



## 4.3. Matplotlib

**Autores:** José María Herrera Fernández, Luis Miguel Sánchez Brea

**Contenidos de este capítulo:**

- [Funciones principales](#)
- [Ejemplo: gráfica simple en una dimensión](#)
- [Ejemplo: gráfica completa en una dimensión](#)
- [Ejemplo: Varias gráficas individuales en una figura](#)
- [Ejemplo: Gráfica en dos dimensiones](#)
- [Ejemplo: Gráfica en tres dimensiones](#)

[Matplotlib](#) es el módulo de dibujo de gráficas 2D y 3D que vamos a utilizar aunque no es el único existente. [Matplotlib](#) tiene multitud de librerías de las cuales nosotros, por semejanza a Matlab, utilizaremos [pyplot](#). Recomendamos al lector visitar <http://matplotlib.sourceforge.net/>, donde puede encontrar multitud de programas y [ejemplos](#) de como hacer dibujos con [Matplotlib](#). La documentación oficial se encuentra en <http://matplotlib.sourceforge.net/contents.html> no obstante, a lo largo de esta sección veremos algunas de las funciones más interesantes.

### TABLA DE CONTENIDOS

Autores
Qué es nuevo
Licencia
Descargas
Referencias
1. Introducción
2. Puesta en Marcha
3. Lenguajes de Programación
4. Módulos Científicos
5. Ecuaciones de Maxwell en el vacío
6. Polarización
7. Interacción radiación-materia
8. Índice de refracción: modelo microscópico
9. Ecuaciones de Fresnel
10. Pulsos de luz
11. Interferencias
12. Módulos de Óptica
13. Difracción en campo cercano
14. Difracción en campo lejano
15. Redes de Difracción

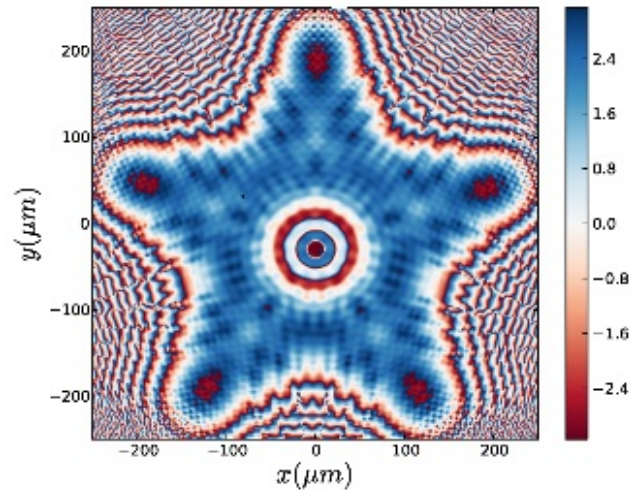


Figura 2. Ejemplo de gráfica realizada con la librería pyplot de Matplotlib (DOE de fase).

Para poder utilizar el módulo Matplotlib en nuestros programas primero debemos importarlo. Por comodidad los módulos de nombre más extenso suelen renombrarse para optimizar su importación. En nuestro caso vamos a importarlo como *plt* incluyendo en el inicio de nuestro programa la siguiente línea de código:

```
import matplotlib.pyplot as plt
```

En la Figura 3 podemos ver que si escribimos esta línea de comando en una consola, con solo poner *plt* se nos muestra en pantalla la ayuda para este módulo. Si además tras *plt* añadimos un punto, se nos muestra un desplegable con todas las funciones en este módulo. Si pinchamos en una de ellas, al igual que antes, se nos abre una ventana de ayuda que nos indica el uso del comando.

## BÚSQUEDA

Introduzca los términos de búsqueda o un nombre de módulo, clase o función.

## PyDev Console [1]

```
>>> import sys; print('%s %s' % (sys.executable or sys.platform, sys.version))
```

Python 2.7.2+ (default, Oct 4 2011, 20:06:09)

Type "copyright", "credits" or "license" for more information.

IPython 0.10.2 -- 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, ?? prints more.

PyDev console: using IPython 0.10

/usr/bin/python2.7 2.7.2+ (default, Oct 4 2011, 20:06:09)

[GCC 4.6.1]

```
>>> import matplotlib.pyplot as plt
```

```
>>> plt
```

plt

Provides a MATLAB-like plotting framework.

:mod:`~matplotlib.pylab` combines pyplot with numpy into a single namespace.

This is convenient for interactive work, but for programming it is recommended that the namespaces be kept separate, e.g.::

```
import numpy as np
import matplotlib.pyplot as plt
```

```
x = np.arange(0, 5, 0.1);
y = np.sin(x)
```

Press Ctrl+Space for templates

```
>>> import sys; print('%s %s' % (sys.executable or sys.platform, sys.version))
```

Python 2.7.2+ (default, Oct 4 2011, 20:06:09)

Type "copyright", "credits" or "license" for more information.

IPython 0.10.2 -- 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, ?? prints more.

