

Introducción a la programación

Prof.Miguel García Silvente

Tema 2 Scipy

Esquema

- Introducción a NumPy
 - Tipo de dato ndarray
 - Funciones universales
 - Vectorización
- Matplotlib
- SciPy

Prof.Miguel García Silvente

3

NumPy (I)

- Un paquete de módulos que permite computación numérica de forma eficiente.
- Python no dispone de vectores multimensionales. NumPy proporciona esta posibilidad.
- El módulo NumPy proporciona un tipo eficiente llamado **ndarray** que permite manejar vectores multimensionales de un determinado tipo.

Prof.Miguel García Silvente

4

NumPy (II)

- Sitio web -- <http://numpy.scipy.org/>
- Proporciona capacidad similar a Matlab
- NumPy reemplaza a Numeric y Numarray
- Fue desarrollado inicialmente por Travis Oliphant
- NumPy 1.0 apareció en octubre de 2006 (existen Numeric desde los 90 y numarray desde el 2000)
- Tiene unas 20.000 descargas/mes desde sourceforge.
Sin contar las distribuciones de linux o MacOS o paquetes como Sage que lo incluyen
- Para utilizarlo:
`import numpy as N`
`from numpy import *`

Prof.Miguel García Silvente

5

Tipo de dato ndarray

- Vector N-dimensional rectangular de datos.
- Los datos pueden ser de cualquier tipo simple.
- Incluye algoritmos eficientes para los tipos de datos básicos (int, float, etc.)
- Las funciones operan elemento a elemento.
- Programación rápida de bucles para datos fundamentales.
- Si N es el número de elementos:
`sin(x)` es equivalente a `[sin(x[i]) for i in range(N)]`
`x+y` es equivalente a `[x[i]+y[i] for i in range(N)]`
- Se crean con `array(<lista o tupla>,dtype)`

Prof.Miguel García Silvente

6

NumPy dtypes

Tipo básico	Tipos NumPy	Comentarios
Boolean 'b'	bool	Tienen 1 byte de tamaño
Integer 'i'	int8, int16, int32, int64, int128, int	int tiene el tamaño de un int en C
Unsigned Integer 'u'	uint8, uint16, uint32, uint64, uint128, uint	uint por defecto coincide con unsigned int de C en la plarafoma
Float 'f', 'd'	float32, float64, float, longfloat,	Float es un real con double precision (64 bits). longfloat representa reales de larga precisión
Complex 'c'	complex64, complex128, complex	Cada real de complex64 tiene 32 bits
Strings 'S', 'a'	str, unicode	Unicode es UTF32 (UCS4)
Object	object	Representa items en el array como Python objects.
Records	void	Una estructura de datos en registros de un array

Ejemplo de NumPy array

```
>>> import numpy as N
>>> a = N.array([[1,2,3],[4,5,6]],float)
>>> print (a)
[[1. 2. 3.]
 [4. 5. 6.]]
>>> print (a.shape, "\n", a.itemsize)
(2, 3)
8
>>> print (a.dtype, a.dtype.type)
'<f8' <type 'float64scalar'>
>>> type(a[0,0])
<type 'float64scalar'>
>>> type(a[0,0]) is type(a[1,2])
True
```

NumPy arrays vs listas

- Una lista puede tener diferentes tipos de elementos y elementos por dimensión:
 - `a = [1,2]`
 - `b = [[1,2],[3,4]]`
 - `c = [[1,2, 'duh'],[3,[4]]]`
- Mientras que los vectores NumPy:
 - `x = array([1,2])`
 - `y = array([[1,2],[3,4]])`
 - Todas las filas deben tener la misma longitud, etc
 - Todas las entradas deben ser del mismo tipo.
- Se usa NumPy para los problemas:
 - Matemáticos – suma de matrices, producto, conjugados, etc
 - Manipular matrices – Cambiar dimensiones, reorganizar, extraer partes, etc.

Prof.Miguel García Silvente

9

Introducción a los ndarrays (I)

- Creación de vectores a partir de una lista

```
>>> a = numpy.array([0,1,2,3])
```

```
>>> a
```

```
array([0, 1, 2, 3])
```

- Comprobación de tipo

```
>>> type(a)
```

```
<class 'numpy.ndarray'>
```

```
>>> a.dtype
```

```
dtype('int64')
```

- Bytes por elemento

```
>>> a.itemsize
```

Introducción a los ndarrays (II)

Dimensiones del vector

```
>>> a.shape
```

```
(4,)
```

```
>>> shape(a)
```

```
(4,)
```

Número de elementos

```
>>> a.size
```

```
4
```

```
>>> size(a)
```

```
4
```

Prof.Miguel García Silvente

11

Introducción a los ndarrays (III)

Número de bytes

```
>>> a.nbytes
```

```
32
```

Número de dimensiones

```
>>> a.ndim
```

```
1
```

Copia del vector

```
>>> b = a.copy()
```

```
>>> b
```

```
array([0, 1, 2, 3])
```

```
>>> c = np.array([[1], [2], [3], [4]])
```

```
>>> c.shape
```

```
(2, 2, 1)
```

Prof.Miguel García Silvente

12

Buscar información

```
np.lookfor('create array')
Search results for 'create arra'
-----y
numpy.array
    Create an array.
numpy.memmap
    Create a memory-map to an array stored in a *binary* file on
disk.
```

Prof.Miguel García Silvente

13

Creación de vectores

- Creación de vectores nulos

```
>>> a = zeros(3)
>>> a
array([ 0.,  0.,  0.])
>>> a = zeros(3, float32)
>>> a
array([ 0.,  0.,  0.], dtype=float32)
>>> Z = zeros((5,9))
```

- Creación de vectores con unos

```
>>> a = ones(3)
>>> a
array([ 1.,  1.,  1.])
```

Prof.Miguel García Silvente

14

Arrays con secuencias (I)

- `linspace(a, b, n)` genera n valores equiespaciados empezando en a y terminando en b (ambos incluidos)

```
>>> x = linspace(-5, 5, 11)
>>> print (x)
array([-5., -4., -3., -2., -1., 0., 1., 2., 3., 4., 5.])
```

- De forma compacta equivalente:

```
>>> a = r_[-5:5:11j] # linspace(-5, 5, 11)
>>> print (a)
array([-5., -4., -3., -2., -1., 0., 1., 2., 3., 4., 5.])
```

Prof.Miguel García Silvente

15

Arrays con secuencias (II)

`arange(a,b,x, tipo)` genera un array con valores entre a y b (sin incluir b) de x en x valores

```
>>> a = arange(-5, 5, 1, float)
>>> print (a)
[-5. -4. -3. -2. -1. 0. 1. 2. 3. 4.]
>>> a = arange(24).reshape(2,3,4)
>>> a
array([[[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]],
      [[12, 13, 14, 15],
       [16, 17, 18, 19],
       [20, 21, 22, 23]]])
```

Prof.Miguel García Silvente

16

Arrays con secuencias (III)

```
>>> a.shape = (12,2)
>>> a
array([[ 0,  1],
       [ 2,  3],
       [ 4,  5],
       [ 6,  7],
       [ 8,  9],
       [10, 11],
       [12, 13],
       [14, 15],
       [16, 17],
       [18, 19],
       [20, 21],
       [22, 23]])
```

Prof.Miguel García Silvente

17

Eficiencia (timeit)

In [1]: l = range(1000)

In [2]: %timeit [i**2 for i in l]

The slowest run took 4.16 times longer than the fastest. This could mean that an intermediate result is being cached
10000 loops, best of 3: 87.7 μ s per loop

In [3]: a = np.arange(1000)

In [4]: %timeit a**2

The slowest run took 101.98 times longer than the fastest. This could mean that an intermediate result is being cached
1000000 loops, best of 3: 1.89 μ s per loop

Prof.Miguel García Silvente

18

Creación a partir de una función

```
>>> def myfunc(i, j):
...     return (i+1)*(j+4-i)
...
# un array 3x6 con a[i,j] = myfunc(i,j):
>>> a = fromfunction(myfunc, (3,6))
>>> a
array([[ 4.,  5.,  6.,  7.,  8.,  9.],
       [ 6.,  8., 10., 12., 14., 16.],
       [ 6.,  9., 12., 15., 18., 21.]])
```

Prof.Miguel García Silvente

19

Creación de arrays aleatorios

```
>>> a = numpy.random.uniform(0,1,9)
array([ 0.20458382,  0.1549881 ,  0.93299498,
       0.59321766,  0.48781365, 0.58141393,  0.00997613,
       0.00435322,  0.37903198])

>>> a = numpy.random.uniform(0,1,(5,9))
array([[ 0.14742689,  0.06398224,  0.66351784,
       0.39107823,  0.58600751, ...,
       0.74637837,  0.89908347,  0.48976073,
       0.14714759]])
```

Prof.Miguel García Silvente

20

Conversión a lista

La forma más eficiente

```
>>> a.tolist()
```

```
[[4.0, 5.0, 6.0, 7.0, 8.0, 9.0], [6.0, 8.0, 10.0,  
12.0, 14.0, 16.0], [6.0, 9.0, 12.0, 15.0, 18.0,  
21.0]]
```

De una forma menos eficiente con list()

Sólo se genera una lista si es un array 1D

```
>>> list(a)
```

```
[array([ 4., 5., 6., 7., 8., 9.]), array([ 6., 8.,  
10., 12., 14., 16.]), array([ 6., 9., 12., 15.,  
18., 21.])]
```

Prof.Miguel García Silvente

21

Conversión a numPy

Dado un objeto a

```
>>> a = asarray(a)
```

```
>>> a = asarray(a, order='Fortran')
```

Lo convierte a array de NumPy.

En una función es posible hacer:

```
def f(sec, ...):  
    a = asarray(sec)
```

```
f([1,2,3], ...)
```

```
f((-1,1), ...)
```

```
f(zeros(10), ...)
```

Prof.Miguel García Silvente

22

Asignación de elementos (I)

```
>>> a[0]  
0  
>>> a[0] = 10  
>>> a  
[10, 1, 2, 3]
```

Rellenar todos los valores de un vector:

```
>>> a.fill(0)  
>>> a  
[0, 0, 0, 0]  
De una forma menos eficiente  
>>> a[:] = 1  
>>> a  
[1, 1, 1, 1]
```

Prof.Miguel García Silvente

23

Asignación de elementos (II)

Hay que tener cuidado con los cambios automáticos de tipo:

```
>>> a.dtype  
dtype('int64')  
>>> a[0] = 10.6  
>>> a  
[10, 1, 2, 3]
```

También ocurre al llenar:

```
>>> a.fill(-4.8)  
>>> a  
[-4, -4, -4, -4]
```

Prof.Miguel García Silvente

24

Vectores multidimensionales (I)

```
>>> a = array([[ 0, 1, 2, 3],  
           [10,11,12,13]])
```

```
>>> a
```

```
array([[ 0, 1, 2, 3],  
      [10,11,12,13]])
```

Número de filas y columnas (como ya vimos)

```
>>> a.shape
```

```
(2, 4)
```

```
>>> shape(a)
```

```
(2, 4)
```

Número de elementos

```
>>> a.size
```

```
8
```

```
>>> size(a)
```

```
8
```

Prof.Miguel García Silvente

25

Vectores multidimensionales (II)

Número de dimensiones

```
>>> a.ndim
```

```
2
```

Acceso a los elementos

```
>>> a[1,3]
```

```
13
```

```
>>> a[1,3] = -1
```

```
>>> a
```

```
array([[ 0, 1, 2, 3],  
      [10,11,12,-1]])
```

Acceso a la primera fila

```
>>> a[1]
```

```
array([10, 11, 12, -1])
```

Prof.Miguel García Silvente

26

Cambiar dimensiones

```
>>> a = array([0, 1.2, 4, -9.1, 5, 8])  
>>> a.shape = (2,3) # convierte a ndarray 2x3  
>>> a.size  
6  
>>> a.shape = (a.size,) # de nuevo a vector  
>>> a.shape  
(6,)  
>>> a = a.reshape(2,3) # igual que usar shape  
>>> a.shape  
(2, 3)
```

Prof.Miguel García Silvente

27

Trocear un vector

Funciona de la forma habitual

```
>>> a[0,3:5]  
array([3, 4])  
>>> a[4:,4:]  
array([[44, 45],  
      [54, 55]])  
>>> a[:,2]  
array([2,12,22,32,42,52])
```

También se pueden extraer tiras

```
>>> a[2::2,::2]  
array([[20, 22, 24],  
      [40, 42, 44]])
```

0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

Prof.Miguel García Silvente

28

Asignación de elementos

Por ejemplo, partiendo de zeros(40).reshape(5,8)
Se puede asignar un valor a un trozo

>>> a[:, 0] = 1



>>> a[:,::2] = 1



>>> a[::2,::2] = 1



>>> a[2:,2:] = 1



Prof.Miguel García Silvente

29

Ejemplos de troceado de vectores

```
>>> b = a[:,::2]
>>> b[0,1] = 100
>>> print (a)
[[ 0  1 100  3]
 [ 10 11 12 13]]
>>> c = a[:,::2].copy()
>>> c[1,0] = 500
>>> print (a)
[[ 0  1 100  3]
 [ 10 11 12 13]]
```

Prof.Miguel García Silvente

30

Trozos como referencias

Si se cambia un valor en un trozo, se hace sobre el original

```
>>> a = array((0,1,2,3,4))
```

```
# trozo con los últimos elementos
```

```
>>> b = a[2:4]
```

```
>>> b[0] = 10
```

```
# se cambia a
```

```
>>> a
```

```
array([ 0, 1, 10, 3, 4])
```

Prof.Miguel García Silvente

31

Indexación indirecta

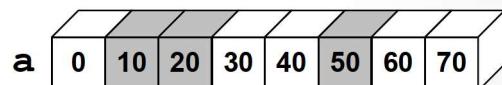
```
>>> a = arange(0,80,10)
```

```
# indexación indirecta
```

```
>>> y = a[[1, 2, -3]]
```

```
>>> print (y)
```

```
[10 20 50]
```



```
# take
```

```
>>> y = take(a,[1,2,-3])
```

```
>>> print (y)
```

```
[10 20 50]
```

