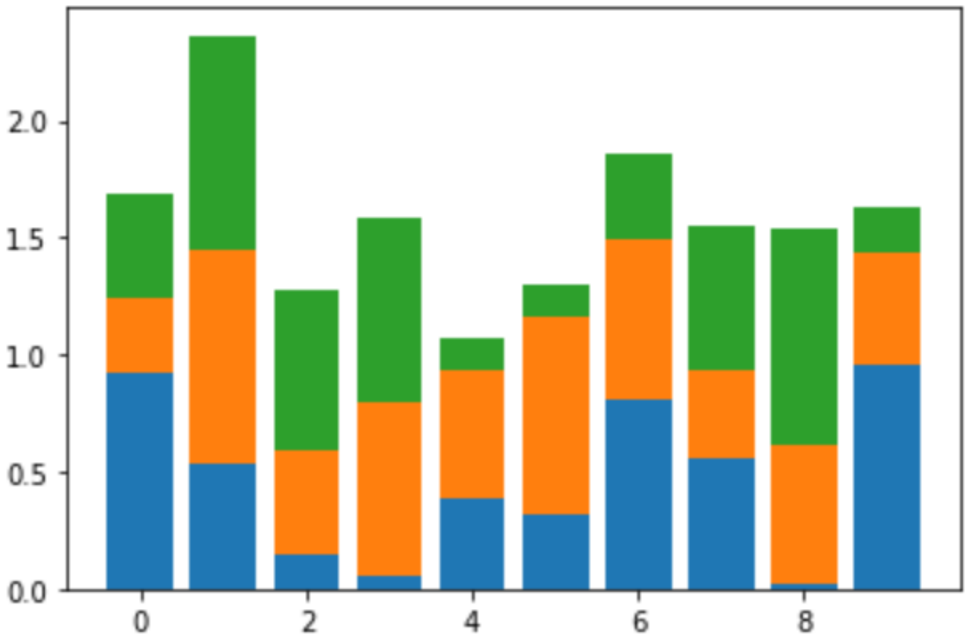


Matplotlib 활용

- 누적 막대 그래프 그리기

```
import matplotlib.pyplot as plt
import numpy as np

x = np.random.rand(10) # 아래 막대
y = np.random.rand(10) # 중간 막대
z = np.random.rand(10) # 위 막대
data = [x, y, z]
x_array = np.arange(10)
for i in range(0, 3): # 누적 막대의 종류가 3개
  plt.bar(
    x_array, # 0부터 10까지의 X 위치에서
    data[i], # 각 높이(10개)만큼 쌓음
    bottom=np.sum(data[:i], axis=0)
  )
plt.show()
```

<실행 결과>

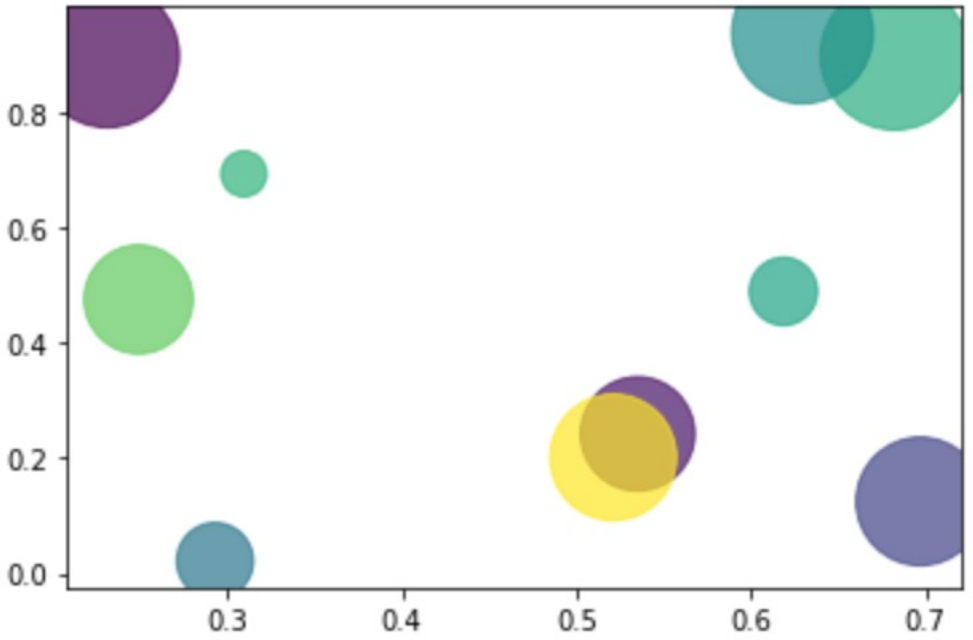


Matplotlib 활용

- 스캐터 그래프 그리기

```
import matplotlib.pyplot as plt
import numpy as np

x = np.random.rand(10)
y = np.random.rand(10)
colors = np.random.randint(0, 100, 10)
sizes = np.pi * 1000 * np.random.rand(10)
plt.scatter(x, y, c=colors, s=sizes, alpha=0.7)
plt.show()
```

