INTRODUCCIÓN A

PARA CÁLCULO CIENTÍFICO

Luis Rández

Universidad de Zaragoza

6 de marzo de 2011

Instituto Universitario de Investigación
de Matemáticas
y Aplicaciones
Universidad Zaragoza

Instituto Universitario de Investigación
de Matemáticas
y Aplicaciones
Universidad Zaragoza

¿Qué es PYTHON?

PYTHON es un lenguaje de programación de tipo *script* creado por Guido van Rossum a principios de los años 90, cuyo nombre proviene del grupo «Monty Python». El objetivo es un lenguaje con una sintaxis muy limpia y con un código legible.

Figura: Los humoristas británicos *Monty Python*

Características de PYTHON

- interpretado
- tipado dinámico: una misma variable puede tomar valores de distinto tipo en distintos momentos
- multiplataforma: Symbian, Unix, Windows, ...
- conexiones con otros lenguajes (FORTRAN, CUDA, ...)
- manejo de excepciones
- orientado a objetos
- código abierto

f2py: PYTHON & FORTRAN

```
! definición de la función fcn en FORTRAN
real*8 function fcn(x)
real*8 x
fcn = sin(x*cos(x**2))
return
end
```

```
unix$ f2py --fcompiler=gfortran -c -m modulo fcn.f
unix$ python
>>> import modulo
>>> modulo.fcn(5.0)
-0.97046990369101838
```

f2py: PYTHON & FORTRAN

```
 subroutine fun(x,n,m,s) ! pasar vectores
 real*8 x(n), s
 integer i,n,m
!f2py intent(in) n,m,x,s
!f2py intent(out) s
!f2py depend(n) x    ! x depende de n
 print*, n,m
 do i=1,m
   s = s + x(i)
 end do
 return
 end
```

```
unix$ f2py --fcompiler=gfortran -c -m modulo fun.f90
unix$ ipython
In[1]: import modulo
In[2]: t=0
In[3]: modulo.fun(array([1.,2.,3.,4.,5.,6.]),6,5,t)
                6             5
Out[3]: 15.0
```

¿Por qué PYTHON?

- desarrollo rápido de código
- lenguaje de muy alto nivel
- sintaxis clara y sencilla. Mantenimiento fácil
- gran cantidad de librerías
- lenguaje de propósito general

Figura: TiraEcol sobre PYTHON

palabras clave de PYTHON

```
and       as        assert    break
class     continue  def       del
elif      else      except    exec
finally   for       from      global
if        import    in        is
lambda    not       or        pass
print     raise     return    try
while     with      yield
```

Quién usa PYTHON

- Google
- Yahoo
- Industrial Light & Magic
- Walt Disney
- NASA
- SGI, Inc.
- ⋮

Librerías útiles para cálculo científico

PYTHON: Para empezar

- NUMPY Librería: proporciona herramientas para la generación y manipulación de `arrays`.
- SCIPY Librería: optimización, Fourier, cuadratura e integración numérica, . . . (depende de NUMPY)
- SYMPY Librería: cálculo simbólico
- MATPLOTLIB Librería: Gráficos 2D y 3D
- PIL Librería: Python Imaging Library
- IPYTHON Intérprete mejorado de PYTHON


De http://www.sagemath.org

Sage is a free open-source mathematics software system licensed under the GPL. It combines the power of many existing open-source packages into a common PYTHON-based interface.

Mission: Creating a viable free open source alternative to Magma, Maple, Mathematica and Matlab.

Cónsola

Intérprete básico de comandos

```
ubuntu-laptop:> python
Python 2.6.5 (r265:79063, Apr 16 2010, 13:57:41)
[GCC 4.4.3] on linux2
Type "help", "copyright", "credits" or "license" ...
>>>
```

Intérprete mejorado de comandos

```
ubuntu-laptop:> ipython
Python 2.6.5 (r265:79063, Apr 16 2010, 13:57:41)
Type "copyright", "credits" or "license"  ...

IPython 0.10 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object'. ?object also works, ...
In [1]:
```

Módulos

Los módulos son programas que amplian las funciones y clases de PYTHON para realizar tareas específicas. Por ejemplo, el módulo `os` permite usar muchas funciones del sistema operativo.

Los módulos se pueden cargar de la forma `from módulo import *` o bien con `import módulo` o `import módulo as modu`. En estos últimos casos, una `función` de `módulo` hay que llamarla `modulo.función` o `modu.función` respectivamente, mientras que en el primero basta con `función`.

Módulos

```
>>> import os # Modulo que provee funciones del s.o.
>>> help (os) # lista todas las funciones (q para salir)
>>> os.listdir("/home/randez") # directorio
['.cache', '.wine', '.kde', '.thunderbird', 'tmp', ...
>>> print os.getcwd() # directorio actual
/home/randez
>>> os.mkdir("ejemplo") # Crea un directorio
#  Modulo para trabajar con fechas/horas
>>> import time  # fecha y/o hora actual con formato
>>> time.strftime("%Y-%m-%d %H:%M:%S")
'2011-01-09 20:19:23'
```

Tipos numéricos y lógico

entero	coma flotante	complejo	lógico
`int` y `long`	`float`	`complex`	`bool`

```
>>> i=2 #  tipo entero
>>> i=pow(2,130)
>>> i=2**130    # L indica long, precision arbitraria
1361129467683753853853498429727072845824L
>>> float(_) # _ es el equivalente a Ans en matlab
1.3611294676837539e+39
>>> a=2.1 # tipo coma flotante
>>> int(a)
2
>>> long(a)
2L
>>> z=1.5 + 3j #  z=complex(1.5,3)  tipo complejo
>>> z=1.5 + 3*1j
(1.5+3j)
```

Tipos numéricos y lógico

```
>>> z.real, z.imag
(1.5, 3.0)
>>> z.conjugate()
(1.5-3j)
>>> abs(z)
3.3541019662496847
>>> z=(3>4) # tipo logico
>>> print z
False
>>> type(z)
<type 'bool'>
```

El tipo de una variable puede cambiar varias veces en el mismo programa.

Operaciones aritméticas $+, -, *, /, **, \%$

```
>>> 3/2 #  ojo! division entre enteros (fortran?)
1
# para que no ocurra lo anterior
>>> from __future__ import division
>>> 3/2 #  ahora funciona
1.5
>>> 11%7 # resto de la division
4
>>> q, r=divmod(11,7) # cociente y resto
>>> print q, r
1 4
>>>  ␣1+2    # CUIDADO CON EL SANGRADO EN PYTHON
  File "<stdin>", line 1
    1+2
    ^
IndentationError: unexpected indent
```

Más operaciones aritméticas

Además existen en PYTHON notación compacta para modificar el valor de una variable.

`+=`	`c += a`	`c = c+a`	`suma`
`-=`	`c -= a`	`c = c-a`	`resta`
`*=`	`c *= a`	`c = c*a`	`producto`
`/=`	`c /= a`	`c = c/a`	`cociente`
`%=`	`c %= a`	`c = c%a`	`resto del cociente c/a`
`**=`	`c **= a`	`c = c**a`	`potencia`
`//=`	`c //= a`	`c = c//a`	`cociente entero`

Operaciones lógicas y comparaciones

Operaciones lógicas

x or y	x and y	not x

Comparaciones

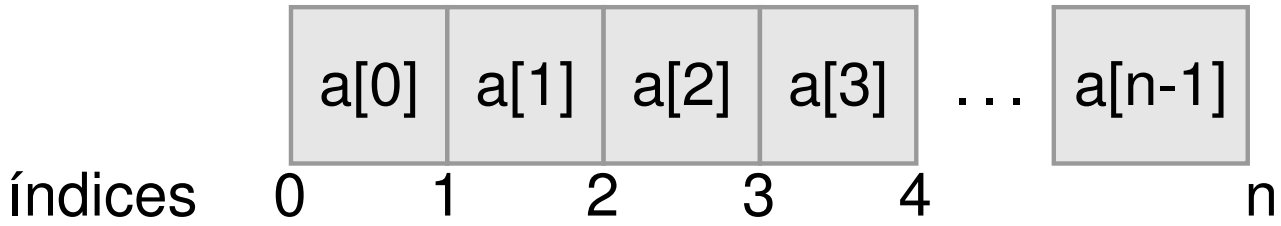
igualdad	menor o igual	mayor o igual	distinto
==	<=	>=	!=

```
>>> (3<4) or (1>0)
True
>>> not (3<4)
False
>>> 4 == 9
False
>>> 4 != 9
True
```

Colecciones de tipos. Listas

Una lista es una colección de objetos que pueden ser de varios tipos.

```
>>> lista=[1,2,2<1,1+3*1j] # se define entre corchetes
>>> type(lista)
<type 'list'>
# CUIDADO programadores fortran/matlab
>>> lista[0] # ojo! los indices empiezan en cero
1
>>> lista[-1]  # ultimo elemento
(1+3j)
>>> lista[1:3] # notacion : igual que en matlab?
[2, False]
>>> lista[1:]
[2, False, (1+3j)]
>>> lista[::2]
[1, False]
```



`a[i:j]` contiene los elementos entre los índices `i` y `j` y consta de `(j-i)` elementos.

Propiedades

- El índice de una lista empieza en **0** como en `C`.
- Las listas son *mutables* [a].
- Los elementos de una lista pueden ser de tipos distintos.
- Se pueden añadir y quitar elementos con `append` y `pop` respectivamente.
- Se pueden concatenar listas con + y `*`.

[a] Los objetos inmutables no pueden ser cambiados tras su definición. Todos los tipos de números son inmutables.

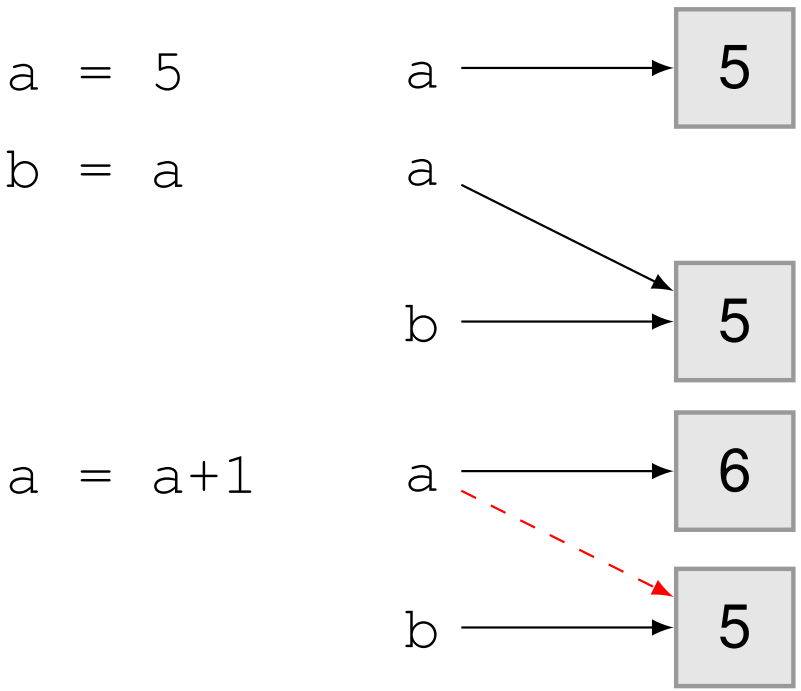
```
>>> lista=[1,2,3,4,False]
>>> lista.pop(4)  # quita el elemento list[4]
False
>>> print lista
[1, 2, 3, 4]
>>> lista.append(True) # poner al final de la lista
>>> print lista
[1, 2, 3, 4, True]
>>> lista+lista[::-1]
[1, 2, 3, 4, True, True, 4, 3, 2, 1]
>>> lista*2
[1, 2, 3, 4, True, 1, 2, 3, 4, True]
>>> lista.sort()
>>> lista   # True == 1   y   False == 0
[1, True, 2, 3, 4]
>>> lista = [ [1,2,3],[[[1j,2]],'a']] # anidar listas
>>> print lista
[[1, 2, 3], [[[1j, 2]], 'a']]
```

Algunas funciones sobre listas

`len(L)`	Dimensión de la lista `L`
`L.append(x)`	Añade el elemento `x` al final de `L`
`L2.extend(L1)`	Añade la lista `L1` al final de la lista `L2`
`L.insert(i,x)`	Inserta el elemento `x` en la posición dada de `L`
`L.remove(x)`	Borra la primera aparición del elemento `x` en `L`
`L.pop([i])`	Borra el elemento en la posición dada de la lista `L`
`L.index(x)`	Devuelve el índice en la lista de la primera aparición del elemento `x`, y si no está se produce un error.
`L.count(x)`	Número de veces que aparece `x` en `L`
`L.sort()`	Ordenar la lista `L`
`L.reverse()`	Devuelve la lista `L` al revés.

Variables mutables/inmutables

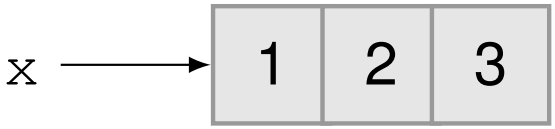
Al ejecutar las sentencias siguientes, las variables `a` y `b` apuntan hacia la misma dirección de memoria. Al redefinir `a=a+1` hace que PYTHON redirija el nuevo valor de `a` hacia otra dirección de memoria donde está el resultado `a+1`.



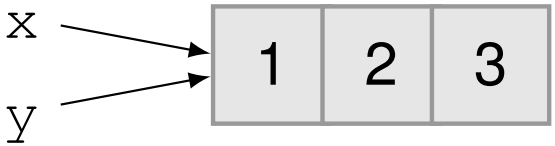
Cuando se modifica una variable inmutable, `Python` genera otro valor dejando el original intacto. Cambiando `a` dejará el valor de `b` intacto ya que direccionan a distintos valores. Si ejecutamos `del a` el valor de `b` sigue siendo `5`.

En el caso de una lista, que es mutable, lo que ocurre es:

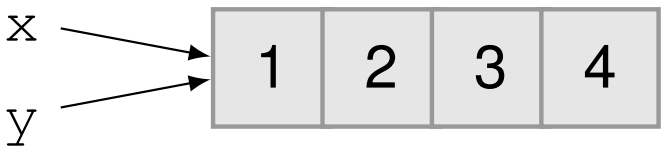
```
x =[1,2,3]
```




```
y=x
```




```
x.append(4)
```



Copiar o no copiar.