Prof.Miguel García Silvente

32

Selección de valores con datos lógicos

```
>>> mask = array([0,1,1,0,0,1,0,0], dtype=bool)
```

```
# selección de valores
```

```
>>> y = a[mask]
```

```
>>> print (y)
```

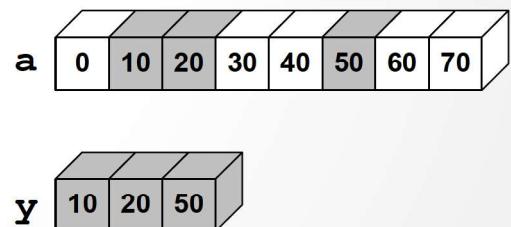
```
[10,20,50]
```

```
# con compress
```

```
>>> y = compress(mask, a)
```

```
>>> print (y)
```

```
[10,20,50]
```



Prof.Miguel García Silvente

33

Ejemplos de selección de valores(I)

```
>>> a = linspace(1, 8, 8)
```

```
>>> a
```

```
array([ 1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.])
```

```
>>> a[[1,6,7]] = 10
```

```
>>> a
```

```
array([ 1., 10.,  3.,  4.,  5.,  6., 10., 10.])
```

```
>>> a[range(2,8,3)] = -2
```

```
>>> a
```

```
array([ 1., 10., -2.,  4.,  5., -2., 10., 10.])
```

```
>>> a[a < 0] # escoge los elementos negativos
```

```
array([-2., -2.])
```

```
>>> a[a < 0] = a.max()
```

```
>>> a
```

```
array([ 1., 10., 10.,  4.,  5., 10., 10., 10.])
```

Prof.Miguel García Silvente

34

Ejemplos de selección de valores(II)

```
>>> a[a % 2 == 1]
array([ 1.,  3.,  5.,  7.])
>>> a = numpy.array([arange(5), arange(5), arange(6)])
>>> a
array([array([0, 1, 2, 3, 4]), array([0, 1, 2, 3, 4]),
       array([0, 1, 2, 3, 4, 5])], dtype=object)
# todas las dimensiones: ...
>>> a[..., 2]
array(array([0, 1, 2, 3, 4, 5]), dtype=object)
>>> a[0]
array([0, 1, 2, 3, 4])
```

Prof.Miguel García Silvente

35

Indexación indirecta en 2D

```
>>> a[(0,1,2,3,4),(1,2,3,4,5)]
array([ 1, 12, 23, 34, 45])
```

```
>>> a[3:,[0, 2, 5]]
array([[30, 32, 35],
       [40, 42, 45]],
      [50, 52, 55]])
```

```
>>> mask = array([1,0,1,0,0,1],
                  dtype=bool)
```

```
>>> a[mask,2]
array([2,22,52])
```

0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

Prof.Miguel García Silvente

36

Bucles

Sobre todos los elementos como si fuera 1D:

```
>>> for e in a.flat:  
    print (e)
```

Sobre todos los índices y valores:

```
>>> for index, value in ndenumerate(a):  
...     print (index, value)
```

...

```
(0, 0) 2.0  
(0, 1) 6.0  
(0, 2) 12.0  
(1, 0) 4.0  
(1, 1) 12.0  
(1, 2) 24.0
```

```
>>> b = a.view('i8')  
>>> [hex(val.item()) for val in b.flat]  
['0x0', '0x1', '0x64', '0x3', '0xa',  
'0xb', '0xc', '0xd']
```

Prof.Miguel García Silvente

37

Modelo de memoria de NumPy

- Se puede acceder a cada dimensión del ndarray saltando un determinado número de bytes.
- Si la memoria es contigua se pueden precalcular los saltos. Si se guardan con el orden de **Fortran** se accede más rápido a la primera dimensión, si es en el orden de **C** sería a la última

```
>>> print a.strides # número de bytes/dimensión  
(32, 8)  
>>> print (a.flags.fortran, a.flags.contiguous)  
False True  
>>> print (a.T.strides)  
(8, 32)  
>>> print (a.T.flags.fortran, a.T.flags.contiguous)  
True False
```

Prof.Miguel García Silvente

38

Operaciones sobre vectores (I)

Las operaciones se realizan elemento a elemento

```
>>> a = array([[1,2,3], [4,5,6]], float)
```

```
>>> b = 3*a - 1
```

Se pueden sumar todos los elementos

```
>>> sum(a)
```

21.

Se puede sumar a lo largo de un eje (o multiplicar con `prod`)

```
>>> sum(a, axis=0)
```

```
array([5., 7., 9.])
```

Para el último eje

```
>>> sum(a, axis=-1)
```

```
array([6., 15.])
```

Prof.Miguel García Silvente

39

Operaciones sobre vectores (II)

```
>>> a = array([1,2,3,4])
```

```
>>> b = array([2,3,4,5])
```

```
>>> a + b
```

```
array([3, 5, 7, 9])
```

Crear un array de 0 a 10

```
>>> x = arange(11.)
```

Multiplicar un array por un escalar

```
>>> a = (2*pi)/10.
```

```
>>> a
```

```
0.62831853071795862
```

```
>>> a*x
```

```
array([ 0., 0.628, ..., 6.283])
```

Operar sobre el mismo array

```
>>> x *= a
```

```
>>> x
```

```
array([ 0., 0.628, ..., 6.283])
```

Aplicar una función

```
>>> y = sin(x)
```

Prof.Miguel García Silvente

40

Calcular Mínimo y Máximo

```
>>> a = array([2.,3.,0.,1.])
```

```
>>> a.min(axis=0)
```

0.

Para NumPy con datos multimensionales se usa amin

```
>>> amin(a, axis=0)
```

0.

También se puede calcular la posición del mínimo

```
>>> argmin(a, axis=0)
```

2

```
>>> argmin(a, axis=0)
```

2

De forma equivalente con **amax**, **argmax**

Prof.Miguel García Silvente

41

Operaciones estadísticas (I)

```
>>> a = array([[1,2,3], [4,5,6]], float)
```

Para calcular la media

```
>>> a.mean()
```

3.5

```
>>> a.mean(axis=0)
```

array([2.5, 3.5, 4.5])

```
>>> mean(a, axis=0)
```

array([2.5, 3.5, 4.5])

```
>>> average(a, axis=0)
```

array([2.5, 3.5, 4.5])

Media ponderada

```
>>> average(a, weights=[1,2],  
...           axis=0)
```

array([3., 4., 5.])

Para calcular la desviación estándar

```
>>> a.std(axis=0)
```

array([1.5, 1.5, 1.5])

La varianza

```
>>> a.var(axis=0)
```

array([2.25, 2.25, 2.25])

```
>>> var(a, axis=0)
```

array([2.25, 2.25, 2.25])

Prof.Miguel García Silvente

42

Operaciones estadísticas (II)

```
a = arange(0,100,2)
b = a + 20
correlacion = np.corrcoef(a,b)
print(correlacion)
array([[ 1.,  1.],
       [ 1.,  1.]])
b = -b
correlacion = np.corrcoef(a,b)
print(correlacion)
array([[ 1., -1.],
       [-1.,  1.]])
```

Prof.Miguel García Silvente

43

Operadores binarios matemáticos

a + b -> add(a,b)
a - b -> subtract(a,b)
a % b -> remainder(a,b)

a * b -> multiply(a,b)
a / b -> divide(a,b)
a ** b -> power(a,b)

Operar con un valor
`>>> a = array((1,2))
>>> a*3.
array([3., 6.])`

Suma con una función
`>>> add(a,b)
array([4, 6])`

Elemento a elemento
`>>> a = array([1,2])
>>> b = array([3,4])
>>> a + b
array([4, 6])`

Producto de matrices
`>>> matmul(a,b)
array([11])`

Prof.Miguel García Silvente

44

Operadores lógicos

<code>equal</code> <code>(==)</code>	<code>not_equal</code> <code>(!=)</code>	<code>greater</code> <code>(>)</code>
<code>greater_equal</code> <code>(>=)</code>	<code>less</code> <code>(<)</code>	<code>less_equal</code> <code>(<=)</code>
<code>logical_and</code>	<code>logical_or</code>	<code>logical_xor</code>
<code>logical_not</code>		

```
>>> a = array(((1,2,3,4),(2,3,4,5)))
>>> b = array(((1,2,5,4),(1,3,4,5)))
>>> a == b
array([[True, True, False, True],
       [False, True, True, True]])
```

De forma equivalente:

```
>>> equal(a,b)
array([[True, True, False, True],
       [False, True, True, True]])
```

Prof.Miguel García Silvente

45

Operadores bit a bit

<code>bitwise_and</code> <code>(&)</code>	<code>invert</code> <code>(~)</code>	<code>right_shift(a,shifts)</code>
<code>bitwise_or</code> <code>()</code>	<code>bitwise_xor</code>	<code>left_shift (a,shifts)</code>

Ejemplos

```
>>> a = array((1,2,4,8))
>>> b = array((16,32,64,128))
>>> bitwise_or(a,b)
array([ 17,  34,  68, 136])
```

Inversión de bits

```
>>> a = array((1,2,3,4), uint8)
>>> invert(a)
array([254, 253, 252, 251], dtype=uint8)
```

Desplazamiento a la izquierda

```
>>> left_shift(a,3)
array([ 8, 16, 24, 32], dtype=uint8)
```

Prof.Miguel García Silvente

46

Otras funciones

<code>sin(x)</code>	<code>sinh(x)</code>	<code>exp(x)</code>	<code>log(x)</code>
<code>cos(x)</code>	<code>cosh(x)</code>	<code>log10(x)</code>	<code>sqrt(x)</code>
<code>arccos(x)</code>	<code>arccosh(x)</code>	<code>absolute(x)</code>	<code>conjugate(x)</code>
<code>arctan(x)</code>	<code>arctanh(x)</code>	<code>negative(x)</code>	<code>ceil(x)</code>
<code>arcsin(x)</code>	<code>arcsinh(x)</code>	<code>floor(x)</code>	<code>fabs(x)</code>
<code>arctan2(x,y)</code>		<code>hypot(x,y)</code>	<code>fmod(x,y)</code>
		<code>maximum(x,y)</code>	<code>minimum(x,y)</code>

Prof.Miguel García Silvente

47

Otros métodos (I)

Se pueden limitar los valores a un rango

```
>>> a = array([[1,2,3],  
[4,5,6]], float)
```

Los valores < 3 pasan a ser 3.

Los valores > 5 pasan a ser 5.

```
>>> a.clip(3,5)  
>>> a  
array([[ 3.,  3.,  3.],  
 [ 4.,  5.,  5.]])
```

Prof.Miguel García Silvente

48

Otros métodos (II)

Numpy redondea al número entero más cercano (en caso de igualdad, al par): 1.5 y 2.5 a 2.

```
>>> a = array([1.35, 2.5, 1.5])
```

```
>>> a.round()
```

```
array([ 1.,  2.,  2.])
```

Redondear a un decimal

```
>>> a.round(decimals=1)
```

```
array([ 1.4,  2.5,  1.5])
```

Es posible calcular la distancia punto a punto por columnas

```
>>> a = array([[1,2,3], [4,5,6]], float)
```

```
>>> a.ptp(axis=0)
```

```
array([ 3.0,  3.0,  3.0])
```

O el máximo para todos los posibles pares

```
>>> a.ptp(axis=None)
```

5.0

Prof.Miguel García Silvente

49

Resumen (I)

Atributos básicos

a.dtype – Tipo numérico de los elementos. float32, uint8, etc.

a.shape – Dimensiones. (m,n,o,...)

a.size – Número total de elementos.

a.itemsize – Número de bytes por elemento.

a.nbytes – Número total de bytes usado por los elementos.

a.ndim – Número de dimensiones

Resumen (II)

Operaciones sobre la forma del ndarray

`a.flat` – Un iterador para recorrer todos elementos.

`a.flatten()` – Devuelve una copia 1D del array multidimensional.

`a.ravel()` – Igual que flatten(), pero como una “vista”

`a.resize(nuevo tamaño)` – Cambia el tamaño

`a.swapaxes(axis1, axis2)` – Intercambia el orden de los ejes, es equivalente a traspuesta, `a.transpose(*axes)` – Intercambia el orden de los distintos ejes. `a.T` – es una forma abreviada de `a.transpose()`

`a.squeeze()` – Elimina las dimensiones de longitud 1

Prof.Miguel García Silvente

51

Resumen (III)

`a.copy()` – Devuelve una copia

`a.fill(valor)` – Rellena el vector con un valor.

Conversión

`a.tolist()` – Convierte un array a lista.

`a.tostring()` – copia a un string.

`a.astype(dtype)` – Devuelve un array convertido a un determinado dtype.

`a.byteswap(False)` – Convierte el orden de byte (big <-> little endian).

Números complejos

`a.real` – Devuelve la parte real.

`a.imag` – Devuelve la parte imaginaria.

`a.conjugate()` – Devuelve el complejo conjugado.

`a.conj()` – Igual al anterior

Prof.Miguel García Silvente

52

Resumen (IV)

- a.dump(fichero) – Guarda un array en un archivo.
- a.dumps() – Devuelve el pickle binario del array como un string.
- a.tofile(fid, sep="", format="%s") Salida ascii formateada a fichero.

- a.nonzero() – Devuelve los índices de los elementos que no son cero.
- a.sort(axis=-1) – Ordena los elementos a lo largo de un eje.
- a.argsort(axis=-1) – Devuelve los índices para los elementos ordenados
- a.searchsorted(b) – Devuelve los índices de dónde se encontrarían los elementos de b en a.

- a.clip(low, high) – Limita los valores a un rango.
- a.round(decimals=0) – Redondea a un número de decimales.
- a.cumsum(axis=None) – Suma acumulada a lo largo de un eje.
- a.cumprod(axis=None) – Producto acumulado a lo largo de un eje.
Prof. Miguel García Silvente ⁵³

Resumen (V)

Métodos de reducción: Reducen el tamaño del array a una dimensión, aplicando una operación a lo largo de un eje. Si el eje es None, se realiza para todos los elementos.