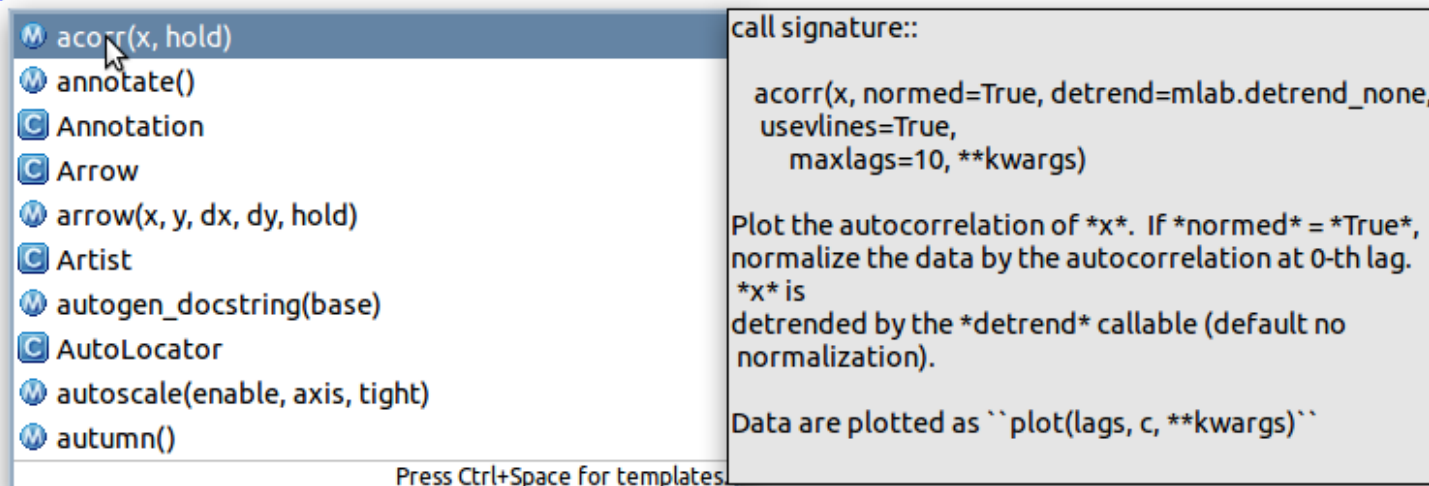
PyDev console: using IPython 0.10

/usr/bin/python2.7 2.7.2+ (default, Oct 4 2011, 20:06:09)

[GCC 4.6.1]

```
>>> import matplotlib.pyplot as plt
```

```
>>> plt.
```



The screenshot shows the IPython console with the following text:

```
M acorr(x, hold)
M annotate()
C Annotation
C Arrow
M arrow(x, y, dx, dy, hold)
C Artist
M autogen_docstring(base)
C AutoLocator
M autoscale(enable, axis, tight)
M autumn()
```

Below the list, it says "Press Ctrl+Space for templates". A tooltip for the `acorr` method is displayed, showing its call signature and description:

call signature::

```
acorr(x, normed=True, detrend=mlab.detrend_none,
      usevlines=True,
      maxlags=10, **kwargs)
```

Plot the autocorrelation of `*x*`. If `*normed* = *True*`, normalize the data by the autocorrelation at 0-th lag. `*x*` is detrended by the `*detrend*` callable (default no normalization).

Data are plotted as ``plot(lags, c, \*\*kwargs)``

Figura 3. Importación del módulo matplotlib como plt en una consola interactiva de Eclipse.

como ya explicamos en temas anteriores también podemos importar matplotlib mediante la sentencia

```
from matplotlib import pyplot
```

donde para ejecutar la función debemos usar

```
pyplot.funcion()
```

o cargar todas las funciones,

```
from matplotlib import *
```

### 4.3.1. Funciones principales

---

El módulo **Matplotlib** (al igual que la mayoría de los módulos) es enorme y explicar todas y cada una de las funciones nos llevaría demasiado tiempo así que analizaremos las más interesantes para un usuario principiante.

**figure(num, figsize, dpi, facecolor, edgecolor, frameon)** → Crea una nueva figura. Se puede utilizar sin argumentos. Devuelve un identificador a la figura.

- num = numeración de la figura, si num = None, las figuras se numeran automáticamente.
- figsize = w, h tuplas en pulgadas. Tamaño de la figura
- dpi = Resolución de la imagen en puntos por pulgada.
- facecolor = Color del rectángulo de la figura.
- edgecolor = Color del perímetro de la figura.
- frameon = Si es falso, elimina el marco de la figura.

```
>>> figure(num = None, figsize = (8, 6), dpi = 80, facecolor = 'w', edgecolor = 'k')
```

que nos crea el objeto

```
<matplotlib.figure.Figure object at 0x3938e50>
```

**subplot(numRows, numCols, plotNum)** → Permite incluir varias gráficas en una única figura.

- numRows = Número de filas
- numCols = Número de columnas
- plotNum = Número de gráfica

```
>>> subplot(211)
```

es la primera gráfica de la primera línea. Ejecutando,

```
<matplotlib.axes.AxesSubplot object at 0x393de90>
```

**plot(x, y, linestyle, linewidth, marker)** → Permite incluir varias gráficas en una única figura.

- x = Abcisas.

