Librerías científicas en Python:

Scipy

SciPy

- Trabaja en conjunto con NumPy
- Utiliza la eficiencia de los arreglos de NumPy para implementar librerías que resuelven problemas comunes en la vida de científicos e ingenieros.
- Contiene gran cantidad de valores constantes usados comunmente en ciencia (http://docs.scipy.org/doc/scipy/reference/constants.html)

SciPy

- http://www.scipy.org/
- Instalación: sudo apt-get install python-scipy

Subpackage	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
`weave`	C/C++ integration

SciPy

- Contiene muy buena documentación

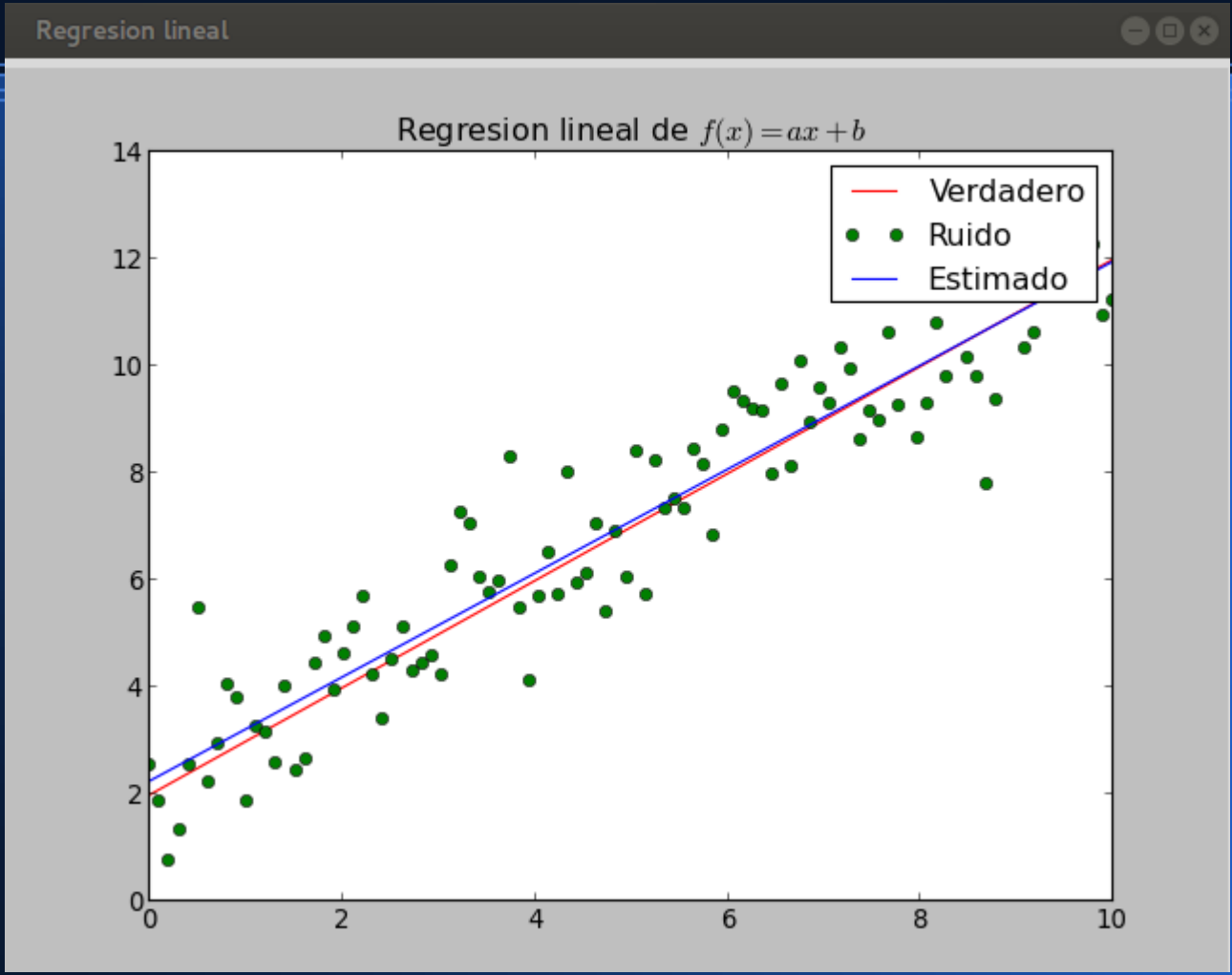
```
In [5]: import scipy as sp

In [6]: from scipy import linalg, optimize

In [7]: sp.info(optimize.fmin)
 fmin(func, x0, args=(), xtol=0.0001, ftol=0.0001, maxiter=None, maxfun=None,
      full_output=0, disp=1, retall=0, callback=None)

Minimize a function using the downhill simplex algorithm.

Parameters
----------
func : callable func(x,*args)
    The objective function to be minimized.
x0 : ndarray
    Initial guess.
args : tuple
    Extra arguments passed to func, i.e. ``f(x,*args)``.
callback : callable
    Called after each iteration, as callback(xk), where xk is the
    current parameter vector.
```

Optimización

- Regresión lineal
- Mínimos y máximos de una función
- Ceros de una función

Regresión lineal (1)

- Tenemos una función f(x) = ax + b.
- Generamos un espacio lineal para x.
- Generamos datos con ruido a partir de la función exacta.
- Generamos datos exactos para comparar.
- Se obtienen los estimados de a y b con curve_fit del módulo optimize.


