Librería Scipy (Introducción)

Antonio Espín Herranz

Contenidos

- Visión general de la librería

- Algoritmos:
  - Estadísticos
  - Algebra lineal
  - Procesado de señal
  - Lectura / Escritura

Visión general de la Librería

- Scipy (librería científica) es una colección de algoritmos matemáticos basados en una extensión de la librería Numpy.
- Proporciona comandos y clases de alto nivel para manipular y visualizar datos.
- Se utiliza sobre todo en aplicaciones científicas.
- Trabaja conjuntamente con numpy y matplotlib

Módulos scipy

import scipy
help(scipy)

```
cluster                      --- Vector Quantization / Kmeans
fftpack                      --- Discrete Fourier Transform algorithms
integrate                    --- Integration routines
interpolate                  --- Interpolation Tools
io                           --- Data input and output
lib                          --- Python wrappers to external libraries
lib.lapack                   --- Wrappers to LAPACK library
linalg                       --- Linear algebra routines
misc                         --- Various utilities that don't have
                                 another home.
ndimage                      --- n-dimensional image package
odr                          --- Orthogonal Distance Regression
optimize                     --- Optimization Tools
signal                       --- Signal Processing Tools
sparse                       --- Sparse Matrices
sparse.linalg                --- Sparse Linear Algebra
sparse.linalg.dsolve         --- Linear Solvers
sparse.linalg.dsolve.umfpack --- :Interface to the UMFPACK library:
                                 Conjugate Gradient Method (LOBPCG)
sparse.linalg.eigen.lobpcg   --- Locally Optimal Block Preconditioned
                                 Conjugate Gradient Method (LOBPCG) [*]
special                      --- Airy Functions [*]
lib.blas                     --- Wrappers to BLAS library [*]
sparse.linalg.eigen          --- Sparse Eigenvalue Solvers [*]
stats                        --- Statistical Functions [*]
lib                          --- Python wrappers to external libraries
                                 [*]
```

Estadísticos

- El módulo es: **scipy.stats**

- Este módulo contiene herramientas estadísticas y descripciones probabilísticas de procesos aleatorios.

- Nos apoyaremos en el módulo de **numpy.random** que disponemos de generadores de números aleatorios.

Distribuciones: histograma y función de densidad de probabilidad

- **scipy.stats.norm** es un objeto de distribución.
  - Cada distribución en scipy.stats se representa **como un objeto**.

- Podemos generar valores aleatorios de una distribución normal a partir del modulo **numpy.random.normal**.

- Generar histogramas con **numpy.histogram**.

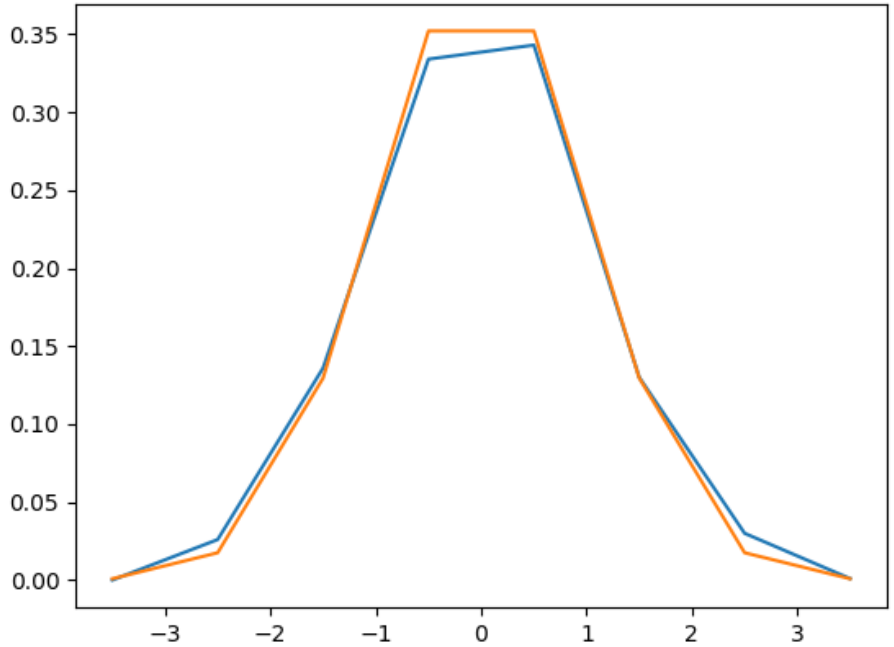
Ejemplo

- import numpy as np
- from scipy import stats
- import matplotlib.pyplot as plt

- **ejemplos = np.random.normal(size=1000)**
- bins = np.arange(-4, 5)