- $y$  = Ordenadas. Tanto  $x$  como  $y$  pueden ser abscisas tuplas, listas o arrays. La única condición es que el tamaño de ambas debe ser el mismo ya que en caso contrario python nos devolverá un fallo de tipo dimesión. También se puede hacer una gráfica sin especificar la coordenada  $x$ .
- `linestyle` = color y tipo de dibujar la gráfica. Por ejemplo `'k- -'`
- `linewidth` = ancho de línea.
- `marker` = Marcador.

por ejemplo,

```
>>> plt.plot(x, y, 'k--', linewidth = 2)
```

Dibuja la curva  $y(x)$  en trazo discontinuo de color negro con ancho de línea de tamaño 2. La función *plot* devuelve una lista. El número de elementos depende del número de plots realizados. El tipo retornado para cada elemento de la lista es *matplotlib.lines.Lines2D*. Analizando cada elemento se puede extraer información que caracteriza al gráfico (formato de marcador, colores usados,...).

```
>>> from matplotlib import pyplot
>>> x = range(10)
>>> y = x
>>> a = pyplot.plot(x,y)
>>> len(a)
1
>>> a[0].get_color()
'b'
>>> a[0].get_ls()
'_-'
>>> a[0].get_marker()
'None'
```

**Tipos** Los tipos más comunes son:

- ‘ - ‘ línea sólida
- ‘ - - ‘ línea a rayas
- ‘ -. ‘ línea con puntos y rayas
- ‘ : ‘ línea punteada

## Colores

‘b’ Azul

‘g’ Verde

‘r’ Rojo

‘c’ Cíán

‘m’ Magenta

‘y’ Amarillo

‘k’ Negro

‘w’ Blanco

También se puede escribir el color de las siguientes formas: nombres (‘green’); cadenas hexadecimales (‘#008000’); tuplas con convención RGB (0,1,0); intensidades de escala de grises (‘0.8’).

## Marcadores

Los tipos principales son:

[ ‘+’ | ‘\*’ | ‘,’ | ‘.’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘<’ | ‘>’ | ‘D’ | ‘H’ | ‘^’ | ‘\_’ | ‘d’ | ‘h’ | ‘o’ | ‘p’ | ‘s’ | ‘v’ | ‘x’ ].

**Fillstyle** Relleno de símbolo elegido para representar cada dato.

[ ‘full’ | ‘left’ | ‘right’ | ‘bottom’ | ‘top’ ]

**show()** Presenta las figuras en pantalla mediante una ventana en la que podemos interactuar. Este comando se debe añadir siempre que necesitemos observar nuestra gráfica ya que sino python realizará el cálculo pero no presentará la imagen. Lo usual es ponerlo al final del programa posibilitando así la llamada a todas las figuras que se encuentren en el mismo.

```
>>> plt.show()
```

**legend(labels, loc)** Coloca una leyenda. Cuando se presentan varias curvas simultáneamente este comando identifica cada una. Puede utilizarse sin argumentos si en cada función plot (o equivalente) se ha incluido el argumento ‘label’, que es un texto para identificar la curva.

```
>>> plt.legend( ('Etiqueta1', 'Etiqueta2', 'Etiqueta3'), loc = 'upper left')
```

**plt.xlabel(‘s’, comandos\_optativos)** Etiqueta el eje de abscisas de la gráfica actual. **plt.ylabel(‘s’, comandos\_optativos)** Etiqueta el eje de abscisas de la gráfica actual. **plt.title(‘s’, comandos\_optativos)** Etiqueta el eje

de abcisas de la gráfica actual.

- `s` = Texto que aparecerá en el título
- `comandos_optativos` = En esta etiqueta englobamos todos los modificadores de la fuente etc.

```
>>> plt.xlabel("Tiempo (s)", fontsize = 20)
>>> plt.ylabel("Distancia (m)", fontsize = 20)
>>> plt.title("velocidad (m/s)", fontsize = 20)
```

**plt.text(x, y, s, comandos\_optativos)** Añade el texto `s` en las coordenadas espaciales `x`, `y`. El texto se puede modificar como otro texto (tamaño, color, etc.).

- `x, y` = Coordenadas espaciales horizontal y vertical.
- `s` = Texto que queremos añadir

```
>>> plt.text(5, 7, "Más texto", fontsize = 12)
```

**axis()** Establece u obtiene las propiedades de los ejes. Podemos establecer el rango de coordenadas en `x` e `y` que queremos mostrar en el gráfico. Del mismo modo, podemos seleccionar la relación de aspecto entre las coordenadas `x` e `y`.

- `axis()`, devuelve los límites de los ejes (`[xmin, xmax, ymin, ymax]`)
- `axis(v)`, establece los valores límites de los ejes a `v = [xmin, xmax, ymin, ymax]`
- `axis('off')`, elimina líneas de ejes y etiquetas
- `axis('equal')`, cambia los límites de `x` e `y` para que los incrementos de `x` e `y` tengan la misma longitud (un círculo parece un círculo)
- `axis('scaled')`, cambia las dimensiones del plot para conseguir la misma longitud de intervalo en `x` e `y`.
- `axis('tight')`, cambia los ejes para que se muestren todos los datos.

**axhline(y, xmin, xmax)** Con esos valores de los parámetros predeterminados establecidos, la línea se ajusta al rango representado por los correspondientes ejes.

- `y`, array con los datos
- `xmin = 0`, valor mínimo



- `xmax = 1`, valor máximo

```
>>> plt.axhline(y = .5, xmin = 0.25, xmax = 0.75)
```

**hold()** Si el parámetro es *True*, podemos poner más curvas en la misma gráfica. Si el parámetro es *False*, entonces al escribir una nueva curva, las anteriores se borran.