Regresion lineal
Regresion lineal de $f(x) = ax + b$
Verdadero
Ruido
Estimado
0
2
4
6
8
10
12
14

Regresión lineal (1)

- De repo sci-libs, bajar archivo regresion_lineal.py

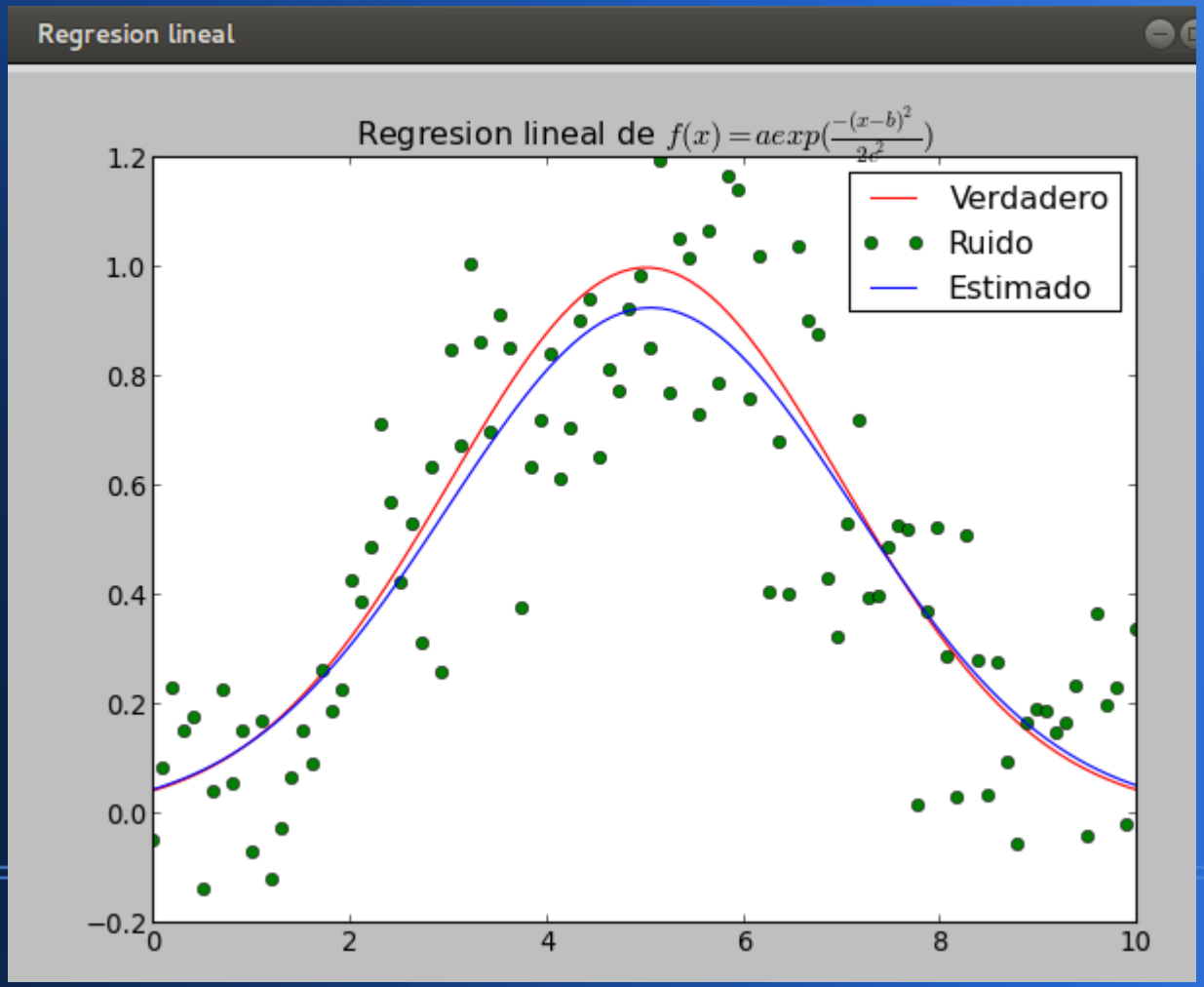
```python
# -*- coding: UTF-8 -*-

# Basado en ejemplo de página 18 de SciPy and NumPy de O'Reilly

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

# Función f(x) = ax + b
def f(x, a, b):
    return a * x + b

# Generating clean data
x = np.linspace(0, 10, 100)
y = f(x, 1, 2)

# Adding noise to the data
yn = y + 0.9 * np.random.normal(size=len(x))

# Executing curve_fit on noisy data
popt, pcov = curve_fit(f, x, yn)

# popt contiene el estimado de a y b a partir de la regresión linea
# de los datos con ruido
```

```python
a_estimado = popt[0]
b_estimado = popt[1]

y_estimado = f(x, a_estimado, b_estimado)

fig = plt.figure('Regresion lineal')

plt.plot(x, y, 'r')          # Valor exacto de la función
plt.plot(x, yn, 'go')    # Datos con ruido
plt.plot(x, y_estimado, 'b') # Best fit con regresioón lineal
plt.title('Regresion lineal de $f(x) = ax + b$')
plt.legend(['Verdadero', 'Ruido', 'Estimado'])
plt.show()
```