- a.sum(axis=None) – Suma los elementos.
- a.prod(axis=None) – Calcula el producto.
- a.min(axis=None) – Calcula el mínimo.
- a.max(axis=None) – Calcula el máximo.
- a.argmin(axis=None) – Calcula el índice del mínimo.
- a.argmax(axis=None) – Calcula el índice del máximo.
- a.ptp(axis=None) – Calcula a.max(axis) – a.min(axis)
- a.mean(axis=None) – Calcula la media.
- a.std(axis=None) – Calcula la desviación estándar.
- a.var(axis=None) – Calcula la varianza.

- a.any(axis=None) – True si algún valor es distinto de cero.
- a.all(axis=None) – True si todos los valores son distintos de cero.

Entrada/Salida

- Para leer datos en formato texto

```
datos = np.loadtxt('datos.txt')
```

- Para guardar datos en formato texto

```
np.savetxt('datos.txt', datos)
```

Propagación (Broadcasting) I

Con múltiples entradas, son “difundibles” a la misma dimensión

Todos los arrays se modifican para tener las mismas dimensiones.

Se expanden las dimensiones de longitud 1

```
>>> x = [1,2,3,4]
>>> y = [[10],[20],[30]]
>>> print (add(x,y))
[[11 12 13 14]
 [21 22 23 24]
 [31 32 33 34]]
>>> x = array(x)
>>> y = array(y)
>>> print (x+y)
[[11 12 13 14]
 [21 22 23 24]
 [31 32 33 34]]
```

x es (4,) y la función lo ve como de (1,4)
y es (3,1)
Resultado (3,4)

Propagación II

The diagram illustrates the propagation of values through three stages of computation:

- Stage 1:** Two 4×3 matrices are added. The first matrix has values 0, 1, 2 on its diagonal and 0 elsewhere. The second matrix has values 10, 10, 10 on its diagonal and 0 elsewhere. The result is a matrix where the diagonal elements are doubled (0, 2, 4) and off-diagonal elements remain 0.
- Stage 2:** The resulting matrix from Stage 1 is added to a 3 vector. The vector has values 0, 1, 2. The result is a matrix where the diagonal elements are tripled (0, 3, 6) and off-diagonal elements remain 0.
- Stage 3:** The resulting matrix from Stage 2 is added to a 4×1 vector. The vector has values 0, 10, 20, 30. The result is a final matrix where all elements are increased by 10, 20, and 30 respectively, resulting in values 0, 11, 21, 31 along the diagonal.

Arrows indicate the flow of data from Stage 1 to Stage 2 and from Stage 2 to Stage 3. The bottom right corner shows the value 57.

Prof.Miguel García Silvente

57

Propagación III

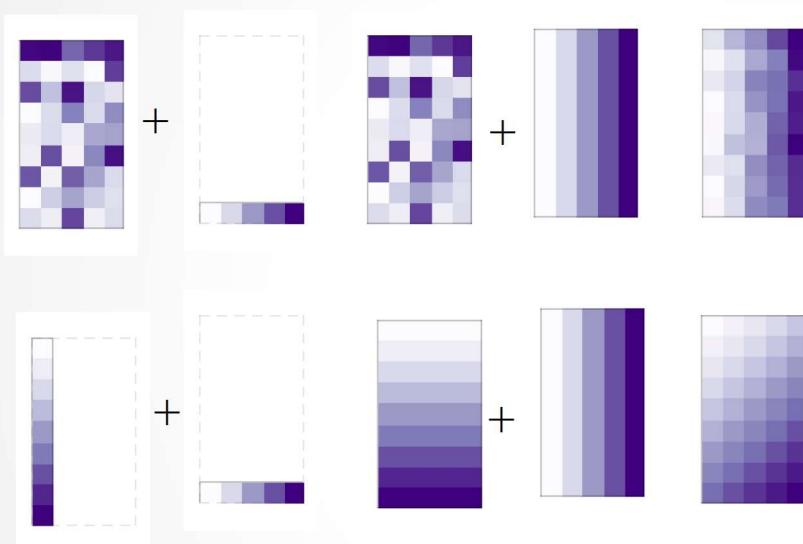
The diagram illustrates the propagation of features through two stages of computation:

- Stage 1:** A 4×4 input image with a checkerboard pattern of dark purple and light purple pixels is added to a 1×4 vector. The vector has values 0, 10, 20, 30. The result is a 4×4 image where the checkerboard pattern is scaled by 10, 20, and 30 respectively.
- Stage 2:** The resulting image from Stage 1 is added to a 1×4 vector. The vector has values 0, 1, 2, 3. The result is a final 4×4 image where the features are scaled by 1, 2, and 3 respectively.

Prof.Miguel García Silvente

58

Propagación IV

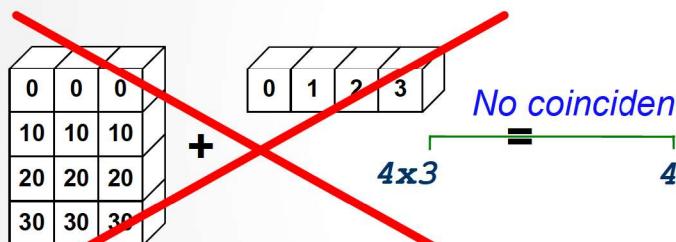


Prof.Miguel García Silvente

59

Normas de propagación

Los ejes de ambos arrays deben ser 1 o tener el mismo tamaño.
En caso contrario:, a “**ValueError: operands could not be broadcast together with shapes (4,3) (4)**”



```
>>> a = array([ [0,0,0], [10,10,10], [20,20,20], [30,30,30] ])
>>> b = array([0,1,2,3])
>>> a + b
```

Traceback (most recent call last):

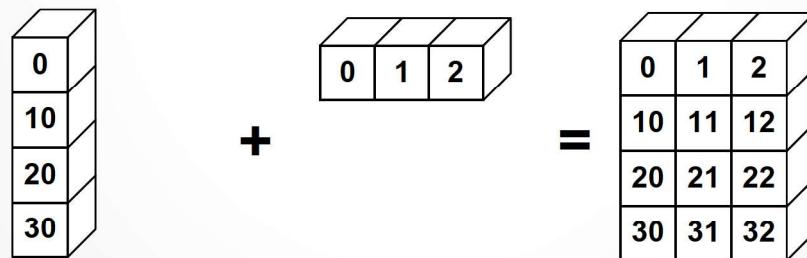
File "<stdin>", line 1, in <module>

ValueError: operands could not be broadcast together with shapes
(4,3) (4)⁶⁰

Prof.Miguel García Silvente

Ejemplo de Propagación

```
a = array((0,10,20,30))  
b = array((0,1,2))  
y = a[:, None] + b
```



Prof.Miguel García Silvente

61

Vectorización (I)

Ejemplo

```
# special.sinc ya existe  
def sinc(x):  
    if x == 0.0:  
        return 1.0  
    else:  
        w = pi*x  
        return sin(w) / w
```

```
>>> sinc(array([1.3,1.5]))  
array([-0.19809085, -0.21220659])
```

Antes...

TypeError: can't multiply
sequence to non-int

Alternativa:

```
>>> from numpy import vectorize  
>>> vsinc = vectorize(sinc)  
>>> vsinc(array([1.3,1.5]))  
array([-0.19809085, -0.21220659])  
>>> x = r_[-5:5:100j]  
>>> y = vsinc(x)
```

Prof.Miguel García Silvente

62

Vectorización (II)

Una solución más rápida (usando *where*)

```
def f_sin(x):
    x1 = zeros(x.size, float)
    x2 = sin(x)
    return where(x < 0, x1, x2)
```

Números aleatorios I

- Generar números aleatorios escalares:

```
import random
random.seed(2198)
print ('número aleatorio uniforme en (0,1):', random.random())
print ('número aleatorio uniforme en (-1,1):', random.uniform(-1,1))
print ('número aleatorio Normal(0,1):', random.gauss(0,1))
print ('número aleatorio entero:', random.randint(1,10))
```

Números aleatorios II

- Generar números aleatorios vectorizado:

```
from numpy import random  
random.seed(12)  
u = random.random(n)  
u = random.uniform(-1, 1, n) # n valores de una uniforme en (-1,1)  
u = random.normal(m, s, n) # n valores de una normal N(m,s)
```

- Los dos módulos tienen el mismo nombre:

```
import random as random_number # renombra random para escalares  
from numpy import * # random es ahora numpy.random
```

Prof.Miguel García Silvente

65

Tipo matriz (I)

- NumPy proporciona un tipo matrix similar a Matlab.
- Sólo se pueden usar arrays 1D y 2D

```
>>> x1 = array([1, 2, 3], float)  
>>> x2 = matrix(x1)          # o mat(x)  
>>> x2  
matrix([[ 1.,  2.,  3.]])  
>>> x3 = mat(x1).transpose()    # vector columna  
>>> x3  
matrix([[ 1.], [ 2.], [ 3.]])  
>>> type(x3)  
<class 'numpy.matrixlib.defmatrix.matrix'>  
>>> isinstance(x3, matrix)  
True
```

Prof.Miguel García Silvente

66

Tipo matriz (II)

- El operador * corresponde a la multiplicación entre matrices

```
>>> A = eye(3)
>>> A = mat(A)
>>> A
matrix([[ 1.,  0.,  0.],
        [ 0.,  1.,  0.],
        [ 0.,  0.,  1.]])
>>> y2 = x2*A
>>> y2
matrix([[ 1.,  2.,  3.]])
>>> y3 = A*x3
```

```
>>> y3
matrix([[ 1.],
        [ 2.],
        [ 3.]])
```

Prof.Miguel García Silvente

67

Python con C/C++/Fortran

- weave (C/C++)
- f2py (Fortran)
- Cython (basado en Pyrex) C/C++
- ctypes (C)
- SWIG (C/C++)
- Boost.Python (C++)
- RPy / RSPython ®
- PyPy

Prof.Miguel García Silvente

68

Representación gráfica en python

- Matplotlib para uso habitual.
<http://matplotlib.sourceforge.net>

Matplotlib está integrado en iPython.

- Chaco para construir aplicaciones interactivas de dibujo
<http://code.enthought.com/projects/chaco>

Es más útil como toolkit que para hacer gráficas simples.

Ambos requieren NumPy

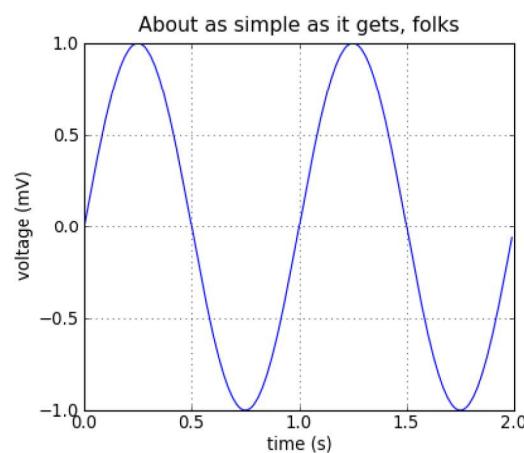
Prof.Miguel García Silvente

69

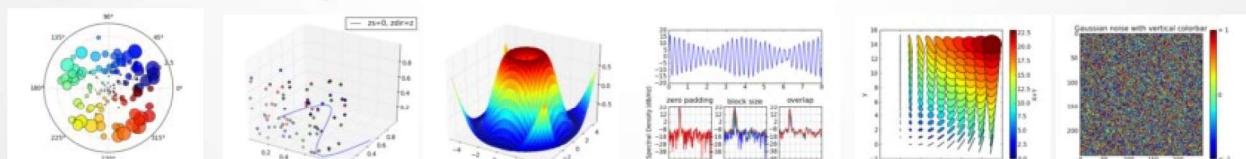
Matplotlib

```
from pylab import *
t = arange(0.0, 2.0, 0.01)
s = sin(2*pi*t)
plot(t, s, linewidth=1.0)

xlabel('time (s)')
ylabel('voltage (mV)')
title('About as simple as it gets, folks')
grid(True)
show()
```



<http://matplotlib.sourceforge.net/users/screenshots.html>

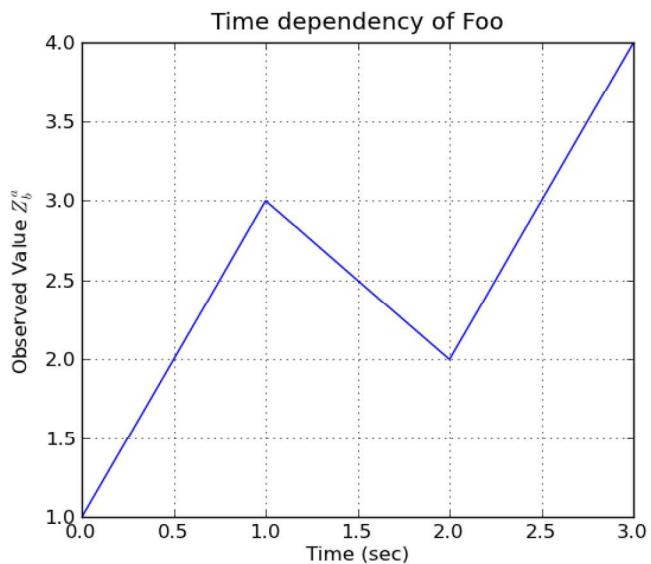


Prof.Miguel García Silvente

70

Gráfica simple con Latex

```
plot([1,3,2,4])
title('Time dependency of Foo')
xlabel('Time (sec)')
ylabel(r'Observed Value $Z^a_{\{a\}}_{\{b\}}$')
grid(True)
```