Para que un vector sea una **copia** totalmente independiente de otro deberemos utilizar el módulo `copy`.

```
>>> import copy
>>> a=[1,2,3,4]
>>> b=a
>>> c=copy.copy(a)
>>> print id(a), id(b), id(c)
140597721749408 140597721749408 140597721874512
>>> a.append(5)
>>> print a, b, c
[1, 2, 3, 4, 5] [1, 2, 3, 4, 5] [1, 2, 3, 4]
```

Variables de tipo caracter (*String*)

Este tipo de variables se puede definir con varios tipos de comillas. Pueden considerarse como listas y son *inmutables*. Entre comillas, el caracter de línea nueva es `\n` y el tabulador `\t`.

```
>>> s='esto es una \tcadena'  # tabulador
>>> print s
esto es una     cadena
>>> s="Nos vemos en pub's"
>>> s="""tambien vale la triple comilla"""
>>> print s[0:10]
tambien va
>>> print s[::2]
tminvl atil oil
>>> s[1]='0'  # es inmutable (este es no modificable)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item
assignment
```

Variables de tipo caracter (*String*)

Un elemento individual de una variable *string* no se puede cambiar ya que son inmutables, pero es posible modificar la variable entera. Por ejemplo, `replace(x,y[,n])`, sustituye el caracter `x` por `y` `n` veces. Si se omite `n`, se sustituyen todas.

```
>>> s="Esta es una cadena inmutable"
>>> s.replace('a','.',2)
'Est. es un. cadena inmutable'
>>> s
'Esta es una cadena inmutable'
>>> s=s.replace('a','.') # asignar de nuevo s
'Est. es un. c.den. inmut.ble'
```

Para formatear la salida de datos, se hace una sustitución de la forma:

```
>>> fmt='El numero premiado con %f %s es %i'
>>> fmt % (123.45,'libras',23456)
'El numero premiado con 123.450000 libras es 23456'
```

Algunas funciones sobre (*Strings*)

`len(s)`	Número de caracteres de la variable `s`
`s.count(x)`	Devuelve el número de veces que aparece `x` en `s`
`s.find(x)`	Devuelve la posición de `x` en `s`. Si no está, el resultado es -1
`s.lower()`	Transforma `s` a minúsculas.
`s.upper()`	Transforma `s` a mayúsculas.
`s.replace(x,y[,n])`	Reemplaza `x` por `y` un determinado número de veces.
`s.split(x)`	Divide la variable `s` en una lista utilizando como separador `x`.
`x.join(L)`	Combina los elementos de una lista `L` de *strings* con `x`
`str(a)`	Conversión a tipo *string* de `a`

```
>>> s="Esta es una cadena inmutable"
>>> len(s)
28
>>> s.count('a')
5
>>> s.find('una')
8
>>> s.find('muy')
-1
>>> s.upper()
'ESTA ES UNA CADENA INMUTABLE'
>>> L=s.split(' ')
['Esta', 'es', 'una', 'cadena', 'inmutable']
>>> '.'.join(L)
'Esta.es.una.cadena.inmutable'
>>> s=str(355./113)
>>> s
'3.14159292035'
```

```
>>> from math import *
>>> s='sin(8.0)'
>>> type(s)
<type 'str'>
>>> eval(t)
0.98935824662338179
>>> s='[1,2,3,4,5]'
>>> v=eval(s)
>>> type(v)
<type 'list'>
>>> i=raw_input('Dame un valor ') # entrada de datos
Dame un valor 12.0
>>> type (i)
<type 'str'>
# Puede usarse i=eval(raw_input(' ... '))
>>> i=input('Dame un valor ')
Dame un valor 12.0
>>> type (i)
<type 'float'>
```

¿Todas las comillas son iguales?

```
>>> from math import *
>>> s='sin(8.0)'
>>> t="sin(8.0)"
>>> t, s
('sin(8.0)', 'sin(8.0)')
>>> eval(t), eval(s)
(0.98935824662338179, 0.98935824662338179)
# las comillas invertidas evaluan la
# expresion entre ellas
>>> x=2.0
>>> r1=`sin(cos(x**2/(1+x)))`
>>> r2='sin(cos(x**2/(1+x)))'
>>> r1, r2
('0.23307402238356428', 'sin(cos(x**2/(1+x)))')
```

Tuplas

Podemos pensar en las tuplas como listas inmutables. Se definen usualmente entre parentésis y se separan por comas.

```
>>> t=1,2,'carlos','jose'     # sin parentesis
>>> type(t)
<type 'tuple'>
>>> t=(1,2,'carlos','jose')   # con parentesis
>>> len(t)
4
>>> t[0]='jorge'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item
assignment
>>> t.count('jose')
1
```

Diccionario (*hash*, vector asociativo)

Un diccionario es un tipo de dato formado por una colección de pares claves/valor, de forma que permite acceder al valor por medio de una clave.

```
>>> h={'juan':98, 'jose':23, 123:84}  # entre llaves
>>> h['javi']='xx'
>>> h
{'jose': 23, 'juan': 98, 123: 84, 'javi': 'xx'}
>>> print h['jose'], h[123], h['javi']
23 84 xx
>>> h.keys()
['jose', 'juan', 123, 'javi']
>>> h.values()
[23, 98, 84, 'xx']
>>> 'jose' in h
True
```

Controles de flujo `if/elif/else`

En PYTHON se usa el sangrado para definir bloques de código, bucles `if`, `for`, `while`, funciones, ...

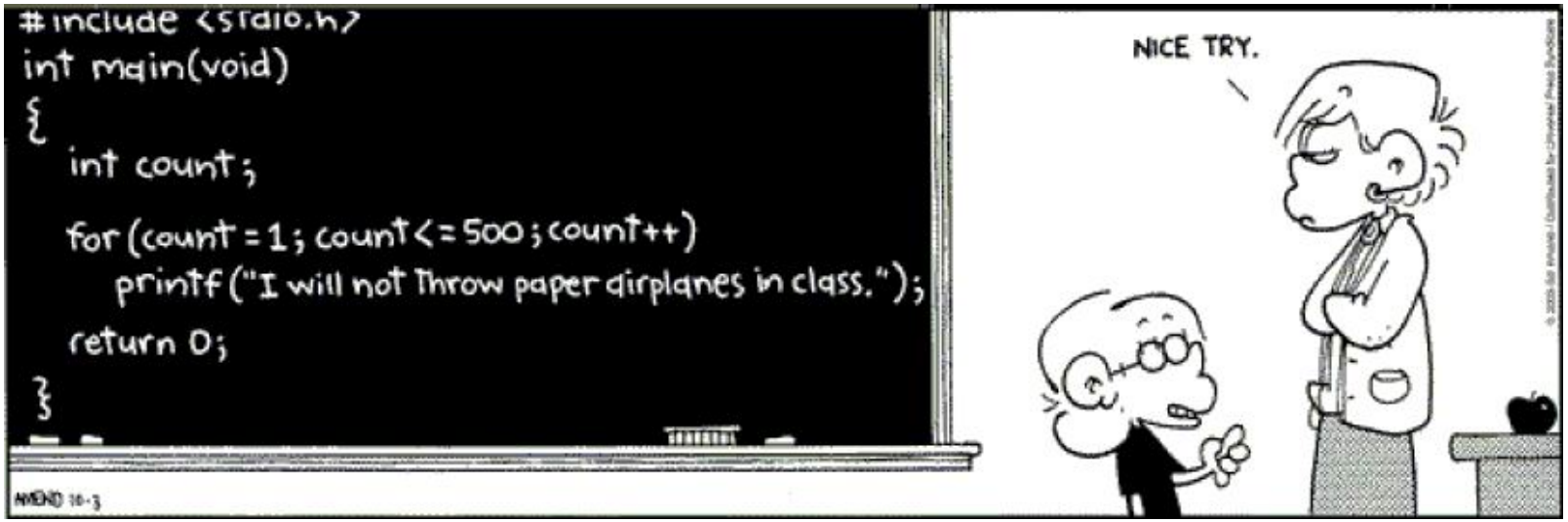
```
if condicion_if:
    bloque_condicion_if
elif condicion_elif:
    bloque_elif
else:
    bloque_else
```

Notas

- El sangrado (4 espacios en blanco) es necesario[a]
- No hay `end` como en `matlab`

[a] Ipython no es tan ortodoxo

Control de flujo `for`



Control de flujo `for`

```
for contador:
    bloque_for
```

```
for i in range(5):    # 0,1,2,3,4
    print(i**2),  # la , escribe en la misma linea
0 1 4 9 16
```

```
for color in ('rojo','ocre','azul'):
    print('el color es ' % color)
el color es rojo
el color es ocre
el color es azul
```

```
for i in 'australopithecus':
    print i,
a u s t r a l o p i t h e c u s
```

```
frase='Escribir cada palabra'
for palabra in frase.split():
    print palabra  # salta linea
Escribir
cada
palabra
```

```
# enumerate(seq) donde seq es una lista o tupla
# da los pares (i, seq[i])
for i, col in enumerate(['verde', 'azul', 'rojo']):
    print i, col
0 verde
1 azul
2 rojo
```

También se puede iterar en un diccionario de una forma similar a la dada por `enumerate` con `iteritems`, que proporciona el par clave/valor de un diccionario.

```
hash={'a':9, 'b':2, 'c':3.0, 'd':2j}
for clave,valor in hash.iteritems():
    print ('clave: %s, valor: %s' % (clave, valor))
clave: a, valor: 9
clave: c, valor: 3.0
clave: b, valor: 2
clave: d, valor: 2j
```

Controles de flujo `while/break/continue`

Tienen el mismo significado que en `matlab`

```
while condicion:
    bloque_while
```

```
# Generacion de la sucesion de Fibonacci
a,b=0,1     # asignacion multiple
while a<90:
    a,b=b,a+b
    if b>10 and b<40:
        continue # continua con la iteracion siguiente
    print b,
1 2 3 5 8 55 89 144 233
```

Listas por comprensión

Una lista por comprensión es una construcción que permite generar una lista basada en listas existentes. La forma de definirla es muy similar a la definición de un conjunto en matemáticas. Por ejemplo,

$$S = \{2n \,|\, n \in \mathbf{N}, n^2 <= 121\}$$

```
>>> S = [2*n for n in range(101) if n**2 <=121 ]
>>> print S
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22]
```

```
>>> S=[ [x,y] for x in range(2) for y in 'abc']
>>> S
[[0, 'a'], [0, 'b'], [0, 'c'], [1, 'a'],
[1, 'b'], [1, 'c']]
>>> R=range(3)
>>> H=[ 1/(i+j+1.) if i<=j else 0 for i in R for j in R]
>>> H
[1.0, 0.5, 0.3333333, 0, 0.333333, 0.25, 0, 0, 0.20]
```

Funciones

La forma típica para definir una función es:

```
def nombre_funcion(args):
    bloque_funcion
```

Por defecto, las funciones en PYTHON devuelven `None`, por lo que suele emplearse **`return`** `valores`. En principio, las variables en la función tienen su propio espacio, i.e., son locales.

```
>>> def fun(x):    # es necesario pasar x
...     return sin(x*cos(x**5))
...
>>> fun(8.0)    # tambien vale fun(x=8.0)
0.15743233177672805
>>> fun()          # da error
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: fun() takes exactly 1 argument (0 given)
```

```
>>> valor_pi=pi
>>> def fun(x=valor_pi):  # parametro opcional.
...     """ Asi se ponen los comentarios en la funcion\
    fun.__doc__ los mostrara """
...     return sin(x*cos(x**5))
...
>>> fun(8.0)
0.15743233177672805
>>> fun()
-0.7739705540130050 7
>>> fun(pi)
-0.7739705540130050 7
>>> fun(x=9*pi)
-0.868365629853879
# el valor por defecto de valor_pi es el que tiene
# cuando se define la funcion fun
>>> valor_pi=0.0
>>> fun()
-0.7739705540130050 7
```

Paso por referencia o por valor

- En el paso por referencia lo que se pasa como argumento es una referencia o puntero a la variable, es decir, la dirección de memoria en la que se encuentra el contenido de la variable, y no el contenido en si.
- En el paso por valor, por el contrario, lo que se pasa como argumento es el valor que contiene la variable.

```
>>> def fun(a,b):
...     a=a + 2.0          # inmutable
...     b.append('1')      # mutable
...     print a, b

>>> a, b = 0.0, ['0']   # En python: los valores mutables
>>> fun(a,b)            # se comportan como paso por
2.0 ['0', '1']          # referencia y los inmutables
>>> print a, b          # como paso por valor.
0.0 ['0', '1']          # --chequea que id(b) es el mismo
```

```
>>> a=8
>>> def fun(b):          # Las variables declaradas fuera de
...     return a+b       # la funcion guardan su valor dentro
...                      # de la funcion
>>> fun(9)
17
```

Si no se pasan como argumentos, para que puedan modificarse fuera del ámbito de la función, emplearemos `global`.

```
>>> a=1
>>> def fun(b):
...     global a   # sin esta linea, a es variable local
...     a = b      # probarlo !!
...     print a, b
...
>>> print a, '--',fun(9), '--', a
1 -- 9 9
None -- 9          # No hay return
```

```
>>> def fun(p1,*resto):    # Lista argumentos variable
...     for par in resto:
...         print par,
...     print '\nel tipo de resto es: ', type(resto)
...
>>> fun(1,2,3,4,'o')   # llamada como tupla
2 3 4 o
el tipo de resto es:  <type 'tuple'>
```

```
>>> def fun(p1,**hash):    # Lista argumentos variable
...     claves = hash.keys()
...     claves.sort()
...     for key in claves:
...         print key, hash[key],
...     print '\nel tipo de hash es: ', type(hash)
...
>>> fun('o',x=1,y=2,z=3)   # llamada como diccionario
x 1 y 2 z 3
el tipo de hash es:  <type 'dict'>
```

Funciones lambda

El operador lambda vale para definir funciones anónimas en línea. El que sean anónimas, «sin nombre», implica que no pueden ser referenciadas posteriormente. Son funciones *ad hoc* que pueden ser escritas de forma simple en una línea.

```
>>> g = lambda x: x**2   # formas equivalentes
>>> def fun(x):          # para definir una
...     return x**2      # funcion sencilla
...
>>> def potencia(n):     # composicion de funciones
...     return lambda x: x**n
...
>>> f=potencia(2)
>>> print f(1), f(2), f(3)
1 4 9
>>> f=potencia(3)
>>> print f(1), f(2), f(3)
1 8 27
```

Funciones lambda

```
>>> lista = [1, 4, 9, 16, 25, 36, 49]
>>> print filter(lambda x: x%3 == 0, lista)
[9, 36]    # multiplos de 3
>>> print filter(lambda x: x%3 == 0 or x%4 == 0, lista)
[4, 9, 16, 36]  # multiplos de 3 o de 4
>>> print map(lambda x: sqrt(x) if x<=25 else -1, lista)
[1.0, 2.0, 3.0, 4.0, 5.0, -1, -1]
#
# un ejemplo algo mas complicado
#
>>> frase ="La puerta es muy alta y grande"
>>> palabras = frase.split()
>>> carac = map(lambda palabra: len(palabra), palabras)
>>> print carac, sum(carac)
[2, 6, 2, 3, 4, 1, 6] 24
```

Excepciones

Las excepciones de manejan en la forma **try-except**

```
>>> def f(x):
...     try:
...         return 1/x
...     except:
...         print "la cosa esta muy mala"
...     else:           # Ha ido todo bien
...         pass        # no hace nada
...
>>> f(8.0)
0.125
>>> print f(0.0)
la cosa esta muy mala
None
```

Ficheros

En PYTHON los ficheros se abren con `open(fichero, modo)`. El primer parámetro es el nombre del fichero y el segundo una variable *string* conteniendo caracteres que describen la forma de abrir el fichero, `r` sólo lectura, `r+` lectura y escritura, `w` sólo escritura, `a` para añadir registros nuevos al final del fichero, etc.

```
>>> f = open("file.txt") # por defecto solo lee, 'r'
>>> f
<open file 'file.txt', mode 'r' at 0x2238660>
>>> f.close()              # cerrar el fichero
>>> f
<closed file 'file.txt', mode 'r' at 0x2238660>
```

Para la lectura del fichero, podemos emplear:

`f.read()`	Devuelve una variable *string* conteniendo el fichero completo. Las líneas están separadas por `\n`. Si se ha alcanzado el final `f.read()` devuelve la *string* vacía.
`f.read(n)`	Devuelve una variable *string* conteniendo `n` bytes del fichero.
`f.readline()`	Devuelve una variable *string* conteniendo una línea del fichero. Cuando el final se ha alcanzado `f.readline()` devuelve la *string* vacía.
`f.readlines()`	Devuelve una lista conteniendo todas las líneas del fichero.

```
>>> f.read()
'primera linea\nsegunda linea\ntercera linea'
>>> f.read()
''
```

```
>>> f.read(10)
'primera li'
>>> f.read()
'nea\nsegunda linea\ntercera linea'
```

```
>>> f.readline()
'primera linea\n'
>>> f.readline()
'segunda linea\n'
>>> f.read()
'tercera linea'
>>> f.read()
''
```

```
>>> f.readlines()
['primera linea\n', 'segunda linea\n', 'tercera linea']
```

Para escribir en un fichero, usamos `fwrite(cadena)`, y si queremos guardar números, antes hay que transformarlos a variables *string*.

```
>>> f=open("file1.txt","w")
>>> linea='una aprox de pi es ' + str(355/113.)+"\n"
>>> f.write( str(linea))
>>> linea=r'segunda linea \n'   # raw string \n
>>> f.write(linea)
>>> linea='tercera linea \n'
>>> f.write(linea)
>>> f.close()
```

El contenido del fichero `file1.txt` es:

```
una aprox de pi es 3.14159292035
segunda linea \ntercera linea
```

Con `f.tell()` sabemos en que posición del fichero estamos y con `f.seek()` nos desplazamos por él, para leer o escribir en una determinada posición.