**grid()** Establece la malla a *True* (visible) o *False* (oculta).

```
>>> plt.grid(True)
>>> plt.grid(color = 'r', linestyle = ' ', linewidth = 2)
```

**savefig(ruta)** Guarda la gráfica en un archivo.

- `ruta`: ruta y nombre de archivo. Los tipos de datos pueden ser `.png`, `.eps`, `.pdf`, `.ps`, `.svg`.
- `dpi = None`: resolución de la imagen en puntos por pulgada
- `facecolor = 'w'`: color del rectángulo de la figura
- `edgecolor = 'w'`: color del perímetro de la figura
- `orientation = 'portrait'`: orientación del papel (landscape)
- `format = None` : (png, pdf, ps, eps y svg).
- `transparent = False`, si es *True*, creará una gráfica de fondo transparente.

```
>>> plt.savefig('figura3.eps', dpi = 300) #guarda la gráfica con 300dpi (puntos por pulgada)
```

**close()** Cierra la gráfica

Asimismo, podemos usar la sintaxis de Latex para hacer más precisas y elegantes nuestras etiquetas. Lo único que debemos hacer es poner nuestra cadena de texto de la siguiente forma: `r'cadena latex'`. Por ejemplo:

```
xlabel(r'$\lambda$ (\AA)')
```

se muestra como  $\lambda(\text{\AA})$

### 4.3.2. Ejemplo: gráfica simple en una dimensión

```
#!/usr/bin/env python
```

```
# -*- coding: utf-8 -*-

# Carga de los módulos necesarios
import scipy as sp
import matplotlib.pyplot as plt

# Creamos el array x de cero a cien con cien puntos
x = sp.linspace(0, 10, 100)

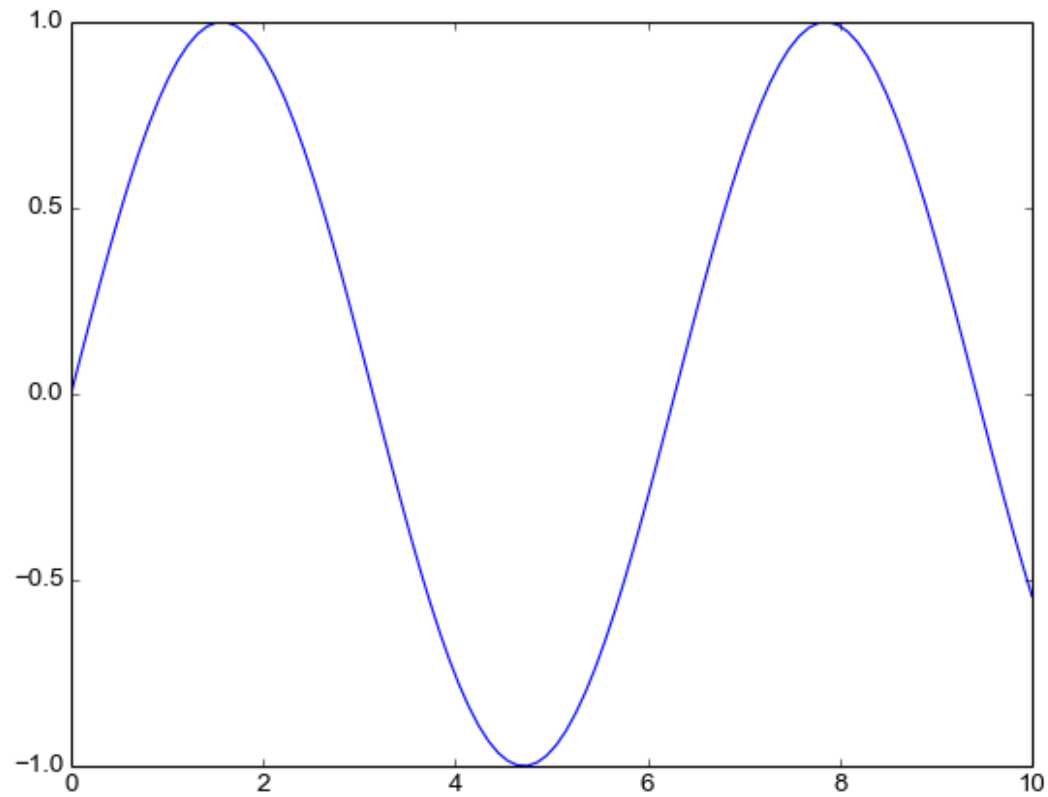
# Creamos el array y donde cada punto es el seno de cada elemento de x
y = sp.sin(x)

# Creamos una figura
plt.figure()

# Representamos
plt.plot(x, y)

# Mostramos en pantalla
plt.show()
```

([Source code](#), [png](#), [hires.png](#), [pdf](#))



### 4.3.3. Ejemplo: gráfica completa en una dimensión

---

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from __future__ import division

#-----
# Nombre:      ejemplo_matplotlib_grafica_completa.py
# Propósito:   Aprender a realizar una gráfica lo más completa posible
#
# Origen:      Propio.
```

```
# Autor: Luis Miguel S  nchez-Brea, Jos   Mar  a Herrera-Fernandez
#
# Creaci  n: 18 de Septiembre de 2013
# Historia:
#
# Dependencias: numpy, matplotlib
# Licencia: GPL
#-----

"""
Descripci  n: Doble gr  fica senoidal donde se muestran la mayor  a de los complementos
que se pueden a  adir a una gr  fica
"""

import numpy as np                # Cargamos numpy como el alias np
import matplotlib.pyplot as plt   # Cargamos matplotlib.pyplot como el alias plt