- histograma = np.histogram(ejemplos, bins=bins, density=True)[0]
- bins = 0.5 * (bins[1:] + bins[:-1]) **# Generar las divisiones**
- **pdf = stats.norm.pdf(bins) # pdf → function density probability**

- plt.plot(bins, histograma)
- plt.plot(bins, pdf)
- plt.show()



Media, mediana y percentiles

- También podemos calcular la **media** y **mediana** a partir de **numpy**:
- import numpy as np
- from scipy import stats

- ejemplos = np.random.normal(size=1000)
- media = np.mean(ejemplos)
- mediana = np.median(ejemplos)

- print("Media:", media)
- print("Mediana:", mediana)

- # La mediana es también el percentil de 50
- mediana2 = **stats.scoreatpercentile**(ejemplos, 50)
- print("Mediana:", mediana2)

Comparar distribuciones normales stats.ttest_ind()

- **Una prueba estadística es un indicador de decisión:**
  - Por ejemplo, si tenemos dos conjuntos de observaciones, que suponemos que se generan a partir de procesos gaussianos. Podemos usar una prueba T para decidir si las medias de dos conjuntos de observaciones son significativamente diferentes:

- La función **stats.ttest_ind**(a, b) devuelve una tupla con:
  - **El valor estadístico T**: es un número cuyo signo es proporcional a la diferencia entre los dos procesos aleatorios y la magnitud están relacionados con la importancia de esta diferencia.

  - **El valor p**: la probabilidad de que ambos procesos sean idénticos. Si está cerca de 1, los dos procesos son casi ciertamente idénticos
    - Cuanto más cerca esté de cero, más probable es que los procesos sean diferentes.

Algebra lineal

- Scipy para el álgebra dispone del módulo: **scipy.linalg**

- Dentro del módulo de álgebra permite realizar operaciones con matrices como son:

  - Se suele trabajar con tipos de la librería **Numpy** (tipo array)
    - Podemos trabajar con **np.array** de más de una dimensión
    - Y también con **np.mat** (tipo de matriz más apropiado soporta el operador * y métodos como T e I para trasponer e invertir).

  - Dentro de numpy también hay un módulo destinado al cálculo algebraico: **numpy.linalg** pero el de scipy contiene todo las funciones presentes en numpy y algunas mas.

Funciones scipy.linalg

- Calcular el determinante: **det**
- Matriz inversa: **inv**
- Resolución de ecuaciones lineales: **solve**
- Calcular la norma de una matriz y un vector: **norm**

Algebra lineal: Determinante

 # Dispone de la función det recibe por parámetro una matriz cuadrada:

```
import numpy as np
from scipy import linalg

# Creamos una matriz:
arr = np.array([[1,2],[3,4]])
determinante = linalg.det(arr)
print(arr)
print("Determinante:", determinante)
```

 # Si la matriz no es cuadrada lanzará una excepción.

Algebra lineal: Inversa

```
from scipy import linalg
import numpy as np

arr = np.array([[1,2],[3,4]])
iarr = linalg.inv(arr)

print("Matriz:")
print(arr)
print("Inversa:")
print(iarr)
```

```
# Utilizamos el tipo: np.matriz
mat = np.mat('[1 2;3 4]')
print(mat)
inversa = linalg.inv(mat)
print(inversa)

inv = mat.I
print("Inversa con numpy:")
print(inv)
```

Resolución de ecuaciones lineales: solve

- Partimos de un sistema de ecuaciones:

3x + 4y = 5

2x - y = 7

```
import numpy as np
from scipy import linalg
a = np.array([[3,4],[2,-1]])
 b = np.array([5,7])
 x = linalg.solve(a,b)
 print("Resultado: ")
 print(x)
```

Calcular la norma de un vector o matriz

```
import numpy as np
from scipy import linalg

a = np.array([[1,2],[3,4]])
norma = linalg.norm(a)
print(a)
print("norma:", norma)
```

Procesado de la señal

- Para esto se dispone del módulo: **scipy.signal**
- Este módulo proporciona funciones de filtrado, remuestreo y tendencia de la señales.
- Se considera que en una señal en Scipy es una matriz de número reales o complejos.