Regresión lineal (2)

- De repo sci-libs, bajar archivo regresion_gauss.py

$$a * \exp\left(\frac{-(x-\mu)^2}{2\,\sigma^2}\right)$$



Interpolación

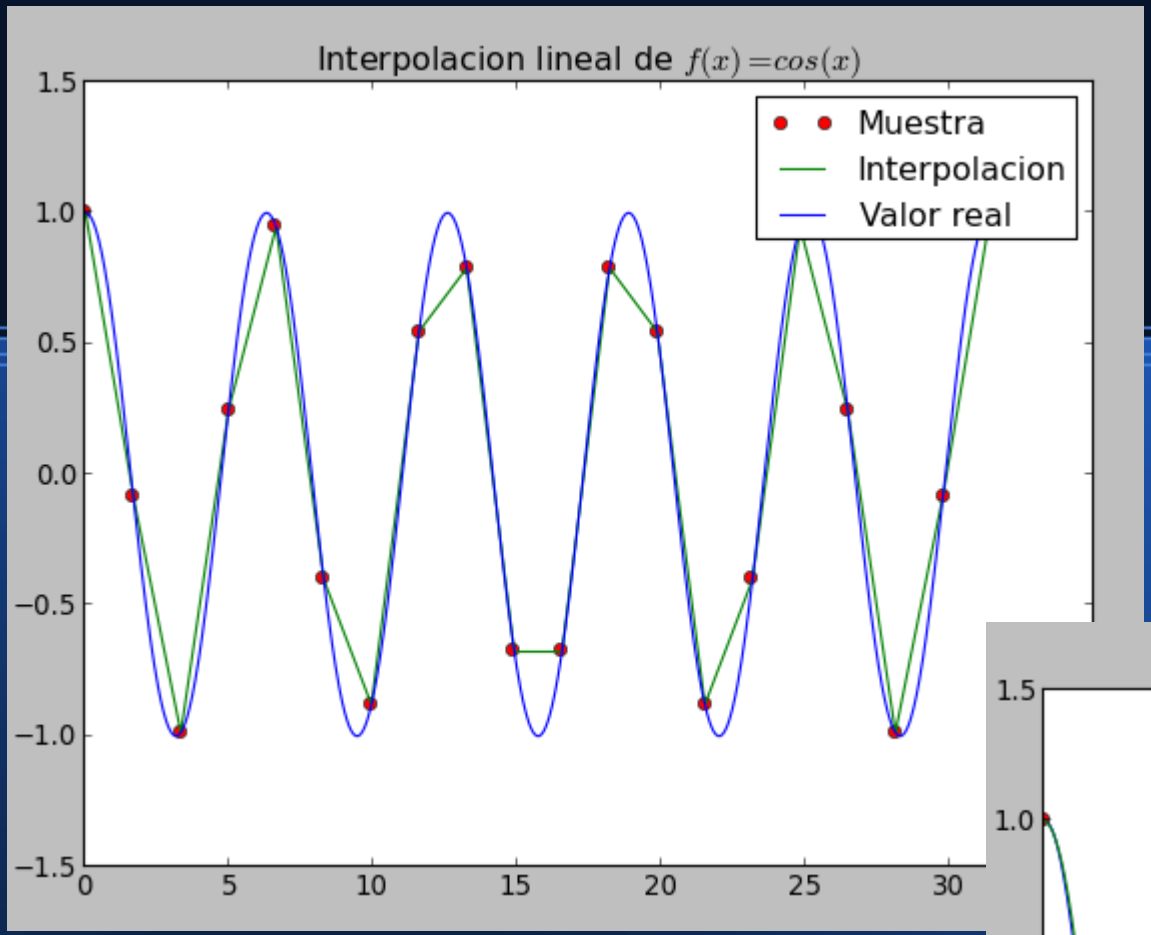
- Dada una muestra de datos, obtener valores intermedios.
- Scipy contiene docenas de funciones para interpolación en scipy.interpolate.
- Con interp1d interpolamos datos generados a partir de una función con una variable independiente.