# Definimos el periodo de la funci  n
periodo = 0.5

# Definimos el array dimensional
x = np.linspace(0, 2, 1000)

# Definimos la funci  n senoidal
y = np.sin(2*np.pi*x/periodo)

# Creamos la figura
plt.figure()

# Dibujamos en negro discontinuo con etiqueta y1
plt.plot(x, y, 'k--', linewidth = 2, label = 'y1')

# Mantenemos la misma figura para la siguiente gr  fica
plt.hold(True)
```

```

# Esta vez dibujamos - y en rojo co etiqueta y2
plt.plot(x,-y,'r', linewidth = 2, label = 'y2')

# Añadimos la leyenda
plt.legend(loc = 2)

# Añadimos las etiquetas poniermo en Latex "mu" sÃmbolo de micras
plt.xlabel(r"$x$ (\mu m)", fontsize = 24, color = (1,0,0))
plt.ylabel(r"$y$ (\mu m)", fontsize = 24, color = 'blue')

# Añadimos texto
plt.text(x = 1, y = 0.0, s = u'T = 0.05', fontsize = 24)

# Añadimos la rejilla
plt.grid(True)
plt.grid(color = '0.5', linestyle = '--', linewidth = 1)

# Añadimos los ejes
plt.axis('tight')

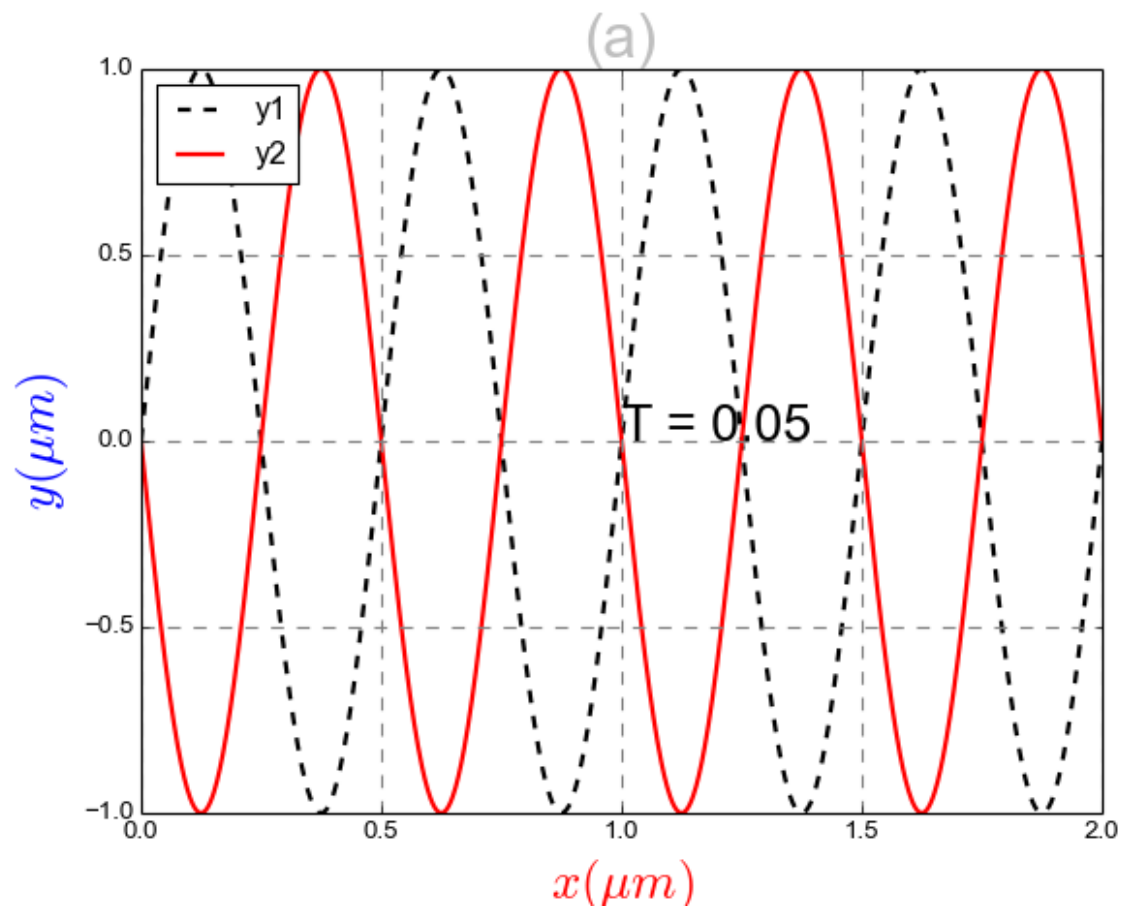
# Añadimos el título
plt.title('(a)', fontsize = 28, color = '0.75', verticalalignment = 'baseline', horizontalalignment = 'center')