Remuestreo y tendencia de una señal

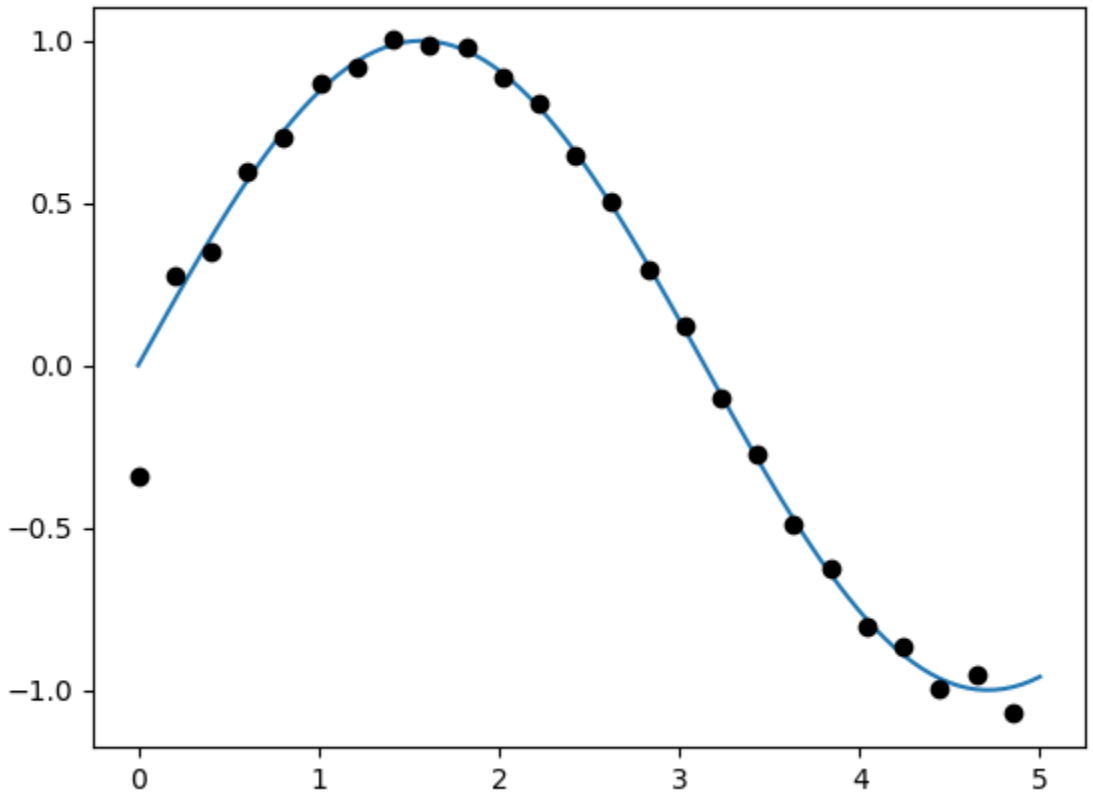
- Dentro del módulo **scipy.signal** disponemos de la función:

  - **scipy.signal.resample**(x, num)
    - Por debajo utiliza la transformada rápida de Fourier (*FFT*)
    - Se asume que la señal x es periódica
    - El parámetro num es el número de ejemplos de la señal remuestreada.

  - **scipy.signal.detrend**(x)
    - Eliminar la tendencia lineal de una señal.
    - Se pasa por parámetro x .