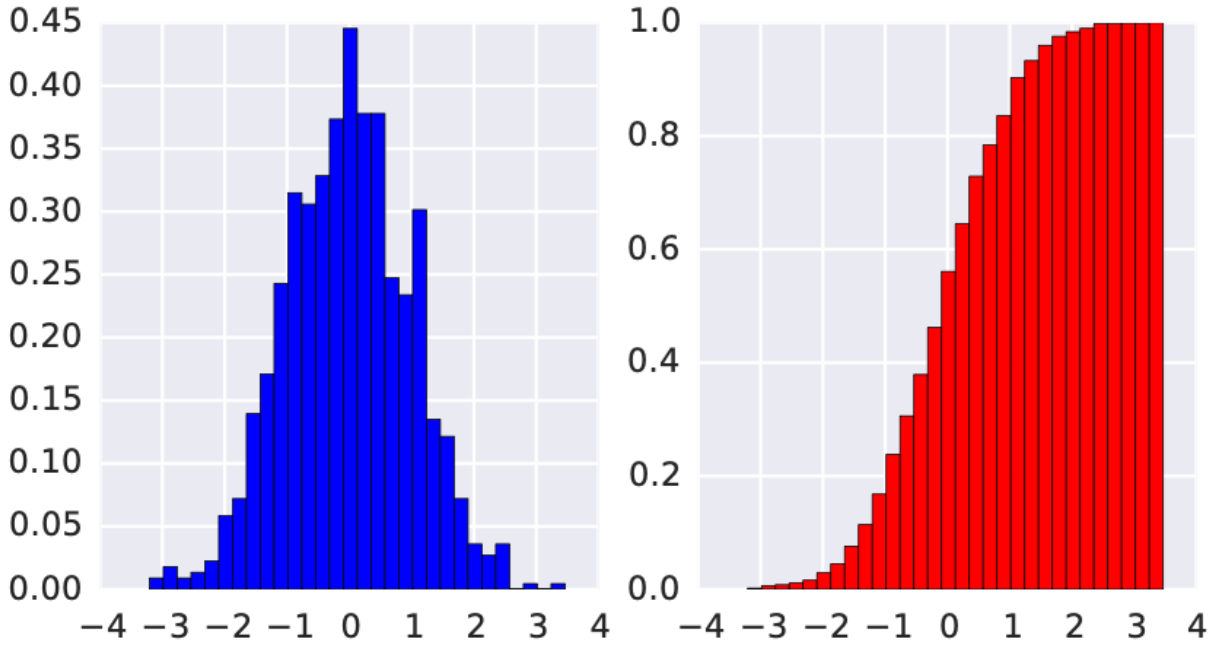
<실행 결과>



Matplotlib 활용

- 히스토그램 그래프 그리기

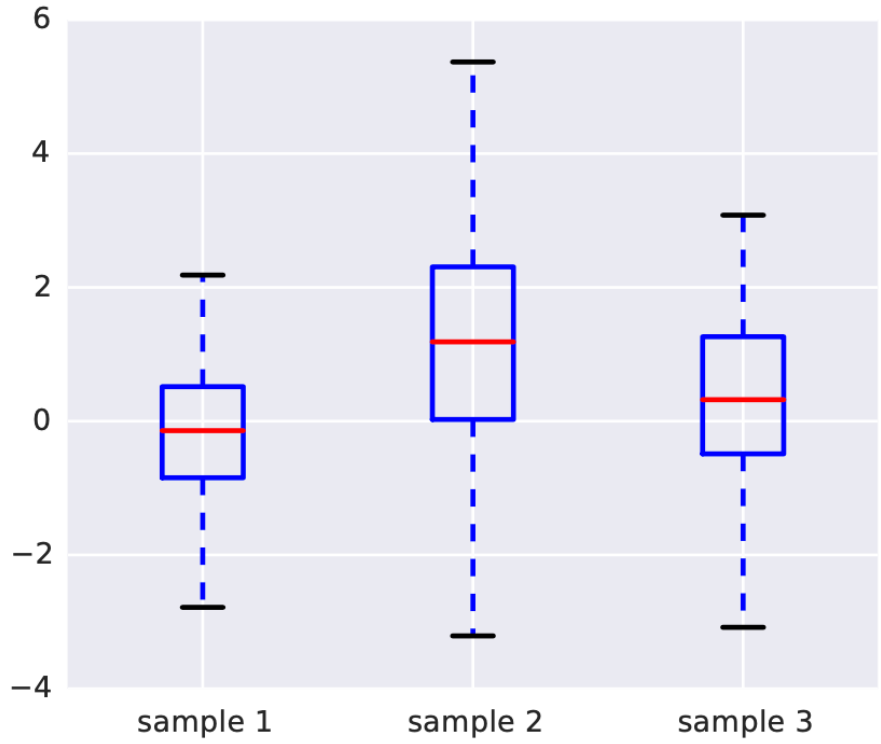
```
data = np.random.randn(1000)

f, (ax1, ax2) = plt.subplots(1, 2, figsize=(6,3))

# histogram (pdf)
ax1.hist(data, bins=30, normed=True, color='b')

# empirical cdf
ax2.hist(data, bins=30, normed=True, color='r',
         cumulative=True)

plt.savefig('histogram.pdf')
```