# Guardamos
plt.savefig('plotCompleta.png')

# Mostramos en pantalla
plt.show()

```

([Source code](#), [png](#), [hires.png](#), [pdf](#))



#### 4.3.4. Ejemplo: Varias gráficas individuales en una figura

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from __future__ import division

#-----
# Nombre:      ejemplo_matplotlib_subplot.py
# Propósito:   Aprender a mostrar varias gráficas separadas en una figura
#
# Origen:      Propio.
```

```

# Autor:          Luis Miguel S  nchez-Brea, Jos   Mar  a Herrera-Fernandez
#
# Creaci  n:      18 de Septiembre de 2013
# Historia:
#
# Dependencias:   numpy, matplotlib
# Licencia:       GPL
#-----

"""
Descripci  n: En este ejemplo se muestra como realizar la presentaci  n de var  as
gr  ficas individuales en una misma figura mediante "subplot"
"""

import numpy as np                # Cargamos numpy como el alias np
import matplotlib.pyplot as plt   # Cargamos matplotlib.pyplot como el alias plt

# Definimos el periodo de la gr  fica senoidal
periodo = 2

# Definimos el array dimensional
x = np.linspace(0, 10, 1000)
# Definimos la funci  n senoidal
y = np.sin(2*np.pi*x/periodo)

# Creamos la figura
plt.figure()

# Primera gr  fica
plt.subplot(2,2,1)
plt.plot(x, y, 'r')

# Segunda gr  fica
plt.subplot(2,2,2)

```

```
plt.plot(x, y, 'g')

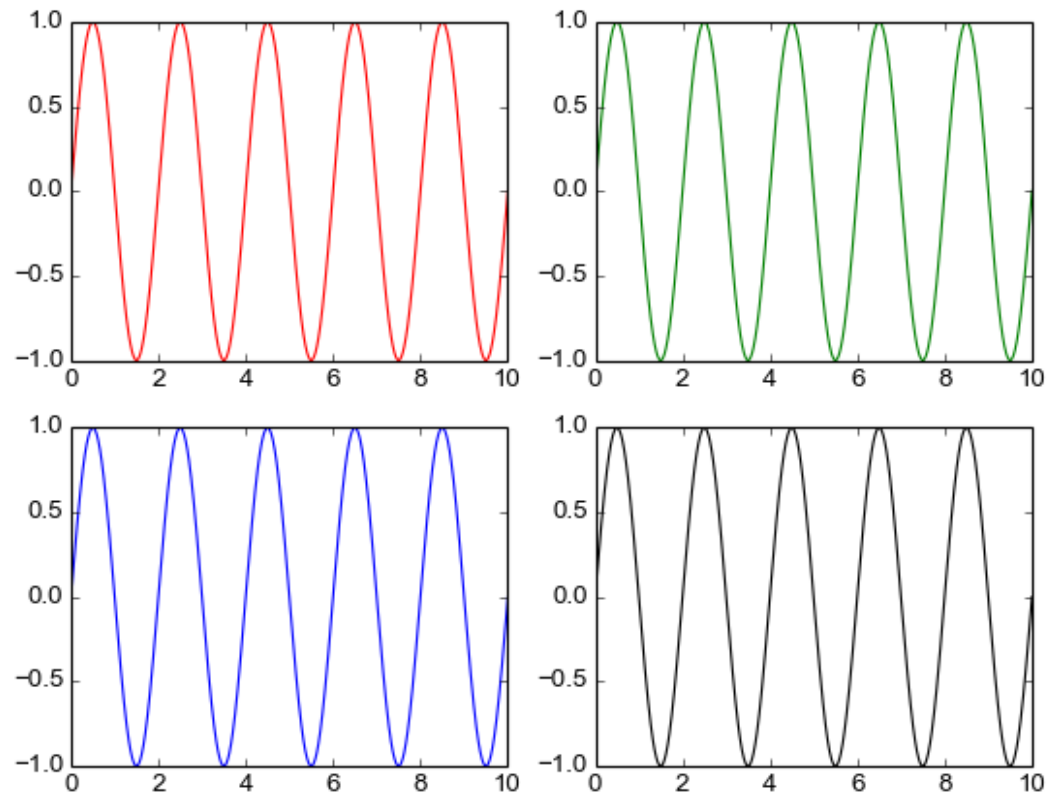
# Tercera grÃ¡fica
plt.subplot(2,2,3)
plt.plot(x, y, 'b')

# Cuarta grÃ¡fica
plt.subplot(2,2,4)
plt.plot(x, y, 'k')

# Mostramos en pantalla
plt.show()
```

([Source code](#), [png](#), [hires.png](#), [pdf](#))





#### 4.3.5. Ejemplo: Gráfica en dos dimensiones

---

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from __future__ import division

#-----
# Nombre:      ejemplo_matplotlib_imshow.py
# Propósito:   Aprender a realizar una gráfica en 2D
#
# Origen:      Propio.
```

```

# Autor:      Luis Miguel SÃ¡nchez-Brea, JosÃ© MarÃ­a Herrera-Fernandez
#
# CreaciÃ³n:   18 de Septiembre de 2013
# Historia:
#
# Dependencias:  numpy, matplotlib
# Licencia:     GPL
#-----

"""
DescripciÃ³n: En este ejemplo se muestra como realizar una grÃ¡fica en dos dimensiones
mediante imshow
"""

import numpy as np                # Cargamos numpy como el alias np
import matplotlib.pyplot as plt   # Cargamos matplotlib.pyplot como el alias plt