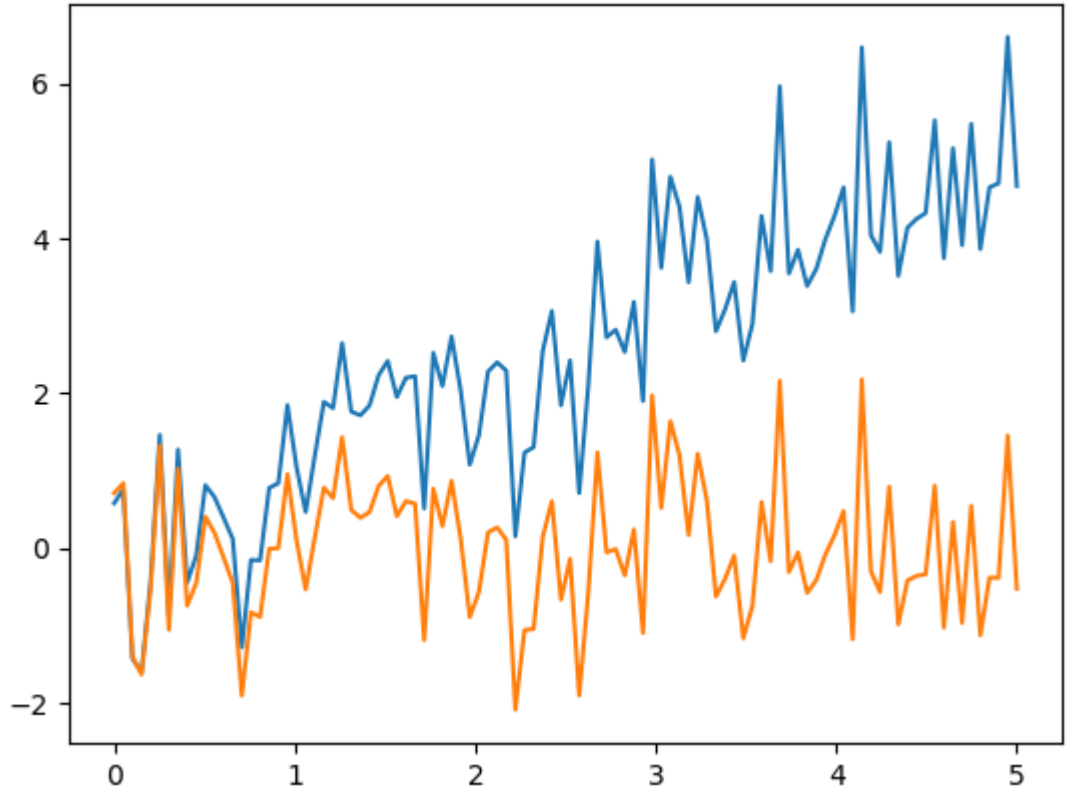
Ejemplo: remuestreo

```
t = np.linspace(0, 5, 100)
x = np.sin(t)
x_resample = signal.resample(x, 25)
plt.plot(t, x)
# Toma de 4 en 4 porque partimos de 100, 100/4 = 25
plt.plot(t[::4], x_resample, 'ko')
# k --> black | o --> circulo relleno
plt.show()
```



Ejemplo: tendencia

- t = np.linspace(0,5,100)
- x = t + np.random.normal(size=100)
- x_detrended = **signal.detrend**(x)
- plt.plot(t, x)
- plt.plot(t, x_detrended)
- plt.show()



Mas funciones para el filtrado de señales

- Calcular la transformada de Fourier
- from scipy import fftpack

- **<u>Funciones:</u>**
  - fftpack.**fft**(signal)
    - Calcular la transformada de Fourier de la señal original
    - Devuelve la transformada de Fourier de una señal.
  - fftpack.**fftfreq**(signal.size, d=time_step)
    - Calcular las frecuencias discretas de la transformada de Fourier.
    - El tamaño de la señal y el espaciado.
  - fftpack.**ifft**(signal_fft)
    - Obtener la inversa de la transformada de Fourier
    - Se utiliza para obtener una señal filtrada a partir de las frecuencias más altas de la tranformada de Fourier.