Lo típico cuando entramos en una sesión de IPYTHON es cargar todas las librerías numéricas y gráficas que podríamos usar en la sesión.

```
In [1]: from scipy import *   # scipy carga numpy
In [2]: from numpy import *   # redundante con scipy
In [3]: from pylab import *
In [4]: help (pylab)   # informacion sobre pylab
In [5]: help (numpy)   # informacion sobre numpy
In [6]: dir (numpy)    # todas las funciones de numpy
In [7]: numpy.info('topico')    # info de topico numpy
In [7]: numpy.info('fft')       # info de fft
```

Las librerias pueden cargarse de la forma `from numpy import *` o bien con `import numpy` o `import numpy as np`. En estos últimos casos, una función de NUMPY hay que llamarla `numpy.funcion` o `np.funcion` respectivamente.

Numpy

En NUMPY el tipo fundamental es el `array`, que básicamente es una lista con un sólo tipo.

```
In [1]: import numpy as np
In [2]: a=np.array([0,1,3.0])
In [3]: print a, a[0], a[-1]
[0. 1. 3.] 0.0 3.0
In [4]: type(a)
Out[4]: <type 'numpy.ndarray'>
In [5]: b=np.array([[1,2],[2,3.0]])
In [6]: print b
[[ 1.  2.]
 [ 2.  3.]]
In [7]: c=np.eye(2) - np.ones(2)
In [8]: print c
array([[ 0., -1.],
       [-1.,  0.]])
```

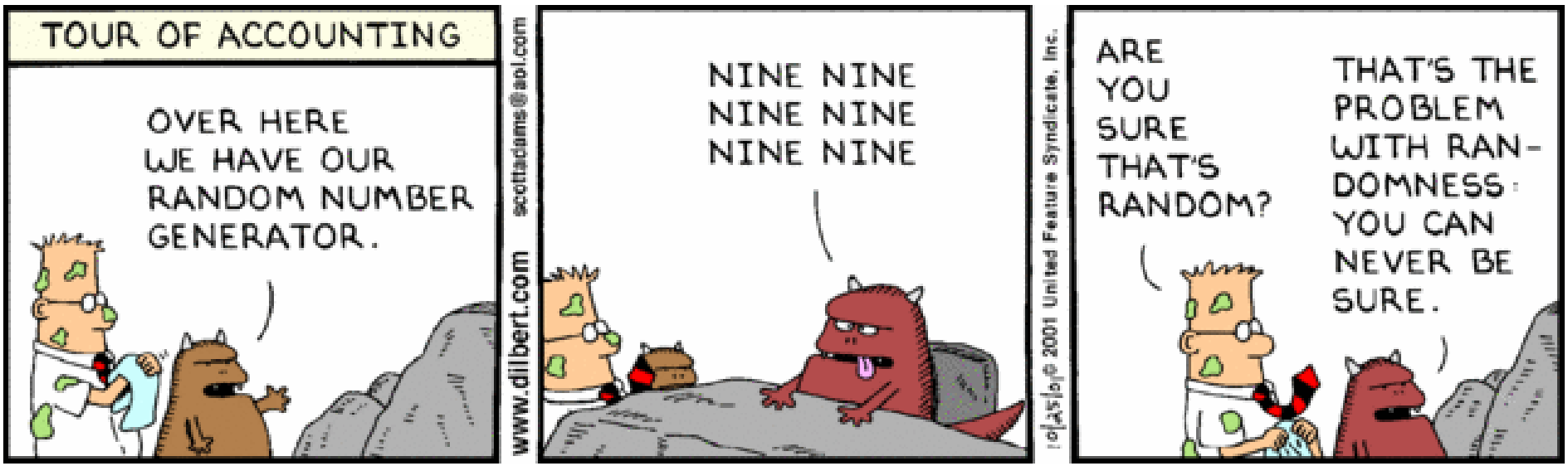
```
In [1]: from numpy import *
In [2]: finfo(float).eps  # eps en matlab
Out[2]: 2.2204460492503131e-16
In [3]: a=arange(8).reshape(2,4)

In [3]: print a
array([[0, 1, 2, 3],
       [4, 5, 6, 7]])
In [4]: a.shape
Out[4]: (2, 4)

In [5]: a.ndim
Out[5]: 2

In [6]: a.dtype
Out[6]: dtype('int64')

In [7]: type(a)
Out[7]: <type 'numpy.ndarray'>
```

```
In [1]: b = array( [ [1.5,2,3], [4,5,6] ] )
In [2]: print b
array([[ 1.5,  2. ,  3. ],
       [ 4. ,  5. ,  6. ]])
In [3]: b.dtype
Out[3]: dtype('float64')
In [4]: c = array( [ [1,2], [3,4]], dtype=complex )
In [5]: print c
array([[ 1.+0.j,  2.+0.j],
       [ 3.+0.j,  4.+0.j]])
In [6]: linspace( 0, 2, 5 )
Out[6]: array([ 0. ,  0.5,  1. ,  1.5,  2. ])
In [7]: A = array( [[1,1], [2.,3]])
In [8]: B = array( [[1,-1], [2.,-4]])
In [9]: print dot(A,B) # producto matricial
array([[  3.,  -5.],
       [  8., -14.]])
In [10]: print A*B     # producto elemento a elemento
array([[  1.,  -1.],
       [  4., -12.]])
```

Números aleatorios

Para poder utilizar números pseudo-aleatorios hay que importar la librería correspondiente

```
# en Ipython no es necesaria la linea siguiente
In [1]: from numpy.random import *
In [2]: a=random((3,4))   # es preciso cargar random
```



```
In [3]: print a
[[0.35692182,0.33172828,0.57910313,0.33401589],
 [0.36782743,0.51413341,0.54869768,0.51155853],
 [0.19688295,0.7342029 ,0.20734874,0.85260151]])
In [4]: print a.sum() # suma total
5.5350222702225951
In [5]: print a.min(), a.max()  # minimo y maximo total
0.19688294684109708, 0.85260150761680042
In [6]: a.max(axis=0) # maximo de cada columna
([0.36782743,0.7342029,0.57910313,0.85260151])
In [7]: a.max(axis=1) # maximo de cada fila
Out[8]: array([ 0.57910313,  0.54869768,  0.85260151])

In [9]: a=pi*ones((3,3), dtype=float)
In [10]: print a
[[ 3.14159265,  3.14159265,  3.14159265],
 [ 3.14159265,  3.14159265,  3.14159265],
 [ 3.14159265,  3.14159265,  3.14159265]])
```

```
In [10]: b=ones((3,3))
In [11]: b+=a    # equivalente a b = b+a
In [12]: print b
[[ 4.14159265,  4.14159265,  4.14159265],
 [ 4.14159265,  4.14159265,  4.14159265],
 [ 4.14159265,  4.14159265,  4.14159265]])
In [13]: c=linspace(0,2*pi,10)
In [14]: print c
[ 0.        ,0.6981317,1.3962634 ,2.0943951 ,2.7925268,
  3.4906585,4.1887902,4.88692191,5.58505361,6.28318531]
In [29]: exp(c*1j)
Out[29]:
array([
 1.00000000+0.00000000e+00j, 0.76604444+6.42787610e-01j,
 0.17364818+9.84807753e-01j,-0.50000000+8.66025404e-01j,
-0.93969262+3.42020143e-01j,-0.93969262-3.42020143e-01j,
-0.50000000-8.66025404e-01j, 0.17364818-9.84807753e-01j,
 0.76604444-6.42787610e-01j, 1.00000000-2.44929360e-16j])
```

```
In [53]: a=arange(5)**2
In [54]: a
Out[54]: array([ 0,  1,  4,  9, 16])
In [55]: a[2:4]
Out[55]: array([4, 9])
In [56]: a[2:4]=-1
In [57]: a
Out[57]: array([ 0,  1, -1, -1, 16])
In [58]: a[::-1]
Out[58]: array([16, -1, -1,  1,  0])
In [59]: a
Out[59]: array([ 0,  1, -1, -1, 16])
In [68]: def f(i,j):
   ....:     return 2*i+j
   ....:
In [69]: b = fromfunction(f,(3,3),dtype=float)
# En una linea como funcion lambda
In [69]: b = fromfunction(lambda x,y: 2*x+y,(3,3))
```

```
In [70]: print b
[[ 0.,  1.,  2.],
 [ 2.,  3.,  4.],
 [ 4.,  5.,  6.]])
In [71]: b[0,0]   # elemento 1,1
Out[71]: 0.0
In [72]: b[1]     # segunda fila
Out[72]: array([ 2.,  3.,  4.])
In [73]: b[-1]    # ultima fila
Out[73]: array([ 4.,  5.,  6.])
In [74]: b[:,1]   # segunda columna
Out[74]: array([ 1.,  3.,  5.])
In [75]: b[:,-1] # ultima columna
Out[75]: array([ 2.,  4.,  6.])
In [76]: b[:,1]+b[1,:] # suma de fila y columna !!!
Out[76]: array([ 3.,  6.,  9.])
```

Es importante notar que en principio no hay diferencia entre vectores fila y columna.

Notación :

a[0,3:5] a[4:,4:] a[:,1] a[2::2,2::2]

Broadcasting

Anteriormente, hemos visto que es posible hacer operaciones con `arrays` de tamaño/forma diferente si NUMPY puede transformarlos para que sea posible. Esto se llama **broadcasting**.

```
In [1]: a = arange(3)
In [2]: print a
[0 1 2]
In [3]: a.shape = (3,1)
In [4]: print a
[[0]
 [1]
 [2]]
In [5]: b=array([1,1,1])
In [6]: print a+b
[[1 1 1]
 [2 2 2]
 [3 3 3]]
```

```
In [1]: a=array([[0],[1],[2]]); b=array([1,1,1]);
In [2]: c=array([[1,2,3],[4,5,6],[7,8,9]])
In [3]: print a+b  # vector columna + fila
[[1 1 1]
 [2 2 2]
 [3 3 3]]
In [4]:  print a+c # vector columna + matriz
[[ 1  2  3]
 [ 5  6  7]
 [ 9 10 11]]
In [5]: print b+c # vector fila + matriz
[[ 2  3  4]
 [ 5  6  7]
 [ 8  9 10]]
In [6]: print dot(c,a) # matriz * vector columna
[[ 8]
 [17]
 [26]]
In [7]: print dot(c,b) # matriz * vector fila
[ 6 15 24]
```

asignación y `copy()`

```
In [1]: from numpy import *
In [2]: a=arange(4)
In [3]: a
Out[3]: array([0, 1, 2, 3])
In [4]: b=a
In [5]: c=a.copy()
In [6]: a[1:3]=999
In [7]: print a, b, c
[  0 999 999 3] [  0 999 999 3] [0 1 2 3]
```

Fancy indexing

```
In [1]: a=arange(5)**2   # numpy.ndarray
In [2]: print a
[ 0 1 4 9 16]
In [3]: i=[1,1,2,2] # vector de indices
In [3]: a[i]
Out[4]: array([1, 1, 4, 4])
In [5]: a=array([[00,01,02],[10,11,12],[20,21,22]])
In [6]: i=array([[0,1],[1,2]])
In [7]: j=array([[2,1],[1,0]])
In [8]: a[i,j]
array([[02, 11],
       [11, 20]])
```

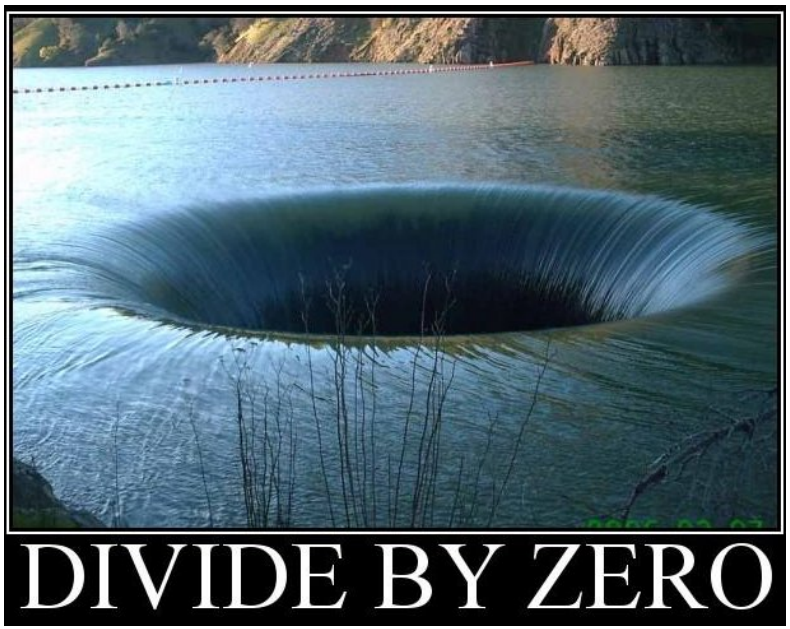
```
In [9]: y=linspace(-1,1,5)**3
In[10]: y
array([-1.   , -0.125,  0.   ,  0.125,  1.   ])
In[11]: i= abs(y)<=0.5
In[12]: i
array([False,  True,  True,  True, False], dtype=bool)
In[13]: y[i]=9
In[14]: y
array([-1.,  9.,  9.,  9.,  1.])
In[15]: a=arange(12).reshape(3,4)
In[16]: i1=array([False,True,True])
In[17]: i2=array([True,False,True,False])
In[18]: a[i1]
array([[ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
In[19]: print a[:,i2], a[i1,i2]
[[ 0  2]
 [ 4  6]
 [ 8 10]] [ 4 10]
```

Aritmética

Al igual que MATLAB, PYTHON trabaja en aritmética de coma flotante que usa el estándar IEEE-754.

```
In [2]: a=-inf       # a= -Inf tambien vale
In [3]: print 2**a, inf/inf, inf-inf
(0.0, nan, nan)
In [4]: print sqrt(-1) # from numpy import *
nan
In [5]: 10.0**400 # en matlab el resultado es inf
OverflowError  Traceback (most recent call last)
/home/user/<ipython console> in <module>()
OverflowError: (34, 'Numerical result out of range')

In [6]: x = 10e200; y = x*x; print y
inf
In [7]: from scipy import *
In [8]: print sqrt(-1) # hay que cargar scipy
1j
```

```
In [1]: sin(0)/0
Out[1]: nan
In [2]: x=1.0/0
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
.
.
.

ZeroDivisionError: float division
```



MATLAB	PYTHON
`fichero[.m]`	`run fichero.py` `execfile('fichero.py')`
`quit`	`CTRL + D`
`help tópico`	`help (tópico) info(tópico)`
	`from numpy import *`
`rand(m,n)`	`random.uniform(a,b,(m,n))`
`randn(m,n)`	`random.standard_normal(a,b,(m,n))`
Matriz $m \times n$ de números aleatorios unifor. o normal. distribuidos en $[a, b]$.	
`a=[1,2,3]`	`a=array([1,2,3])`
`a=[1;2;3]`	`a=array([1,2,3]).reshape(-1,1)`
`a(2:end)`	`a[1:]`
Elimina el primer elemento de un vector	
`[v,i] = max(a)`	`v,i = a.max(0),a.argmax(0)`
`[a; b]`	`vstack((a,b))`
`[a, b]`	`hstack((a,b))`
`zeros(m,n)`	`zeros((m,n), float)`
`eye(n)`	`identity(n)` o `eye(n)`
`diag([4, 5, 6])`	`diag((4,5,6))`

MATLAB	PYTHON
`reshape(a,m,n)`	`a.reshape(m,n)`
`a(:)'` `a(:)`	`a.flatten(0)` `a.flatten(1)`
Convierte una matriz a un vector, (0,filas), (1,columnas)	

`a(1,:)`
`a(:,1)`
`a(end-1:end,:)`
	`a[0,]` Primera fila
	`a[:,0]` Primera columna
	`a[-2:,]` Últimas dos filas

`a(a>0.1) = 9;`	`a[a>0.1]=9`
`a'`	`a.conj().transpose()` o `a.conj().T`
`a.'` o `transpose(a)`	`a.transpose()` o `a.T`
`det(a)`	`linalg.det(a)` Determinante
`inv(a)`	`linalg.inv(a)` Inversa
`pinv(a)`	`linalg.pinv(a)` Pseudo-inversa
`norm(a)`	`norm(a)` Normas
`svd(a)`	`linalg.svd(a)` Valores singulares
`chol(a)`	`linalg.cholesky(a)` factorización

