Cómo graficar con Matplotlib

Ricardo Gil-Hutton
Septiembre 2016

Matplotlib es una librería de Python para hacer gráficos en 2D que hace un uso intensivo de **numpy** y que tiene ciertos puntos de contacto con MATLAB, aunque es completamente independiente.

Hay diferentes maneras de instalar **matplotlib** dependiendo de la plataforma en que se quiera utilizar. En el caso de Linux, es suficiente con instalar la librería desde el repositorio de paquetes (se encuentra con el nombre python-matplotlib).

Matplotlib es facilmente configurable con una enorme variedad de opciones que se pueden modificar en el archivo *matplotlibrc*. Para encontrar dónde está este archivo de configuración se puede ejecutar desde Python:

```
In [1]: import matplotlib

In [2]: matplotlib.matplotlib_fname()
Out[2]: u'/home/rgh/.config/matplotlib/matplotlibrc'
```

1 Interactivo o no interactivo?

Por default **matplotlib** no realiza inmediatamente las modificaciones que se ejecutan en un gráfico hasta que todos los cambios están listos debido a que realizar cada tarea inmediatamente es muy costoso en un lenguaje que se interpreta. Esta resistencia de **matplotlib** a efectuar los cambios inmediatamente es útil cuando los gráficos a desplegar son muy complejos y requeriría bastante tiempo actualizarlos, pero en general si se opera desde algún intérprete de Python y los gráficos son sencillos esto no es necesario.

IPython tiene una función mágica que permite cambiar automáticamente el modo a interactivo:

```
In [3]: %pylab
Using matplotlib backend: Qt4Agg
Populating the interactive namespace from numpy and matplotlib
```

que setea todo para que inmediatamente después de ejecutar el comando se genere o modifique el gráfico de la manera solicitada. Usando esta función mágica no es necesario importar **numpy** o **matplotlib** porque IPython lo hace automáticamente y, además, setea el modo interactivo para la sesión.

En general la función *pylab* de IPython se utiliza en situaciones especiales (en un notebook, por ejemplo) y es recomendable importar la librería con la interface más usual (*pyplot* siguiendo el método usual:

```
In [3]: import matplotlib.pyplot as plt
```

Para controlar el modo en que se encuentra **matplotlib** y actualizar la figura, *pyplot* tiene 4 funciones útiles:

- *isinteractive()*: retorna *True* o *False* si está en modo interactivo o no.
- *ion()*: se pone en interactivo.
- *ioff()*: se pone en no interactivo.
- *draw()*: fuerza a redibujar una figura.

De todos modos, la figura no aparecerá en la pantalla hasta que no se ejecute la función *show()*, pero eso lo veremos en la próxima sección.

Si se quiere que **matplotlib** se inicie en un modo particular es conveniente modificar la línea correspondiente en el archivo de configuración *matplotlibrc* poniendo "interactive" a "True" o "False".

2 Pyplot

Pyplot es una colección de funciones que permiten operar sobre una figura creándola o modificando sus características. Hay una importante lista de funciones disponibles (si matplotlib.pyplot fue importado como se indicó más arriba, en IPython escribir "plt." y presionar TAB) pero las funciones que se utilizan más frecuentemente son:

- *figure()*: crea una ventana para dibujar la figura.
- *plot()*: dibuja la figura en la ventana.
- *subplot()*: distribución de subfiguras en la ventana.
- *axis()*: valores mínimo y máximo de los ejes de la figura.
- *grid()*: dibuja una grilla.
- *title()*: título de la figura.
- *xlabel()*: etiqueta para el eje X.
- *ylabel()*: etiqueta para el eje Y.
- *text()*: posiciona y escribe texto en la figura.
- *annotate()*: posiciona y escribe texto referido a un punto de la figura en particular.

las cuales iremos viendo a continuación en una serie de ejemplos. El primero es un ejemplo muy sencillo que se muestra en la figura 1 y que nos permite ver la diferencia entre los modos de **matplotlib**. En el ejemplo que se muestra en la terminal de IPython se crean dos listas ("x" e "y"), se utiliza la función *plot(x,y)* para graficar una en función de la otra, y finalmente la función *show()* para abrir y mostrar la ventana con el gráfico. Nótese que inicialmente forzamos a **matplotlib** a trabajar en modo no interactivo (*ioff()*) y que después de ejecutar *show()* python no nos devuelve el prompt para seguir operando. Esta es una característica del modo no interactivo donde **matplotlib** queda esperando que decidamos hacer algo con la figura (un zoom, guardarla en disco, modificar los ejes, etc.) con alguno de los botones que presenta el menú y el prompt de python reaparece luego de cerrar la ventana de