Mas funciones para el filtrado de señales

- scipy.signal.**kaiserord**(ripple, width):
  - https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.kaiserord.html
  - Determina los parámetros del filtro para el método de la ventana de Kaiser
  - Los parámetros devueltos por esta función generalmente se usan para crear un filtro de respuesta de impulso finito (FIR) usando el método de ventana **firwin**.

  - **Parámetros:**
    - **ripple: float**. Límite superior para la desviación (en dB) de la magnitud de la respuesta de frecuencia del filtro con respecto a la del filtro deseado (sin incluir las frecuencias en ningún intervalo de transición). Es decir, si w es la frecuencia expresada como una fracción de la frecuencia de Nyquist, A (w) es la respuesta de frecuencia real del filtro y D (w) es la respuesta de frecuencia deseada, el requisito de diseño es que:
      - abs(A(w) - D(w))) < 10**(-ripple/20)

    - **width: float**. El ancho. Normalizado de modo que 1 corresponde a PI radianes / muestra. La frecuencia se expresa como una fracción de la frecuencia de Nyquist

Mas funciones para el filtrado de señales

- scipy.signal.**firwin**:
  - https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.firwin.html
  - La función diseña un filtro FIR utilizando un método de ventana:
  - Calcula los coeficientes de un filtro de respuesta de impulso finito.

  - **Parámetros:**
    - numtaps: número de coeficientes.
    - Cutoff: Frecuencia de corte del filtro

    - window: la ventana por defecto es haming.

  - Con la función **scipy.signal.get_window** podemos obtener una lista con todos los tipos de ventana que hay:
    - https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.get_window.html#scipy.signal.get_window

Mas funciones para el filtrado de señales

- scipy.signal.**lfilter**(b,a,x):
  - https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.lfilter.html
  - Filtra datos en una dimensión con un filtro IIR o FIR.
  - Filtra una secuencia de datos x, utilizando filtro digital.

  - **Parámetros:**
    - b = array_like (un array de numpy, por ejemplo)
      - El vector coeficiente del numerador en una secuencia 1-D

    - a = array_like
      - El vector coeficiente del denominador en una secuencia 1-D

    - X = array_like
      - Un array N-dimensional

  - Filtro IIR: Respuesta infinita al impulso
  - Filtro FIR: Respuesta finita al impulso

Mas funciones para el filtrado de señales

- scipy.signal.**freqz**:
  - https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.freqz.html
  - Computa la frecuencia de respuesta de un filtro digital.

  - **Parámetros:**
    - b: Coeficientes obtenidos del filtro.
    - a: 1. Denominador de un filtro lineal
    - wordN = 512

Lectura / Escritura

- Para este cometido disponemos en **scipy** del módulo **scipy.io**
- Permite lectura / escritura de archivos de Matlab
- De imágenes, datos binarios, de texto
- Nos apoyaremos en funciones de numpy

Lectura / Escritura

- Aporta las funciones **savemat** y **loadmat** para ficheros **MATLAB**

- Ejemplo de uso:

```
import scipy.io as sio
import numpy as np

a = np.ones((3,3))
print(a)

sio.savemat('file.mat', {'a':a})
print('Fichero grabado ...')

a2 = sio.loadmat('file.mat')
print('Contenido del fichero:')
print(a2['a'])
```

Lectura / Escritura

- Otras posibilidades para leer ficheros:
  - Dentro de la librería numpy tenemos para **texto**:
    - numpy.**loadtxt**() / numpy.**savetxt**()
  - Para ficheros **CSV** y **texto**:
    - numpy.**genfromtxt**()
    - numpy.**recfromcsv**()
  - Para formatos **binarios**:
    - numpy.**save**()
    - numpy.**load**()

Lectura / Escritura de imágenes

```
>>> from scipy import misc
>>> misc.imread('fname.png')
array(...)

>>> # Matplotlib also has a similar function
>>> import matplotlib.pyplot as plt
>>> plt.imread('fname.png')
array(...)
```

Enlaces

- Doc
  - https://www.scipy.org/
- Scipy Cookbook
  - https://scipy-cookbook.readthedocs.io/
- Ejemplos filtrado de señales:
  - https://www.programcreek.com/python/example/100533/scipy.signal.medfilt