MATLAB	PYTHON
`[v,V] = eig(a)`	`v,V=linalg.eig(a)` Valores, vectores propios
`[l,u,p] = lu(a)`	`from scipy import linalg` `p,l,u=linalg.lu(a)` Factorización LU
`eps`	`finfo(float).eps`
`1i, 1j`	`1j` Unidad imaginaria
`a * b`	`dot(a,b)`
`a .* b`	`a * b`
`a.^3`	`a**3`
`[Q,R,P]=qr(a,0)`	`Q,R = linalg.qr(a)` Factorización QR
`b = a` `b=a(2,:)`	`b = a.copy()` Copia de un vector `b = a[1,:].copy()`
`a .* (a>0.5)`	`a * (a>0.5)`
`atan2(x,y)`	`from math import *` `atan2(x,y)`
`kron(a,b)`	`kron(a,b)`

`sum(a)`
`sum(a')`
`sum(a(:))`
	`a.sum(axis=0)` suma cada columna
	`sum(a,axis=1)` suma cada fila
	`a.sum()` o `sum(a)` suma total

MATLAB	PYTHON
`abs(z)`	`abs(z)` valor absoluto
`real(z)`	`z.real` Real parte real
`imag(z)`	`z.imag` parte imaginaria
`conj(z)`	`z.conj()` complejo conjugado
	`from scipy import linalg`
`expm(a)`	`scipy.linalg.expm(a)` exponencial matricial
`logm(a)`	`scipy.linalg.logm(a)` logaritmo matricial
`size(a)`	`a.shape`
`size(a,2)`	`a.shape[1]`
`length(a)`	`size(a, axis=1)`
`length(a(:))`	`a.size`
`ndims(a)`	`a.ndim`

¿Cómo calculamos la potencia n-ésima de una matriz ?

```
In [1]: from numpy import random
In [2]: a=random.random((4,4))
In [3]: M=mat(a.copy())          # convertir a clase matriz
In [4]: type(M)
Out[4]: <class 'numpy.core.defmatrix.matrix'>
In [5]: b=array(M**8)    # solo potencias enteras !!
In [6]: type(b)
Out[6]: <type 'numpy.ndarray'>
```

Ejercicio: ¿Cómo calcularías M^n, cuando $n \in \mathbb{R}$?

```
# vectores con una dimension nula
In [1]: a=arange(1,-1,11)
In [2]: a.size
0
In [3]: a.shape
Out[3]: (0,)
```

Ficheros. `loadtxt/savetxt` en NUMPY

En `Ipython` es posible utilizar comandos del tipo `cd`, `pwd`, ...

```
In [1]: mkdir directorio
In [2]: cd directorio/
/home/user/directorio
In [3]: savetxt('fiche.txt', random.random((3,2)) )
# guardar en un fichero
In [4]: cat fiche.txt
3.25916017078421882 4.1460603037109744e-02
9.8844718122868713 8.5280813900026830e-01
8.3488971348487995 5.6735287069147622e-01
In [5]: ls
fiche.txt
In [6]: b=loadtxt('fiche.txt') # cargar un fichero en un ar
In [7]: b
array([[ 0.32591602,  0.0414606 ],
       [ 0.98844718,  0.85280814],
       [ 0.83488971,  0.56735287]])
```

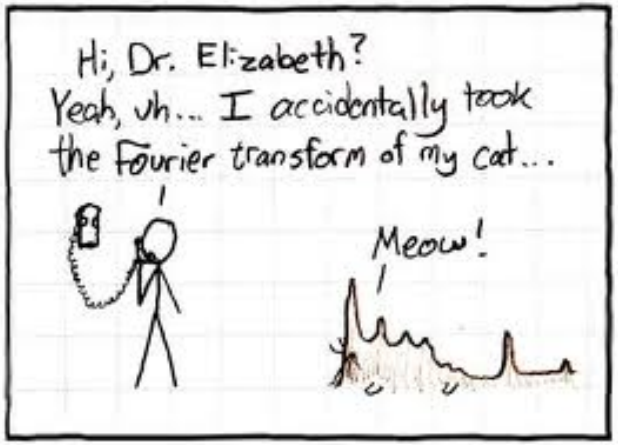
El fichero `fiche.txt` contiene los datos:

```
Esta linea es un comentario
3.25916017e-01  4.14606030e-02   as
9.88447181e-01  8.52808139e-01   as
8.34889713e-01  5.67352870e-01  sdd
```

```
In [1]: b=loadtxt('fiche.txt',skiprows=1,usecols=(0,1),\
        unpack=True)
In [2]: print b
[[ 0.32591602,  0.98844718,  0.83488971],
 [ 0.0414606 ,  0.85280814,  0.56735287]]
In [1]: b=loadtxt('fiche.txt',skiprows=1,usecols=(0,1),\
        unpack=False)
In [2]: print b
[[ 0.32591602,  0.0414606 ],
 [ 0.98844718,  0.85280814],
 [ 0.83488971,  0.56735287]]
```

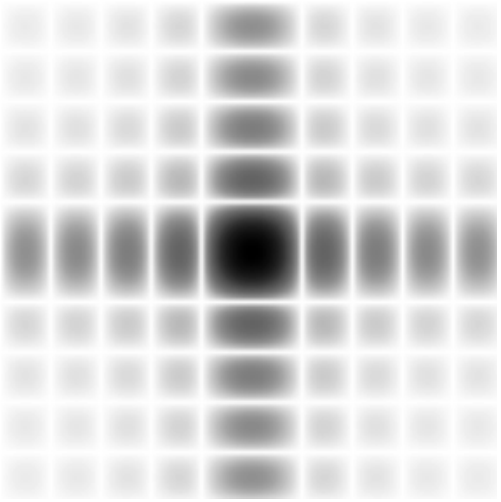
Transformada rápida de Fourier

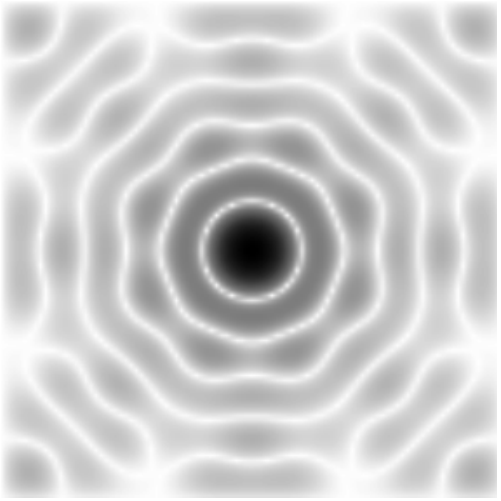
```
In [1]: from scipy import *
In [2]: a=random.random((1,4))
In [3]: b=fft(a)
In [4]: print a
[[ 0.97374222,  0.72134957,  0.26577133,  0.64617774]]
In [5]: print b
[[  2.60704086+0.j    0.70797089-0.07517182j
   -0.12801376+0.j    0.70797089+0.07517182j]]
In [6]: print ifft(b)
[[ 0.97374222+0.j  0.72134957+0.j
   0.26577133+0.j  0.64617774+0.j]]
```



Transformada rápida de Fourier

```
# help mgrid cuando el paso es complejo
x,y = mgrid[-2:2:100j, -2:2:100j]
# Cuadrado rotado
z1 = zeros((100,100));
z1[45:55,45:55]=1.0
A1 = fftshift(fft2(z1))
S1 = log(1 + abs(A1))
# circulo de radio 0.25
z2 = sqrt(x**2 + y**2) < 0.25
A2 = fftshift(fft2(z2))
S2 = log(1 + abs(A2))
##########################################################
fig=figure(figsize=(8,8))
subplot(221); imshow(z1, cmap=cm.Greys); axis('off')
subplot(222); imshow(S1, cmap=cm.Greys); axis('off')
subplot(223); imshow(z2, cmap=cm.Greys); axis('off')
subplot(224); imshow(S2, cmap=cm.Greys); axis('off')
```





Midiendo tiempos

```
# fichero timing.py
import time # modulo para medir tiempos
from numpy import *
a = linspace(0, 1, 5E+06) # generar un vector
t0 = time.clock()
b = 5*a - 3
t1 = time.clock() # t1-t0 is the CPU time of 5*a-3
for i in xrange(a.size): b[i] = 5*a[i] - 3
t2 = time.clock()
print '5*a-3: %g seg, bucle: %g seg' % (t1-t0, t2-t1)
```

```
In [1]: run timing.py
3*a-1: 0.04 sec, loop: 13 sec
```

Al igual que en MATLAB, podemos apreciar la diferencia en velocidad de ambos métodos al vectorizar las asignaciones.

Cuadratura numérica

```
from scipy.integrate import quad
a = float(raw_input('Extremo inf. a: '))# no permite pi
b = eval(raw_input('Extremo sup. b: ')) # permite pi
"""
Cuadratura numerica con quad
"""
aprox = quad(lambda x: exp(-x**2)*sin(x),a,b)
disp(['aproximacion: ', aprox[0]])
disp(['estim. error: ', aprox[1]])
```

```
In [1]: a = float(raw_input('Extremo inf. a: '))
Extremo sup. a: 0
In [2]: b = eval(raw_input('Extremo sup. b: '))
Extremo inf. b: pi
['aproximacion: ', 0.42443751077246467]
['estim. error: ', 2.2123286753150863e-09]
```

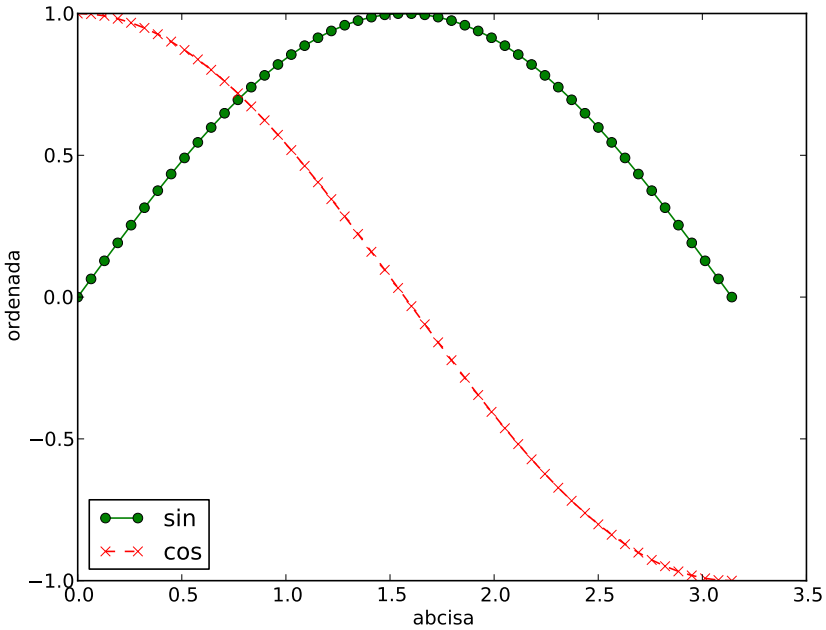
Llamando a funciones desde IPYTHON

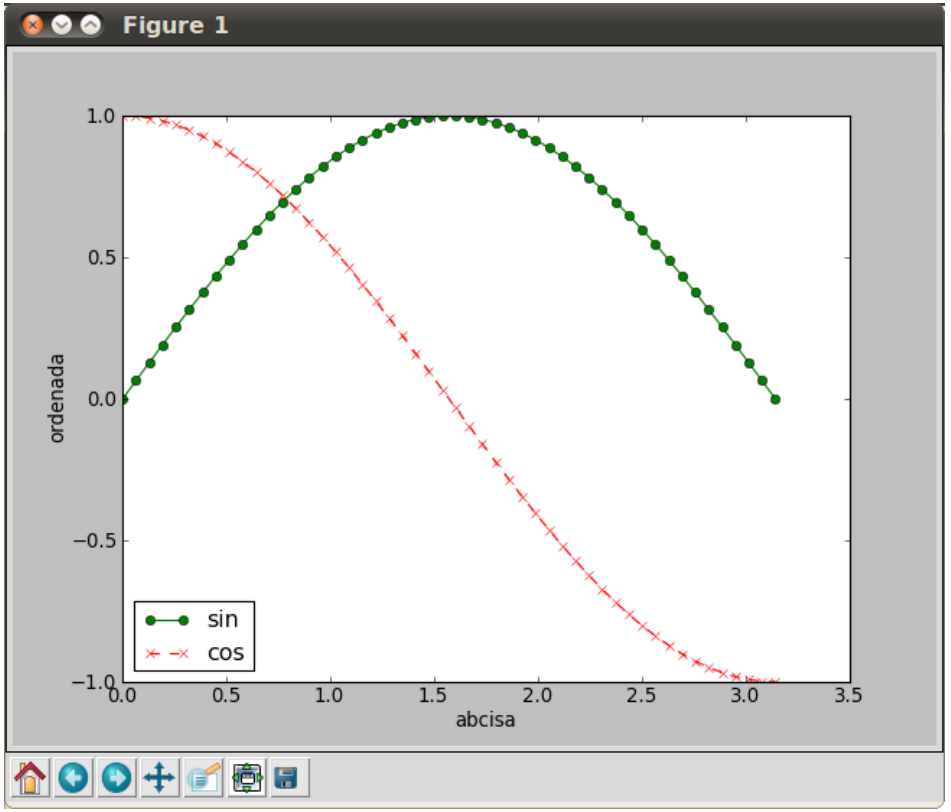
```
# fichero py1.py     # puede haber varias funciones
from numpy import * # necesario para usar pi
def esfera(r):
    """             # help py1.esfera
    Este programa calcula el area y el volumen de
    una esfera de radio r
    """
    volumen, area=4/3*pi*r**3, 4*pi*r**2
    return volumen, area
```

```
In [1]: import py1
In [2]: vol, area=py1.esfera(array([1,2,3]))
In [3]: print vol
[  3.14159265  25.13274123  84.82300165]
In [4]: print area
[  12.56637061   50.26548246  113.09733553]
```

Gráficos 2D

```
In [1]: x = linspace(0,pi,50)
In [2]: y = sin(x)
In [3]: z = cos(x)
In [4]: plot(x,y,'g-o',x,z,'r--x')
Out[4]:
[<matplotlib.lines.Line2D object at 0x2cda750>,
 <matplotlib.lines.Line2D object at 0x323e750>]

# si acabamos con ; no imprime nada

In [5]: legend(('sin', 'cos'), loc='lower left')
Out[5]: <matplotlib.legend.Legend object at 0x3243c90>
In [6]: xlabel('abcisa');

In [7]: ylabel('ordenada')
Out[7]: <matplotlib.text.Text object at 0x296b0d0>
```



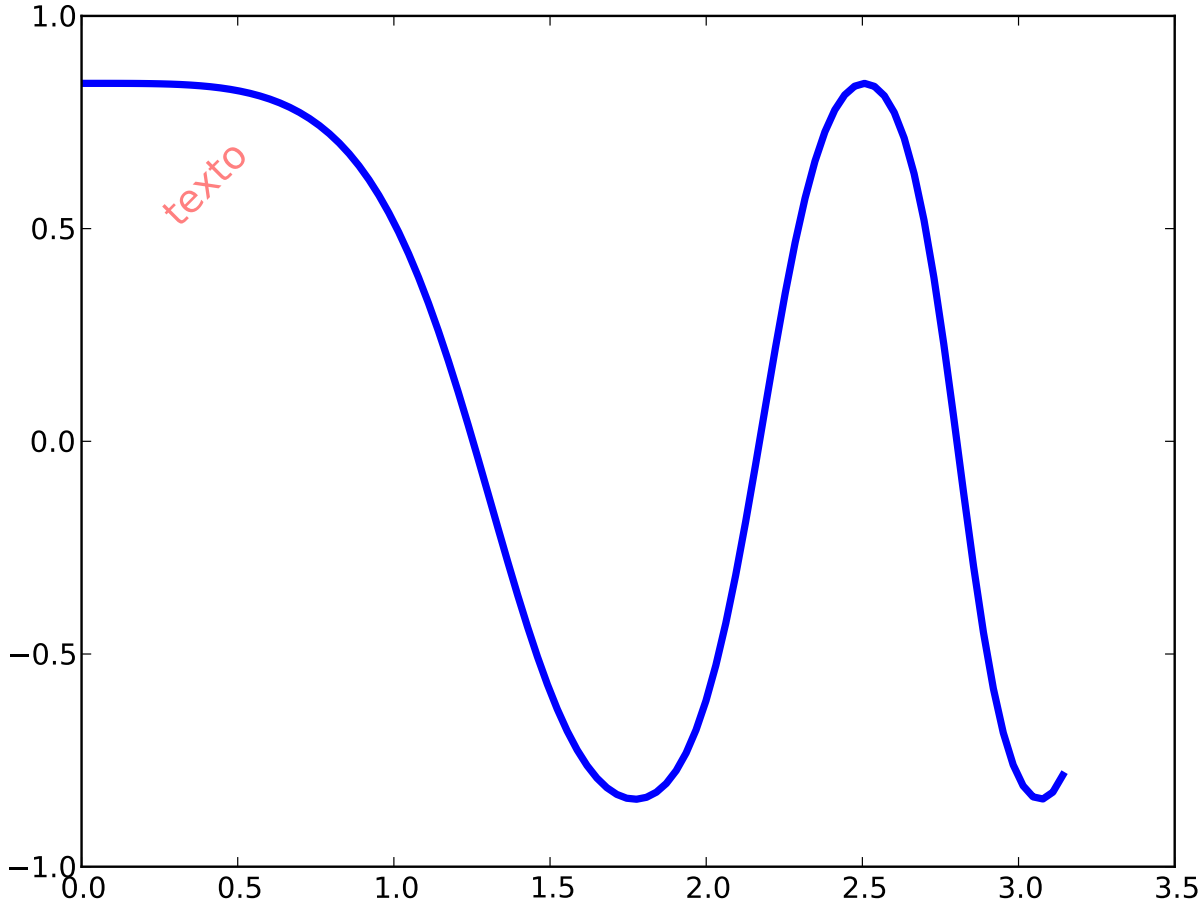


En la figura de la derecha es posible modificar y guardar el gráfico en distintos formatos (eps, png, pdf, . . .)