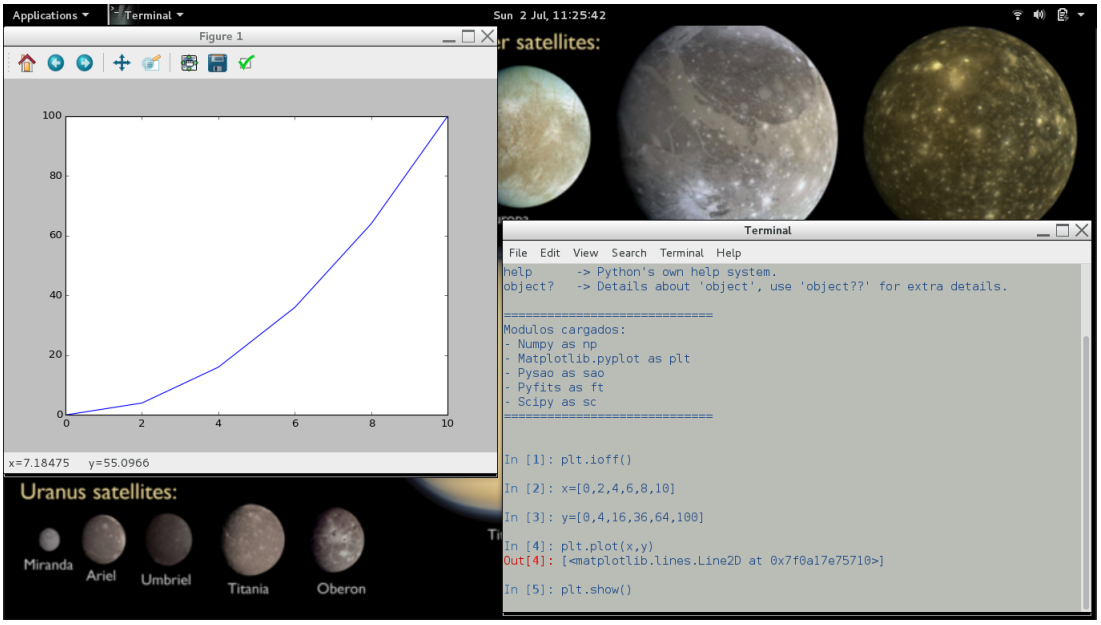

Figure 1:

la figura. En modo interactivo se obtiene el prompt inmediatamente después de dibujar pero cada operación sobre algún elemento de la figura forzará que se dibuje completa cada vez, lo que fuerza a que el proceso sea muy lento si la figura tiene muchos elementos. A partir de este momento asumiremos que **matplotlib** está en modo interactivo, asi que verifique el modo ejecutando *plt.isinteractive()* y, si es necesario, cambie el modo con *plt.ion()*.

3 Ejemplos de gráficos:

Utilizando funciones de **numpy** creamos dos arrays para graficar. Por ejemplo:

```
In [1]: x=np.arange(20)

In [2]: y=x+2
```

También se pueden utilizar listas con valores numéricos pero en ese caso no podemos operar matemáticamente con ellas lo que representa una limitación. Ahora creamos la ventana para hacer el gráfico con:

```
In [3]:  plt.figure(1,figsize=(6,4.5))
Out[3]: <matplotlib.figure.Figure at 0x7fac6b728110>
```

que abre una ventana identificada como "Figure 1". Los parámetros posibles para *plt.figure* son varios, pero aquí sólo indicamos el número de ventana donde se quiere graficar (si no existe la crea) y el tamaño de la ventana (un tuple con las dimensiones en pulgadas).

A continuación ya podemos graficar con *plt.plot*:

```
In [4]: plt.plot(x,y)
Out[4]: [<matplotlib.lines.Line2D at 0x7fac6ae50850>]
```

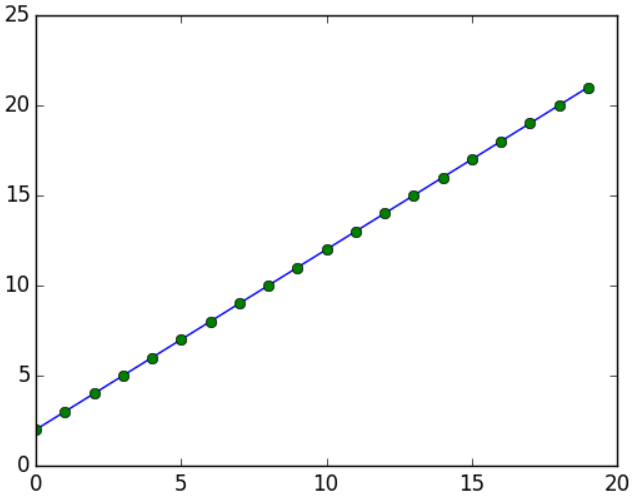

Figure 2:

Los puntos indicados con los arrays x e y se dibujan unidos por una línea azul que es el default, pero es posible utilizar otros colores y símbolos para graficar. Nótese que no fue necesario utilizar *plt.show()* porque estamos en modo interactivo. El formato general para *plt.plot()* es por ejemplo:

plot(x, y, color='blue', linestyle='solid', marker='o', linewidth=1, markerfacecolor='blue', markersize=12).

donde *color* puede ser "blue" (o su abreviatura "b"), "green" ("g"), "red" ("r"), "cyan" ("c"), "magenta" ("m"), "yellow" ("y"), "black" ("k") o "white" ("w"); *linestype* puede ser "solid" (o su abreviatura "-"), "dashed" ("–"), "dash-dot" ("-.") o "dotted" (":"); y *marker* puede ser "." para un punto, "o" para un círculo, "v" para un triángulo hacia abajo, "^" para un triángulo hacia arriba, "<" para un triángulo hacia la izquierda, ">" para un triángulo hacia la derecha, "s" para un cuadrado, "p" para un pentágono, "*" para una estrella, "h" o "H" para dos tipos de hexágonos, "+" para un mas, "x" para una cruz, y "d" o "D" para dos tipos de diamantes diferentes. Por ejemplo:

```
In [6]: plt.plot(x,y,color='g',marker='o', linewidth=0)
Out[6]: [<matplotlib.lines.Line2D at 0x7fac69ef5c50>]
```

va a sobre-escribir el gráfico marcando los puntos con círculos verdes (figura 2). Este es un detalle importante: si se utilizan funciones de **pyplot** las mismas graficarán sobre la figura que se abrió en última instancia y la sobre-escribirán. Esto es útil cuando se quiere incluir el título o alguna etiqueta, o cuando se quieren graficar dos funciones en el mismo gráfico, pero si uno desea graficar algo diferentes hay que borrar los datos graficados con *plt.cla()* que deja los ejes que se han dibujado o borrar la figura entera con *plt.clf()*.