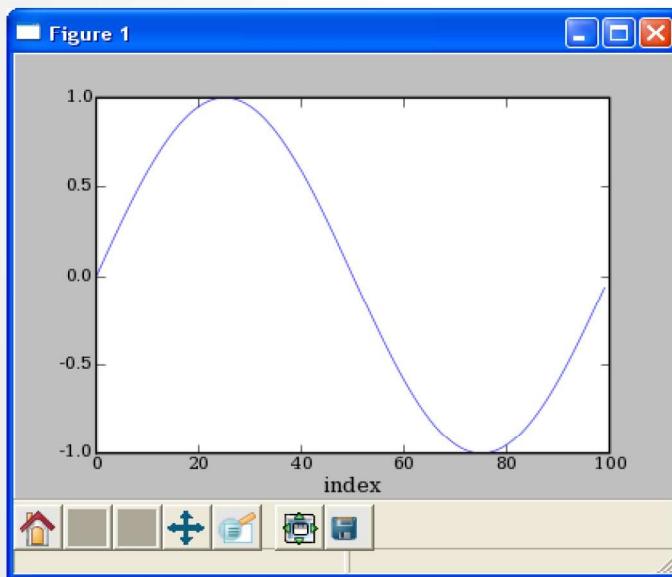


Prof.Miguel García Silvente

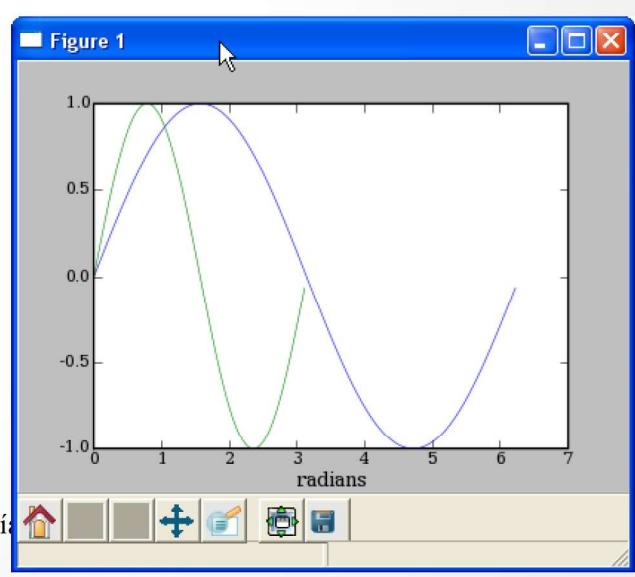
71

Gráficas de datos

```
>>> x = arange(50)*2*pi/50.
>>> y = sin(x)
>>> plot(y)
>>> xlabel('index')
```



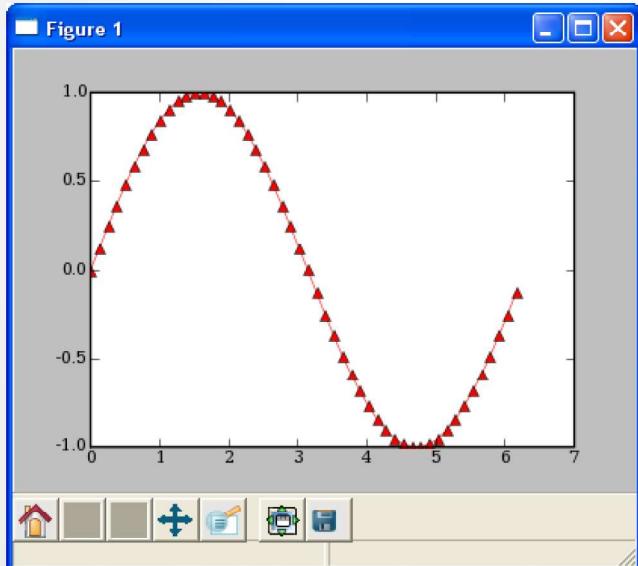
```
>>> plot(x,y,x2,y2)
>>> xlabel('radians')
```



Gráficas de datos

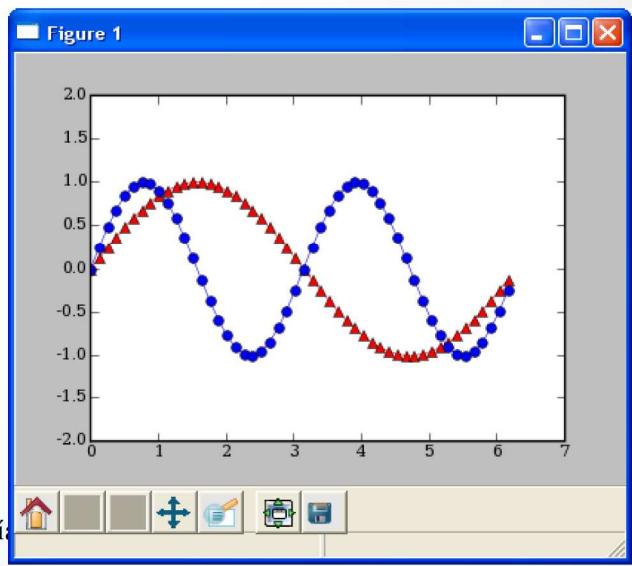
Formatos de línea

```
# color rojo, punteado, triángulos  
>>> plot(x,sin(x),'r^-')
```



Varias gráficas

```
>>> plot(x,y1,'b-o', x,y2, r-^')  
>>> axis([0,7,-2,2])
```



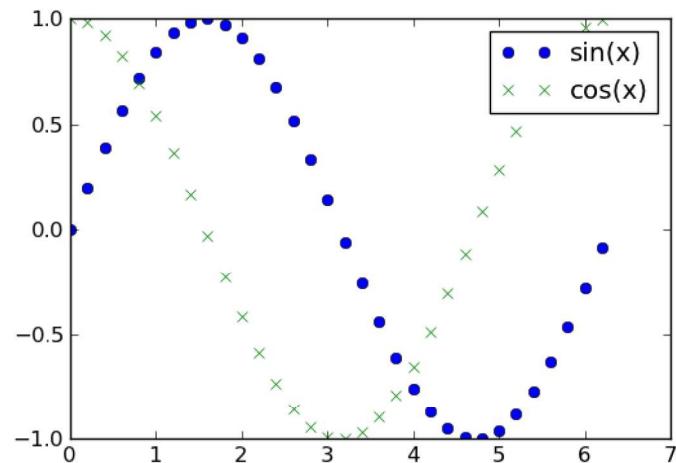
Gráfica dispersa con leyenda

```
x = arange(0.,2.*math.pi,0.2)  
print "Shape de x",x.shape
```

```
plot(x, sin(x), 'o', label="sin(x)")  
plot(x, cos(x), 'x', label="cos(x)")
```

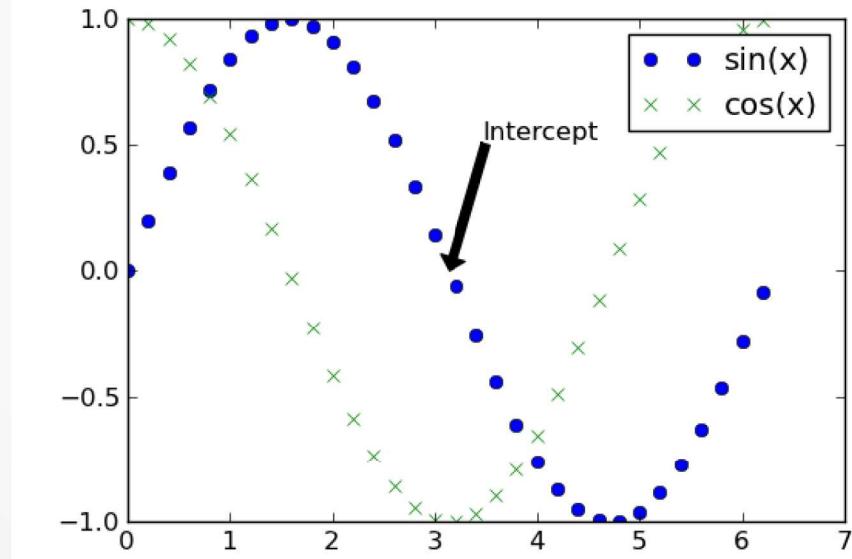
```
legend()
```

```
savefig("plot.svg")  
# formatos pdf,png,ps,eps
```



Anotaciones

```
annotate('Intercept', xy=(math.pi,0), xytext=(3.5,0.5),  
        arrowprops=dict(facecolor='black'))
```



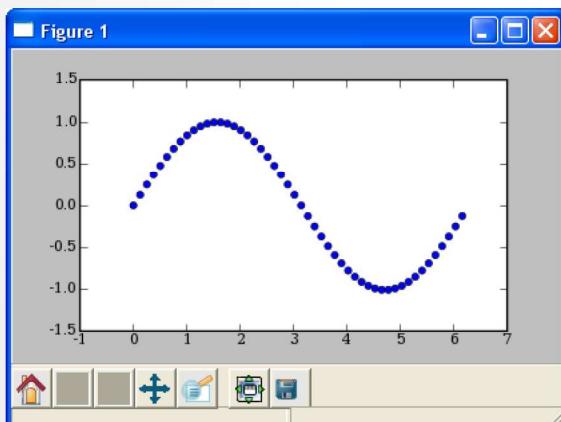
Prof.Miguel García Silvente

75

Gráficas dispersas

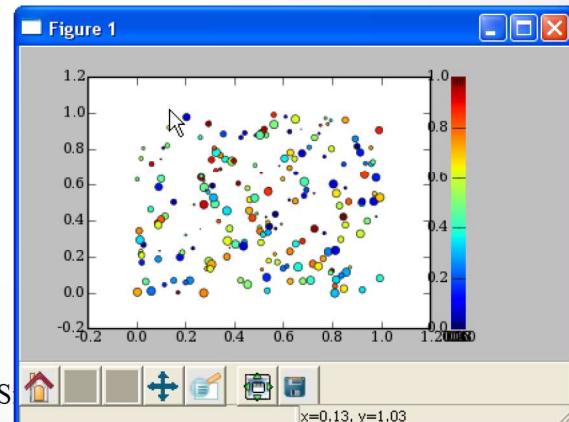
Gráficas simples

```
>>> x = arange(50)*2*pi/50.  
>>> y = sin(x)  
>>> scatter(x,y)
```



Mapas dispersos coloreados

```
# Tamaño / color  
>>> x = rand(200)  
>>> y = rand(200)  
>>> size = rand(200)*30  
>>> color = rand(200)  
>>> scatter(x, y, size, color)  
>>> colorbar()
```

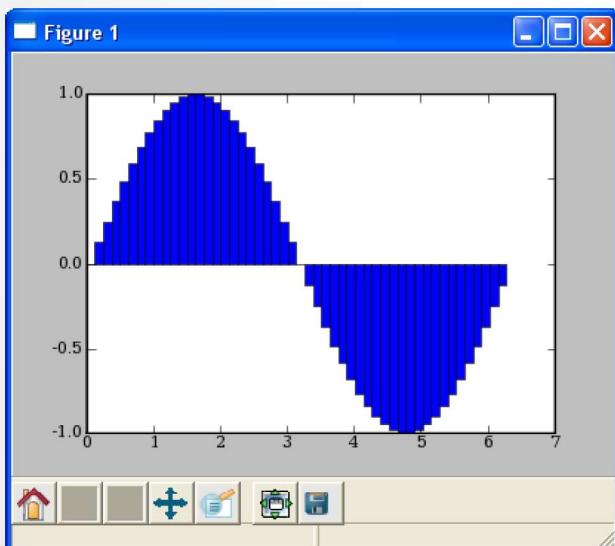


Miguel García S

Gráficos de barras

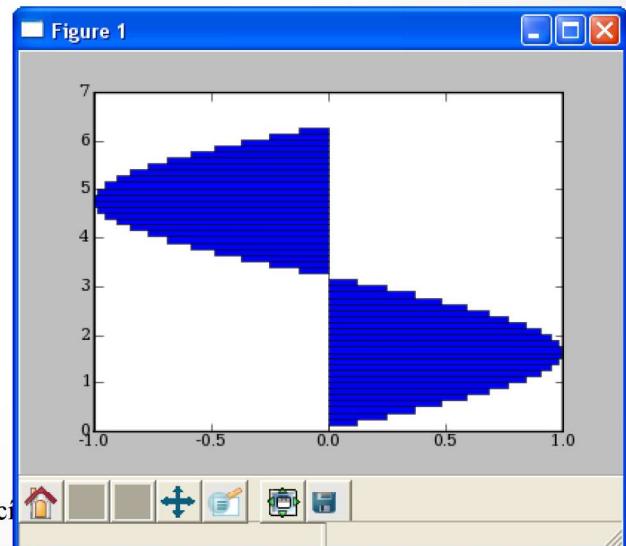
Verticales

```
>>> bar(x,sin(x),  
       width=x[1]-x[0])
```



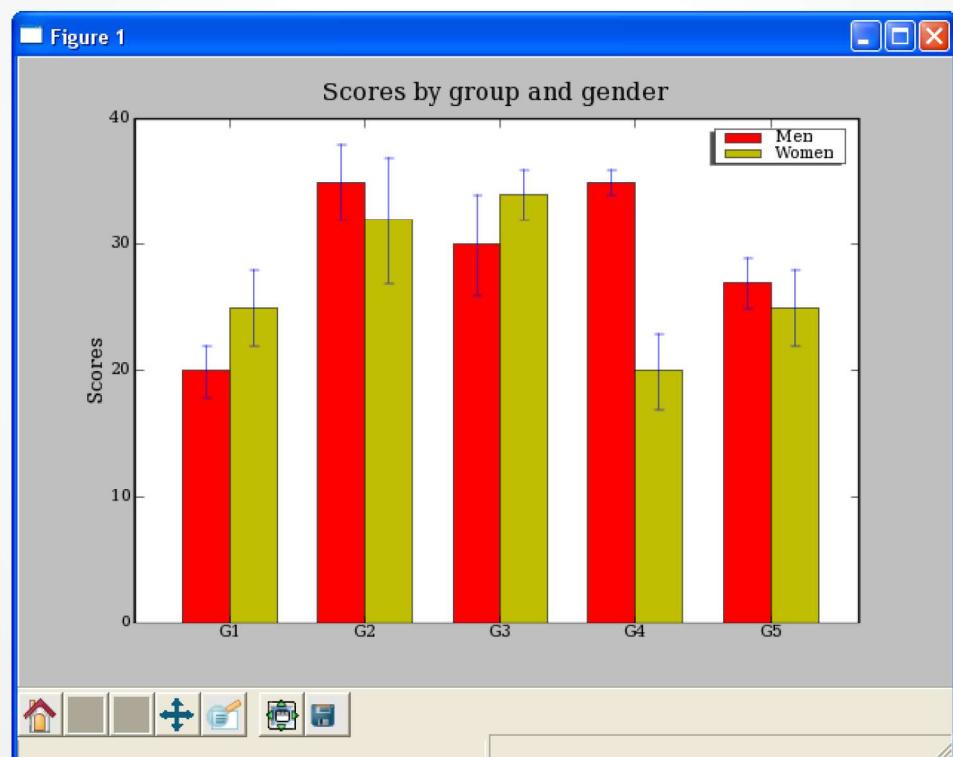
Horizontales

```
>>> barh(x,sin(x),  
        height=x[1]-x[0])
```