Los tipos de líneas y símbolos que pueden utilizarse son:

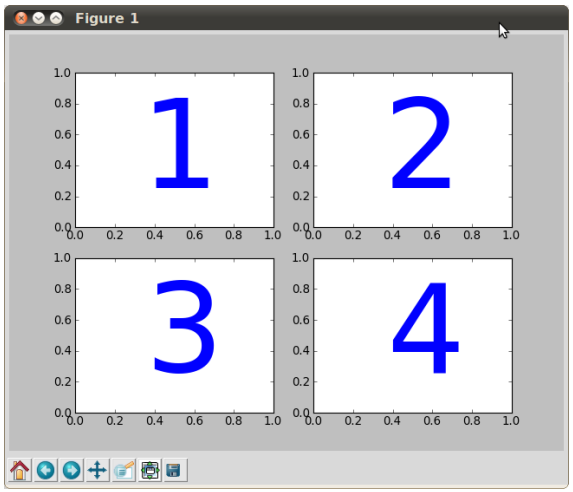
Símbolos y tipos de línea			
-	línea contínua	–	línea de segmentos discontínuos
-.	punto-segmento	:	línea de puntos
.	puntos	,	píxeles
o	○	^	△
v	▽	<	◁
>	▷	s	□
+	+	x	×
D	◇	d	◇
Colores			
b	azul	g	verde
r	rojo	c	cian
m	magenta	y	amarillo
k	negro	w	blanco

```
In [1]: x = linspace(0,pi,100)
In [2]: y = sin(cos(x**2))
In [3]: plot(x,y,linewidth=3);
In [4]: text(0.25, 0.5, 'Texto', color='red', \
        size='16', alpha=0.5, rotation=45);
```



subplot

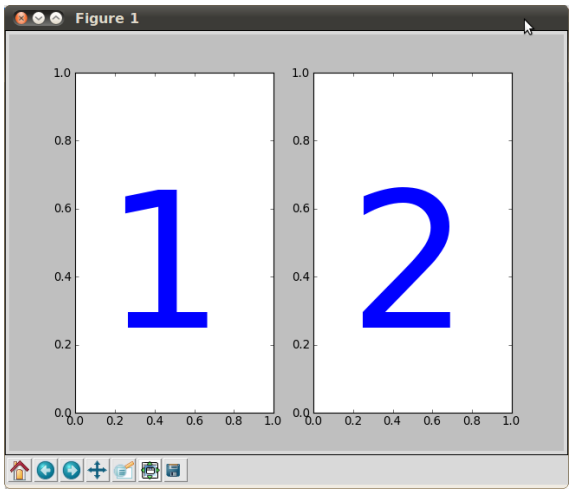
Con `subplot(mnk)` se crea una matriz `m×n` de subventanas gráficas y el índice `k` varía desde 1 hasta `m×n` por filas.



`subplot(221)`

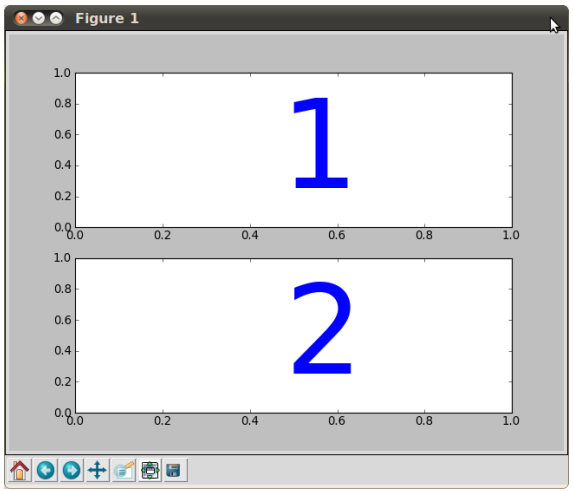
⋮

`subplot(224)`



`subplot(121)`

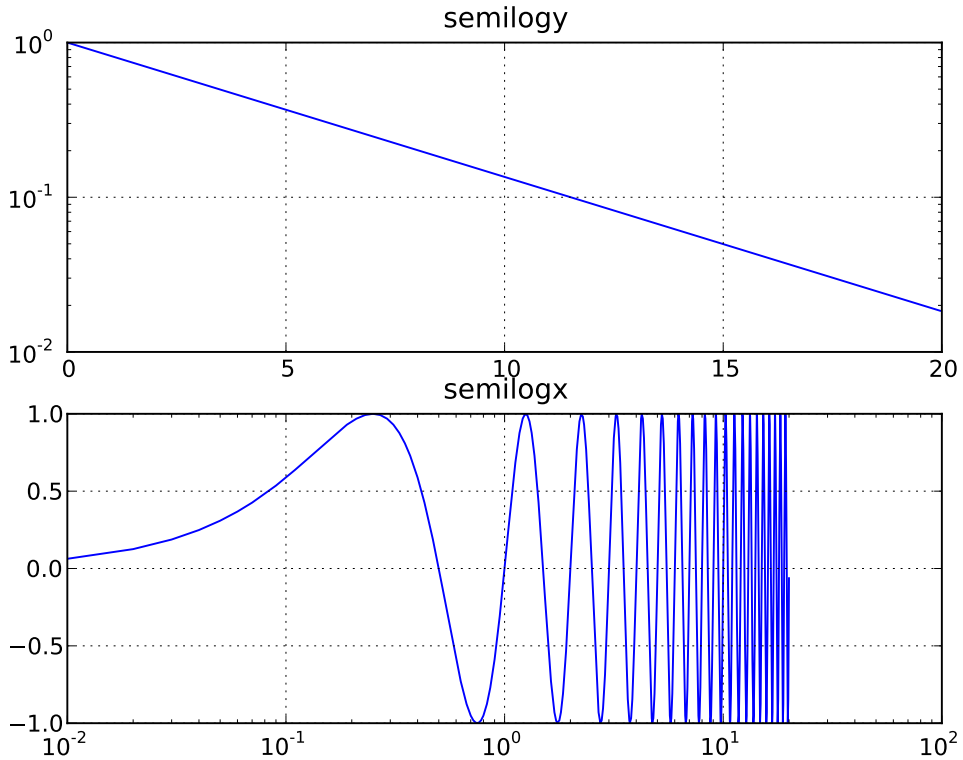
`subplot(122)`



`subplot(211)`

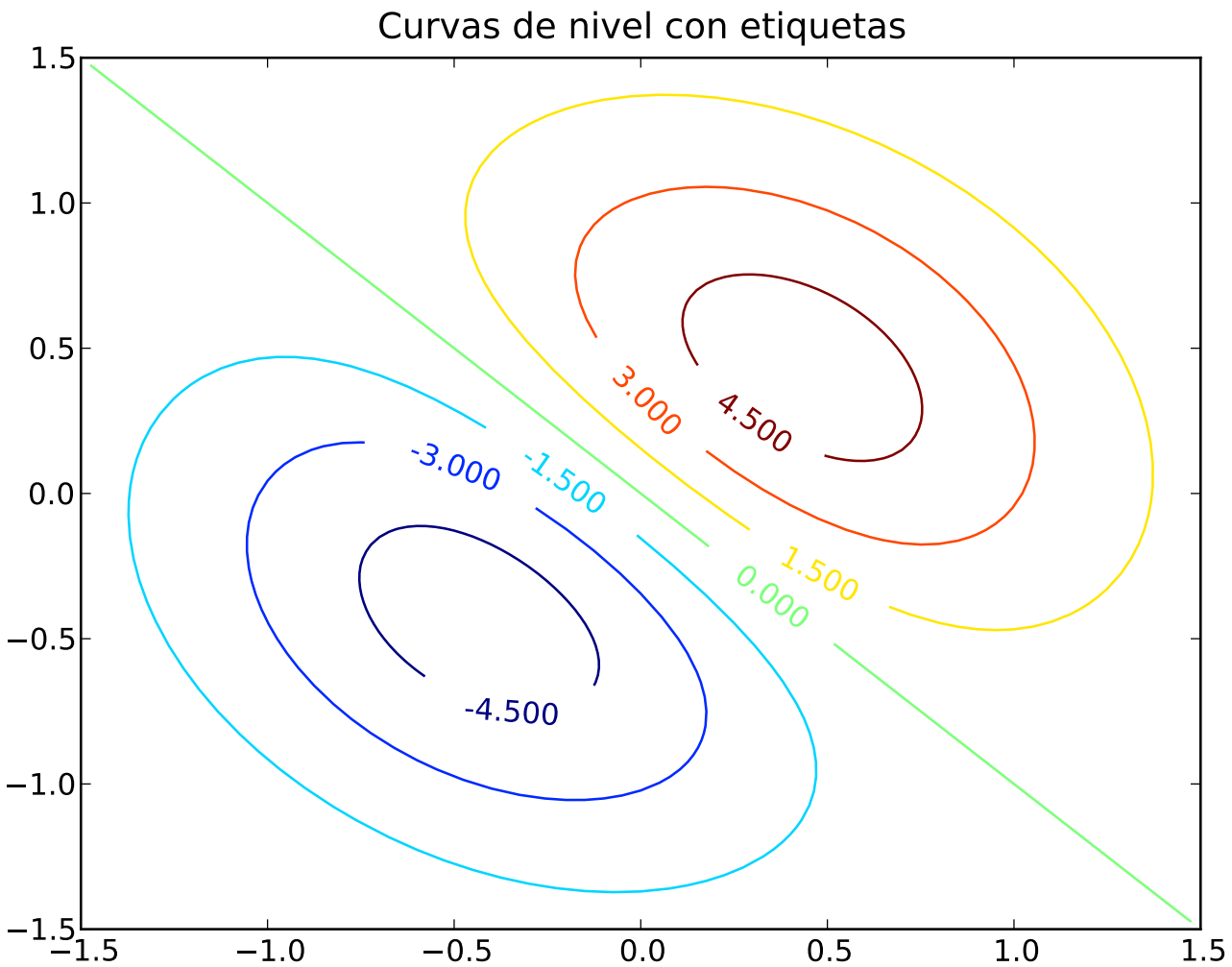
`subplot(212)`

```
t = arange(0.01, 20.0, 0.01)
subplot(211)
semilogy(t, exp(-t/5.0))
title('semilogy'); grid(True)
subplot(212)
semilogx(t, sin(2*pi*t))
title('semilogx'); grid(True)
```

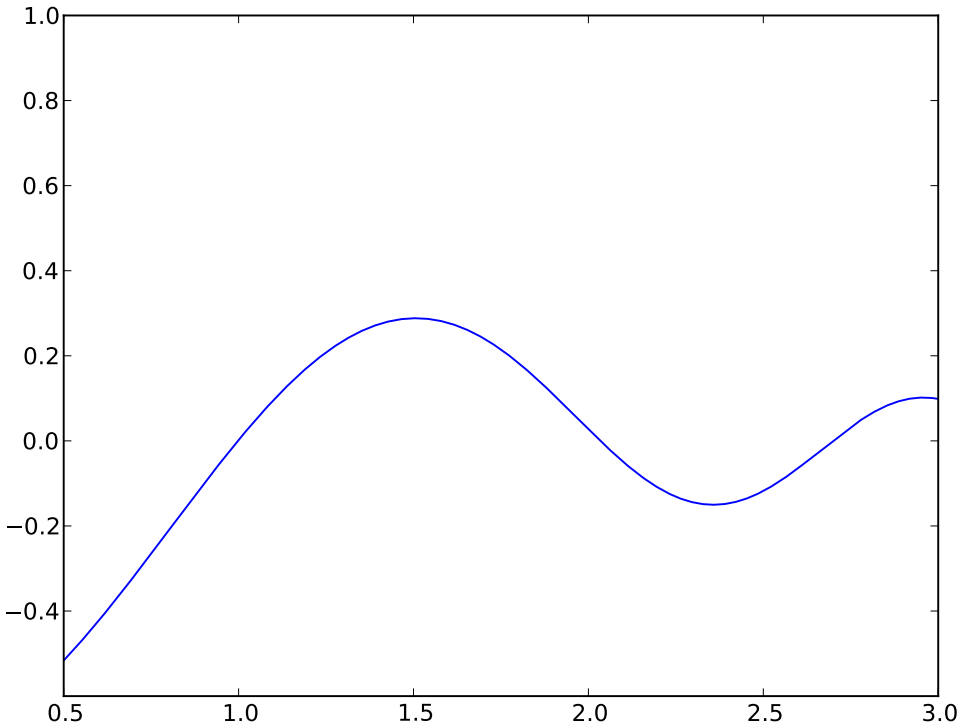


```
# fichero contouring.py
from numpy import *
from pylab import *
delta = 0.025
x = arange(-1.5, 1.5, delta)
y = arange(-1.5, 1.5, delta)
X, Y = meshgrid(x, y)
Z = 10*exp(-X**2-Y**2)*sin(X+Y)

figure()
CS = contour(X, Y, Z, 8)  # 8 curvas de nivel
clabel(CS, inline=1, fontsize=12)
title('Curvas de nivel con etiquetas')
show()
```