Además de utilizar los parámetros indicados es posible emplear un formato breve para indicar el color y tipo de línea o símbolo. Por ejemplo, si quiero hacer el mismo gráfico que antes pero con diamantes de color magenta:

```
In [7]: plt.clf()
```

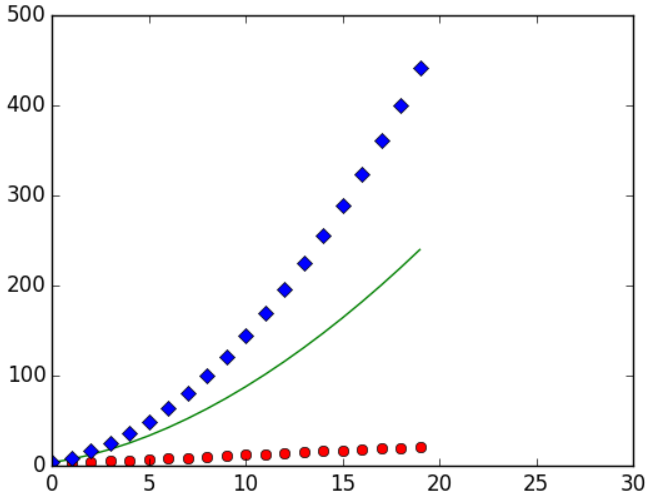

Figure 3:

```
In [8]: plt.plot(x,y,'dm')
Out[8]: [<matplotlib.lines.Line2D at 0x7fac621a4b50>]
```

lo que facilita graficar varias funciones en la misma figura aprovechando la propiedad de sobre-escritura:

```
In [9]: plt.clf()

In [10]plt.plot(x,y,'ro',x,y**1.8,'g-',x,y**2,'bD')
Out[10]:
[<matplotlib.lines.Line2D at 0x7fac61f5c2d0>,
 <matplotlib.lines.Line2D at 0x7fac61f5c510>,
 <matplotlib.lines.Line2D at 0x7fac61f5cbd0>]
```

Los valores límite para cada eje fueron elegidos automáticamente en base a los datos que hay que graficar pero pueden ser modificados utilizando la función *plt.axis()*:

```
In [11]: plt.axis([0,30,0,500])
```

cuyo argumento es una lista con *[xmin,xmax,ymin,ymax]* (figura 3).

4 Múltiples gráficos en una figura:

Es posible realizar una figura con varios gráficos distribuídos en ella utilizando la función *plt.subplot()*. El formato de esta función es:

plt.subplot(filas, columnas, nro. de gráfico)

donde número de gráfico identifica a la subfigura. Por ejemplo:

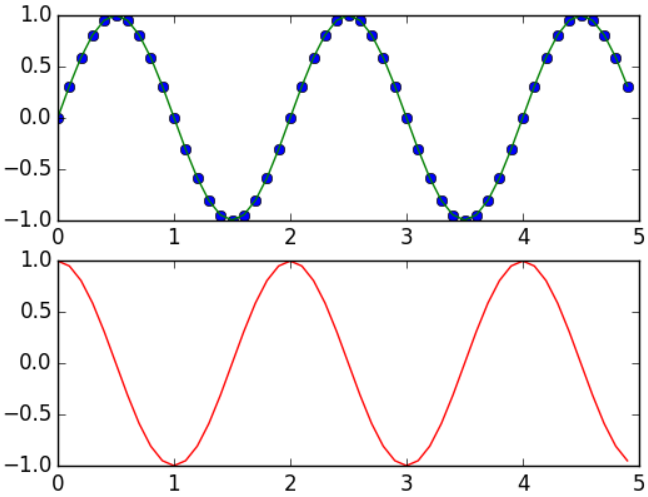

Figure 4:

```
In [12]: plt.figure(2,figsize=(6,4.5))
Out[12]: <matplotlib.figure.Figure at 0x7fda1105c410>

In [13]: plt.subplot(211)
Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x7fda1105c3d0>

In [14]: plt.plot(x,y,'ob',x,y,'r-')
Out[14]:
[<matplotlib.lines.Line2D at 0x7fda10ecca10>,
 <matplotlib.lines.Line2D at 0x7fda10eccbd0>]

In [15]: plt.subplot(212)
Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x7fda10ecc5d0>

In [16]: plt.plot(x,y2,'r-')
Out[16]: [<matplotlib.lines.Line2D at 0x7fda10e5bf90>]
```

En el caso de gráficos múltiples en la misma figura (figura 4), la función *plt.cla()* opera sólo sobre la subfigura activa, en este caso la última. Si se desea borrar la primera subfigura es necesario ponerla activa primero:

```
In [17]: plt.subplot(211)
Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x7fda1105c3d0>

In [18]: plt.cla()

In [19]: plt.plot(x,y2,'oc')
Out[19]: [<matplotlib.lines.Line2D at 0x7fda10dfef10>]
```