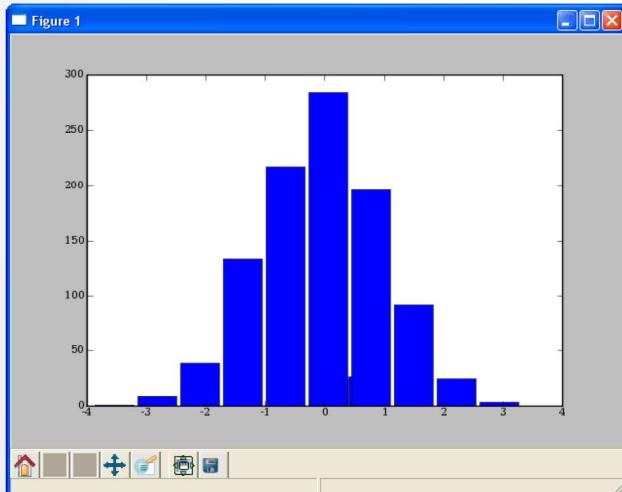
Gráficos de barras

http://matplotlib.org/examples/pylab_examples/barchart_demo.html

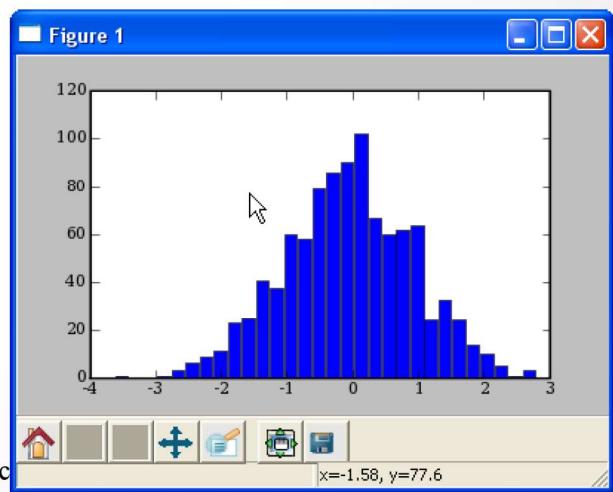


Histogramas

```
# dibujar histograma  
# por defecto hay 10 grupos  
>>> hist(np.random.randn(1000))
```



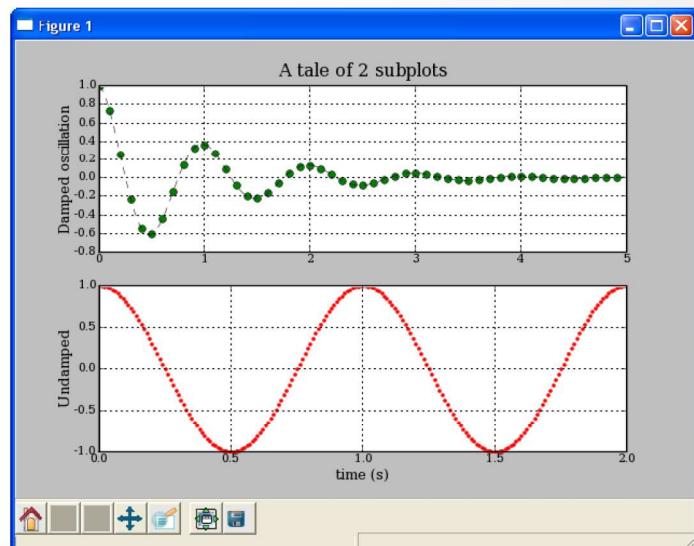
```
# cambiar el número de grupos  
>>> hist(np.random.randn(1000), 30)
```



Varias gráficas con subplot

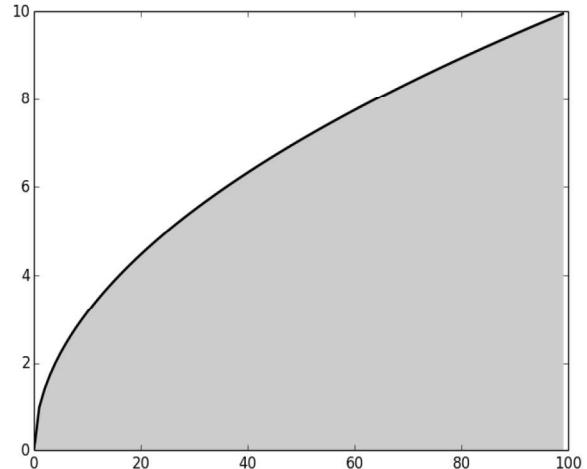
http://matplotlib.org/examples/pylab_examples/subplot_demo.html

```
import numpy as np  
import matplotlib.pyplot as plt  
  
x1 = np.linspace(0.0, 5.0)  
x2 = np.linspace(0.0, 2.0)  
  
y1 = np.cos(2 * np.pi * x1) * np.exp(-x1)  
y2 = np.cos(2 * np.pi * x2)  
  
plt.subplot(2, 1, 1)  
plt.plot(x1, y1, 'ko-')  
plt.title('A tale of 2 subplots')  
plt.ylabel('Damped oscillation')  
  
plt.subplot(2, 1, 2)  
plt.plot(x2, y2, 'r.-')  
plt.xlabel('time (s)')  
plt.ylabel('Undamped')  
plt.show()
```



Gráficas con relleno

```
import matplotlib.pyplot as plt
from math import sqrt
x = range(100)
y = [sqrt(i) for i in x]
plt.plot(x,y,color='k',lw=2)
plt.fill_between(x,y,0,color='0.8')
plt.show()
```

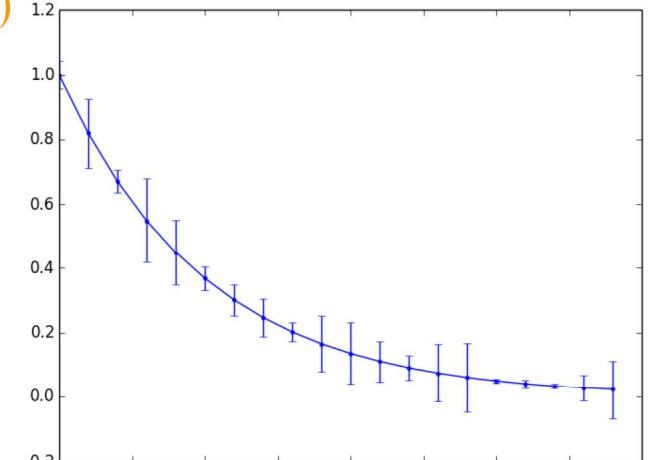


Prof.Miguel García Silvente

81

Gráficos de barras de error

```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(0, 4, 0.2)
y = np.exp(-x)
el = 0.1 * np.abs(np.random.randn(len(y)))
plt.errorbar(x, y, yerr=el, fmt='.-')
plt.show()
```

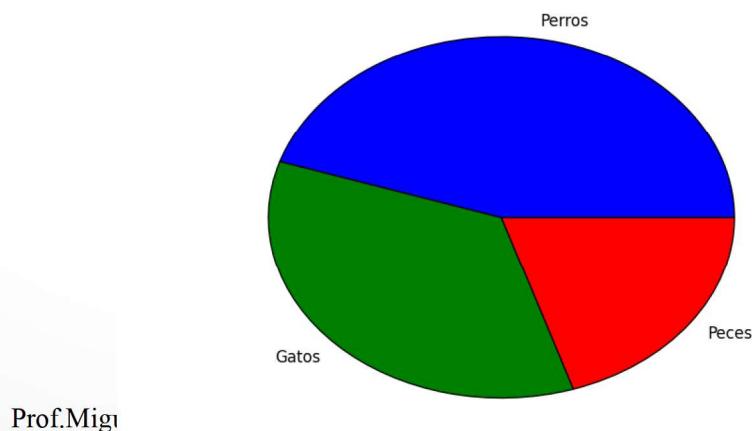


Prof.Miguel García Silvente

81

Gráficos de tarta (I)

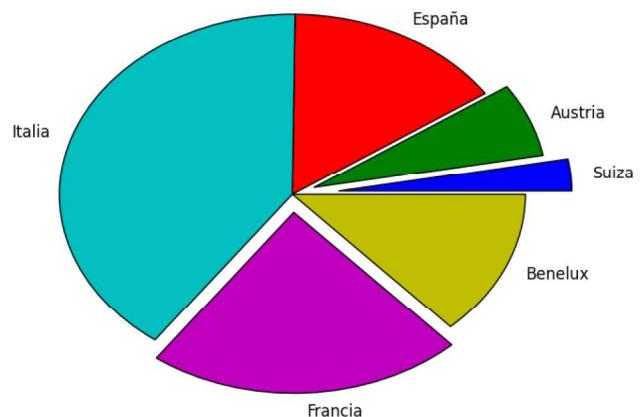
```
import matplotlib.pyplot as plt  
plt.figure(figsize=(3,3))  
x = [45, 35, 20]  
labels = ['Perros', 'Gatos', 'Peces']  
plt.pie(x, labels=labels)  
plt.show()
```



Prof.Miguel

Gráficos de tarta (II)

```
import matplotlib.pyplot as plt  
plt.figure(figsize=(3,3))  
x = [4, 9, 21, 55, 30, 18]  
labels = ['Suiza', 'Austria', 'España', 'Italia', 'Francia', 'Benelux']  
explode = [0.2, 0.1, 0, 0, 0.1, 0]  
plt.pie(x, labels=labels, explode=explode)  
plt.show()
```



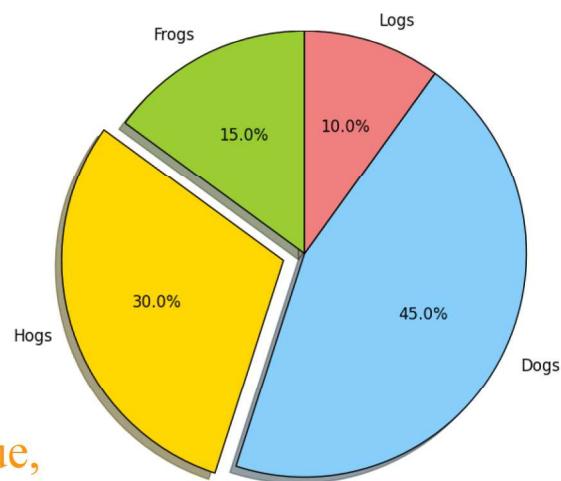
Prof.Miguel

Gráficos de tarta (III)

```
import matplotlib.pyplot as plt  
labels = 'Frogs', 'Hogs', 'Dogs', 'Logs'  
sizes = [15, 30, 45, 10]  
colors = ['yellowgreen', 'gold',  
'lightskyblue', 'lightcoral']  
explode = (0, 0.1, 0, 0)
```

```
plt.pie(sizes, explode=explode,  
labels=labels, colors=colors,  
 autopct='%.1f%%', shadow=True,  
 startangle=90)
```

```
plt.axis('equal')  
plt.show()
```

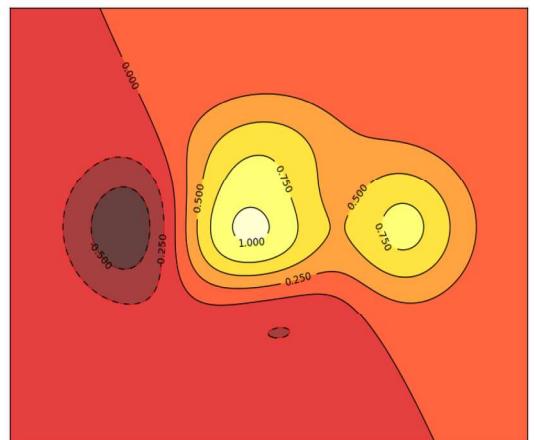


Prof.Miguel García Silvente

85

Gráficos de contornos

```
def f(x, y):  
    return (1 - x / 2 + x ** 5 + y ** 3) * \  
           np.exp(-x ** 2 -y ** 2)  
  
n = 256  
x = np.linspace(-3, 3, n)  
y = np.linspace(-3, 3, n)  
X, Y = np.meshgrid(x, y)  
plt.contourf(X, Y, f(X, Y), 8, alpha=.75, cmap='jet')  
C = plt.contour(X, Y, f(X, Y), 8, colors='black', linewidth=.5)
```



Prof.Miguel García Silvente

86

Gráficos de caminos

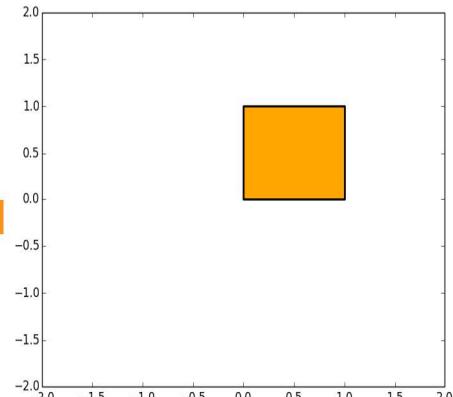
```
import matplotlib.pyplot as plt
from matplotlib.path import Path
import matplotlib.patches as patches

verts = [(0., 0.), (0., 1.), (1., 1.), (1., 0.), (0., 0.)]

codes = [Path.MOVETO, Path.LINETO,
         Path.LINETO, Path.LINETO, Path.CLOSEPOLY,]

path = Path(verts, codes)
fig = plt.figure()
ax = fig.add_subplot(111)
patch = patches.PathPatch(path, facecolor='orange', lw=2)
ax.add_patch(patch)
ax.set_xlim(-2,2)
ax.set_ylim(-2,2)

plt.show()
```