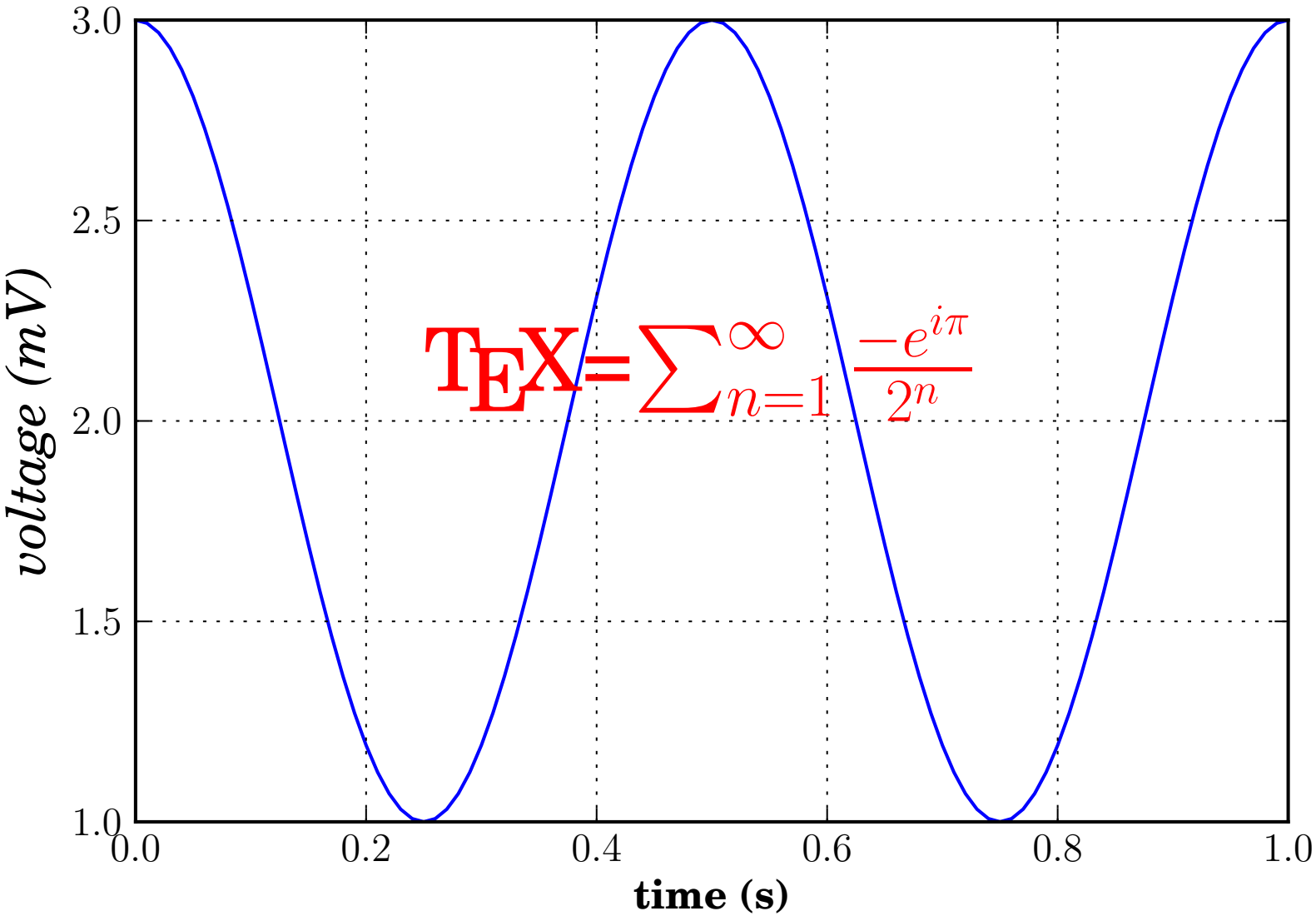
```
In [1]: run contouring.py
```


Curvas de nivel con etiquetas
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
3.000
4.500
-3.000
-1.500
1.500
0.000
-4.500

```
In [1]: from numpy import *
In [2]: x=linspace(0,pi,1000);
In [3]: y=sin( cos(x**2 + 10)/(1+x**2))
In [4]: plot(x,y);
In [5]: xlim(xmin=0.5, xmax=3);   # cambiar limites en x
In [6]: ylim(ymin=-0.6, ymax=1); # cambiar limites en y
```

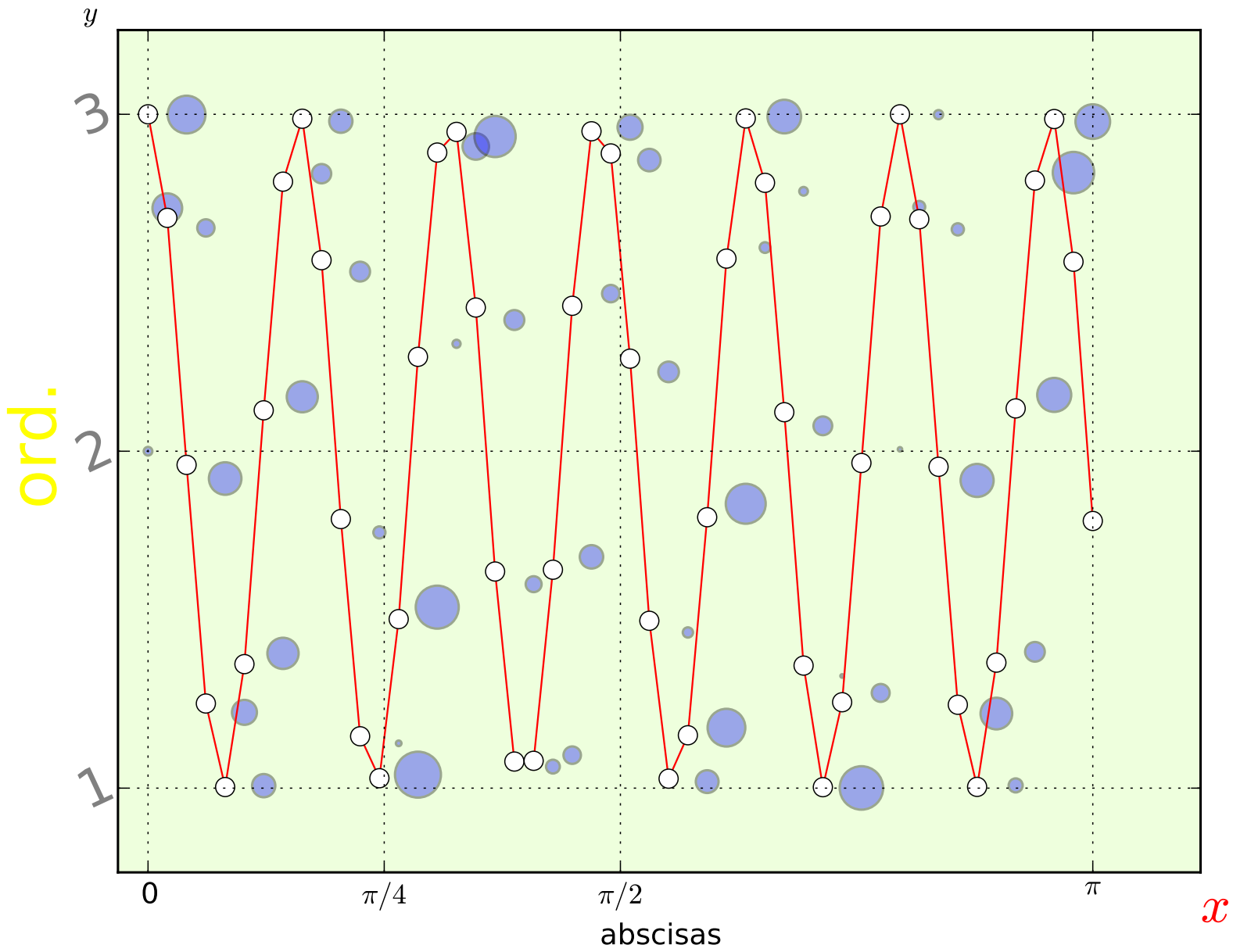


```
# fichero plot_tex.py
#
# rc sirve para modificar varios ajustes a la vez
from matplotlib import rc
from numpy import arange, cos, pi
rc('text', usetex=True, 'font', family='serif')
figure(1, figsize=(6,4))

t = arange(0.0, 1.0+0.01, 0.01)
s = cos(2*2*pi*t)+2
plot(t, s)
xlabel(r'\textbf{time (s)}')
ylabel(r'\textit{voltage (mV)}',fontsize=16)
text(0.25, 2,\
r'\TeX =$\sum_{n=1}^\infty\frac{-e^{i\pi}}{2^n}$',\
fontsize=26, color='r')
grid(True)
show()
```

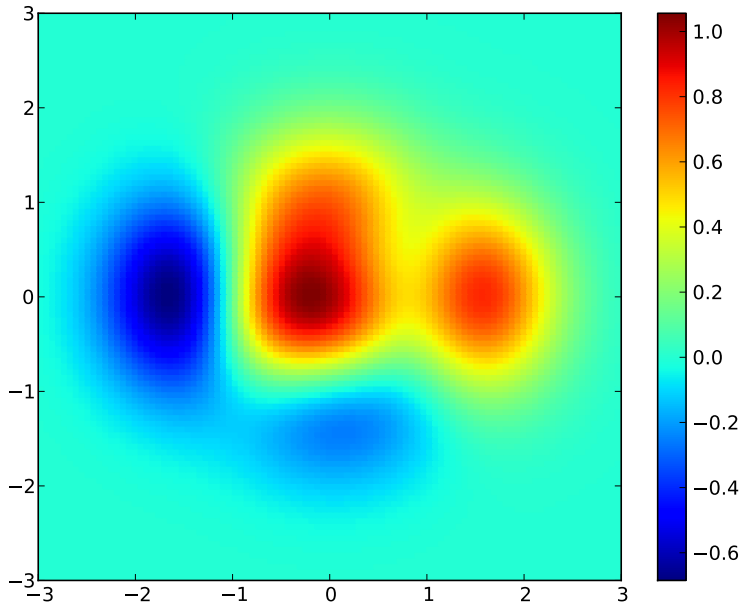

3.0
2.5
2.0
1.5
1.0
0.0
0.2
0.4
0.6
0.8
1.0
voltage (mV)
time (s)
TEX=$\sum_{n=1}^{\infty} \frac{-e^{i\pi}}{2^n}$

Ejemplo de dibujo 2D

```
from pylab import *  # run estefichero.py desde Ipython
ax = subplot(111, axisbg="#eeffdd") # (0.9,0.3,0.5)
t=linspace(0,pi,50)
y1=cos(4*pi*t)+2; y2=sin(4*pi*t)+2
area = pi*(10*rand(50))**2
plot(t,y1,'ro-',lw=0.75, ms=8, mfc='white');
scatter(t,y2,s=area,alpha=0.35);

axis([-0.1,3.5, 0.75, 3.25]);
figtext(0.9, 0.05, '$x$',color='red',size=22);
figtext(0.1, 0.9, '$y$');
yticks((1,2,3),(1,2,3),size=22,family='sans=serif',\
    rotation=25,alpha=0.5,weight='bold');
xticks((0,pi/4,pi/2,pi),(0,r'$\pi/4$',r'$\pi/2$',\
    r'$\pi$'));
xlabel('abscisas');
ylabel('ord.',color='yellow',size=27);
grid(); show()
```

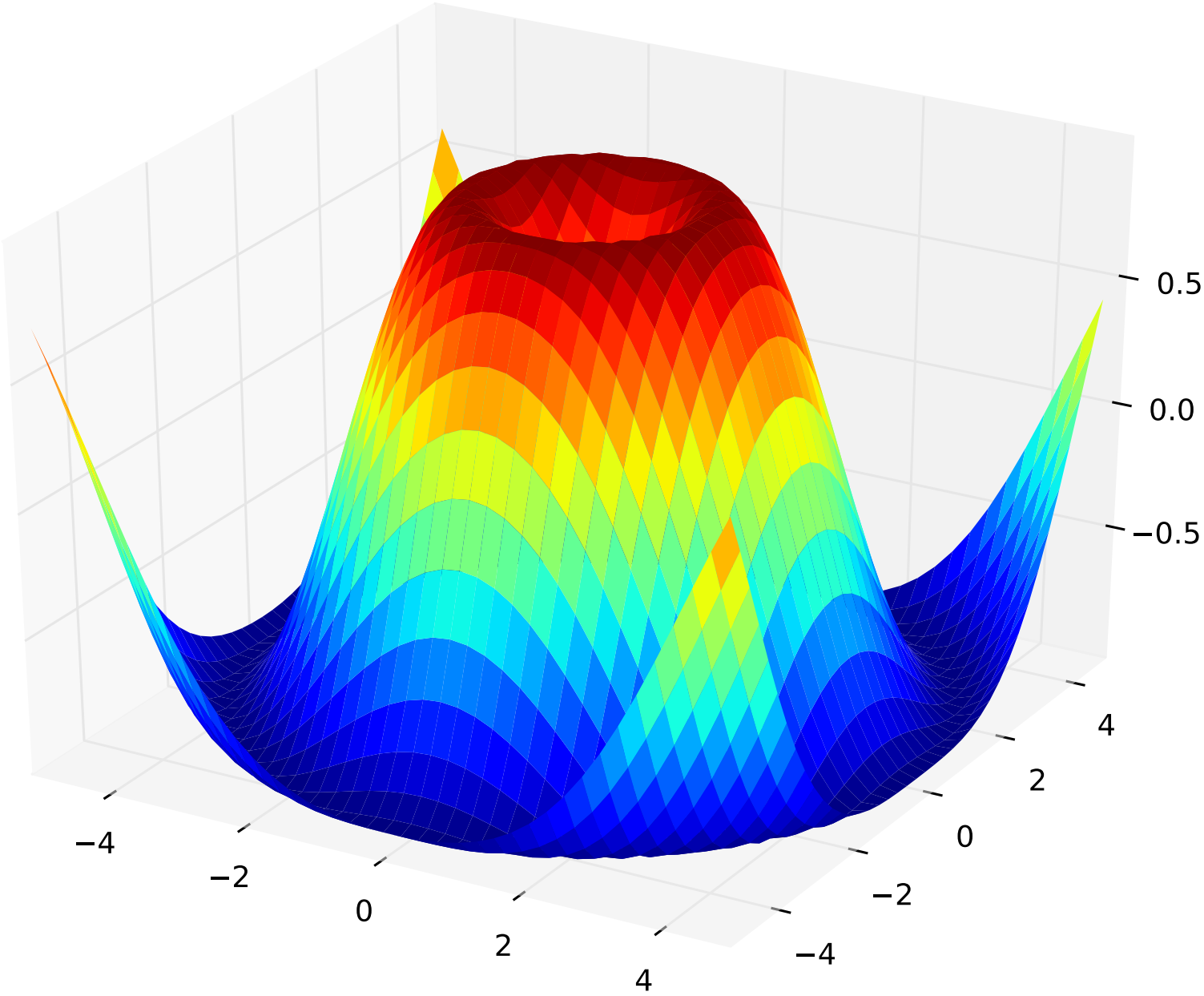

y
3
2
1
ord.
0
π/4
π/2
π
x
abscisas

```
# fichero contorno.py
from numpy import *
from pylab import *
def func3(x,y):
    return (1- x/2 + x**5 + y**3)*exp(-x**2-y**2)

x=linspace(-3,3,100); y=x
X,Y = meshgrid(x, y); Z = func3(X, Y)
pcolor(X, Y, Z); colorbar(); show()
```

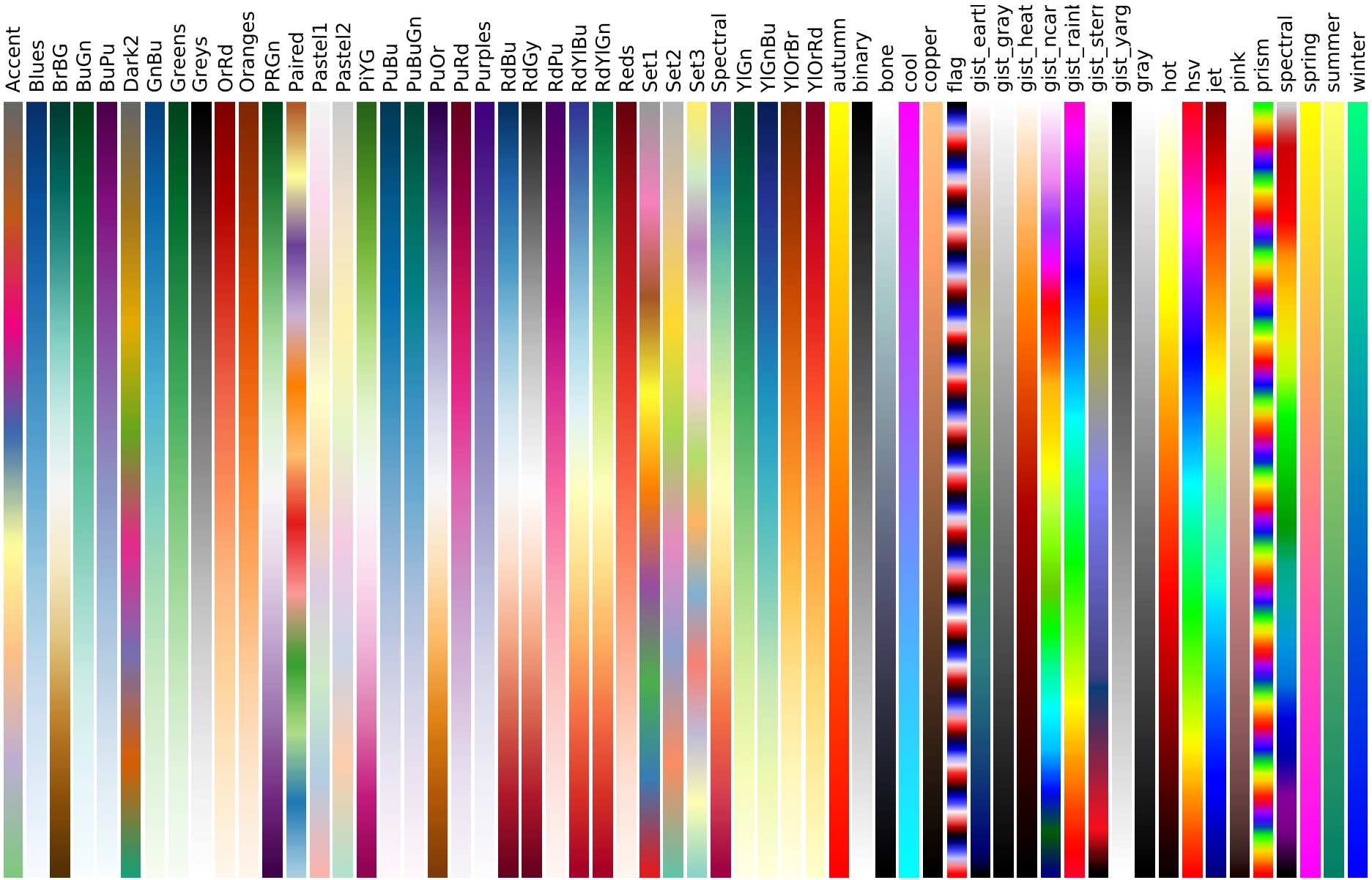


```python
# fichero sombrero.py
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
import matplotlib.pyplot as plt
import numpy as np

fig = plt.figure()
ax = Axes3D(fig)
X = np.arange(-5, 5, 0.25)
Y = np.arange(-5, 5, 0.25)
X, Y = np.meshgrid(X, Y)
R = np.sqrt(X**2 + Y**2)
Z = np.sin(R)
ax.plot_surface(X,Y,Z, rstride=1, cstride=1, cmap=cm.jet)
# cm es el colormap utilizado. Ver mas adelante
# escribir alpha=0.3 para transparencias
plt.show()
```

El dibujo de la superficie se puede rotar con ayuda del ratón.