Matplotlib 활용

- 박스 플롯 그래프 그리기

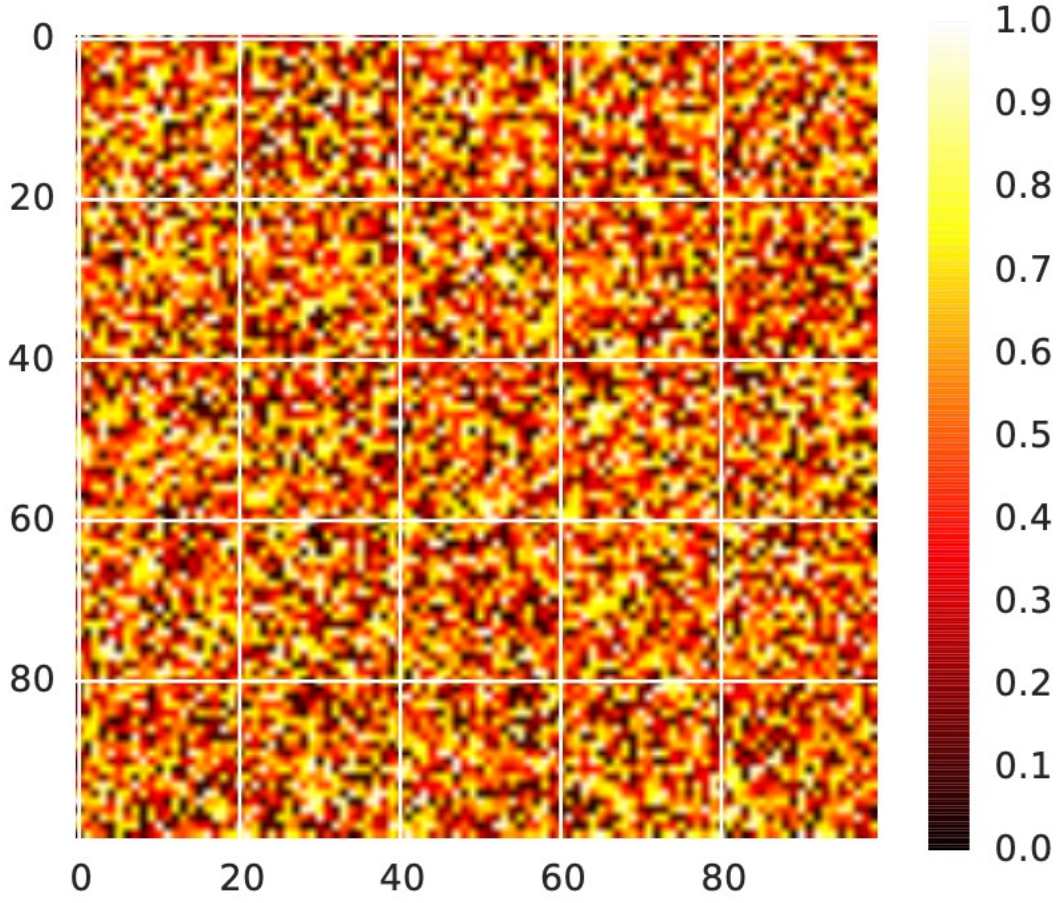
```
samp1 = np.random.normal(loc=0., scale=1., size=100)
samp2 = np.random.normal(loc=1., scale=2., size=100)
samp3 = np.random.normal(loc=0.3, scale=1.2, size=100)

f, ax = plt.subplots(1, 1, figsize=(5,4))

ax.boxplot((samp1, samp2, samp3))
ax.set_xticklabels(['sample 1', 'sample 2', 'sample 3'])
plt.savefig('boxplot.pdf')
```



Matplotlib 활용

- 이미지 플롯 그래프 그리기

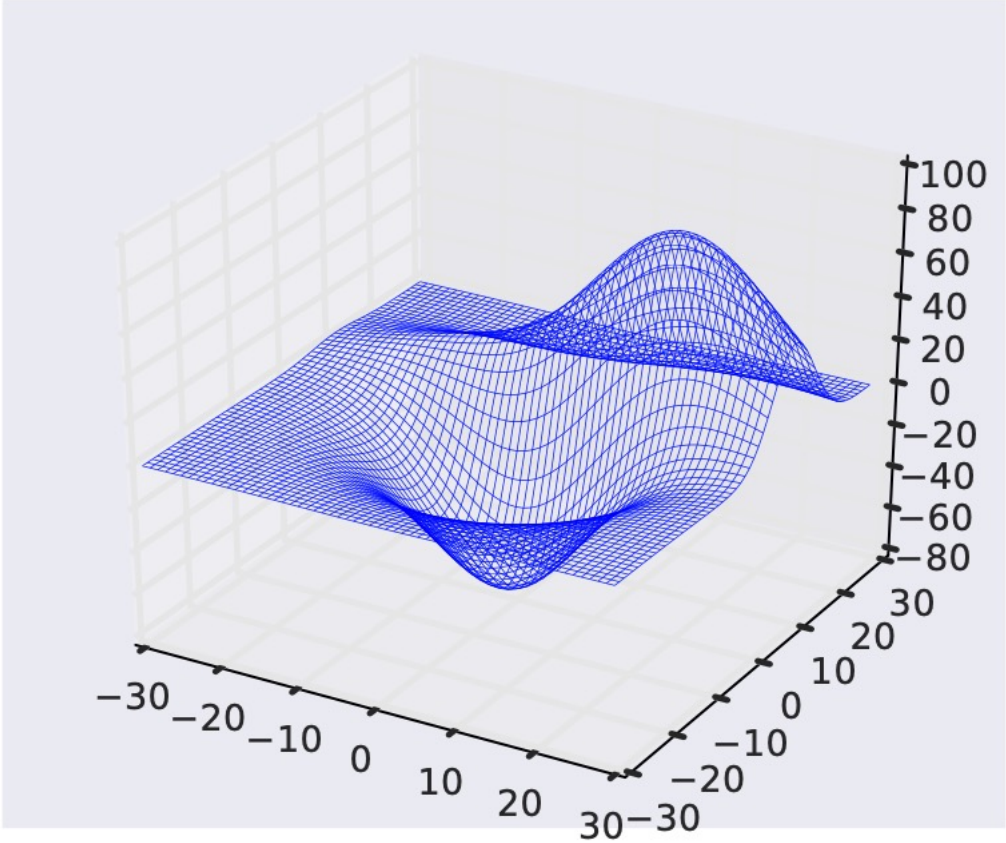
```
A = np.random.random((100, 100))

plt.imshow(A)
plt.hot()
plt.colorbar()

plt.savefig('imageplot.pdf')
```



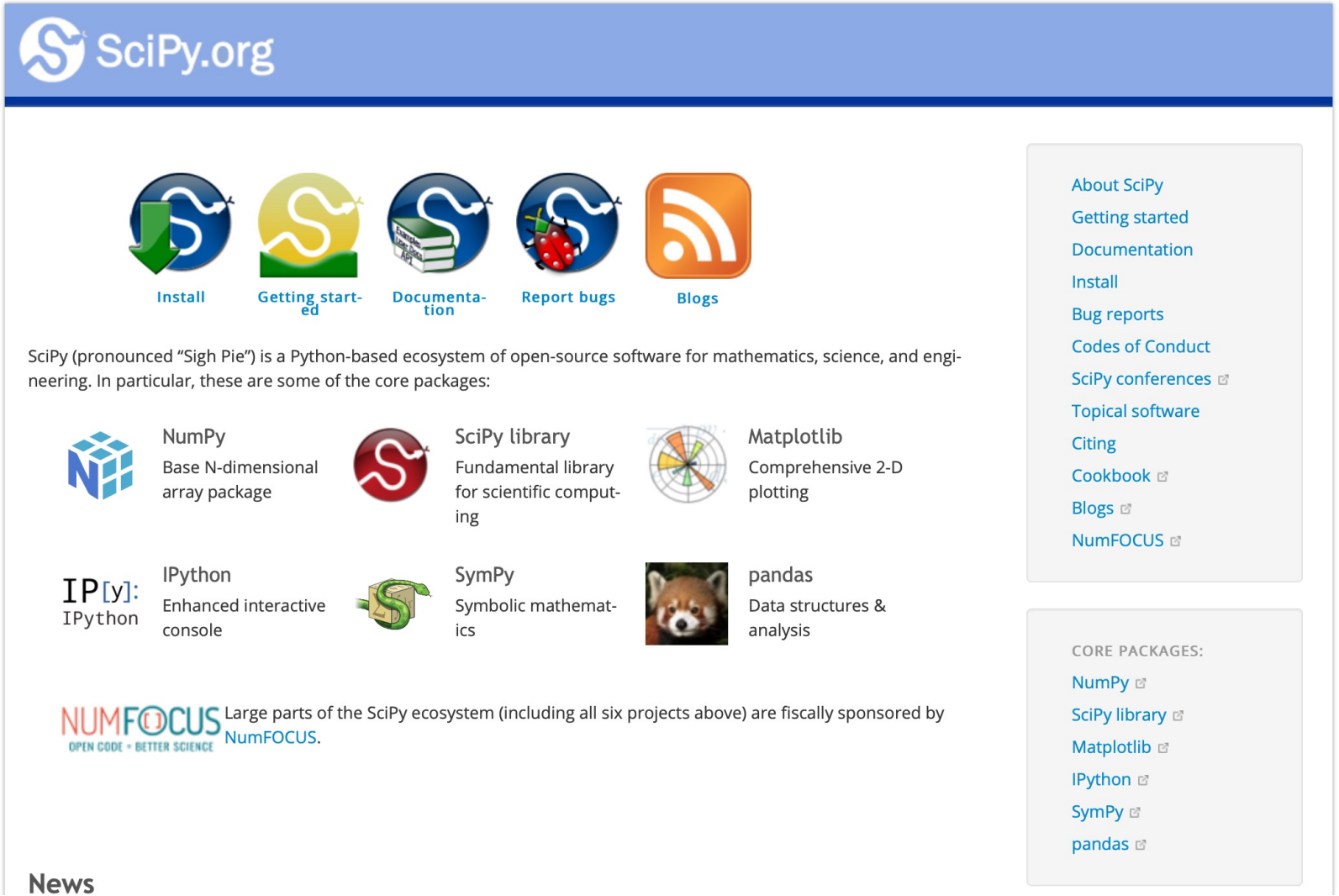
Matplotlib 활용

- 와이어 플롯 그래프 그리기

```
from mpl_toolkits.mplot3d import axes3d

ax = plt.subplot(111, projection='3d')
X, Y, Z = axes3d.get_test_data(0.1)
ax.plot_wireframe(X, Y, Z, linewidth=0.1)

plt.savefig('wire.pdf')
```



SciPy 소개



Source: https://www.scipy.org

SciPy 소개

- SciPy는 과학 계산용 함수를 모아놓은 Python 라이브러리
- SciPy를 사용하기 위해 NumPy 설치는 필수
- 고성능 선형대수, 함수 최적화, 신호처리, 특수한 수학 함수와 통계 분포 등의 기능을 제공
  - linear algebra - scipy.linalg
  - statistics - scipy.stats
  - optimization - scipy.optimize
  - sparse matrices - scipy.sparse
  - signal processing - scipy.signal

Name	Description
cluster	Clustering algorithms
constants	Physical and mathematical constants
fftpack	Fast Fourier Transform routines
integrate	Integration and ordinary differential equation solvers
interpolate	Interpolation and smoothing splines
io	Input and Output
linalg	Linear algebra
ndimage	N-dimensional image processing
odr	Orthogonal distance regression
optimize	Optimization and root-finding routines
signal	Signal processing
sparse	Sparse matrices and associated routines
spatial	Spatial data structures and algorithms
special	Special functions
stats	Statistical distributions and functions

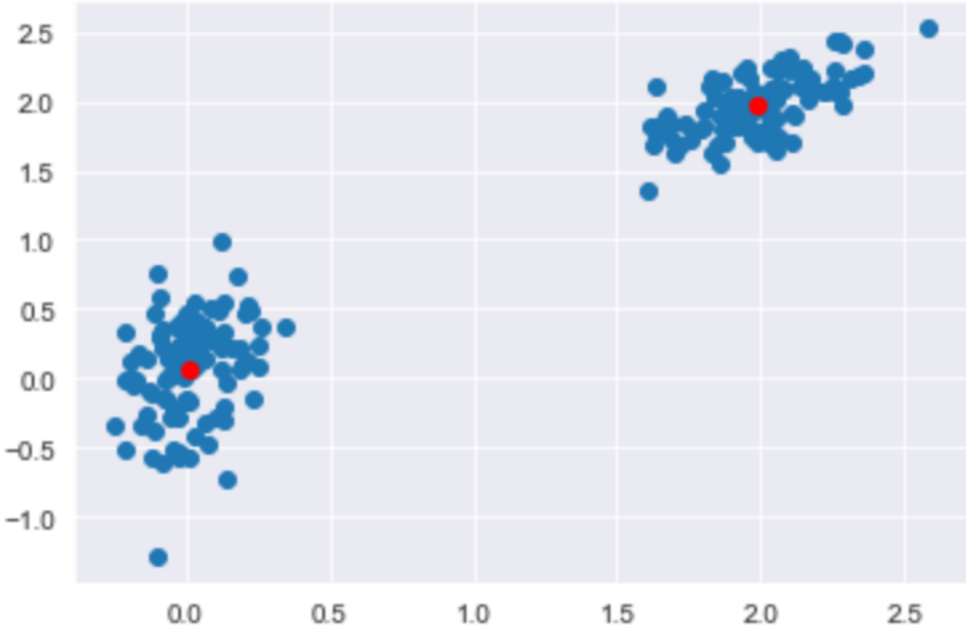
Source: https://www.edureka.co/blog/scipy-tutorial/

SciPy 기본 함수

- SciPy Cluster

```
import numpy as np
from scipy.cluster.vq import  kmeans, whiten
import matplotlib.pyplot as plt
import seaborn as sns

sns.set_style("darkgrid")
# Create 50 datapoints in two clusters a and b
pts = 100
a = np.random.multivariate_normal([0, 0],
                                  [[4, 1], [1, 4]],
                                  size=pts)
b = np.random.multivariate_normal([30, 10],
                                  [[10, 2], [2, 1]],
                                  size=pts)
features = np.concatenate((a, b))
# Whiten data
whitened = whiten(features)
# Find 2 clusters in the data
codebook, distortion = kmeans(whitened, 2)
# Plot whitened data and cluster centers in red
plt.scatter(whitened[:, 0], whitened[:, 1])
plt.scatter(codebook[:, 0], codebook[:, 1], c='r')
plt.show()
```

<실행 결과>



SciPy 기본 함수

- Special Functions

```
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)
```

<실행 결과>

```
1000.0
8.0
1.0
0.7071067811865475
```

SciPy 기본 함수

- SciPy integrate

```
help(integrate)
```

```
    ==========================================

    .. autosummary::
       :toctree: generated/

       quad          -- General purpose integration
       quad_vec      -- General purpose integration of vector-valued functions
       dblquad       -- General purpose double integration
       tplquad       -- General purpose triple integration
       nquad         -- General purpose N-D integration
       fixed_quad    -- Integrate func(x) using Gaussian quadrature of order n
       quadrature    -- Integrate with given tolerance using Gaussian quadrature
       romberg       -- Integrate func using Romberg integration
       quad_explain  -- Print information for use of quad
       newton_cotes  -- Weights and error coefficient for Newton-Cotes integration
       IntegrationWarning -- Warning on issues during integration
       AccuracyWarning  -- Warning on issues during quadrature integration

    Integrating functions, given fixed samples
    ==========================================
```

SciPy 기본 함수

- SciPy integrate

scipy.integrate.quad(f,a,b)

f - Function to be integrated.

a-lower limit.

b- upper limit.

```
from numpy import exp

f = lambda x:exp(-x**2)
print(f)
```

<실행 결과>

```
<function <lambda> at 0x1213ee1f0>
```

```
import scipy

i = scipy.integrate.quad(f, 0, 1)
print(i)
```

<실행 결과>

```
(0.7468241328124271, 8.291413475940725e-15)
```

SciPy 기본 함수

- SciPy integrate

```
import scipy.integrate
from numpy import exp
from math import sqrt

f = lambda x, y : 2*x*y
g = lambda x : 0
h = lambda y : 4*y**2
i = scipy.integrate.dblquad(f, 0, 0.5, g, h)
print(i)
```

<실행 결과>

```
(0.04166666666666667, 5.491107323698757e-15)
```

SciPy 기본 함수

- SciPy optimize
  - Least Square Minimization

```
from scipy.optimize import least_squares
import numpy as np

input = np.array([2, 2])
def rosenbrock(x):
    return np.array([10 * (x[1] - x[0]**3), (1 - x[0])])
res = least_squares(rosenbrock, input)
print(res)
```

<실행 결과>

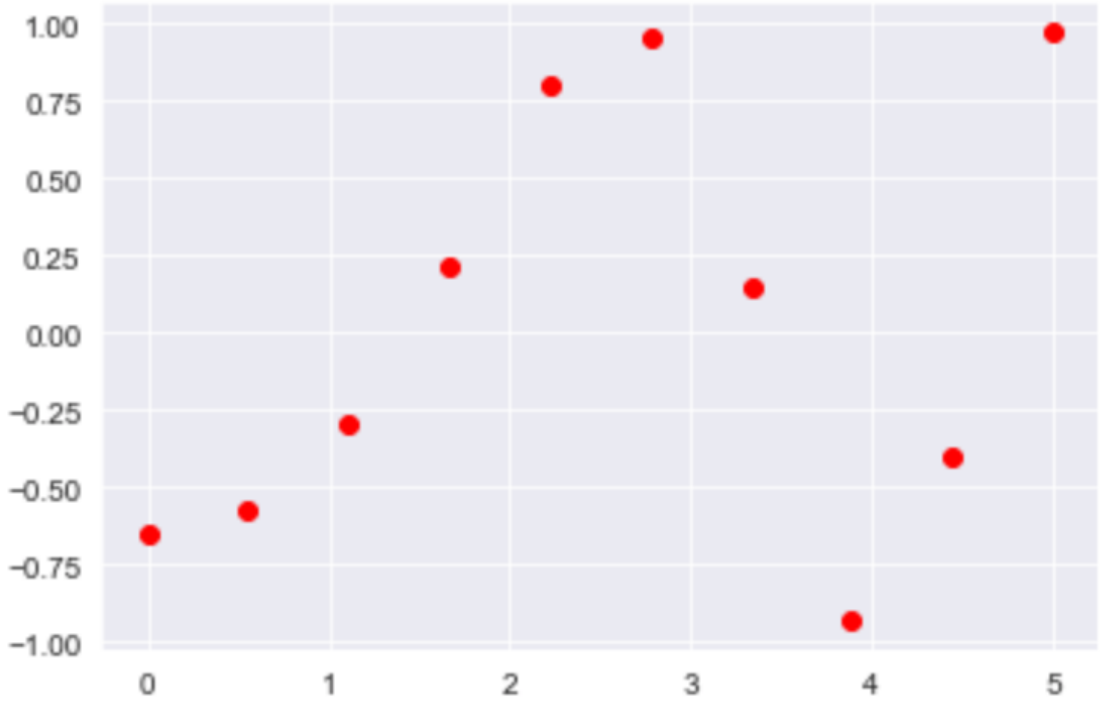
```
 active_mask: array([0., 0.])
        cost: 6.162975822039155e-31
         fun: array([-1.11022302e-15,  0.00000000e+00])
        grad: array([ 3.33066912e-14, -1.11022302e-14])
         jac: array([[-30.00000045,  10.        ],
       [ -1.        ,   0.        ]])
     message: '`gtol` termination condition is satisfied.'
        nfev: 4
        njev: 4
  optimality: 3.3306691235063064e-14
      status: 1
     success: True
           x: array([1., 1.])
```

SciPy 기본 함수

- SciPy Interpolation – 1

```
import numpy as np
from scipy import interpolate
import matplotlib.pyplot as plt

x = np.linspace(0, 5, 10)
y = np.cos(x**2/3+4)
plt.scatter(x,y,c='r')
plt.show()
```

<실행 결과>

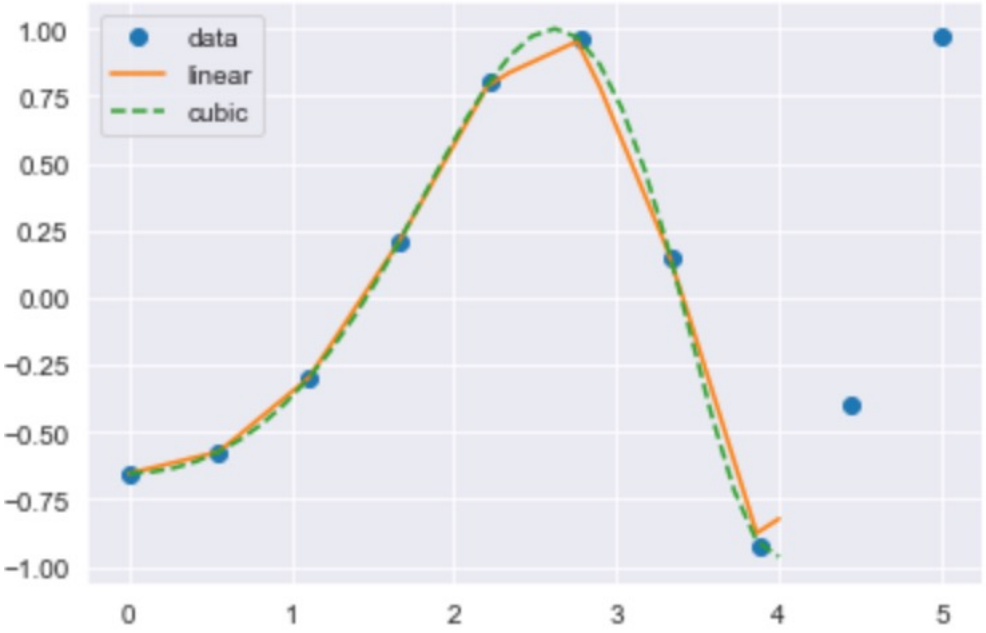


SciPy 기본 함수

- SciPy Interpolation – 1

```
from scipy.interpolate import interp1d
import matplotlib.pyplot as plt

fun1 = interp1d(x, y,kind = 'linear')
fun2 = interp1d(x, y, kind = 'cubic')
#we define a new set of input
xnew = np.linspace(0, 4,30)
plt.plot(x, y, 'o', xnew, fun1(xnew), '-', xnew, fun2(xnew), '--')
plt.legend(['data', 'linear', 'cubic','nearest'], loc = 'best')
plt.show()
```

<실행 결과>

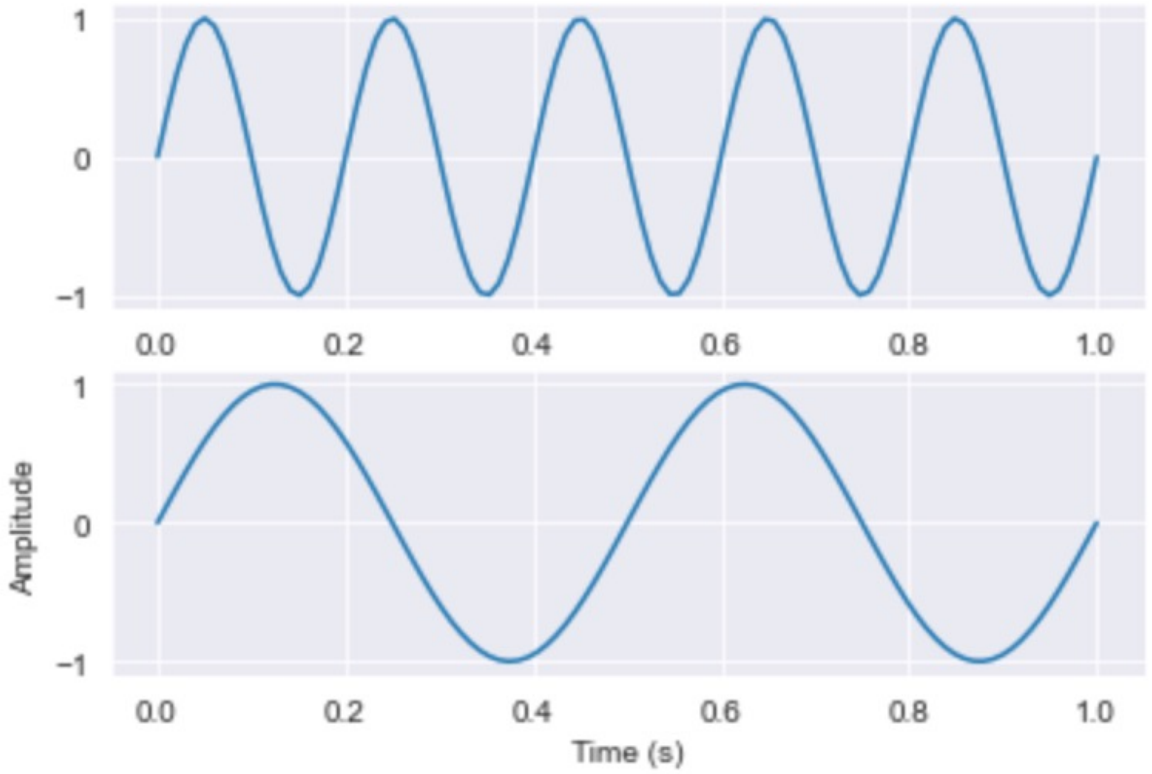


SciPy 기본 함수

- Signal Processing Functions – 1

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

#Time
t = np.linspace(0,1,100)
#Frequency
f1, f2 = 5, 2
#Two signals of different frequencies
first_signal = np.sin(f1*2*np.pi*t)
second_signal = np.sin(f2*2*np.pi*t)

#Plotting both signals
plt.subplot(2,1,1)
plt.plot(t, first_signal)
plt.subplot(2,1,2)
plt.plot(t, second_signal)
plt.ylabel('Amplitude')
plt.xlabel('Time (s)')
```

<실행 결과>

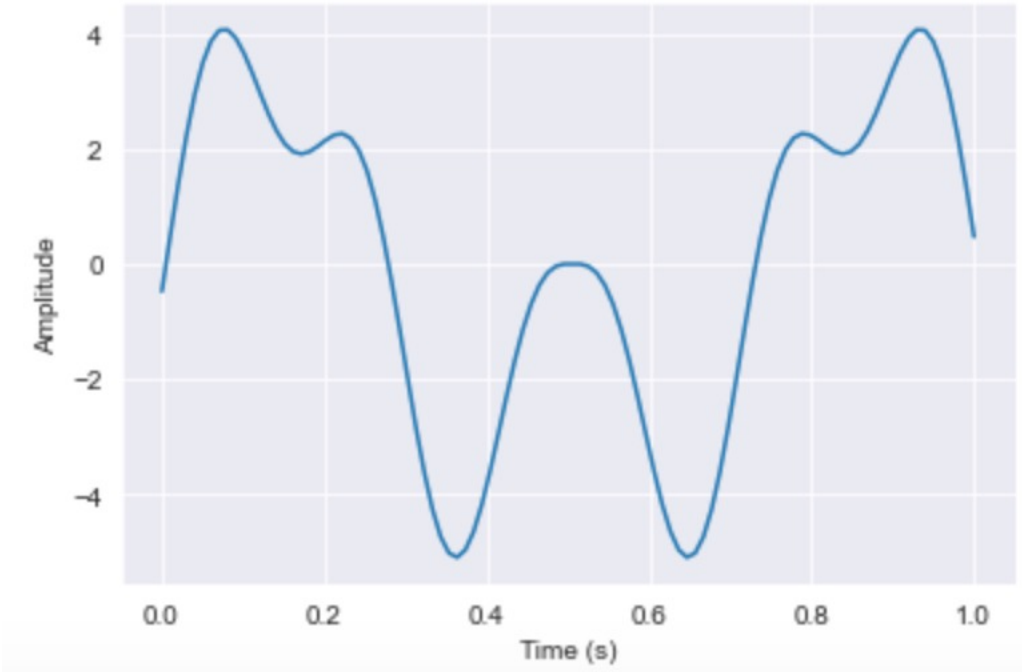


SciPy 기본 함수

- Signal Processing Functions – 1

```
#Convolving two signals
convolution = signal.convolve(first_signal, second_signal, mode='same')

#Plotting the result
plt.plot(t, convolution)
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
```

<실행 결과>



SciPy 기본 함수

- Signal Processing Functions – 2

```
from scipy import signal
import numpy as np

x = np.arange(35).reshape(7, 5)
domain = np.identity(3)
print(x)
print(signal.order_filter(x, domain, 1))
```

<실행 결과>

```
[[ 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]]
[[ 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.]]
```

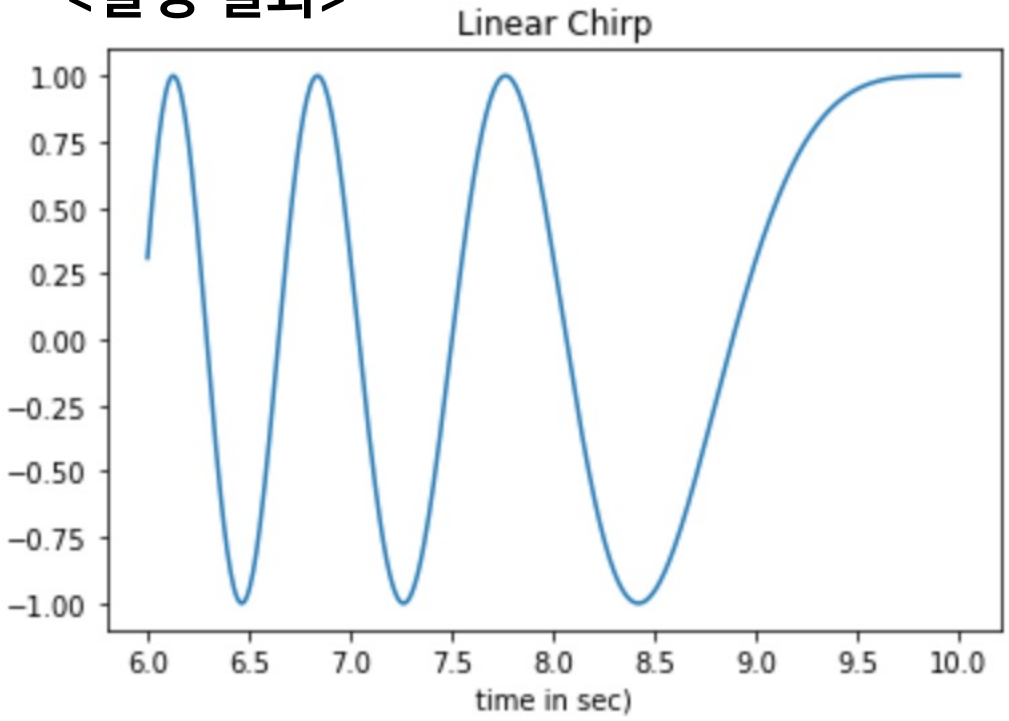
SciPy 기본 함수

- Signal Processing Functions – 3

```
from scipy.signal import chirp, spectrogram
import matplotlib.pyplot as plt
import numpy as np

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()
```

```
t : array_like
    Times at which to evaluate the waveform.
f0 : 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 `t1`.
method : {'linear', 'quadratic', 'logarithmic', 'hyperbolic'}, optional
    Kind of frequency sweep.  If not given, `linear` is assumed.  See
    Notes below for more details.
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.
```

<실행 결과>



SciPy 기본 함수

- SciPy linalg (Linear Algebra)

```
import numpy as np
from scipy import linalg
# We are trying to solve a linear algebra system which can be given as
#         x + 2y - 3z = -3
#         2x - 5y + 4z = 13
#         5x + 4y - z = 5
#We will find values of x,y and z for which all these equations are zero
#Also finally we will check if the values are right by substituting them
#in the equations
# Creating input array
a = np.array([[1, 2, -3], [2, -5, 4], [5, 4, -1]])
# Solution Array
b = np.array([[-3], [13], [5]])
# Solve the linear algebra
x = linalg.solve(a, b)

# Print results
print(x)

# Checking Results
print("\n Checking results,must be zeros")
print(a.dot(x) - b)
```

<실행 결과>

```
[[ 2.]
 [-1.]
 [ 1.]]

 Checking results,must be zeros
[[0.]
 [0.]
 [0.]]
```

SciPy 기본 함수

- SciPy linalg (Linear Algebra)
  - Finding a determinant of a square matrix

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$|A| = ad - bc$$

$$A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

$$|A| = a(ei - fh) - b(di - fg) + c(dh - eg)$$

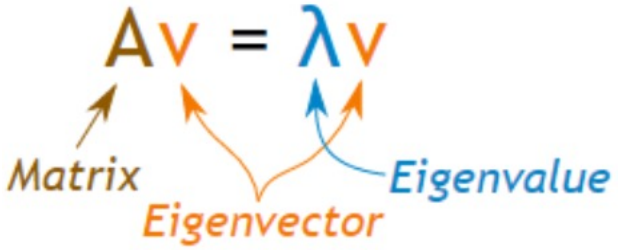
```
from scipy import linalg
import numpy as np

#Declaring the numpy array
A = np.array([[1,2,9],[3,4,8],[7,8,4]])
#Passing the values to the det function
x = linalg.det(A)

#printing the result
print('Determinant of \n{} \n is {}'.format(A,x))
```

<실행 결과>

```
Determinant of
[[1 2 9]
 [3 4 8]
 [7 8 4]]
 is 3.999999999999986
```

SciPy 기본 함수

- SciPy linalg (Linear Algebra)
  - Eigenvalues and Eigenvectors




```
from scipy import linalg
import numpy as np

#Declaring the numpy array
A = np.array([[2,1,-2],[1,0,0],[0,1,0]])

#Passing the values to the eig function
values, vectors = linalg.eig(A)

#printing the result for eigenvalues
print(values)

#printing the result for eigenvectors
print(vectors)
```

<실행 결과>

```
[-1.+0.j  2.+0.j  1.+0.j]
[[-0.57735027 -0.87287156  0.57735027]
 [ 0.57735027 -0.43643578  0.57735027]
 [-0.57735027 -0.21821789  0.57735027]]
```

SciPy 기본 함수

- SciPy linalg (Linear Algebra)
  - Finding the inverse of a matrix

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad-bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

determinant

```
import numpy as np
from scipy import linalg

A = np.array([[1,2], [4,3]])
B = linalg.inv(A)
print(B)
```

<실행 결과>

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

Matplotlib & SciPy 유용한 참고자료

- Matplotlib Tutorials
  - https://matplotlib.org/stable/tutorials/index.html
  - https://github.com/rougier/matplotlib-tutorial
- SciPy Reference Guide
  - https://docs.scipy.org/doc/scipy/reference/index.html
  - https://docs.scipy.org/doc/scipy-0.13.0/scipy-ref.pdf