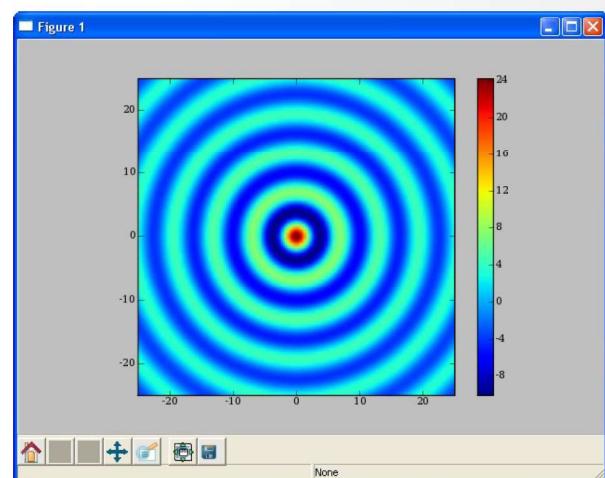
Prof.Miguel García Silvente

87

Visualizar imágenes

```
# Creamos una matriz donde los valores
# corresponde a la distancia al centro
# de la matriz.
from numpy import mgrid
from scipy import special
x,y = mgrid[-25:25:100j, -25:25:100j]
r = numpy.sqrt(x**2+y**2)
# Calculamos la función bessel
# de cada punto de la matriz y
# escalamos
s = special.j0(r)*25

# Mostrar la superficie.
plt.imshow(s, extent=[-25,25,-25,25])
plt.colorbar()
```

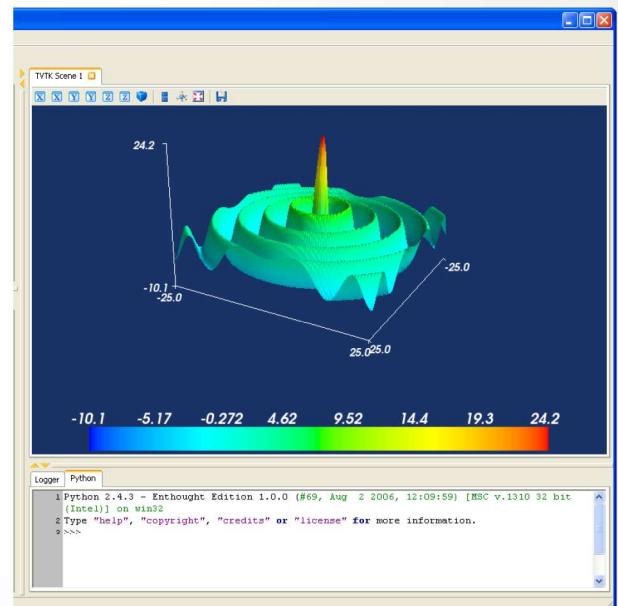


Prof.Miguel García Silvente

88

Superficies con mlab

```
# Create 2d array where values  
# are radial distance from  
# the center of array.  
from numpy import mgrid  
from scipy import special  
x,y = mgrid[-25:25:100j,-25:25:100j]  
r = numpy.sqrt(x**2+y**2)  
# Calculate bessel function of  
# each point in array and scale  
s = special.j0(r)*25  
  
# Display surface plot.  
from enthought.mayavi import mlab  
mlab.surf(x,y,s)  
mlab.scalarbar()  
mlab.axes()
```



Prof.Miguel García Silvente

89

Gráficas en la web

```
import sys  
import cgi  
form = cgi.FieldStorage()  
form_data = form.getFirst('data', '1,3,2,2,4')  
data = [int(x) for x in form_data.split(',')]
```

[http://arrecife.ugr.es/cgi-bin/grafica_cgi.py?
data=1,2,3,5,10](http://arrecife.ugr.es/cgi-bin/grafica_cgi.py?data=1,2,3,5,10)

```
import matplotlib  
matplotlib.use('Agg')  
import matplotlib.pyplot as plt  
plt.plot(data)
```

```
print("Content-Type: image/png")  
print("")  
sys.stdout.flush()  
plt.savefig(sys.stdout, format='png')
```

Prof.Miguel García Silvente

90

SciPy

- Está disponible en www.scipy.org
- Es Open Source con licencia BSD

Paquetes

- Funciones especiales (scipy.special)
- Procesamiento de señales (scipy.signal)
- Procesamiento de imágenes (scipy.ndimage)
- Transformada de Fourier (scipy.fftpack)
- Optimización (scipy.optimize)
- Integración Numérica (scipy.integrate)
- Algebra Lineal (scipy.linalg)

Prof.Miguel García Silvente

- Entrada/Salida (scipy.io)
- Estadística (scipy.stats)
- Ejecución rápida (scipy.weave)
- Algoritmos de Clustering (scipy.cluster)
- Matrices Dispersas (scipy.sparse)
- Interpolación (scipy.interpolate)
- Otros (e.g. scipy.odr, scipy.maxentropy)

91

Interpolación con splines 1D

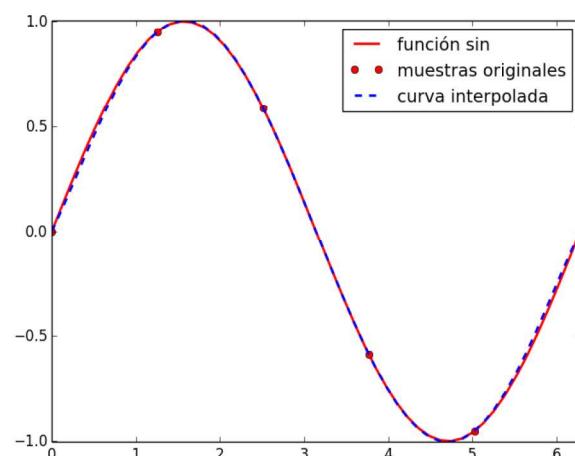
```
from scipy.interpolate import interp1d
from pylab import plot, axis, legend
from numpy import linspace

# valores
x = linspace(0,2*pi,6)
y = sin(x)

# Crear un spline para interpolar.
# kind=5 spline de 5th grado.
# kind=0 -> orden cero.
# kind=1 o 'linear' -> interpolación lineal

spline_fit = interp1d(x,y,kind=5)
xx = linspace(0,2*pi, 50)
yy = spline_fit(xx)

# mostrar los resultados.
plot(xx, sin(xx), 'r-', x,y,'ro',xx,yy, 'b--',linewidth=2)
axis('tight')
legend(['función sin', 'muestras originales', 'curva interpolada'])
```



Prof.Miguel García Silvente

92

Procesamiento de señales

scipy.signal --- Signal and Image Processing

Filtrado

Convolución 2D generalizada (con condiciones de borde)

Convolución N-D

Filtrado B-spline

Filtro de orden N-D, filtro mediana N-D,

Filtrado IIR y FIR y diseño de filtros

Sistemas LTI

Simulación de sistemas

Respuestas impulso y salto

Expansión

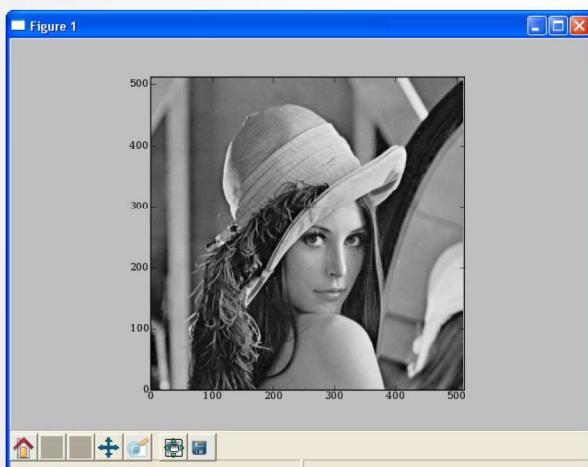
Prof.Miguel García Silvente

93

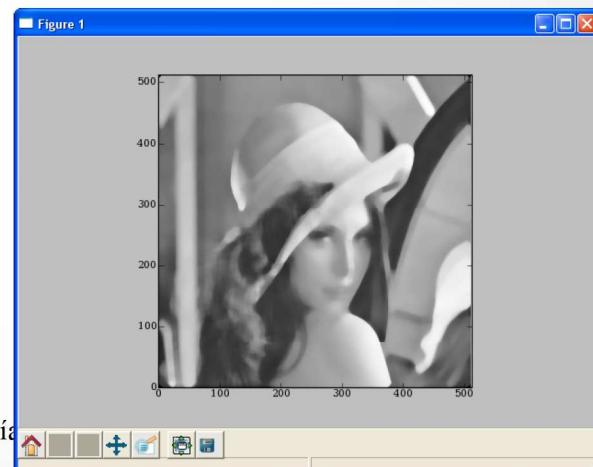
Procesamiento de imágenes (I)

```
from scipy.misc import lena
from scipy import signal
import matplotlib.pyplot as plt

lena = lena().astype(float32)
plt.imshow(lena, cmap=cm.gray)
# Emborronamiento usando la media
fl = signal.medfilt2d(lena, [15,15])
plt.imshow(fl, cmap="gray")
```

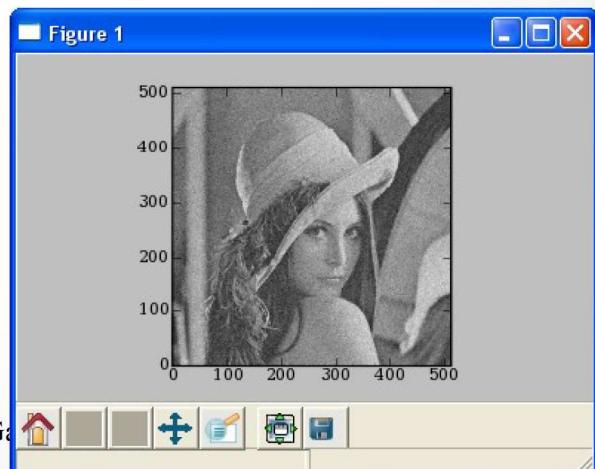
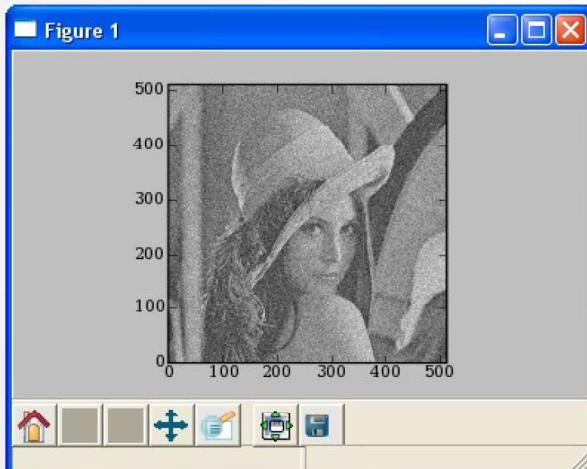


Miguel García



Procesamiento de imágenes (II)

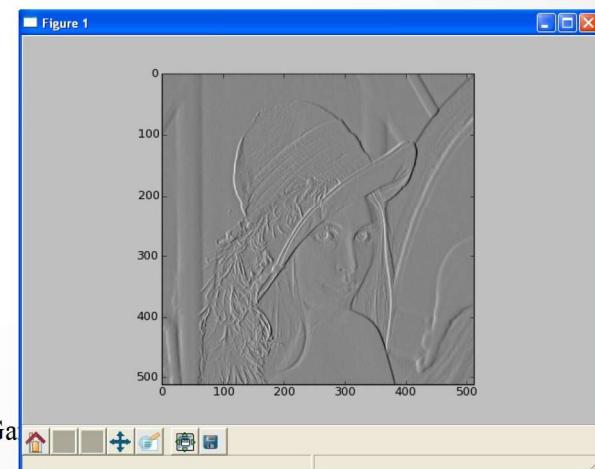
```
# Eliminar ruido usando filtro de wiener
from scipy.stats import norm
ln = lena + norm(0,32).rvs(lena.shape)
imshow(ln)
cleaned = signal.wiener(ln)
imshow(cleaned)
```



95

Procesamiento de imágenes (III)

```
# Detección de fronteras usando el filtro de Sobel
from scipy.ndimage.filters import sobel
imshow(lena)
edges = sobel(lena)
imshow(edges)
```



96

Optimización

scipy.optimize --- minimización sin restricciones y cálculo de raíces

- Optimización sin restricciones

`fmin` (Nelder-Mead simplex), `fmin_powell` (método Powell), `fmin_bfgs` (método BFGS quasi-Newton), `fmin_ncg` (gradiente conjugado de Newton), `leastsq` (Levenberg-Marquardt), `anneal` (minimizador global por simulated annealing), `brute` (minimizador global por fuerza bruta), `brent` (Minimizador 1-D), `golden`, `bracket`

- Optimizacion con restricciones

`fmin_l_bfgs_b`, `fmin_tnc` (código Newton truncado), `fmin_cobyla` (optimización con restricciones por aproximación lineal), `fminbound` (minimizador por intervalos con restricciones por intervalo 1-d)

- Encontrar raíces

`fsolve` (usando MINPACK), `brentq`, `brenth`, `ridder`, `newton`, `bisect`, `fixed_point` (resuelve ecuaciones de punto fijo)

Prof.Miguel García Silvente

97

Optimización (I)

Minimizar función BESSEL

```
# minimiza función bessel
# de primer orden entre 4 y 7
from scipy.special import j1
from scipy.optimize import \
    fminbound

x = r_[2:7.1:.1]
j1x = j1(x)
plot(x,j1x,'-')
hold(True)
x_min = fminbound(j1,4,7)
j1_min = j1(x_min)
plot([x_min],[j1_min],'ro')
```



Prof.Miguel García Silvente

98

Optimización (II)

Resolver ecuaciones no lineales

$$\begin{aligned}3x_0 - \cos(x_1 x_2) + a &= 0 \\x_0^2 - 81(x_1 + 0.1)^2 + \sin(x_2) + b &= 0 \\e^{-x_0 x_1} + 20x_2 + c &= 0\end{aligned}$$

Localización de inicio de búsqueda

```
from scipy.optimize import fsolve
from math import *
def nonlin(x,a,b,c):
    x0,x1,x2 = x
    return [3*x0-cos(x1*x2)+ a,
            x0*x0-81*(x1+0.1)**2 + sin(x2)+b,
            exp(-x0*x1)+20*x2+c]
a,b,c = -0.5,1.06,(10*pi-3.0)/3
root = fsolve(nonlin,
[0.1,0.1,-0.1],args=(a,b,c))
print (root)
[ 0.5,1.38106244e-13,-5.23598776e-01]
print (nonlin(root,a,b,c))
[4.4408920985006262e-16,
-2.2311041902867146e-12,
7.460698725481052e-14]
```

Prof.Miguel García Silvente

99

Optimización (III)

<http://docs.scipy.org/doc/scipy-0.10.0/reference/tutorial/optimize.html>

Minimizar función de ROSEN BROCK

Función Rosenbrock $f(\mathbf{x}) = \sum_{i=1}^{N-1} 100 \left(x_i - x_{i-1}^2 \right)^2 + (1 - x_{i-1})^2$.