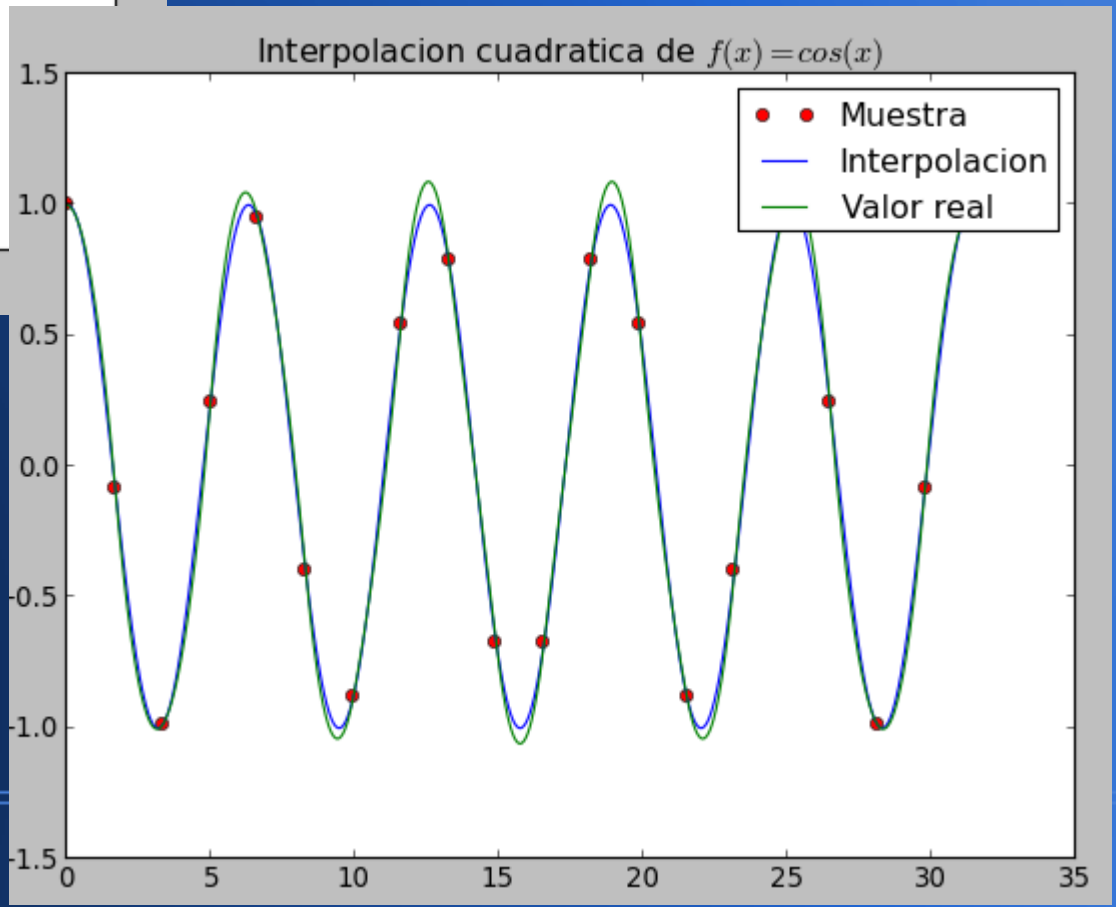
Interpolación – Ejemplo: cos(x)

- Tenemos 20 muestras de datos correspondientes a la función cos(x).
- Con función interp1d se generan 2 funciones: una con método lineal, otra con cuadrático.
- Se generan datos de interpolación con más puntos intermedios, en el mismo rango que la muestra de 20.

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d

x = np.linspace(0, 10 * np.pi, 20)      # Se crea el espacio lineal con 20 puntos
y = np.cos(x)                           # Se obtienen los datos para los 20 puntos

# Se realiza la interpolación con 2 métodos diferentes, se obtienen
# 2 funciones fl y fq con las que podemos estimar los valores para
# más puntos dentro del rango del espacio lineal

fl = interp1d(x, y, kind='linear')
fq = interp1d(x, y, kind='quadratic')
# x.min and x.max are used to make sure we do not
# go beyond the boundaries of the data for the
# interpolation.
xint = np.linspace(x.min(), x.max(), 1000)  # Se crea un nuevo espacio, mismo rango, pero mas puntos
yintl = fl(xint)
yintq = fq(xint)
yreal = np.cos(xint)
```