# Creamos una figura
plt.figure()

# Creamos los arrays dimensionales
x = np.arange(-5, 5, 0.01)
y = np.arange(-5, 5, 0.01)

# Obtenemos las corrdenadas resultantes de esos arrays
X, Y = np.meshgrid(x, y)

# Definimos la grÃ¡fica sen (x^2 + y^2)
fxy = np.sin(X**2+Y**2)

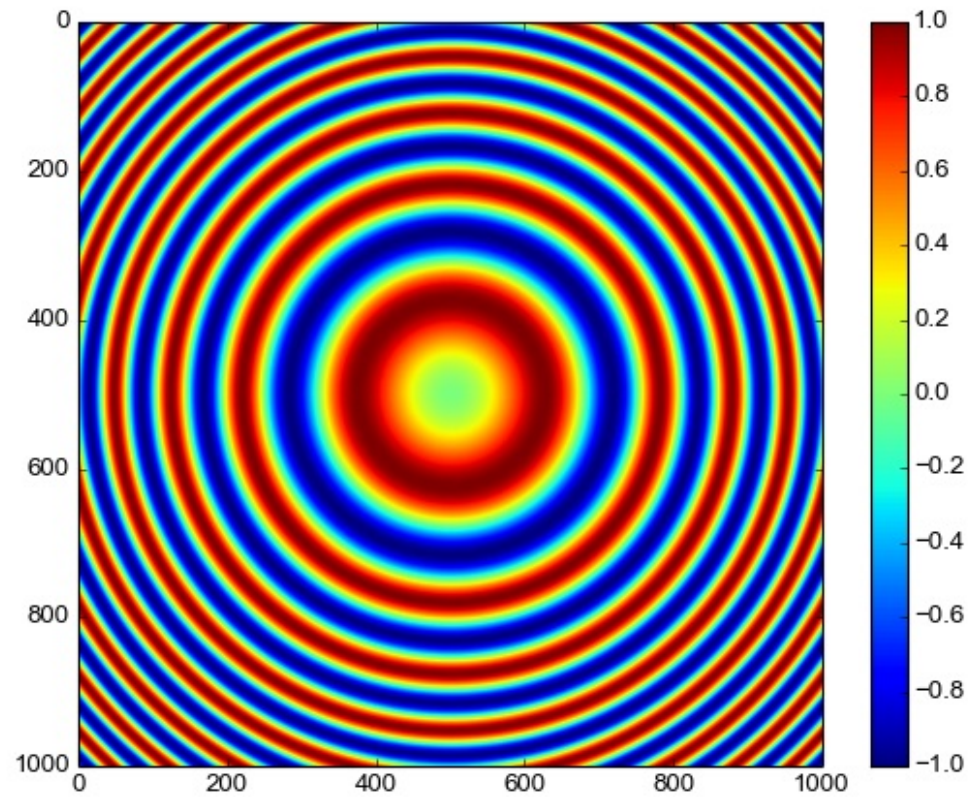
# Representamos
plt.imshow(fxy);

# AÃ±adimos una colorbar

```

```
plt.colorbar();  
  
# Mostramos en pantalla  
plt.show()
```

([Source code](#), [png](#), [hires.png](#), [pdf](#))



#### 4.3.6. Ejemplo: Gráfica en tres dimensiones

---

```
#!/usr/bin/env python  
# -*- coding: utf-8 -*-  
from __future__ import division
```

```

#-----
# Nombre:      ejemplo_matplotlib_grafica_superficie3d.py
# Propósito:   Aprender como leer una image y representar una grafica en 3D.
#
# Origen:      Propio .
# Autor:       Jos   Mar  a Herrera-Fernandez
#
# Creaci  n:   18 de Septiembre de 2013
# Historia:
#
# Dependencias:  scipy, mpl_toolkits, matplotlib, numpy
# Licencia:      GPL
#-----

"""
Descripci  n: Ejemplo de c  mo usar la funci  n integrate para realizar integrales
num  ricas en python.
"""

from mpl_toolkits.mplot3d import Axes3D      # Cargo Axes3D de mpl_toolkits.mplot3d
from scipy.misc import imread                # Cargo imread de scipy.misc
import numpy as np                          # Cargo numpy como el aliaas np
import matplotlib.pyplot as plt              # Cargo matplotlib.pyplot en el alias sp

# Leo una imagen y la almaceno en imagen_superficial
imagen_superficial = imread('fondo.jpg')

# Creo una figura
plt.figure()

# Muestro la imagen en pantalla

```

```

plt.imshow(imagen_superficial)

# Añadido etiquetas
plt.title('Imagen que usaremos de superficie')
plt.xlabel(u'# de pñ xeles')
plt.ylabel(u'# de pñ xeles')

# Creo otra figura y la almaceno en figura_3d
figura_3d = plt.figure()

# Indicamos que vamos a representar en 3D
ax = figura_3d.gca(projection = '3d')

# Creamos los arrays dimensionales de la misma dimensiñ que imagen_superficial
X = np.linspace(-5, 5, imagen_superficial.shape[0])
Y = np.linspace(-5, 5, imagen_superficial.shape[1])

# Obtenemos las coordenadas a partir de los arrays creados
X, Y = np.meshgrid(X, Y)

# Defino la funciñ que deseo representar
R = np.sqrt(X ** 2 + Y ** 2)
Z = np.sin(R)

# Reescalamos de RGB a [0-1]
imagen_superficial = imagen_superficial.swapaxes(0, 1) / 255.

# meshgrid orienta los ejes al revñs luego hay que voltear
ax.plot_surface(X, Y, Z, facecolors = np.flipud(imagen_superficial))