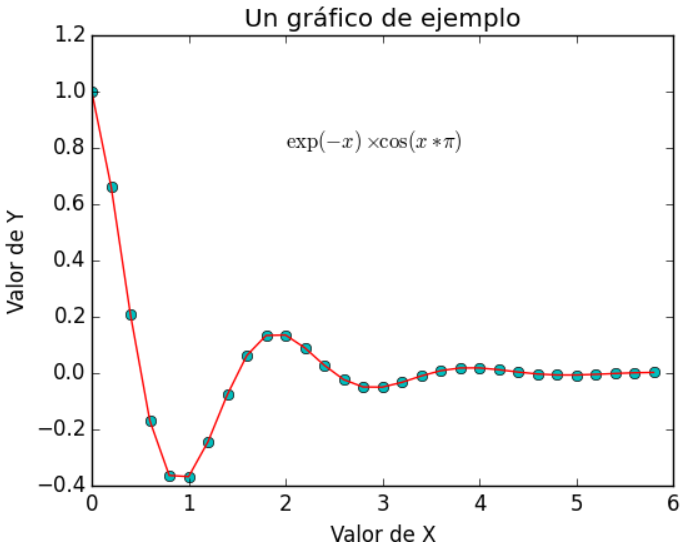

Figure 5:

5 Insertando texto en las figuras:

Para insertar texto en un lugar arbitrario de la figura se utiliza la función *plt.text()*, además de las funciones *plt.title()*, *plt.xlabel()*, y *plt.ylabel()* que insertan texto en los lugares correspondientes:

```
In [20]: plt.figure(1,figsize=(6,4.5))
Out[20]: <matplotlib.figure.Figure at 0x7fda10e60a50>

In [21]: x=np.arange(0,6,0.2)

In [22]: y=np.exp(-x)*np.cos(np.pi*x)

In [23]: plt.plot(x,y,'oc',x,y,'r-')
Out[23]:
[<matplotlib.lines.Line2D at 0x7fda113ec610>,
 <matplotlib.lines.Line2D at 0x7fda112b2810>]

In [24]: plt.xlabel("Valor de X")
Out[24]: <matplotlib.text.Text at 0x7fda0ca4c9d0>

In [25]: plt.ylabel("Valor de Y")
Out[25]: <matplotlib.text.Text at 0x7fda110ca150>

In [26]: plt.title(u"Un grfico de ejemplo")
Out[26]: <matplotlib.text.Text at 0x7fda1108ef50>

In [27]: plt.text(2.,0.8,r"$\exp(-x)\times \cos(x*\pi)$")
Out[27]: <matplotlib.text.Text at 0x7fda0c847210>
```

En la figura 5 se puede ver el resultado. Los parámetros para la función *plt.text()* son las coor-

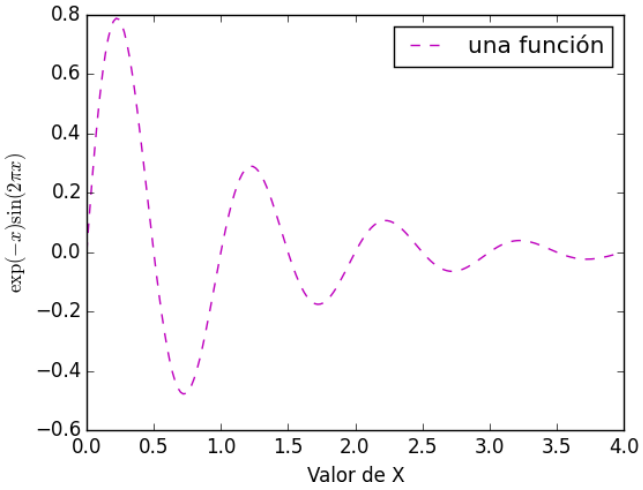

Figure 6:

denadas x e y donde se insertará el texto y el string a insertar. En el caso del título utilizamos una codificación del string en *unicode* para poner la letra acentuada y en el caso del texto insertado en la figura utilizamos "r" antes del string para informar a Python que imprima el texto tal como esta sin considerar caracteres de control o de escape. Además, es posible incluir expresiones en Tex para representar funciones matemáticas.

Una opción algo diferente para insertar texto la da la función *plt.annotate()*, que permite insertar texto e indicar con una flecha algún punto específico de la figura. Por ejemplo:

```
In [28]: plt.annotate("maximo local",xy=(1.9,0.15),xytext=(3,0.4), \
         arrowprops=dict(facecolor="red",shrink=0.05),)
Out[28]: <matplotlib.text.Annotation at 0x7fda0c87bc10>
```

donde *xy* indica el punto a donde apunta la flecha, *xytext* donde se inserta el texto, y *arrowprops* es un dictionary con las propiedades de la flecha.

Todas las funciones que manejan texto aceptan diferentes keywords, como *fontsize* o *color*. Un listado completo se puede obtener en el help.

6 Leyendas:

Es posible incluir leyendas en las figuras con la función *plt.legend()*. Para que la función incluya la leyenda es necesario que cuando se grafique con *plt.plot()* se incluya un keyword *label* con una etiqueta para identificar la figura. Por ejemplo:

```
In [141]: plt.clf()

In [142]: x=np.arange(0,4,0.02)

In [143]: y=np.exp(-x)*np.sin(2.*np.pi*x)

In [144]: plt.plot(x,y,'m--',label=u'una funcin')
```

```
Out[144]: [<matplotlib.lines.Line2D at 0x7fd9f79c69d0>]

In [145]: plt.legend()
Out[145]: <matplotlib.legend.Legend at 0x7fd9f7916210>

In [146]: plt.ylabel(r"$\exp(-x) \sin(2\pi x)$")
Out[146]: <matplotlib.text.Text at 0x7fd9f7a64810>

In [147]: plt.xlabel("Valor de X")
Out[147]: <matplotlib.text.Text at 0x7fd9f7a115d0>
```

cuyo resultado se muestra en la figura 6. Hay varias posibilidades de posicionamiento y formas de incluir la leyenda en la figura mediante diferentes parámetros y keywords, por lo que se recomienda consultar el help de la función.