```python
plt.figure("Interpolacion1")
plt.plot(x, y, 'ro')          # Muestras
plt.plot(xint, yintl, 'g')    # Interpolación lineal
plt.plot(xint, yreal, 'b')    # Datos reales
plt.title('Interpolacion lineal de $f(x) = cos(x)$')
plt.ylim(-1.5, 1.5)
plt.legend(['Muestra', 'Interpolacion', 'Valor real'])

plt.figure("Interpolacion2")
plt.plot(x, y, 'ro')          # Muestras
plt.plot(xint, yreal, 'b')    # Datos reales
plt.plot(xint, yintq, 'g')    # Interpolación cuadrática
plt.title('Interpolacion cuadratica de $f(x) = cos(x)$')
plt.legend(['Muestra', 'Interpolacion', 'Valor real'])
plt.ylim(-1.5, 1.5)

plt.show()
```

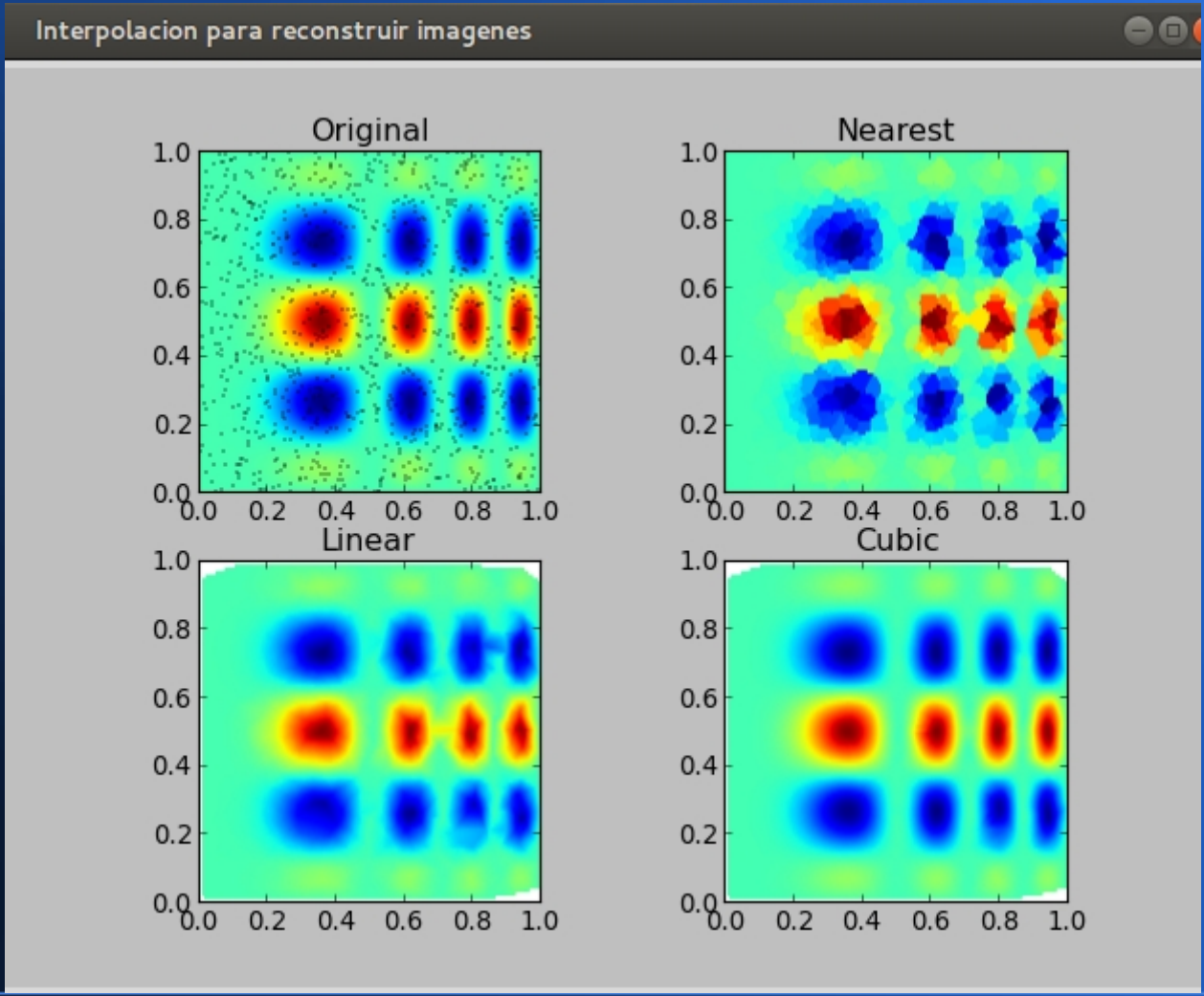

Interpolacion lineal de $f(x) = cos(x)$
Muestra
Interpolacion
Valor real
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0
5
10
15
20
25
30


Interpolacion cuadratica de $f(x) = cos(x)$
Muestra
Interpolacion
Valor real
1.5
1.0
0.5
0.0
-0.5
-1.0
-1.5
0
5
10
15
20
25
30
35

Interpolación - Imágenes

- Del repositorio sci-libs, bajar interpolacion_imagen.py
- Función scipy.interpolate.griddata puede manejar datos N dimensionales.
- Por ejemplo, una imágen puede ser reconstruida a partir de N puntos seleccionados aleatoriamente.

Interpolación - Imágenes



Integración

- Scipy permite integrar tanto ecuaciones como conjuntos de datos.
- from scipy.integrate import quad para integrar funciones de 1 variable.
- Función quad se basa en paquete QUADPACK de Fortran.

Integración: quad 1D

$$\int_0^3 \cos^2(e^x)\,dx$$