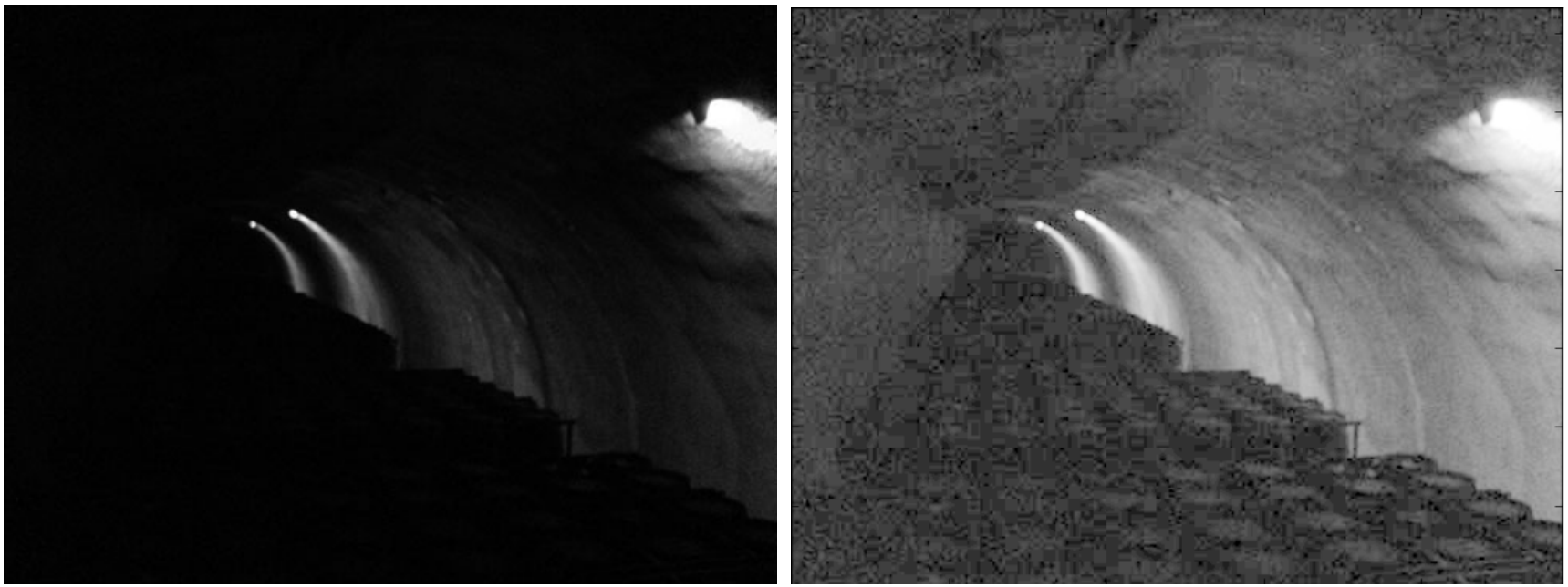

0.5
0.0
−0.5
−4
−2
0
2
4
4
2
0
−2
−4

Colormaps


Accent
Blues
BrBG
BuGn
BuPu
Dark2
GnBu
Greens
Greys
OrRd
Oranges
PRGn
Paired
Pastel1
Pastel2
PiYG
PuBu
PuBuGn
PuOr
PuRd
Purples
RdBu
RdGy
RdPu
RdYlBu
RdYlGn
Reds
Set1
Set2
Set3
Spectral
YlGn
YlGnBu
YlOrBr
YlOrRd
autumn
binary
bone
cool
copper
flag
gist_eartl
gist_gray
gist_heat
gist_ncar
gist_raint
gist_sterr
gist_yarg
gray
hot
hsv
jet
pink
prism
spectral
spring
summer
winter

Aclarando una imagen

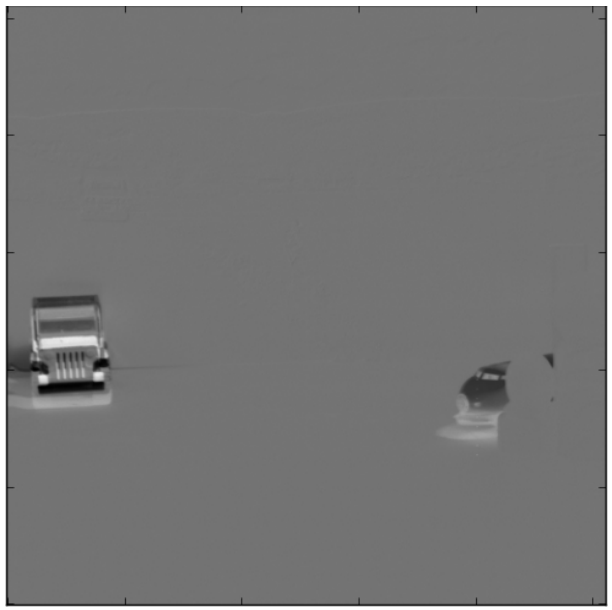
```
In [1]: X=imread('oscura.jpg')
In [2]: Z=105*log10(1+X)
In [3]: figure()  # dibujo sin ejes ni ticks
In [4]: ax = axes([0,0,1,1], frameon=False)
In [5]: ax.set_axis_off()
In [6]: imshow(Z,cmap=cm.gray,origin='left')
```



Diferencia de imágenes

```
In [1]: im1=imread('motion01.tiff')
In [2]: im2=imread('motion02.tiff')
In [3]: Q=single(im2)-single(im1)
In [4]: b=Q.max(); a=Q.min()
In [5]: Q1=floor(255-255*(Q-a)/(b-a))
In [6]: imshow(Q1, cmap=cm.gray,origin='left')
In [7]: savefig('diferencia.png')
```



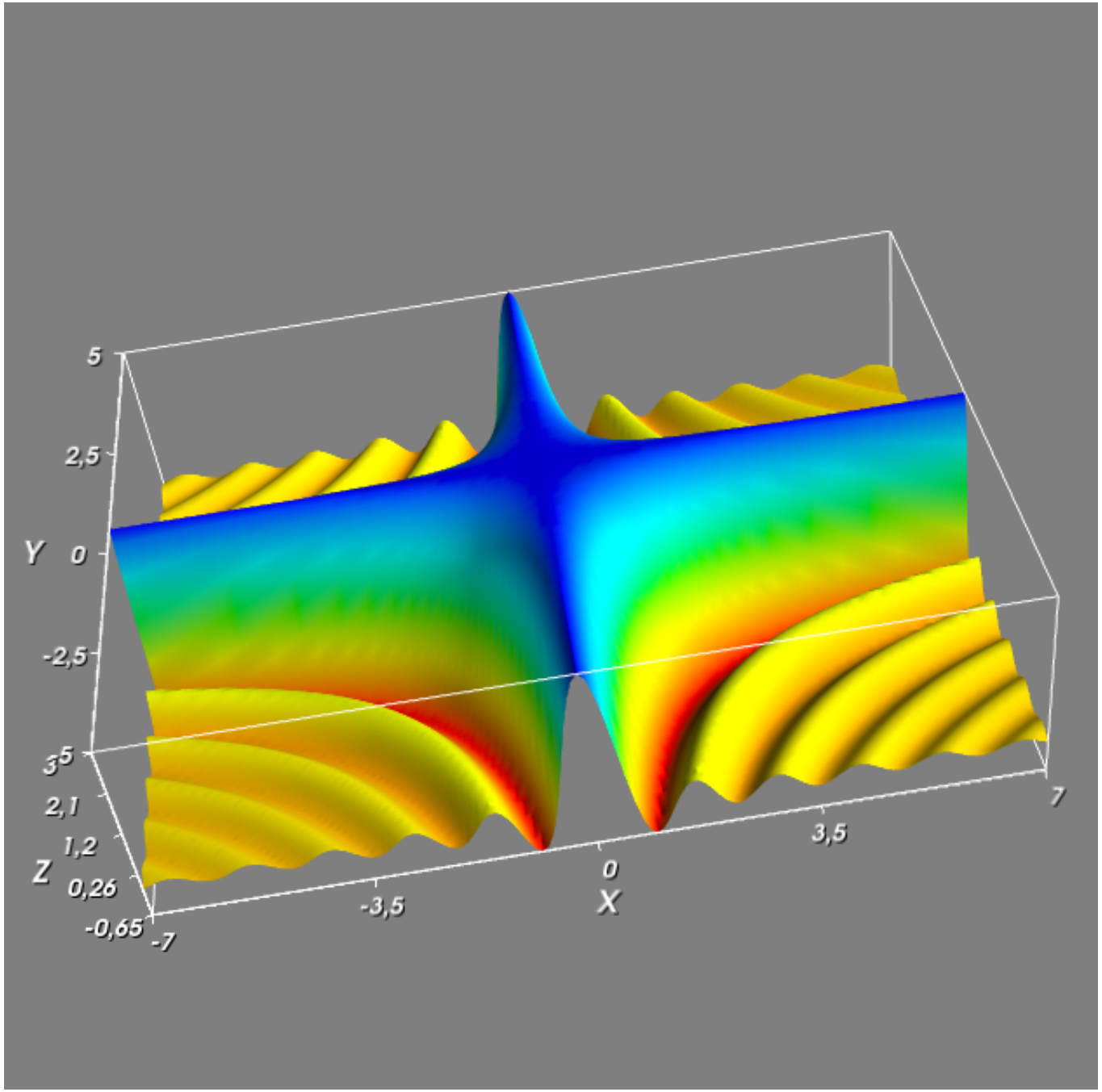


Hay programas en PYTHON que pueden resultar complicados de leer.

```
import string
def pi(x):
    _ = [0] * 10000
    a = ['@!&ABCDE?FG','_[999','_[998','(_)','while ','\n','\t',
         'return string.join','.append(str','99','.insert','for i in[']
    b = "*A@8]&:_[?77]&BCA_[?70]&:_[?71]&BC!7]F(1,'.')BCD(!7],'')$-!6]<!1]$*G?72,?74,?78,?75,?76,?73]:_[i]&$"\
        "*!9],!5]=0,!2]$*!6]+=1$*A@8]&:!0]&$*if !4]==10:_[?79]&$*if !6]:!7]E(@1]))$*_[@5]&],!5]=@4]&$*@1]=!4"\
        "]BC!4]=!3]+(!9]/10)BC!3]=!9]%10$*@1],!4]=@1]+1,0$*@0]=@9]&BC!9]=@3]&BC_[@5]&]=@2]&BC!5]=@5]&$x$(!1]"\
        "*10)/3$0$0$!2]$0$[]$2$0$0$0$-@0]%@7](_,!5])$-@0]/@7](_,@6]&)$-(!8],@5]&)$-!5]-1$-!5]$-x*!8]-1$-!5]>"\
        "0$-_[!5]-1]*10+(!9]*@6]&)"
    c={}
    for i in range(256):c[chr(i)]=chr(i)
    for i in range(1,len(a)):c[a[0][i-1]]=a[i]
    b = string.join(map(lambda x,_=c:_[x],list(b)),'').split('$')
    r = len(_)-len(b)
    for i in range(r,len(_)):
        _[1],_[2],_[3],=b[i-r],"def f%d(_,x=%d):\n\t"%(i,x),"f%d"%i
        if _[1][0]=='-':exec(_[2]+"return %s\n"%(_[1][1:]))
        elif _[1][0] == '*':exec(_[2]+"%s\n"%(_[1][1:]))
        else: _[3]=b[i-r]
        _[i]=eval(_[3])
    return _[9969](_)
print "PI=",pi(20)
print
```

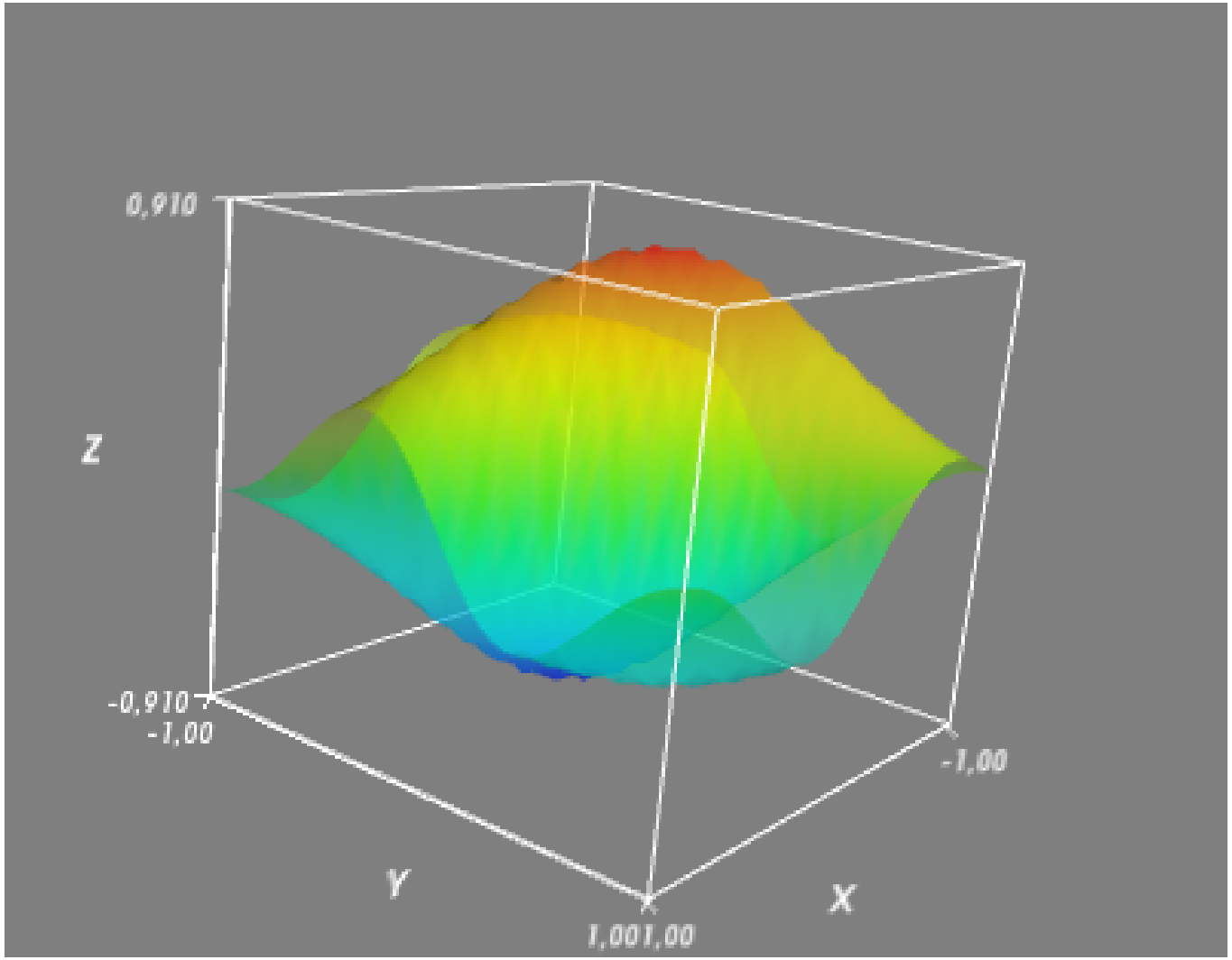
MayaVi es un visualizador de datos científicos y puede utilizarse desde PYTHON para dibujar gráficos 3D de muy alta calidad e integrado con las librerías científicas de PYTHON. Utiliza la librería de código abierto TVTK.

```
from scipy import * # ipython -wthread
def f(x, y):        # definir la funcion
    return 3.0*sin(x*y+1e-4)/(x*y+1e-4)

x = arange(-7., 7.05, 0.1)
y = arange(-5., 5.05, 0.1)

# utiliza la libreria VTK
from enthought.tvtk.tools import mlab
fig = mlab.figure()
s = mlab.SurfRegular(x, y, f)
fig.add(s)
```