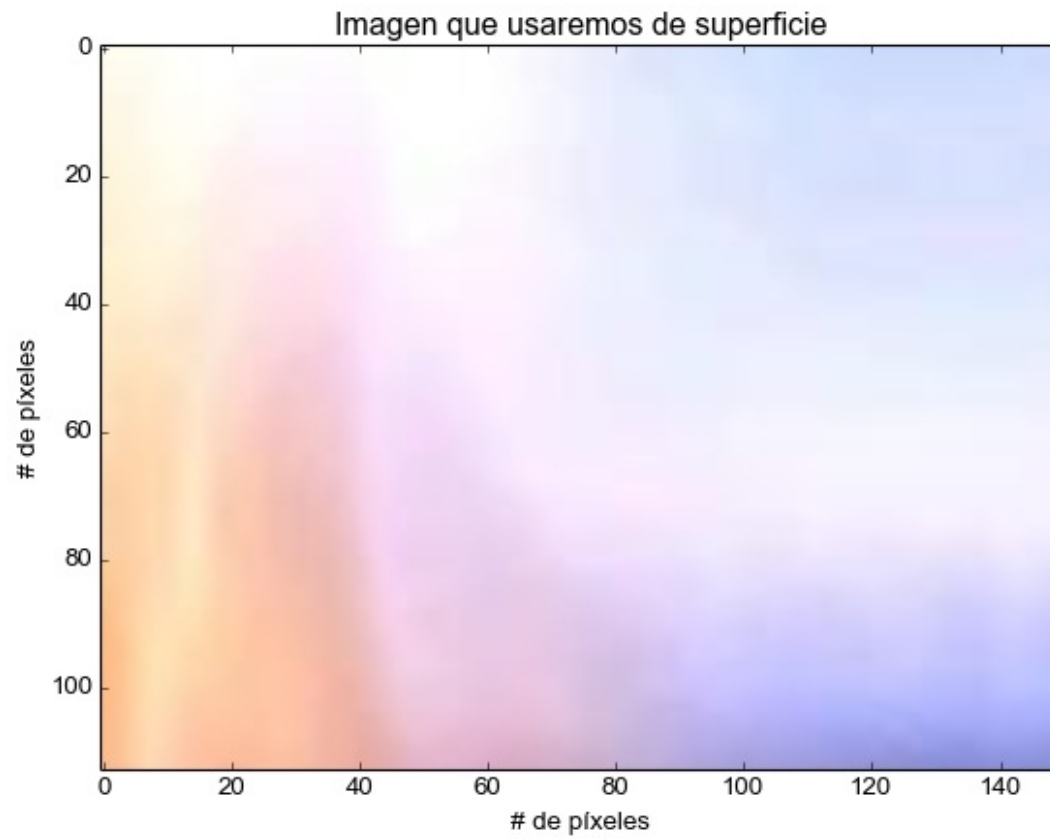
# Fijamos la posiciñ inicial de la grafica
ax.view_init(45, -35)

# Añadimos etiquetas
plt.title(u'Imagen sobre una grafica 3D')

```

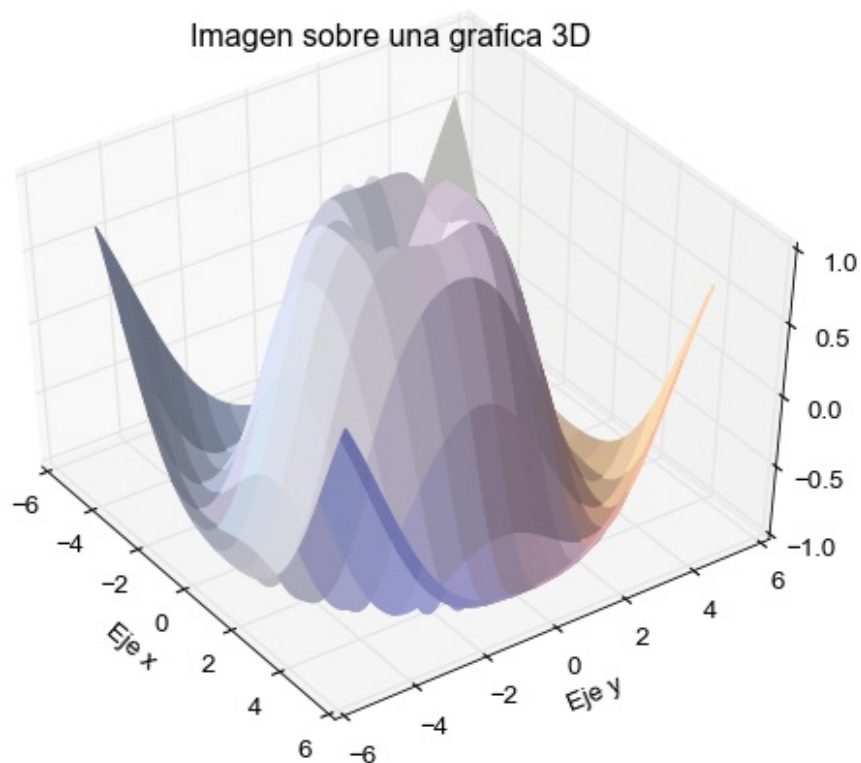
```
plt.xlabel('Eje x')  
plt.ylabel('Eje y')  
# Mostramos en pantalla  
plt.show()
```

([Source code](#))



([png](#), [hires.png](#), [pdf](#))

Imagen sobre una grafica 3D



([png](#), [hires.png](#), [pdf](#))

[ANTERIOR](#) | [SIGUIENTE](#)  
[MOSTRAR EL CÓDIGO](#)

© Copyright 2013, Luis Miguel Sánchez Brea. Creado con [Sphinx](#) 1.2.2.