```python
# -*- coding: UTF-8 -*-

import numpy as np
from scipy.integrate import quad


# Funcion a integrar: cos²(e^x)
def f(x):
    return np.cos(np.exp(x)) ** 2

# Integrar la función en los límites 0, 3
# quad devuelve un arreglo con 2 elementos: el primero, la solución
# el segundo, el error
solucion = quad(f, 0, 3)

print "La solución es: %s " % (str(solucion))
```

```
Terminal
File  Edit  View  Search  Terminal  Help

In [66]: %run integracion_quad1D.py
La solución es: (1.296467785724373, 1.3977971863744082e-09)
```

Integración: límites infinitos

- Integrar f(x) = e^x de -infinito a 0

```python
# -*- coding: UTF-8 -*-

import numpy as np
from scipy.integrate import quad, Inf


# Funcion a integrar: cos²(e^x)
def f(x):
    return np.exp(x)

# Integrar la función en los límites 0, 3
# quad devuelve un arreglo con 2 elementos: el primero, la solución
# el segundo, el error
solucion = quad(f, -Inf, 0)

print "La solución es: %s " % (str(solucion))

```

```
In [70]: %run integracion_infinito.py
La solución es: (1.0000000000000002, 5.842606742906004e-11)
```

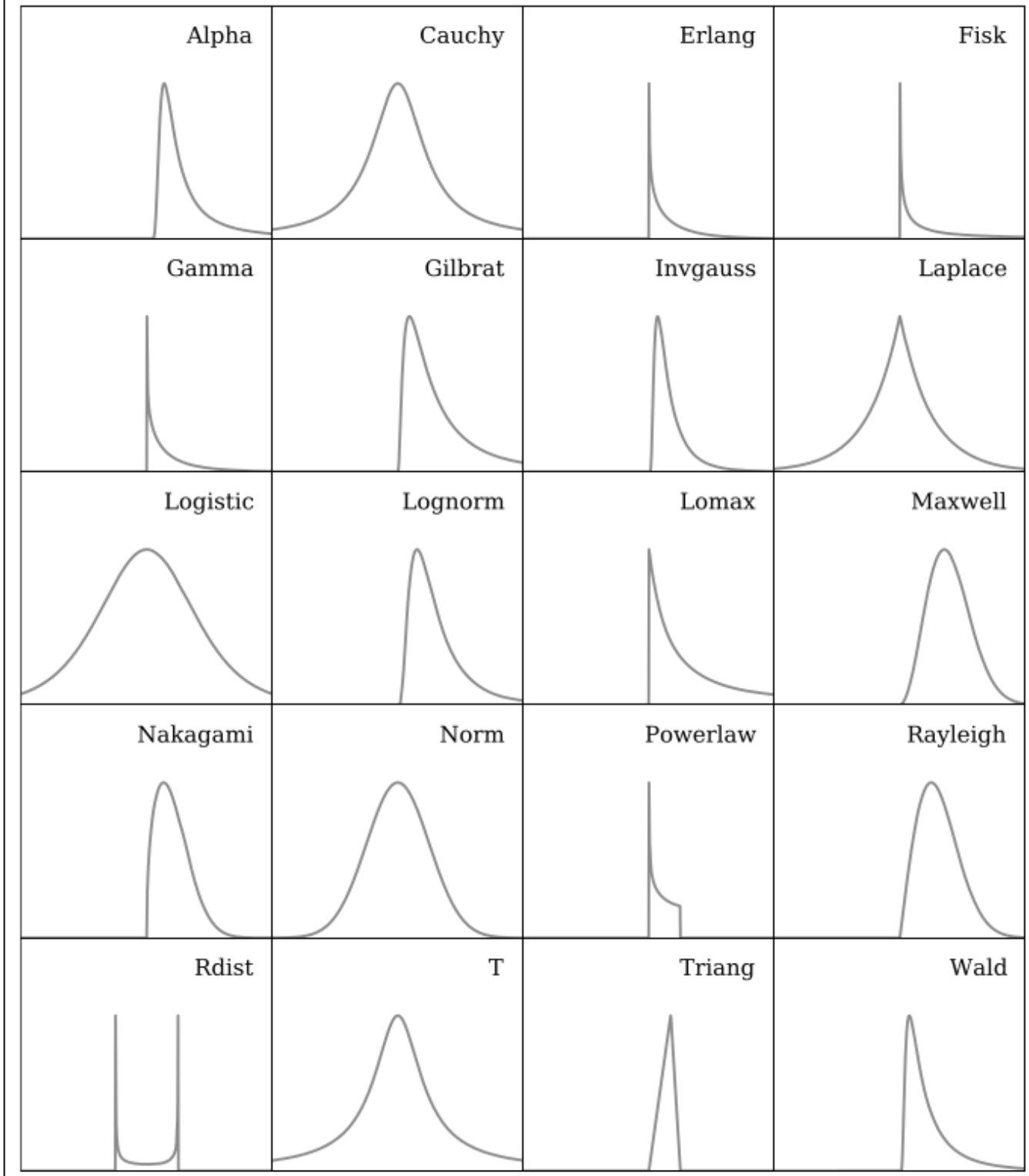
Integración numérica

- En lugar de una función, tenemos un conjunto de datos (ej. muestras, observaciones, etc)
- scipy.integrate.trapz → regla trapezoidal.