Sin derivar (Nelder-Mead)

```
from scipy.optimize import *
rosen = optimize.rosen
import time
x0 = [1.3,0.7,0.8,1.9,1.2]
start = time.time()
xopt = fmin(rosen, x0)
stop = time.time()
print(xopt)
print("Calculado en ", stop-start, "s.")
```

Optimization terminated successfully.

Current function value: 0.000066

Iterations: 141

Function evaluations: 243

[0.99910115 0.99820923 0.99646346
0.99297555 0.98600385]

Calculado en 0.05863213539123535 s.

Derivando (Broyden-Fletcher-Goldfarb-Shanno)

```
from scipy.optimize import *
rosen = optimize.rosen
import time
x0 = [1.3,0.7,0.8,1.9,1.2]
start = time.time()
xopt = optimize.fmin_bfgs(rosen, x0, fprime=rosen_der)
stop = time.time()
print(xopt)
print("Calculado en ", stop-start, "s.")
```

Optimization terminated successfully.

Current function value: 0.000000

Iterations: 51

Function evaluations: 63

Gradient evaluations: 63

[1. 0.9999999 0.9999999 0.9999997 0.9999994]

Calculado en 0.12189006805419922 s.

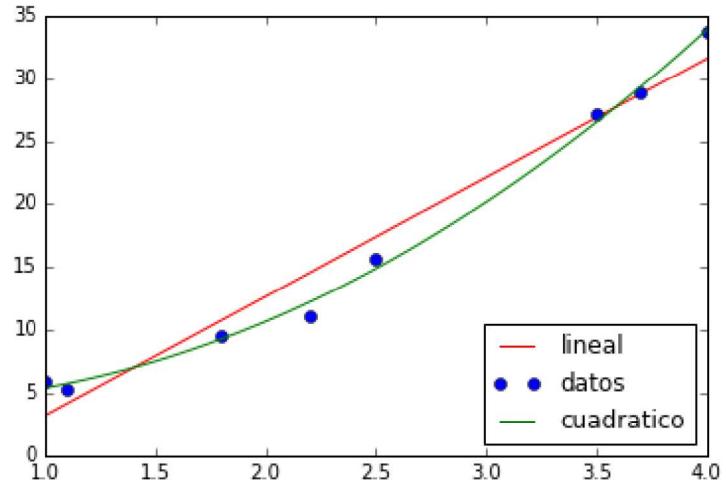
100

Prof.Miguel García Silvente

Optimización (IV)

Ajuste no lineal por mínimos cuadrados

```
x=np.array([1.0,2.5,3.5,4.0,1.1,1.8,2.2,3.7])
y=np.array([6.008,15.722,27.130,33.772,5.2
           ,11.098,28.828])
funcLine=lambda tpl,x : tpl[0]*x+tpl[1]
funcQuad=lambda tpl,x : tpl[0]*x**2+tpl[1]*x
func=funcLine
ErrorFunc=lambda tpl,x,y: func(tpl,x)-y
TpIInitial1=(1.0,2.0)
tplFinal1,success=leastsq(ErrorFunc,tplInitial1
                           =(x,y))
print(" linear fit ",tplFinal1)
xx1=np.linspace(x.min(),x.max(),50)
yy1=func(tplFinal1,xx1)
...
plt.plot(xx1,yy1,'r-',x,y,'bo',xx2,yy2,'g-')
plt.show()
```



Prof.Miguel García Silvente

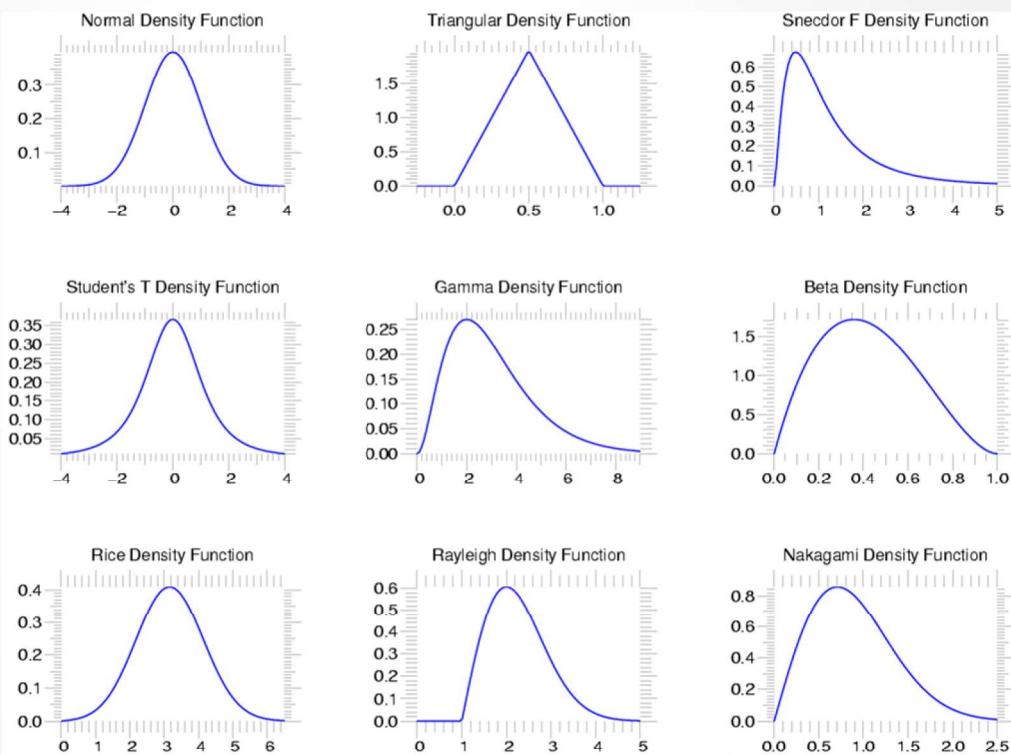
101

Estadística (I)

scipy.stats --- Distribuciones continuas

Alrededor de
80
distribuciones
continuas

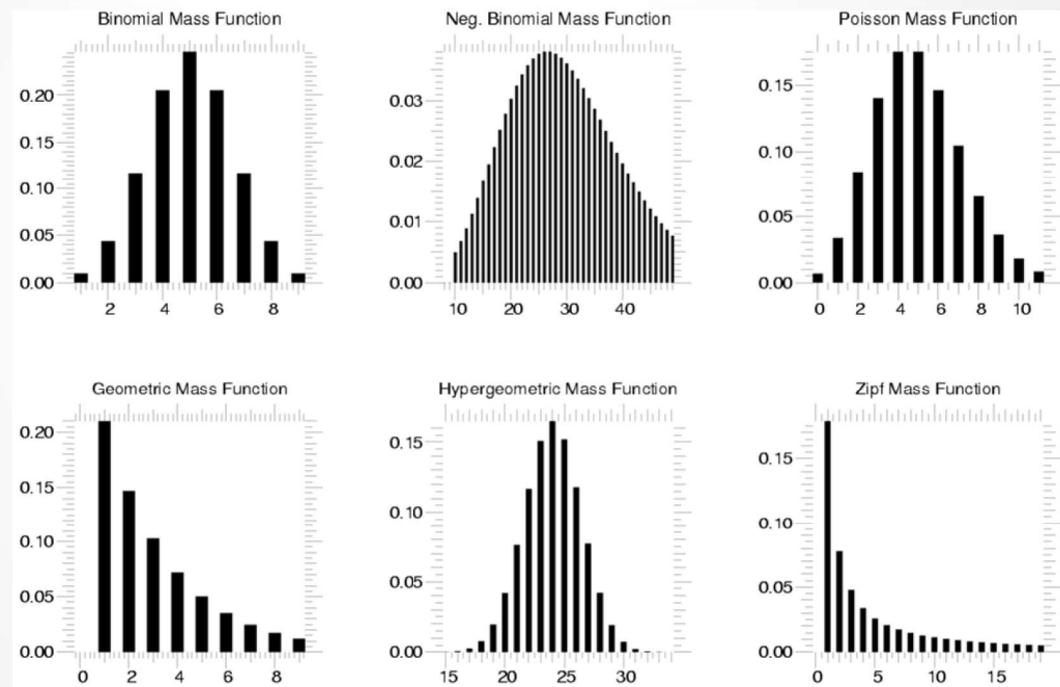
Métodos
pdf
cdf
rvs
ppf
stats



Estadística (II)

scipy.stats --- Distribuciones discretas

10
distribuciones
discretas



Métodos

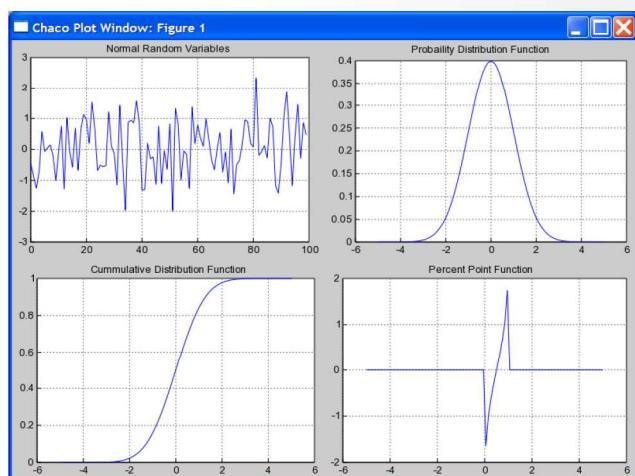
pdf
cdf
rvs
ppf
stats

Estadística (III)

Distribuciones

```
# Muestreo de una dist.normal
samp = stats.norm.rvs(size=100)

x = r_[-5:5:100j]
# Calcular dist. probabilidad
pdf = stats.norm.pdf(x)
# Calcular dist.acumulada
cdf = stats.norm.cdf(x)
# Calcular función Percent Point
ppf = stats.norm.ppf(x)
```



Estadísticas (IV)

scipy.stats --- Cálculos estadísticos básicos sobre datos

- `numpy.mean, numpy.std, numpy.var, numpy.cov`
- `stats.skew, stats.kurtosis, stats.moment`

scipy.stats.bayes_mvs --- Media, varianza y desviación estándar bayesianas

```
# Distribución Gamma con a=2.5
grv = stats.gamma(2.5)
grv.stats()      # Media y varianza teóricas
(array(2.5), array(2.5))
# media, varianza y desviación estándar estimados con 95% de confianza
vals = grv.rvs(size=100)
stats.bayes_mvs(vals, alpha=0.95)
((2.52887906081, (2.19560839724, 2.86214972438)),
 (2.87924964268, (2.17476164549, 3.8070215789)),
 (1.69246760584, (1.47470730841, 1.95115903475)))
# (media e intervalo de confianza para cada estadístico)
```

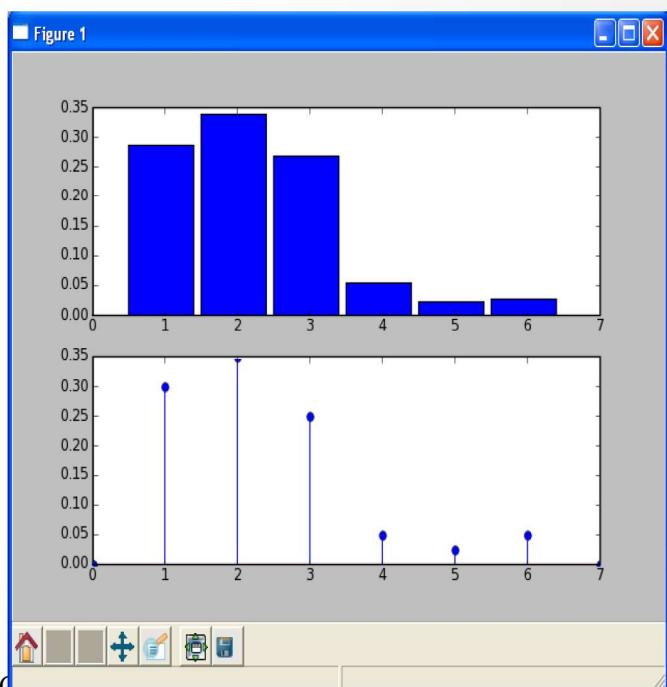
105

Objetos estadísticos

Crear nuevas distribuciones discretas

```
# Crear un dado cargado.
from scipy.stats import rv_discrete
xk = [1,2,3,4,5,6]
pk = [0.3,0.35,0.25,0.05,
      0.025,0.025]
new = rv_discrete(name='creada',
                   values=(xk,pk))

# Calcular histograma
samples = new.rvs(size=1000)
bins=linspace(0.5,5.5,6)
subplot(2,1,1)
hist(samples,bins=bins,normed=True)
# Calcular pmf (probability mass func)
x = range(0,8)
subplot(2,1,2)
stem(x,new.pmf(x))
```