5
2,5
Y 0
-2,5
-5
3
2,1
1,2
Z
0,26
-0,65
-7
-3,5
0
X
3,5
7

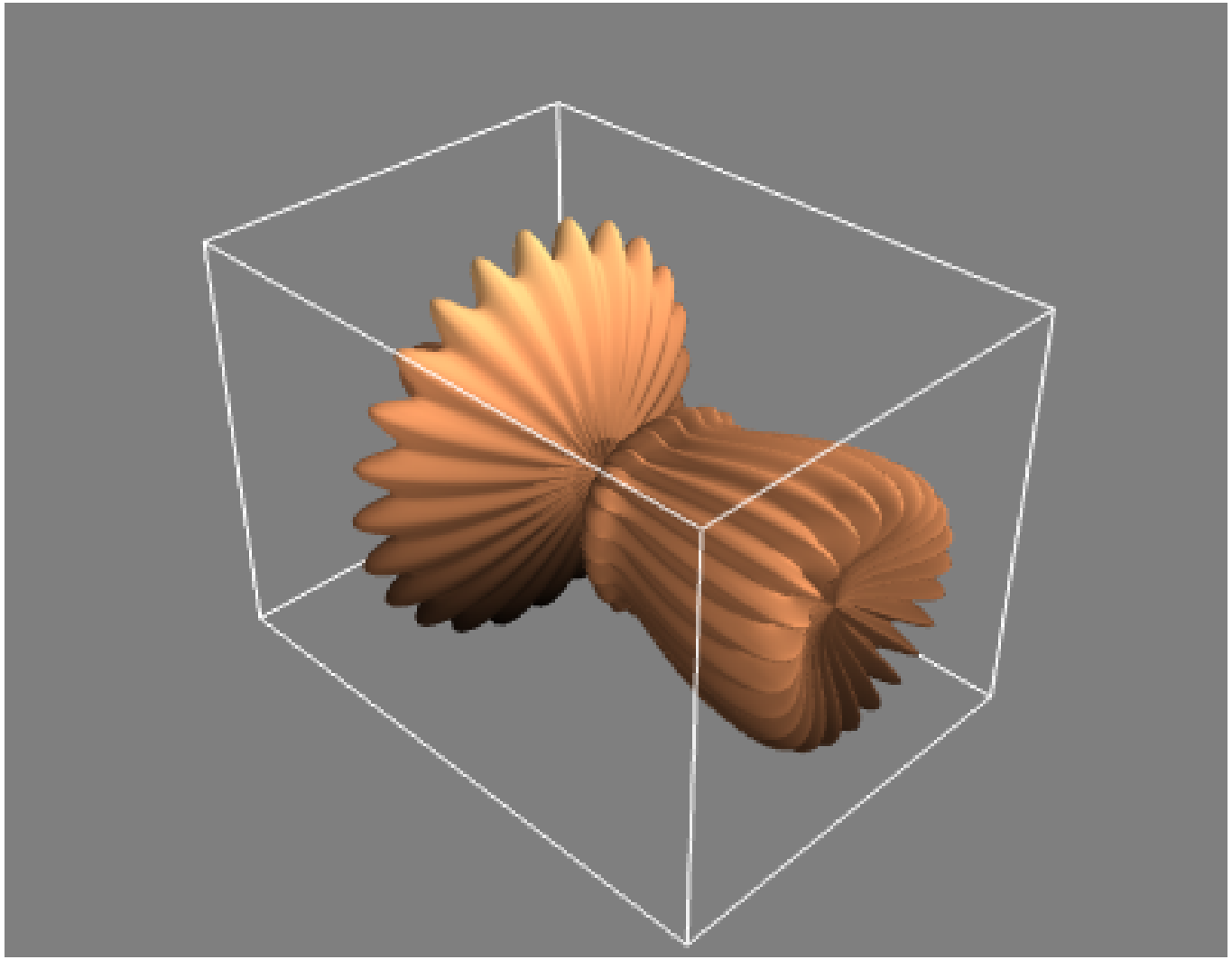
surf

```
from numpy import *
from enthought.mayavi.mlab import *
def f(x, y):
    return exp(-x**2-y**2)*sin(3*x+4*y)

x, y = mgrid[-1.:1.05:0.1, -1.:1.05:0.05]

mlab.surf(x, y, f, opacity=0.5)
mlab.outline()  # dibuja la caja
mlab.axes()
```


0,910
Z
-0,910
-1,00
Y
1,001,00
X
-1,00

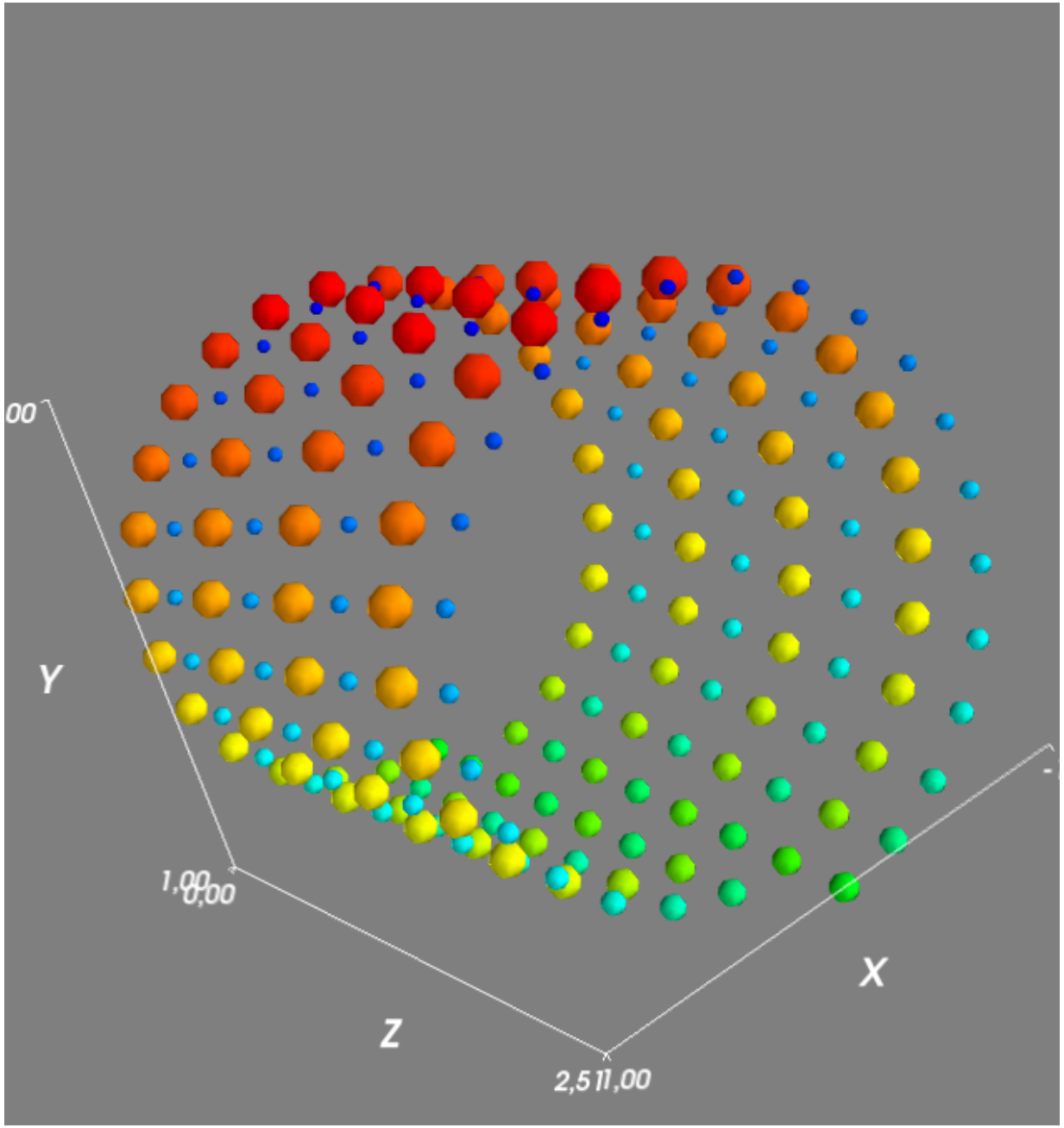
mesh

```
from numpy import *
from enthought.mayavi.mlab import *

dphi, dtheta = pi/250.0, pi/250.0
[phi,theta] = mgrid[0:pi+dphi*1.5:dphi,\
0:2*pi+dtheta*1.5:dtheta]
m0 = 4; m1 = 3; m2 = 2; m3 = 3; m4 = 6;
m5 = 2; m6 = 6; m7 = 4;
r = sin(m0*phi)**m1 + cos(m2*phi)**m3 +\
    sin(m4*theta)**m5 + cos(m6*theta)**m7
x = r*sin(phi)*cos(theta)
y = r*cos(phi)
z = r*sin(phi)*sin(theta);

mlab.mesh(x, y, z, colormap="copper")
mlab.outline()
```



points3d

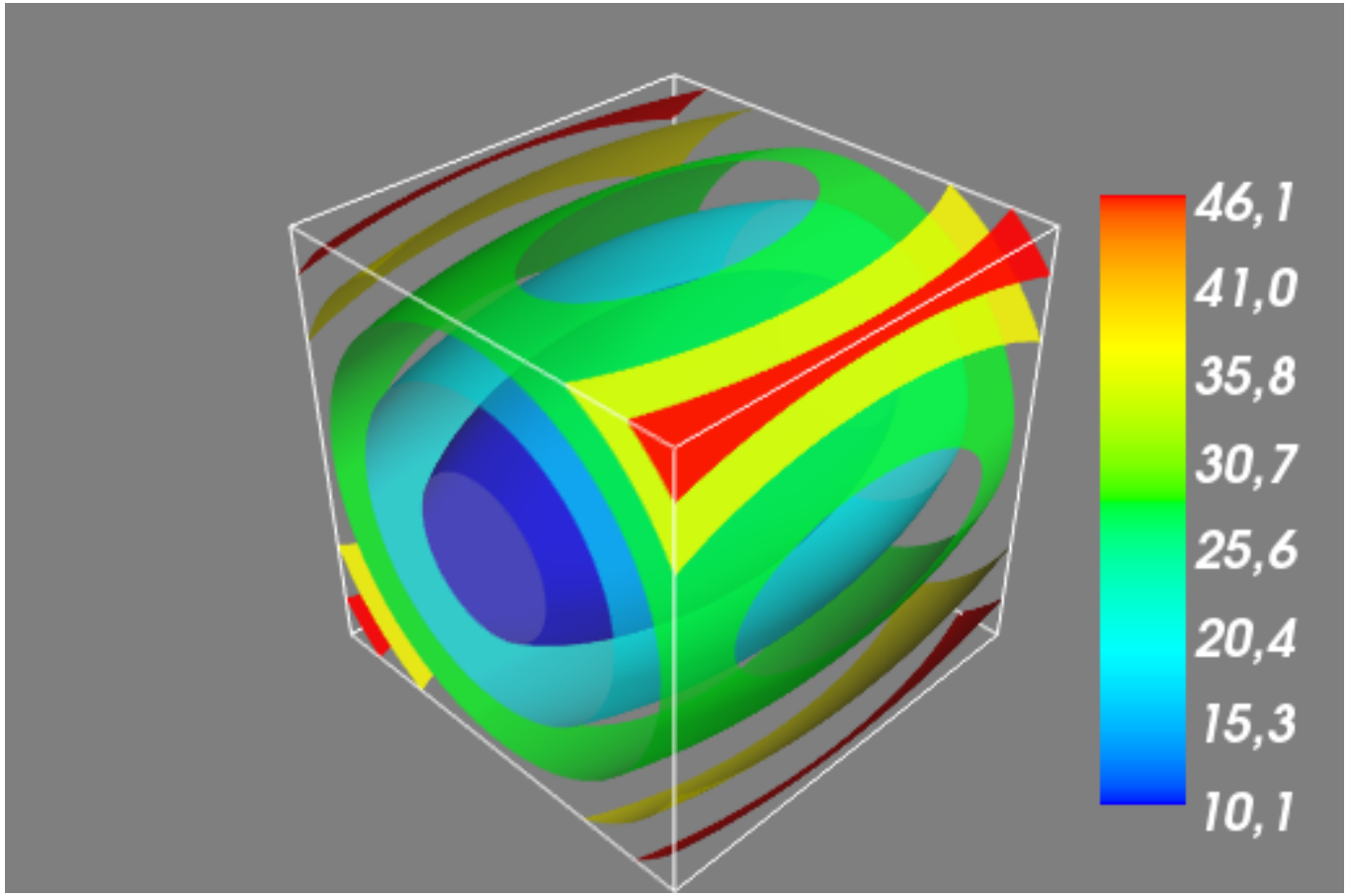
```
from numpy import *
from enthought.mayavi.mlab import *

t = linspace(0, 8*pi, 200)

x = sin(2*t)
y = cos(2*t)
z = t/10
s = 2+sin(t) # radio de las esferas

mlab.points3d(x, y, z, s, scale_factor=.05)
mlab.xlabel("X")
```


Y
1,00
0,00
Z
2,5
11,00
X

contour3d

```
from numpy import *
from enthought.mayavi.mlab import *

x, y, z = ogrid[-5:5:64j, -5:5:64j, -5:5:64j]
fun = x*x/4 + y*y + z*z
mlab.contour3d(fun, contours=7, transparent=True)
mlab.outline()
mlab.colorbar(orientation="vertical")
```

Pueden verse más ejemplos de gráficas en esta dirección.


46,1
41,0
35,8
30,7
25,6
20,4
15,3
10,1

Bibliografía


http://docs.python.org/


http://www.tutorialspoint.com/python/


http://mathesaurus.sourceforge.net/matlab-numpy.html


Python Tutorial Release 2.6.5
Guido van Rossum Fred L. Drake, Jr., editor, 2010


SciPy Reference Guide. Release 0.10.0.dev6665
Written by the SciPy community 2010


NumPy Reference. Release 2.0.0.dev-cdac120
Written by the NumPy community 2010


Python Scientific lecture notes Release 2010
EuroScipy tutorial team Editors: Emmanuelle Gouillart, Gaël Varoquaux