```
In [107]: x = np.arange(0.0, 20.5, 0.5)

In [108]: y = x * x   # Esta es nuestra "muestra" de datos

In [109]: resultado = scipy.integrate.trapz(y, x)

In [110]: print resultado
2667.5

In [111]: resultado2 = scipy.integrate.trapz(y, dx=0.5)

In [112]: print resultado2
2667.5
```

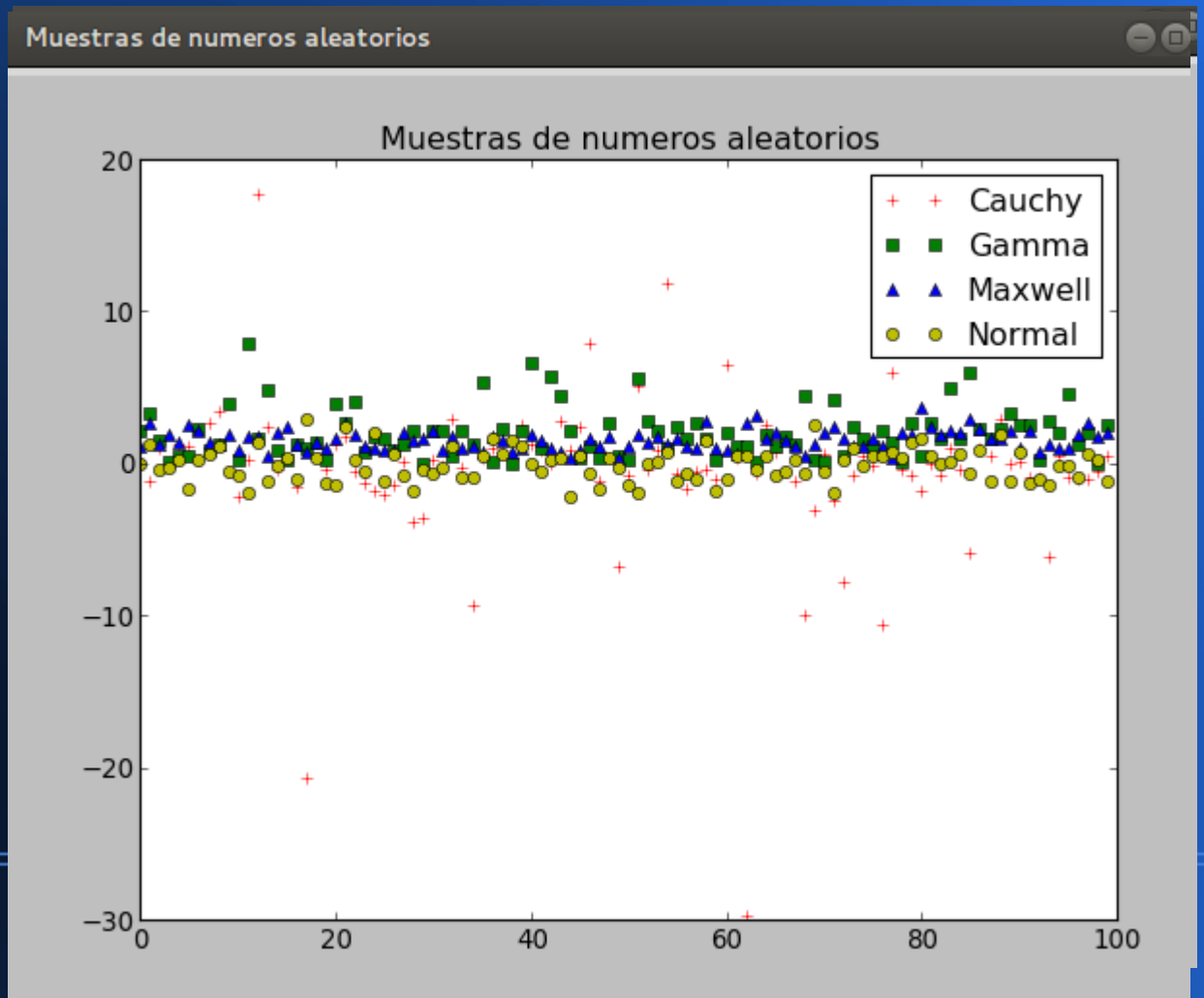
Estadística: distribuciones

- Decenas de distribuciones continuas y discretas (http://docs.scipy.org/doc/scipy/reference/tutorial/stats.html).
- Útil para generar números aleatorios (similar a numpy.random) pero más allá de las distribuciones Gaussianas de numpy.
- Se puede obtener información de las distribuciones, por ejemplo, la función de densidad de probabilidad (PDF)


Alpha
Cauchy
Erlang
Fisk
Gamma
Gilbrat
Invgauss
Laplace
Logistic
Lognorm
Lomax
Maxwell
Nakagami
Norm
Powerlaw
Rayleigh
Rdist
T
Triang
Wald

Ejemplo: distribuciones

- Del repositorio sci-libs bajar distribuciones.py



Resolver sistemas de ecuaciones lineales

- Con función solve de módulo scipy.linalg
- Recibe una matriz y un vector (lado derecho de la ecuación).
- Devuelve el vector solución

http://docs.scipy.org/doc/scipy/reference/tutorial/linalg.html#solving-linear-system

Sistemas de ecuaciones lineales

$$
\begin{aligned}
x + 3y + 5z &= 10 \\
2x + 5y + z &= 8 \\
2x + 3y + 8z &= 3
\end{aligned}
$$

```
8
9   import numpy as np
10  from scipy import linalg
11
12  matriz_coeficientes = np.array(
13      [[1, 3, 5],
14      [2, 5, 1],
15      [2, 3, 8]]
16  )
17
18  vector_lado_derecho = np.array([10., 8., 3.])
19
20  soluciones = linalg.solve(matriz_coeficientes, vector_lado_derecho)
21
22  print "Solucion al sistema: %s" % (soluciones)
23
24  # Probamos que la solucion sea la correcta, multiplicando la matriz de coeficientes por
25  # la solucion, debemos obtener el vector de lado derecho
26
27  prueba = matriz_coeficientes.dot(soluciones)
28
29  print "Probando las soluciones: %s." % (prueba)
```

Ejercicio

- Resuelva el siguiente sistema de ecuaciones utilizando numpy y scipy

$$
\begin{aligned}
3x + 6y - 5z &= 12 \\
x - 3y + 2z &= -2 \\
5x - y + 4z &= 10
\end{aligned}
$$

Constantes

- http://docs.scipy.org/doc/scipy/reference/constants.html
- from scipy.constants import *

```
In [40]: scipy.constants.c # velocidad de la luz
Out[40]: 299792458.0

In [41]: scipy.constants.pi
Out[41]: 3.141592653589793

In [42]: scipy.constants.h  # constante de Planck
Out[42]: 6.62606896e-34

In [43]: scipy.constants.G   # constante de gravitación universal
Out[43]: 6.67428e-11

In [44]: scipy.constants.e  # Carga del electron
Out[44]: 1.602176487e-19

In [45]: scipy.constants.k  # Constante de Boltzmann
Out[45]: 1.3806504e-23

In [46]: scipy.constants.g  # Aceleracion de la gravedad
Out[46]: 9.80665
```

Physical constants

`c`	speed of light in vacuum
`mu_0`	the magnetic constant μ_0
`epsilon_0`	the electric constant (vacuum permittivity), ϵ_0
`h`	the Planck constant h
`hbar`	$\hbar = h/(2\pi)$
`G`	Newtonian constant of gravitation
`g`	standard acceleration of gravity
`e`	elementary charge
`R`	molar gas constant
`alpha`	fine-structure constant
`N_A`	Avogadro constant
`k`	Boltzmann constant
`sigma`	Stefan-Boltzmann constant σ
`Wien`	Wien displacement law constant
`Rydberg`	Rydberg constant
`m_e`	electron mass
`m_p`	proton mass
`m_n`	neutron mass

Conversión de unidades de temperatura

Temperature

`zero_Celsius`	zero of Celsius scale in Kelvin
`degree_Fahrenheit`	one Fahrenheit (only differences) in Kelvins
`C2K`(C)	Convert Celsius to Kelvin
`K2C`(K)	Convert Kelvin to Celsius
`F2C`(F)	Convert Fahrenheit to Celsius
`C2F`(C)	Convert Celsius to Fahrenheit
`F2K`(F)	Convert Fahrenheit to Kelvin
`K2F`(K)	Convert Kelvin to Fahrenheit

```
In [49]: scipy.constants.C2F(25)
Out[49]: 77.0

In [50]: scipy.constants.F2C(100)
Out[50]: 37.777777777777779
```

Próxima(s) clase(s)

- PyFITS
- SymPy
- Ejercicios NumPy, SciPy y Matplotlib