7 La barra de herramientas:

Todas las ventanas de figuras tienen una barra de herramientas, que se muestra en la figura 7, y que permiten hacer ciertas operaciones sobre las figuras que contienen:

- Los tres primeros botones desde la izquierda indican *home*, *hacia atrás* y *hacia adelante* y permiten moverse por las diferentes visualizaciones que se han hecho de un gráfico. Funcionan solo si previamente se ha modificado la vista con el botón de arrastre y zoom.

- el siguiente botón es el de arrastre y zoom. Luego de clickear sobre el botón de la barra de herramientas coloque el mouse sobre algún punto de la figura y mueva el mouse presionando su botón izquierdo para arrastrar la figura. Si se repite la operación pero presionando ahora el botón derecho se modifican las escalas de los ejes comprimiendo o expandiendo la figura en una de las dos direcciones si se mueve el mouse en esa dirección. Si durante la operación se presiona la tecla "x" o "y" el efecto se reduce a el eje indicado.

- el próximo botón es el de zoom en una cierta área. Presione ese botón de la barra de herramientas y posiciones el mouse en una de las esquinas del área a aumentar. Presione el botón izquierdo del mause y arrastrelo hacia la esquina opuesta para delimitar un área de la figura. Al soltar el botón el área definida se desplegará en la figura.

- el siguiente botón de la barra de herramientas permite configurar varios parámetros de las subfiguras, como espacio entre las filas o columnas.

- el anteúltimo botón abre un diálogo para guardar la figura en disco. Se pueden guardar figuras con extensiones .png, .ps, .eps, .svg y .pdf.

- el último botón permite configurar los valores para los ejes de las figuras, cambiar la escala a logarítmica, etc.

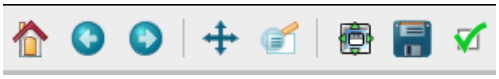

Figure 7:

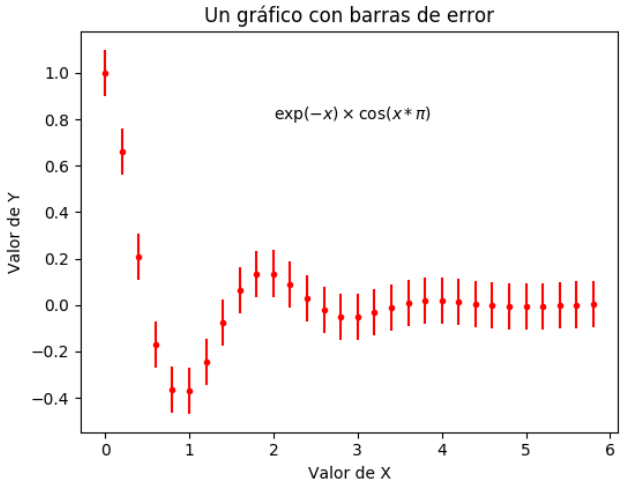

Figure 8:

8 Otros tipos de gráficos:

Existe una enorme variedad de gráficos que se pueden hacer con **matplotlib** pero no es posible mostrar aqui las diferentes opciones. Si bien a continuación daremos dos ejemplos usuales, en la página oficial (https://matplotlib.org/1.5.3/examples) existen ejemplos y scripts para toda la gama de posibilidades.

8.1 Incluyendo barras de error:

En un gráfico de datos dispersos es posible incluir barras de error en una o ambas coordenadas utilizando la funcin *plt.errorbar* (figura 8). Para todas las opciones de parámetros y keywords ver el help.

```
In [20]: plt.figure(1,figsize=(6,4.5))
Out[20]: <matplotlib.figure.Figure at 0x7fda10e60a50>

In [21]: x=np.arange(0,6,0.2)

In [22]: y=np.exp(-x)*np.cos(np.pi*x)

In [23]: yerr=np.ones(len(x))*0.1

In [24]: plt.errorbar(x,y,yerr=yerr,xerr=None,fmt='.r')
Out[24]: <Container object of 3 artists>
```

```
In [24]: plt.xlabel("Valor de X")
Out[24]: <matplotlib.text.Text at 0x7fda0ca4c9d0>

In [25]: plt.ylabel("Valor de Y")
Out[25]: <matplotlib.text.Text at 0x7fda110ca150>

In [26]: plt.title(u"Un grfico con barras de error")
Out[26]: <matplotlib.text.Text at 0x7fda1108ef50>

In [27]: plt.text(2.,0.8,r"$\exp(-x)\times \cos(x*\pi)$")
Out[27]: <matplotlib.text.Text at 0x7fda0c847210>
```

8.2 Histogramas:

Se pueden hacer histogramas con la función *plt.hist* (figura 9). Para las opciones de parámetros y keywords ver el help.

```
In [29]: plt.figure(1,figsize=(6,4.5))
Out[29]: <matplotlib.figure.Figure at 0x7fda0c8b2150>

In [30]: mu=40

In [31]: sig=7

In [32]: x=mu+sig*np.random.randn(10000)

In [33]: n,b,p=plt.hist(x,50,normed=1,facecolor='b',alpha=0.75)

In [34]: plt.xlabel("Superficie")
Out[34]: <matplotlib.text.Text at 0x7fda0c90bd50>

In [35]: plt.ylabel("Probabilidad")
Out[35]: <matplotlib.text.Text at 0x7fda0c70b5d0>

In [36]: plt.text(60,0.05,r"$\mu=40,\; \sigma=7$",fontsize=12, color="red")
Out[36]: <matplotlib.text.Text at 0x7fd9f7dc6c50>
```

8.3 Gráfico en coordenadas polares:

Se pueden hacer gráficos en coordenadas polares con la función *plt.polar* (figura 10). Para las opciones de parámetros y keywords ver el help.

```
In [37]: plt.clf()

In [38]: r=np.arange(0,1,0.002)

In [39]: th=4.*np.pi*r
```

