SciPy

Es un conjunto de librerías para el cálculo científico, e incluye herramientas de software para matemáticas, ciencia e ingeniería. La siguiente tabla da una idea de lo amplia que es esta colección de librerías:

Módulo	Contenido
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

Además de ello, pueden considerarse los siguientes módulos incluidos en scipy, aunque por su relevancia se suelen estudiar aparte:

Módulo	Contenido
NumPy	Paquete básico de arrays n-dimensionales
Matplotlib	Trazado de gráficos bidimensionales
IPython	Consola interactiva mejorada
SymPy	Matemática simbólica
Pandas	Estructuras y análisis de datos

Funciones estadísticas

El módulo `stats` contiene una gama amplia de distribuciones de probabilidad y de funciones estadísticas. La página de referencia es la siguiente:

```
https://docs.scipy.org/doc/scipy/reference/stats.html
```

En esta página se puede ver las funciones y distribuciones disponibles. Para introducirse en ellas, lo mejor es cargar el modulo y solicitar información sobre una distribución o función cualquiera:

In [*]:

```
from scipy import stats

help(stats.norm) # distibución normal
```

In [*]:

```
# Distribución normal

from scipy.stats import norm
mean, var, skew, kurt = norm.stats(moments='mvsk')
print("Media, varianza, sesgo, kurtosis = ", mean, var, skew, kurt)
print("Probabilidad acumulada en el punto de abcisa 0 = ", norm.cdf(0))
```

A manera de referencia, todas las funciones incluidas en `SciPy.stats.norm` pueden verse en la siguiente dirección:

`https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.norm.html`

Helas aquí:

Uso	Función
rvs(loc=0, scale=1, size=1, random_state=None)	Random variates.
pdf(x, loc=0, scale=1)	Probability density function.
logpdf(x, loc=0, scale=1)	Log of the probability density function.
cdf(x, loc=0, scale=1)	Cumulative distribution function.
logcdf(x, loc=0, scale=1)	Log of the cumulative distribution function.
sf(x, loc=0, scale=1)	Survival function (also defined as 1 - cdf, but sf is sometimes more accurate).
logsf(x, loc=0, scale=1)	Log of the survival function.
ppf(q, loc=0, scale=1)	Percent point function (inverse of cdf — percentiles).
isf(q, loc=0, scale=1)	Inverse survival function (inverse of sf).
moment(n, loc=0, scale=1)	Non-central moment of order n
stats(loc=0, scale=1, moments='mv')	Mean('m'), variance('v'), skew('s'), and/or kurtosis('k').
entropy(loc=0, scale=1)	(Differential) entropy of the RV.
fit(data, loc=0, scale=1)	Parameter estimates for generic data.
expect(func, args=(), loc=0, scale=1, lb=None, ub=None, conditional=False, kwds)	Expected value of a function (of one argument) with respect to the distribution.
median(loc=0, scale=1)	Median of the distribution.
mean(loc=0, scale=1)	Mean of the distribution.
var(loc=0, scale=1)	Variance of the distribution.
std(loc=0, scale=1)	Standard deviation of the distribution.
interval(alpha, loc=0, scale=1)	Endpoints of the range that contains alpha percent of the distribution

Vamos a probar con una de ellas, `cdf` (*Cumulative distribution function*), a manera de ejemplo:

In [*]:

```
import numpy as np
xs = np.linspace(-3.,3.,20)
ys = norm.cdf(xs) # probabilidad acumulada en una colección de puntos
print(xs)
print(ys)
```

Veámoslo gráficamente:

In [*]:

```
import matplotlib.pyplot as plt
plt.plot(xs, ys)
plt.show()
```

In [*]:

```
ys_3_2 = norm.cdf(xs, loc=3, scale=2)  # media 3, desviación 2.
print(ys_3_2)
```

La gráfica sube ahora hasta 0.5. Es lógico, porque la distribución se centra en 3, que es hasta donde llegan las abcisas.

In [*]:

```
import matplotlib.pyplot as plt
plt.plot(xs, ys_3_2)
plt.show()
```

In [*]:

```
# Variables aleatorias (dentro de la distribución normal)
# rvs(loc=0, scale=1, size=1, random_state=None)

norm.rvs(loc= 100, scale=1, size=10) # genera 10 números aleatorios, centrados en 100 y con
```

In [*]:

```
# Los parámetros son los definidos por defecto:
np.random.seed(1000)
norm.rvs(size=10)
```

In [*]:

```
# Cada ejecución es distinta, salvo cuando nosotros deseamos reproducir un experimento idén
np.random.seed(1000)
norm.rvs(size=10)
```

Optimización

Este módulo es muy flexible. Centrándonos únicamente en la función que busca un mínimo, el módulo permite aplicar métodos variasídimos de optimización sin y con restricciones, y parametrizarse para adaptarse a la multitud de situaciones que se pueden presentar.

In [*]:

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

f = lambda x: np.sin(x) * np.exp(-abs(x/10))
sol = optimize.minimize(f, 0.5)
x = np.linspace(-10, 10, 5000)
plt.plot(x, f(x), '-', sol.x, sol.fun, 'o')
plt.show()
```

Optimización de funciones de 2 variables

In [*]:

```
# http://scipy-lectures.org/intro/scipy/auto_examples/plot_2d_minimization.html

# Define the function that we are interested in
def sixhump(x):
    return ((4 - 2.1*x[0]**2 + x[0]**4 / 3.) * x[0]**2 + x[0] * x[1]
            + (-4 + 4*x[1]**2) * x[1] **2)

# Make a grid to evaluate the function (for plotting)
x = np.linspace(-2, 2)
y = np.linspace(-1, 1)
xg, yg = np.meshgrid(x, y)

from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
surf = ax.plot_surface(xg, yg, sixhump([xg, yg]), rstride=1, cstride=1, cmap=plt.cm.jet, li

ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('f(x, y)')
ax.set_title('Six-hump Camelback function')

from scipy import optimize

x_min = optimize.minimize(sixhump, x0=[0, 0])

plt.figure()

# Show the function in 2D
plt.imshow(sixhump([xg, yg]), extent=[-2, 2, -1, 1])
plt.colorbar()
# And the minimum that we've found:
plt.scatter(x_min.x[0], x_min.x[1])

plt.show()
```

Interpolación

In [*]:

```
# http://scipy-lectures.org/intro/scipy/auto_examples/plot_interpolation.html

# Generate data
import numpy as np
np.random.seed(0)
measured_time = np.linspace(0, 1, 10)
noise = 1e-1 * (np.random.random(10)*2 - 1)
measures = np.sin(2 * np.pi * measured_time) + noise

# Interpolate it to new time points
from scipy.interpolate import interp1d
linear_interp = interp1d(measured_time, measures)
interpolation_time = np.linspace(0, 1, 50)
linear_results = linear_interp(interpolation_time)
cubic_interp = interp1d(measured_time, measures, kind='cubic')
cubic_results = cubic_interp(interpolation_time)

# Plot the data and the interpolation
from matplotlib import pyplot as plt
plt.figure(figsize=(6, 4))
plt.plot(measured_time, measures, 'o', ms=6, label='measures')
plt.plot(interpolation_time, linear_results, label='linear interp')
plt.plot(interpolation_time, cubic_results, label='cubic interp')
plt.legend()
plt.show()
```

Ajuste de curvas

In [*]:

```
# http://scipy-lectures.org/intro/scipy/auto_examples/plot_curve_fit.html

import numpy as np
import matplotlib.pyplot as plt

# Empezamos por generar algunos datos:
x_data = np.linspace(-5, 5, num=50)
y_data = 2.9 * np.sin(1.5 * x_data) + np.random.normal(size=50)

# Y los representamos:
plt.figure(figsize=(6, 4))
plt.scatter(x_data, y_data)
plt.show()
```

In [*]:

```
from scipy import optimize

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

params, params_covariance = \
    optimize.curve_fit(test_func, x_data, y_data, p0=[2, 2])

print(params)
print(params_covariance)
```

In [*]:

```
plt.figure(figsize=(6, 4))
plt.scatter(x_data, y_data, label='Datos')
plt.plot(x_data, test_func(x_data, params[0], params[1]),
         label='Función de ajuste')

plt.legend(loc='best')

plt.show()
```

Álgebra lineal

In [*]:

```
from scipy import linalg
a = np.array([[1,3,5],[2,5,1],[2,3,8]])
a
```

In [*]:

```
# Determinante:
print("|A| = ", linalg.det(a))
# Inversión de matrices:
linalg.inv(a)
```

In [*]:

```
# Producto de matrices:
a @ linalg.inv(a)
```

Solución de sistemas de ecuaciones:

$$\begin{pmatrix} 1 & 3 & 5 \\ 2 & 5 & 1 \\ 2 & 3 & 8 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 22 \\ 15 \\ 32 \end{pmatrix}$$

In [*]:

```
a = np.array([[1, 3, 5],
       [2, 5, 1],
       [2, 3, 8]])
#b = np.array([[10],[8],[3]])
b = np.array([[22],[15],[32]])
linalg.solve(a, b)
```

Referencias

La referencia principal, obligada, es la siguiente:

    https://www.scipy.org/