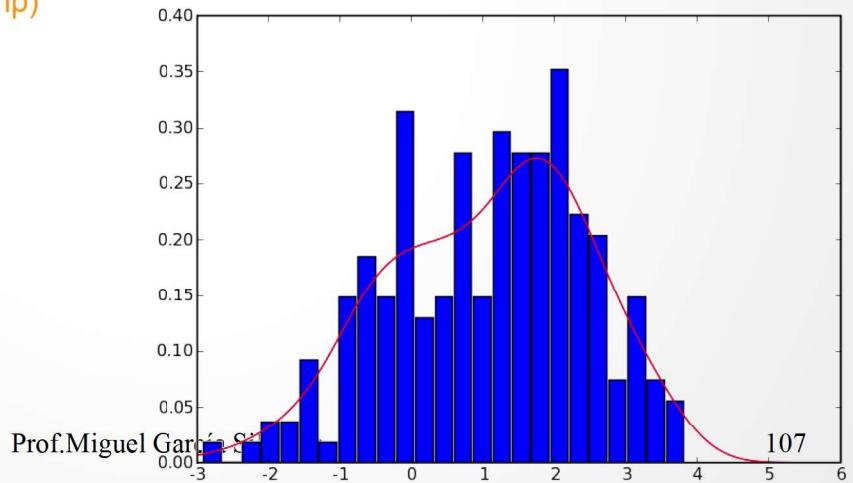


Prof.Miguel García Silvente

Estadísticas

Estimación de una PDF continua usando estimación de densidad de núcleos gaussianos

```
# Muestra dist. normal dist. 100 veces.  
rv1 = stats.norm()  
rv2 = stats.norm(2.0,0.8)  
samp = r_[rv1.rvs(size=100), rv2.rvs(size=100)]  
# Estimación del núcleo (suavizar histograma)  
apdf = stats.kde.gaussian_kde(samp)  
x = linspace(-3,6,200)  
plot(x, apdf(x),'r')  
  
# Histograma  
hist(samp, bins=25, normed=True)
```



Algebra Lineal (I)

scipy.linalg --- Algebra lineal rápida

- Es recomendable usar ATLAS
- Acceso a bajo nivel a las rutinas de BLAS y LAPACK de los módulos `linalg.fblas` y `linalg.flapack` (con orden FORTRAN)
- Rutinas sobre matrices a alto nivel
 - Algebra Lineal Básica: `inv`, `solve`, `det`, `norm`, `lstsq`, `pinv`
 - Descomposiciones: `eig`, `lu`, `svd`, `orth`, `cholesky`, `qr`, `schur`
 - Funciones sobre matrices: `expm`, `logm`, `sqrtm`, `cosm`, `coshm`, `funm` (general matrix functions)

Algebra Lineal (II)

Factorización LU

```
from scipy import linalg
a = array([[1,3,5],
           [2,5,1],
           [2,3,6]])
# lu es una combinación de L y U
# piv es la matriz de permutación
# a = P L U
lu, piv = linalg.lu_factor(a)

# solución rápida de un sistema
b = array([10,8,3])
linalg.lu_solve((lu, piv), b)
array([-7.82608696,  4.56521739,
       0.82608696])
```

Autovalores y autovectores

```
from scipy import linalg
a = array([[1,3,5], [2,5,1], [2,3,6]])
# calcula auto valores/vectores
>>> vals, vecs = linalg.eig(a)
# autovalores
print(vals)
array([ 9.39895873+0.j,
        -0.73379338+0.j,
        3.33483465+0.j])
# autovectores en columnas
# el primer autovector
print(vecs[:,0])
array([-0.57028326,
       -0.41979215,
       -0.70608183])
# la norma debe ser 1.0
print( linalg.norm(vecs[:,0]) )
1.0
```

Prof.Miguel García Silvente

109

Integración (I)

scipy.integrate --- Integración de propósito general

- Ordinary Differential Equations (ODE)

integrate.odeint, integrate.ode

- Usando muestreo de funciones 1D:

integrate.trapz (método trapezoidal), **integrate.simps** (Método Simpson), **integrate.romb** (Método Romberg)

- Arbitrary callable function

integrate.quad (propósito general), **integrate.dblquad** (integración doble), **integrate.tplquad** (integración triple), **integrate.fixed_quad** (inegración gaussiana de orden fijo), **integrate.quadrature** (Cuadratura Gaussian), **integrate.romberg** (Romberg)

Prof.Miguel García Silvente

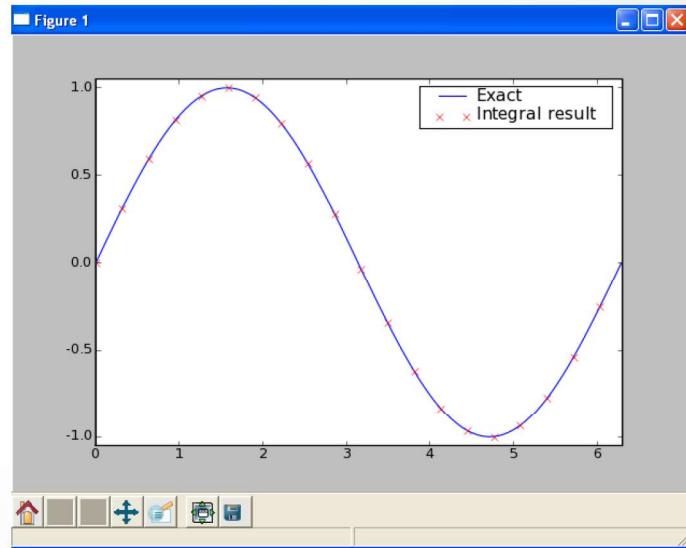
110

Integración (II)

scipy.integrate --- Example

```
# Comparar sin con integral(cos)
def func(x):
    return integrate.quad(cos,0,x)[0]
vecfunc = vectorize(func)

x = r_[0:2*pi:100j]
x2 = x[::5]
y = sin(x)
y2 = vecfunc(x2)
plot(x,y,x2,y2,'rx')
legend(['Exact', 'Integral Result'])
```



Prof.Miguel García Silvente

111

Funciones especiales (I)

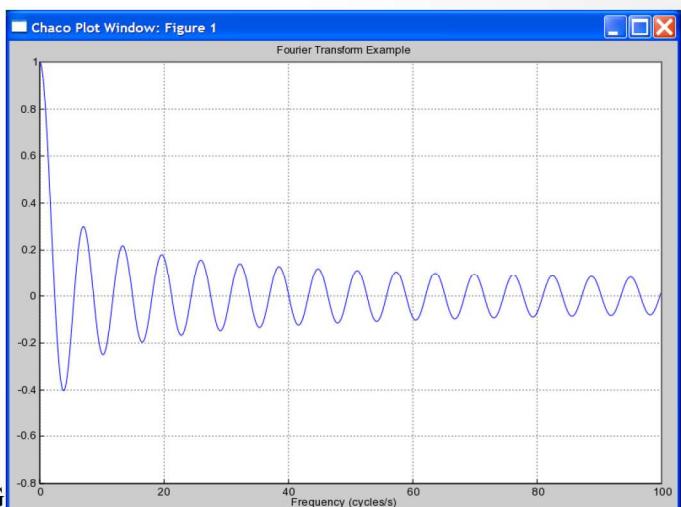
scipy.special

Incluye alrededor de 200 funciones:

Airy, Elliptic, Bessel, Gamma, HyperGeometric, Struve, Error, Orthogonal Polynomials, Parabolic Cylinder, Mathieu, Spheroidal Wave, Kelvin

Función Bessel de primer orden

```
from scipy import special
x = r_[0:100:0.1]
j0x = special.j0(x)
plot(x,j0x)
```

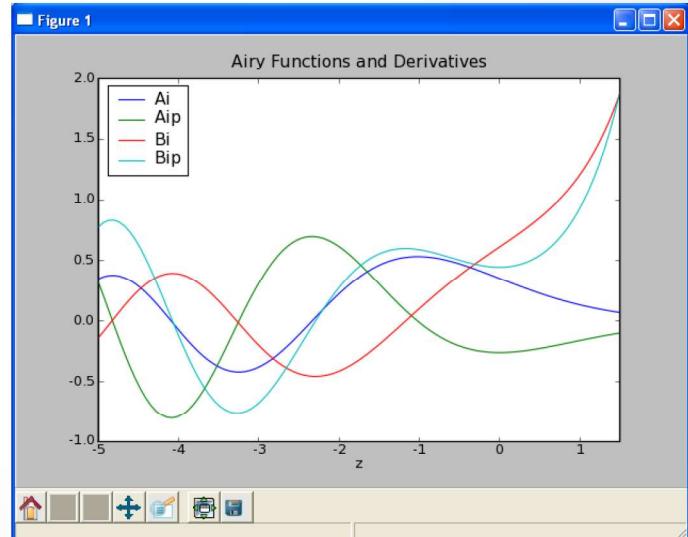


Funciones especiales (II)

scipy.special

Funciones AIRY

```
z = r_[-5:1.5:100j]
vals = special.airy(z)
plot(z,array(vals).T)
legend(['Ai', 'Aip', 'Bi', 'Bip'])
xlabel('z')
title('Airy Functions and Derivatives')
```



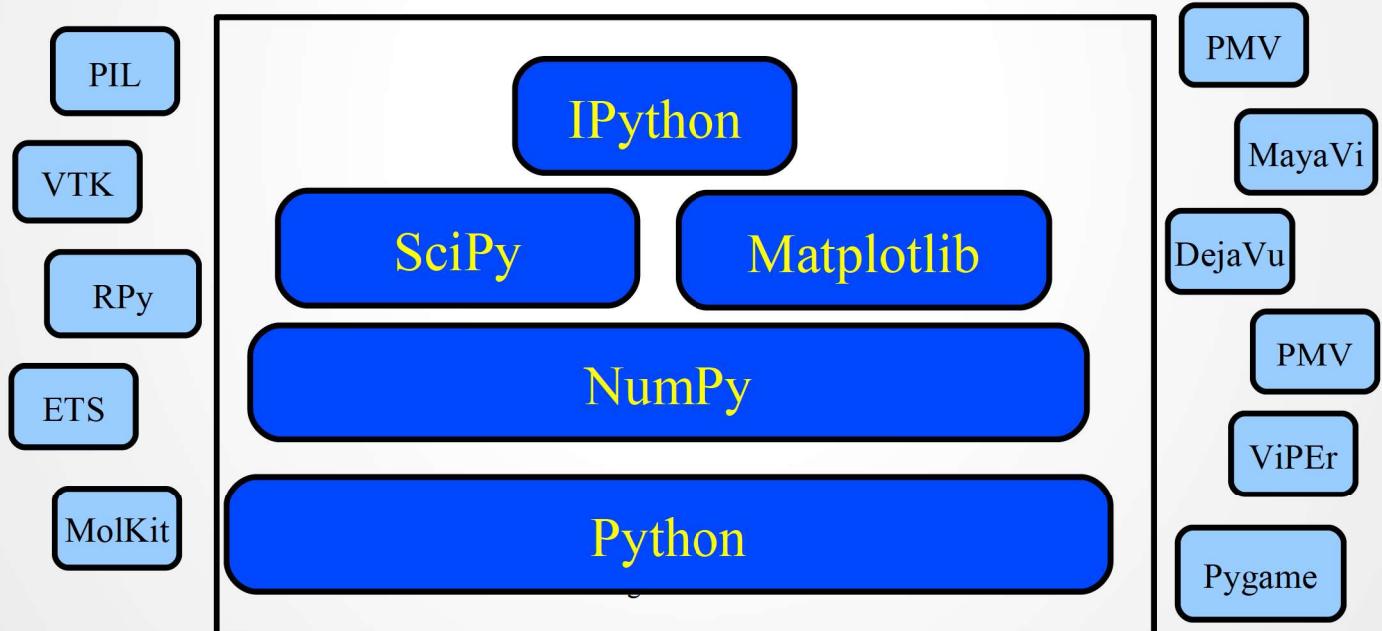
Prof.Miguel García Silvente

113

PyLab

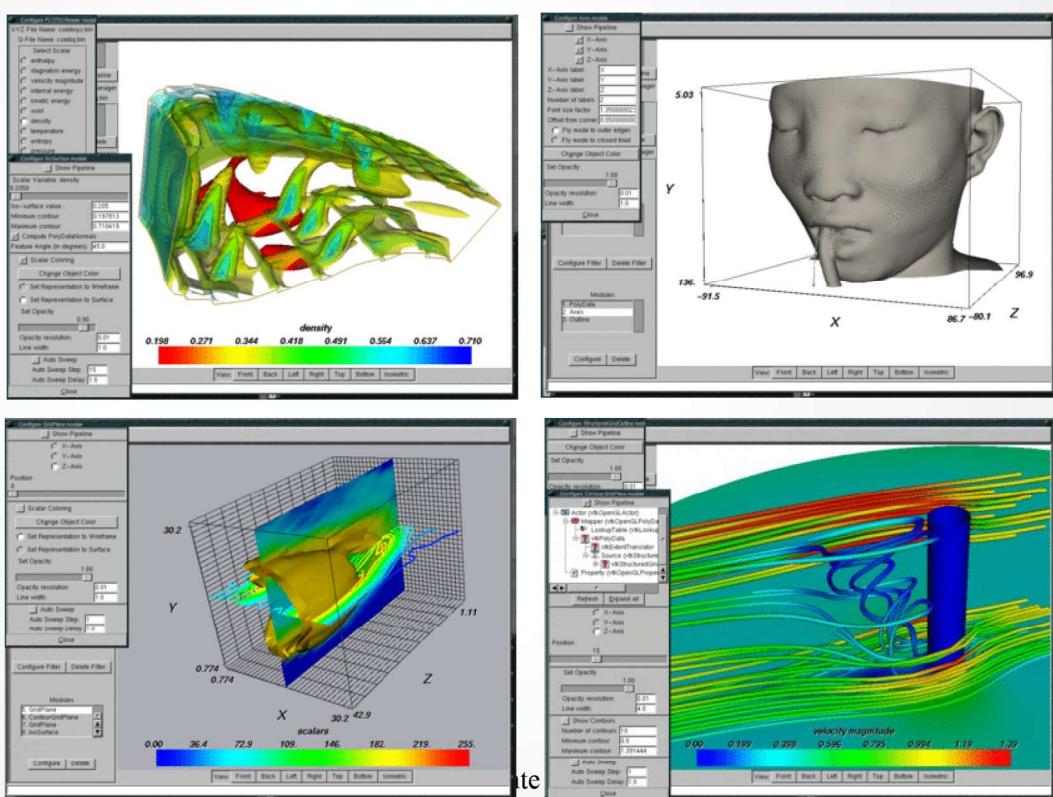
Algunas veces se usa PyLab como una forma de llamar a los cinco paquetes

Más de 1000 módulos/paquetes para Python



Extras: MayaVi

Prabu
Ramachandran



Extras: Enthought Tool Suite

<http://www.enthought.com>

<http://code.enthought.com>