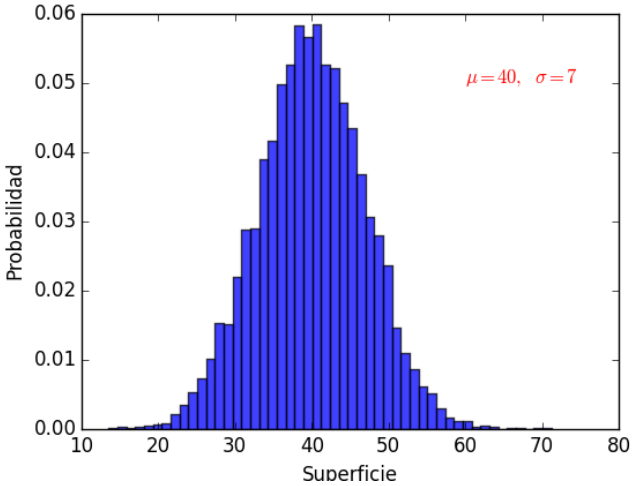

Figure 9:

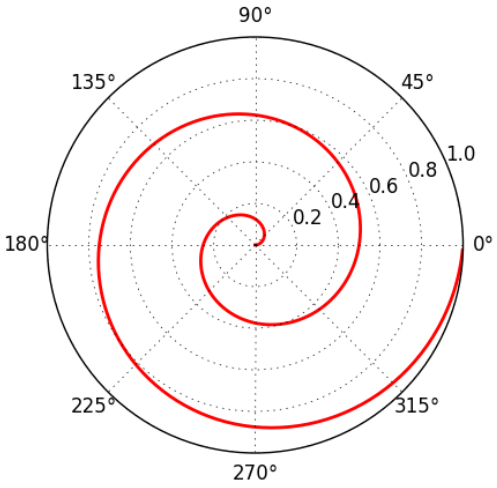

Figure 10:

```
In [40]: plt.polar(th,r,'r-',linewidth=2)
Out[40]: [<matplotlib.lines.Line2D at 0x7fd9f7d46390>]
```

8.4 Gráfico de torta:

Se pueden hacer gráficos de torta con la función *plt.pie* (figura 11). El keyword *explode* indica que porción del gráfico sobresale y en qué medida. Para todas las opciones de parámetros y keywords ver el help.

```
In [41]: lab = 'Despejado', 'Velado', 'Nublado', 'Tormenta'

In [42]: por=[60,15,15,10]

In [43]: explode=[0.1,0,0,0]

In [44]: plt.figure(1,figsize=(6,4.5))
Out[44]: <matplotlib.figure.Figure at 0x7f64d543b710>

In [45]: plt.pie(por, explode=explode, labels=lab, autopct='%1.1f%%',
         shadow=True, startangle=90)
Out[45]:
([<matplotlib.patches.Wedge at 0x7f64d523e550>,
  <matplotlib.patches.Wedge at 0x7f64d5250f50>,
  <matplotlib.patches.Wedge at 0x7f64d51e7210>,
  <matplotlib.patches.Wedge at 0x7f64d51f3510>],
 [<matplotlib.text.Text at 0x7f64d52255d0>,
  <matplotlib.text.Text at 0x7f64d51dc910>,
  <matplotlib.text.Text at 0x7f64d51e7c10>,
  <matplotlib.text.Text at 0x7f64d51f3f10>],
 [<matplotlib.text.Text at 0x7f64d5250bd0>,
  <matplotlib.text.Text at 0x7f64d51dcd90>,
  <matplotlib.text.Text at 0x7f64d51f30d0>,
  <matplotlib.text.Text at 0x7f64d52013d0>])
```

8.5 Gráfico de valores dispersos:

Se pueden hacer gráficos de valores dispersos con la función *plt.scatter* (figura 12) donde los puntos pueden representarse en diversos colores y tamaños. Para todas las opciones de parámetros y keywords ver el help.

```
In [46]: plt.figure(1,figsize=(6,4.5))
Out[46]: <matplotlib.figure.Figure at 0x7f64d52d3790>

In [47]: x=np.random.randn(100)

In [48]: x=np.random.randn(100)
```

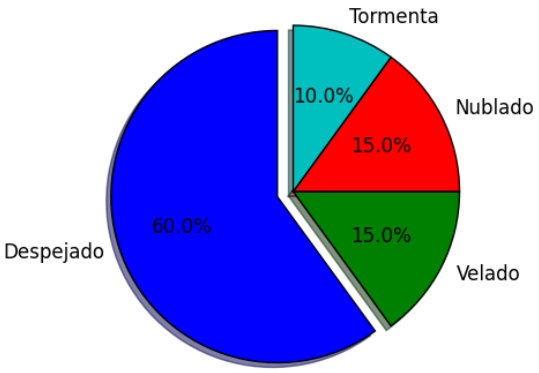

Figure 11:

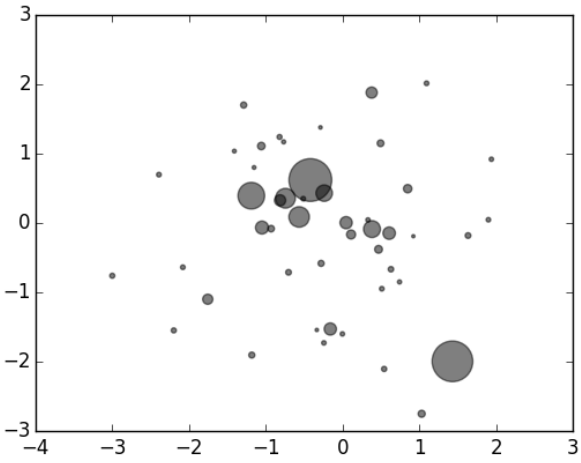

Figure 12:

```
In [49]: y=np.random.randn(100)

In [50]: rad=np.random.randn(100)

In [51]: rad=(10.*rad)/rad**2

In [52]: col=np.random.randn(100)

In [53]: col=(0.003*col+0.001)/col**2

In [54]: plt.scatter(x,y, c=col, s=rad, alpha=0.5)
Out[54]: <matplotlib.collections.PathCollection at 0x7f64cd55f650>
```

8.6 Gráfico de contornos:

Para graficar contornos se debe utilizar la función *plt.contour* (figura 13) donde los niveles de brillo a graficar se pueden identificar de diversas maneras. Para todas las opciones de parámetros y keywords ver el help.

```
In [55]: ff=ft.open('test.fits')

In [56]: img=ff[0].data

In [57]: ff.close()

In [58]: plt.clf()

In [59]: img.min()
Out[59]: 928.01996

In [60]: img.max()
Out[60]: 20955.811

In [61]: plt.contour(img,levels=[950,1050,1100,1150,1200,1250,1300,1350,1400,1450,1500])
Out[61]: <matplotlib.contour.QuadContourSet instance at 0x7f69dd590a28>
```

9 Gráficos en dos dimensiones e imágenes:

Matplotlib tiene la capacidad de graficar en dos dimensiones y, por lo tanto, también puede graficar imágenes de diferente tipo y formato. La función que permite graficar en dos dimensiones es *plt.imshow()* y el único parámetro estrictamente necesario es un array bidimensional.Por ejemplo:

```
In [62]: plt.figure(1,figsize=(6,4.5))
Out[62]: <matplotlib.figure.Figure at 0x7f64d52d3790>
```

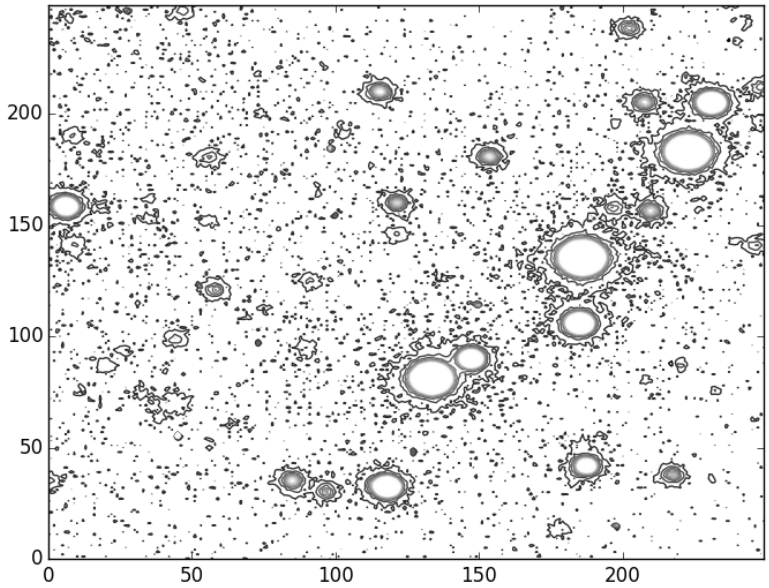

Figure 13:

```
In [63]: plt.subplot(211)
Out[63]: <matplotlib.axes._subplots.AxesSubplot at 0x7f64cce249d0>

In [64]: plt.imshow(np.random.rand(10,15))
Out[64]: <matplotlib.image.AxesImage at 0x7f64ccd44cd0>

In [65]: plt.subplot(212)
Out[65]: <matplotlib.axes._subplots.AxesSubplot at 0x7f64ccce3b90>

In [66]: plt.imshow(np.random.rand(10,15),interpolation='nearest',cmap='Reds')
Out[66]: <matplotlib.image.AxesImage at 0x7f64cd4673d0>
```

cuyo resultado se puede ver en la figura 14.

El mismo procedimiento se aplica para imágenes, con la salvedad de que primero hay que leerlas de memoria con la función *plt.imread()* y posiblemente sea necesario invertir la imagen con la función de bf numpy *np.flipud()*:

```
In [67]: plt.figure(1,figsize=(6,4.5))
Out[67]: <matplotlib.figure.Figure at 0x7f64d52d3790>

In [68]: img=plt.imread('lena.png')

In [68]: img=np.flipud(img)

In [69]: plt.imshow(img, cmap='pink')
Out[69]: <matplotlib.image.AxesImage at 0x7f64ccc96990>
```

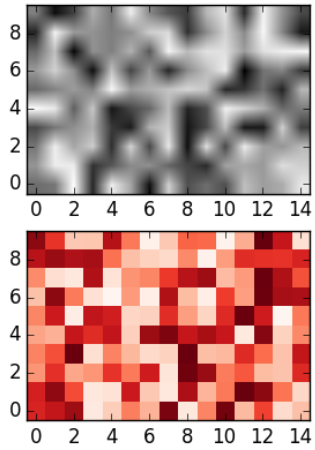

Figure 14:

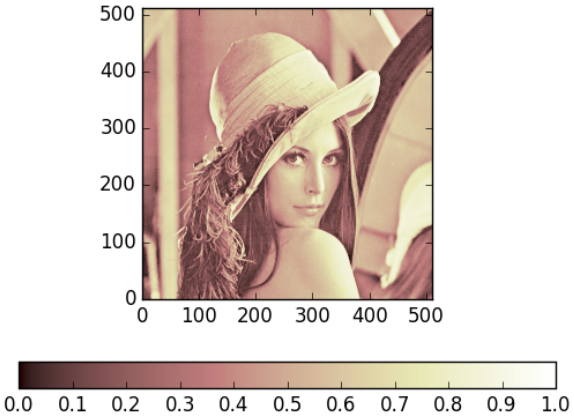

Figure 15:

```
In [70]: plt.colorbar(orientation='horizontal')
Out[70]: <matplotlib.colorbar.Colorbar instance at 0x7f64cc947f38>
```

Como se ve en la figura 15 se desea incluir la paleta de colores en la figura hay que utilizar la función *plt.colorbar()* que acepta diferentes keywords como, por ejemplo *orientation* ('horizontal' o 'vertical'), *fraction* (fracción de la figura usada para la paleta. El default es 0.15), etc.

10 Algunas referencias rápidas:

10.1 Propiedades de líneas:

Property	Description	Appearance
alpha (or a)	alpha transparency on 0-1 scale	
antialiased	True or False - use antialised rendering	Aliased Anti-aliased
color (or c)	matplotlib color arg	
linestyle (or ls)	see Line properties	
linewidth (or lw)	float, the line width in points	
solid_capstyle	Cap style for solid lines	
solid_joinstyle	Join style for solid lines	
dash_capstyle	Cap style for dashes	
dash_joinstyle	Join style for dashes	
marker	see Markers	
markeredgewidth (mew)	line width around the marker symbol	
markeredgecolor (mec)	edge color if a marker is used	
markerfacecolor (mfc)	face color if a marker is used	
markersize (ms)	size of the marker in points	

Figure 16:

10.2 Estilos de líneas y marcadores:

Figure 17:

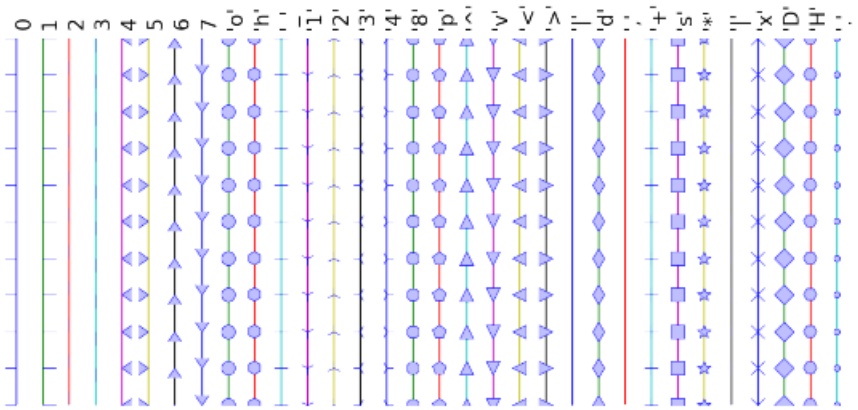

Figure 18:

10.3 Paletas de colores:

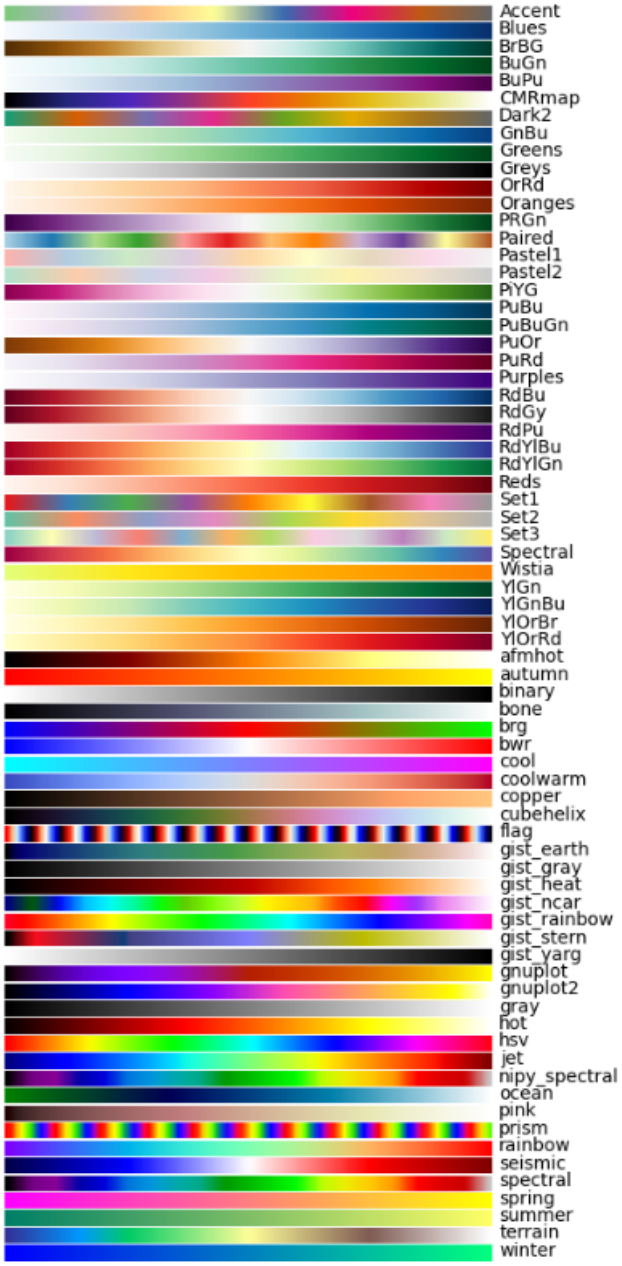

Figure 19: