2-matplotlib

October 22, 2020

1 Matplotlib

Es un paquete de graficación en 2D (aunque tiene capacidades *muy básicas*, 3D)

NOTA El material está basado en las notas de **SciPy**

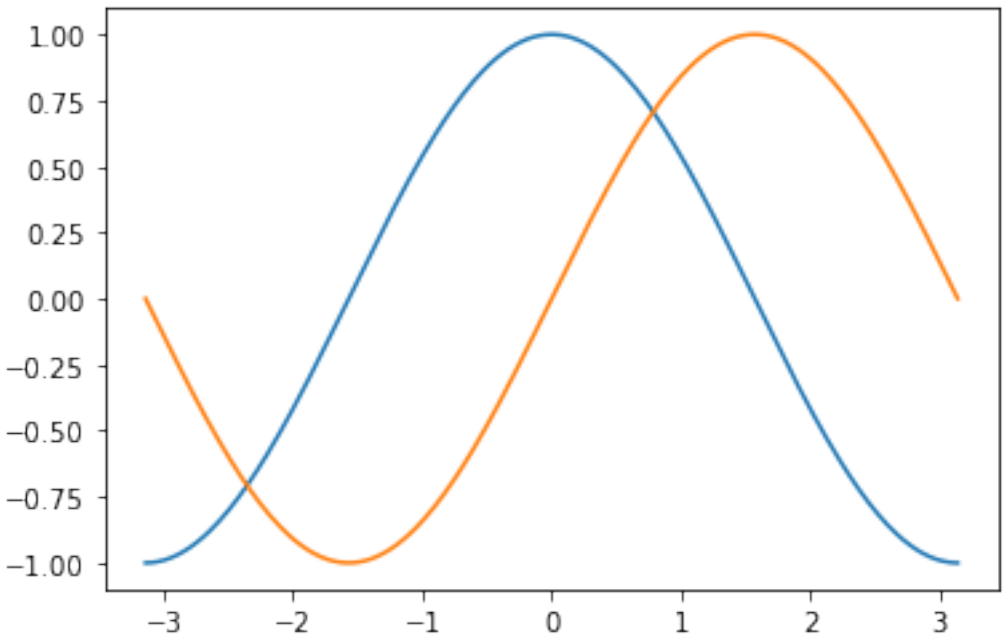
[1]:
```
%pylab inline
#Quiero que grafique todo aqui dentro de jupyter
import numpy as np
import matplotlib.pyplot as plt
```

```
Populating the interactive namespace from numpy and matplotlib
```

1.1 Básico

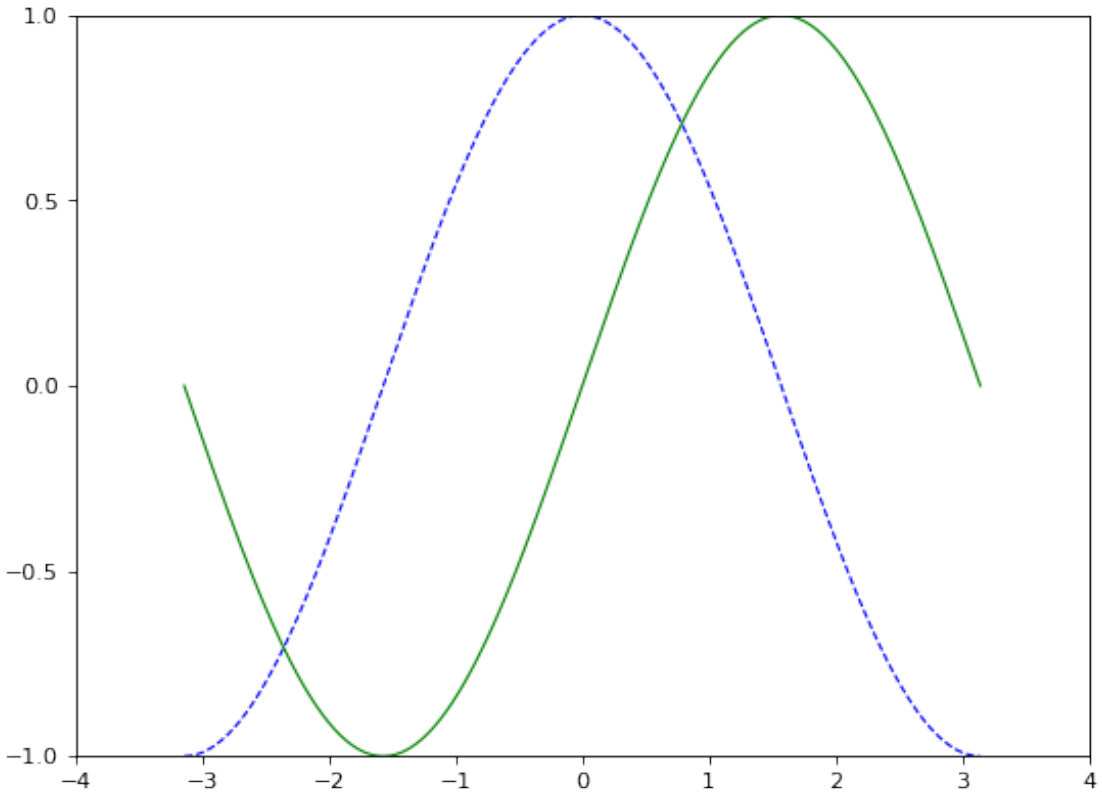
[2]:
```
X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
C, S = np.cos(X), np.sin(X)
```

[3]:
```
plt.plot(X,C);
#Eje X y eje Y
plt.plot(X,S);
```



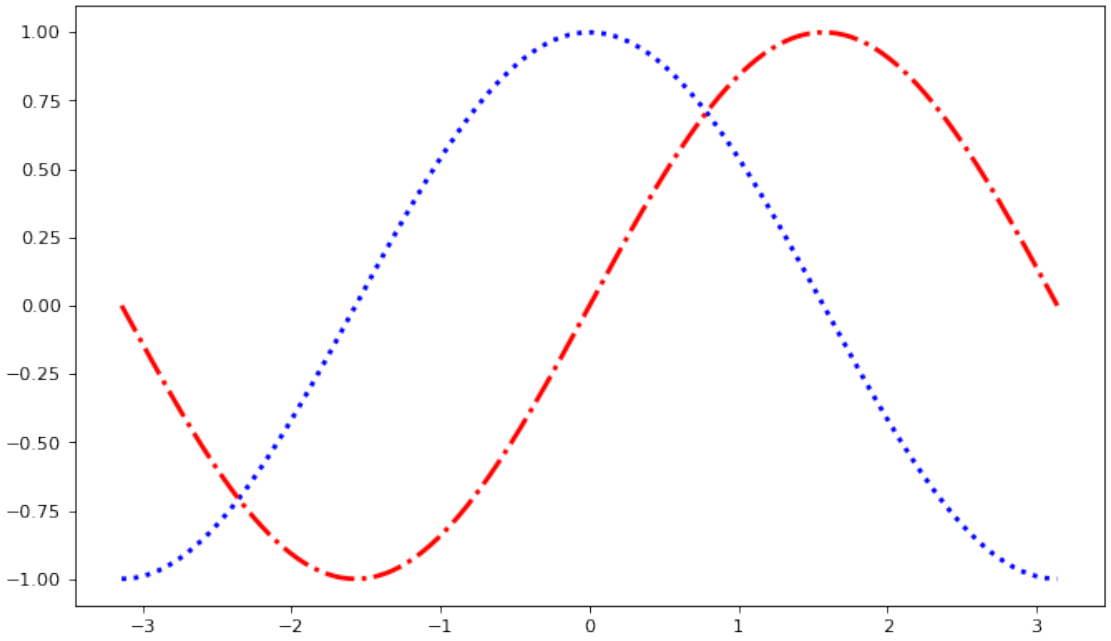
[4]:
```
# Crea una figura de 8x6 pulgadas, con 80 puntos por pulgada (dots per inch =␣
↪dpi)
plt.figure(figsize=(8, 6), dpi=80)

# Crea un subplot en una malla de 1x1.
plt.subplot(1, 1, 1)

# Dibuja el coseno en azul, con una línea contínua.
plt.plot(X, C, color="blue", linewidth=1.0, linestyle="--")

# Dibuja el seno en verde, con una línea contínua, con un ancho de 1 píxel.
plt.plot(X, S, color="green", linewidth=1.0, linestyle="-")

# Los límites de x
plt.xlim(-4.0, 4.0)

# Marcas ('ticks') del eje x (Como lo esta partiendo)
plt.xticks(np.linspace(-4, 4, 9, endpoint=True))

# Lo mismo para y, límites primero
plt.ylim(-1.0, 1.0)

# ticks
plt.yticks(np.linspace(-1, 1, 5, endpoint=True))

# Para guardar usando menor resolucion (menos dpi)
```

```
savefig("Salidas-matplotlib/exercice_2.png", dpi=72)

# Para mostrar en pantalla (si no hubiésemos usado el %pylab inline)
plt.show()
```



[5]:
```
plt.figure(figsize=(10, 6), dpi=80)
#dpi me cambia la calidad
plt.plot(X, C, color="blue", linewidth=2.5, linestyle=":")
plt.plot(X, S, color="red",  linewidth=2.5, linestyle="-.")
```

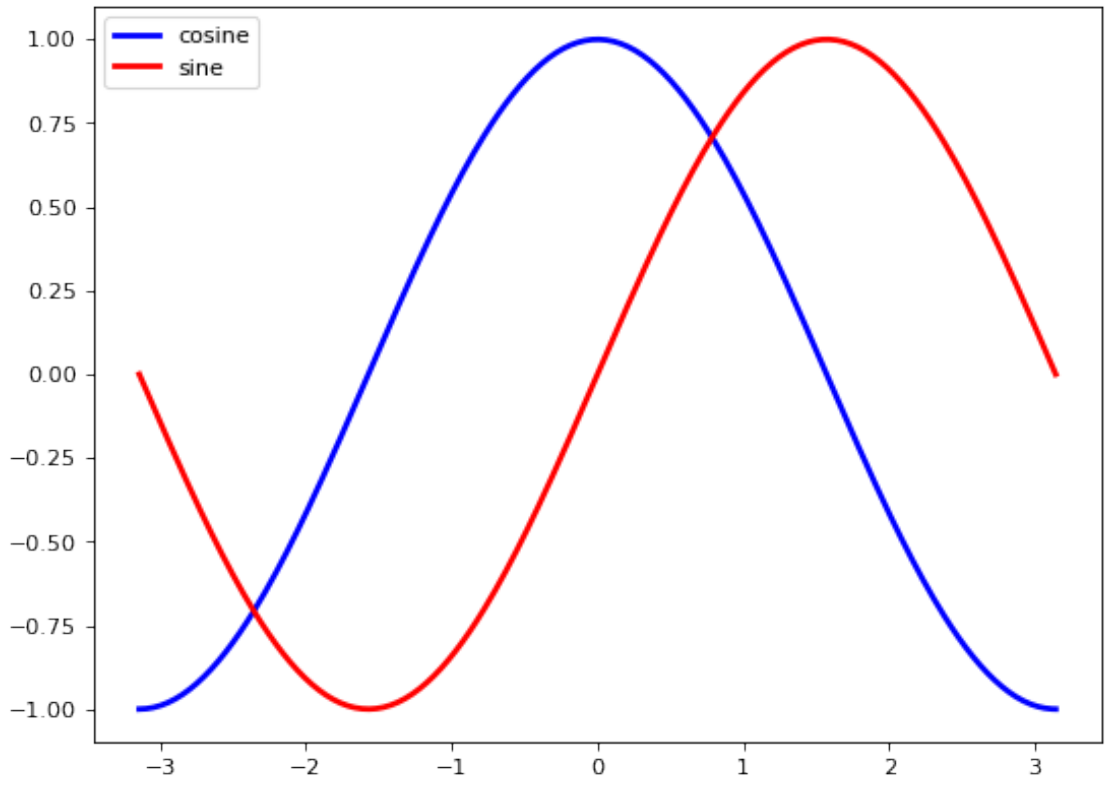
[5]: [<matplotlib.lines.Line2D at 0x7fc33122e9a0>]



1.2 Leyenda

[6]:
```
# Crea una figura de 8x6 pulgadas, con 80 puntos por pulgada (dots per inch =␣
↪dpi)
plt.figure(figsize=(8, 6), dpi=80)

# Crea un subplot en una malla de 1x1.
plt.subplot(1, 1, 1)

plt.plot(X, C, color="blue", linewidth=2.5, linestyle="-", label="cosine")
plt.plot(X, S, color="red",  linewidth=2.5, linestyle="-", label="sine")

plt.legend(loc='upper left')
```

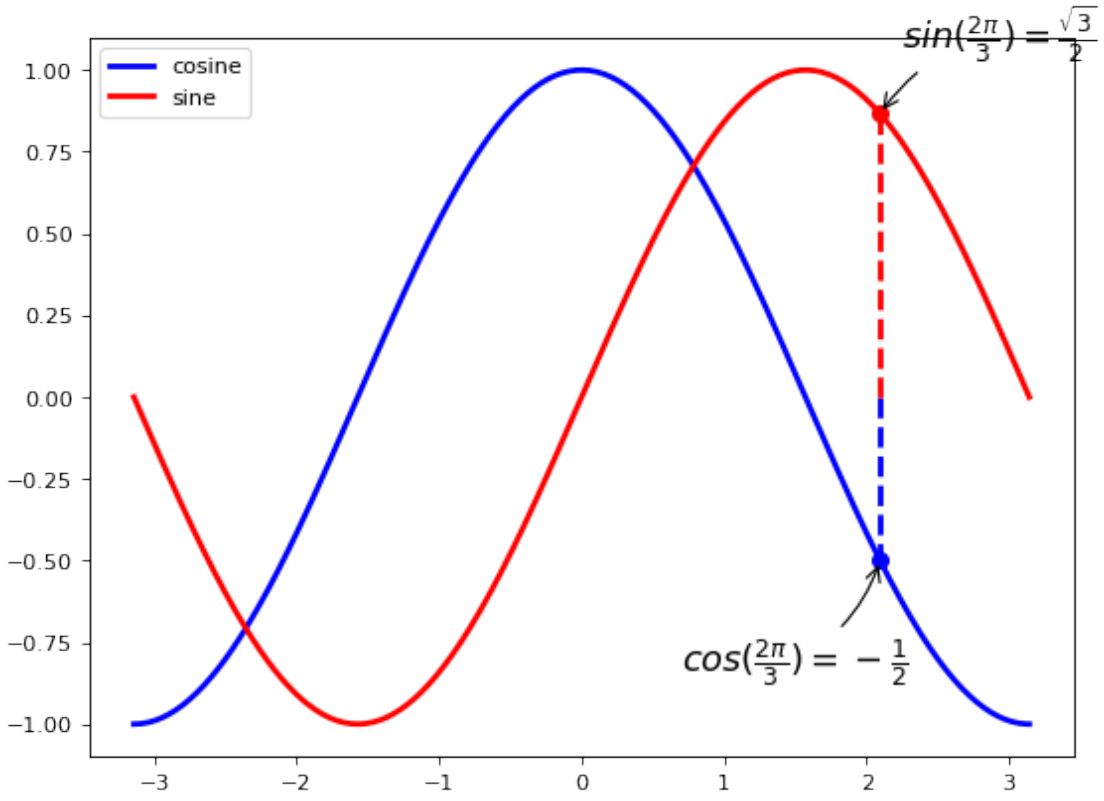
[6]:
```
<matplotlib.legend.Legend at 0x7fc33120bd90>
```



[7]:
```
# Crea una figura de 8x6 pulgadas, con 80 puntos por pulgada (dots per inch =␣
 ↪dpi)
plt.figure(figsize=(8, 6), dpi=80)

# Crea un subplot en una malla de 1x1.
plt.subplot(1, 1, 1)

t = 2 * np.pi / 3
plt.plot(X, C, color="blue", linewidth=2.5, linestyle="-", label="cosine")
plt.plot(X, S, color="red",  linewidth=2.5, linestyle="-", label="sine")

plt.legend(loc='upper left')
plt.plot([t, t], [0, np.cos(t)], color='blue', linewidth=2.5, linestyle="--")
plt.scatter([t, ], [np.cos(t), ], 50, color='blue')

#Se puede agregar LATEX
plt.annotate(r'$sin(\frac{2\pi}{3})=\frac{\sqrt{3}}{2}$',
             xy=(t, np.sin(t)), xycoords='data',
             xytext=(+10, +30), textcoords='offset points', fontsize=16,
             arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))

plt.plot([t, t],[0, np.sin(t)], color='red', linewidth=2.5, linestyle="--")
```

```
plt.scatter([t, ],[np.sin(t), ], 50, color='red')

plt.annotate(r'$cos(\frac{2\pi}{3})=-\frac{1}{2}$',
             xy=(t, np.cos(t)), xycoords='data',
             xytext=(-90, -50), textcoords='offset points', fontsize=16,
             arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))
```

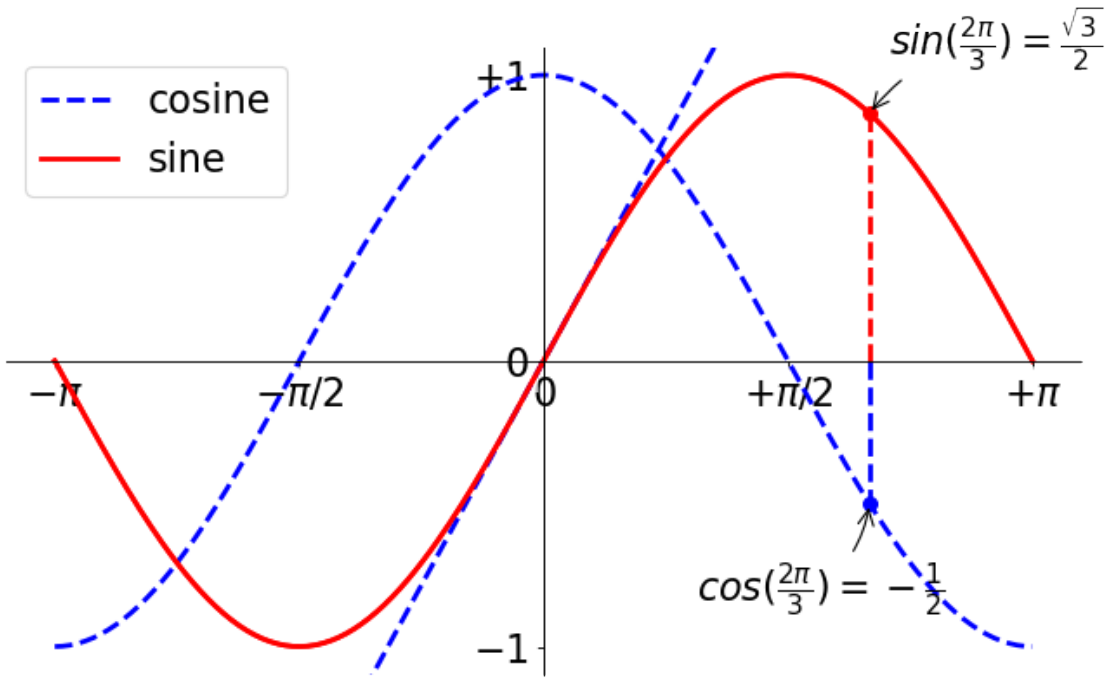
[7]: Text(-90, -50, '$cos(\\frac{2\\pi}{3})=-\\frac{1}{2}$')



1.3 Ajustando detalles

[8]:
```
plt.figure(figsize=(10, 6), dpi=80)
#-----Changed line style to "--"
plt.plot(X, C, color="blue", linewidth=2.5, linestyle="--")
#-----
plt.plot(X, S, color="red",  linewidth=2.5, linestyle="-")
plt.xlim(X.min() * 1.1, X.max() * 1.1)
plt.ylim(C.min() * 1.1, C.max() * 1.1)

ax = plt.gca()  # gca stands for 'get current axis'
ax.spines['right'].set_color('none')
```

```
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position(('data',0))
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position(('data',0))

plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi],
          [r'$-\pi$', r'$-\pi/2$', r'$0$', r'$+\pi/2$', r'$+\pi$'])

plt.yticks([-1, 0, +1],
          [r'$-1$', r'$0$', r'$+1$'])

#-----Changed line style to "--" Es la linea tangente
plt.plot(X, X, color="blue", linewidth=2.5, linestyle="--", label="cosine")
#-----
plt.plot(X, S, color="red",  linewidth=2.5, linestyle="-", label="sine")
plt.legend(loc='upper left', fontsize=20)

t = 2 * np.pi / 3
plt.plot([t, t], [0, np.cos(t)], color='blue', linewidth=2.5, linestyle="--")
plt.scatter([t, ], [np.cos(t), ], 50, color='blue')

plt.annotate(r'$sin(\frac{2\pi}{3})=\frac{\sqrt{3}}{2}$',
            xy=(t, np.sin(t)), xycoords='data',
            xytext=(+10, +30), textcoords='offset points', fontsize=20,
            arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))

plt.plot([t, t],[0, np.sin(t)], color='red', linewidth=2.5, linestyle="--")
plt.scatter([t, ],[np.sin(t), ], 50, color='red')

plt.annotate(r'$cos(\frac{2\pi}{3})=-\frac{1}{2}$',
            xy=(t, np.cos(t)), xycoords='data',
            xytext=(-90, -50), textcoords='offset points', fontsize=20,
            arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))

#-----Start new code
for label in ax.get_xticklabels() + ax.get_yticklabels():
    label.set_fontsize(20)
    label.set_bbox(dict(facecolor='white', edgecolor='None', alpha=0.65))
#-----End new code

plt.show()
```

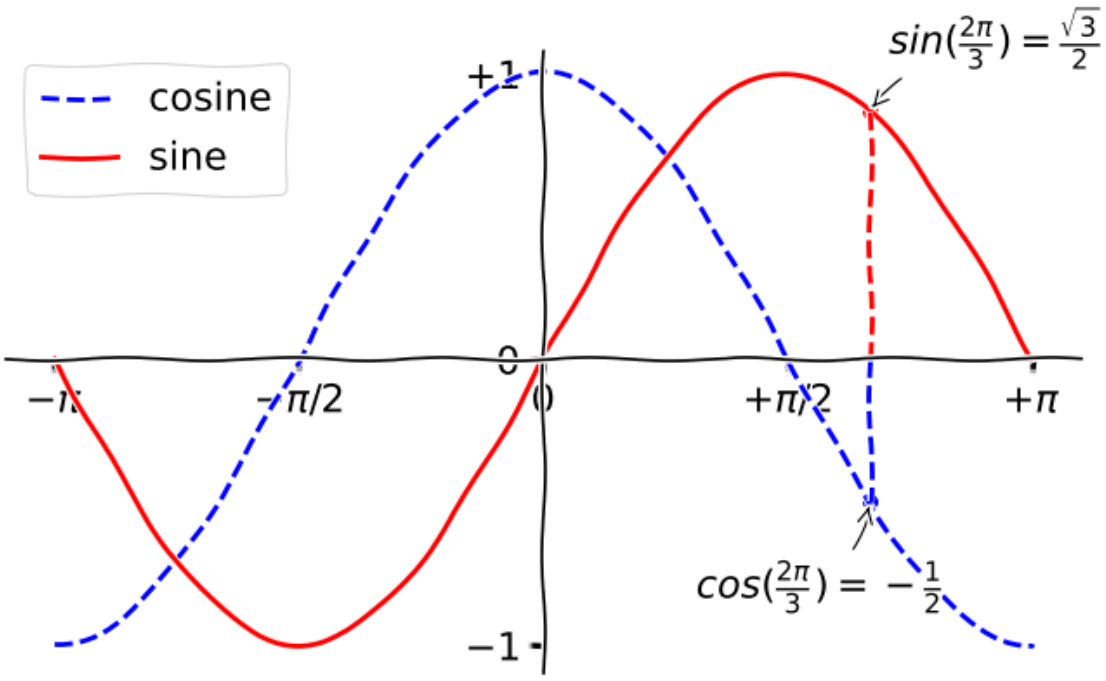


1.4 Xkcd...

[9]:
```
plt.figure(figsize=(10, 6), dpi=80)

#-----Start new code
#Hace graficos divertidos
plt.xkcd()
#-----End new code

plt.plot(X, C, color="blue", linewidth=2.5, linestyle="--")
plt.plot(X, S, color="red",  linewidth=2.5, linestyle="-")
plt.xlim(X.min() * 1.1, X.max() * 1.1)
plt.ylim(C.min() * 1.1, C.max() * 1.1)

ax = plt.gca()  # gca stands for 'get current axis'
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position(('data',0))
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position(('data',0))

plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi],
           [r'$-\pi$', r'$-\pi/2$', r'$0$', r'$+\pi/2$', r'$+\pi$'])
```

```
plt.yticks([-1, 0, +1],
          [r'$-1$', r'$0$', r'$+1$'])

plt.plot(X, C, color="blue", linewidth=2.5, linestyle="--", label="cosine")
plt.plot(X, S, color="red",  linewidth=2.5, linestyle="-", label="sine")
plt.legend(loc='upper left', fontsize=20)

t = 2 * np.pi / 3
plt.plot([t, t], [0, np.cos(t)], color='blue', linewidth=2.5, linestyle="--")
plt.scatter([t, ], [np.cos(t), ], 50, color='blue')

plt.annotate(r'$sin(\frac{2\pi}{3})=\frac{\sqrt{3}}{2}$',
            xy=(t, np.sin(t)), xycoords='data',
            xytext=(+10, +30), textcoords='offset points', fontsize=20,
            arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))

plt.plot([t, t],[0, np.sin(t)], color='red', linewidth=2.5, linestyle="--")
plt.scatter([t, ],[np.sin(t), ], 50, color='red')

plt.annotate(r'$cos(\frac{2\pi}{3})=-\frac{1}{2}$',
            xy=(t, np.cos(t)), xycoords='data',
            xytext=(-90, -50), textcoords='offset points', fontsize=20,
            arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))

for label in ax.get_xticklabels() + ax.get_yticklabels():
    label.set_fontsize(20)
    label.set_bbox(dict(facecolor='white', edgecolor='None', alpha=0.65))

plt.show()
```

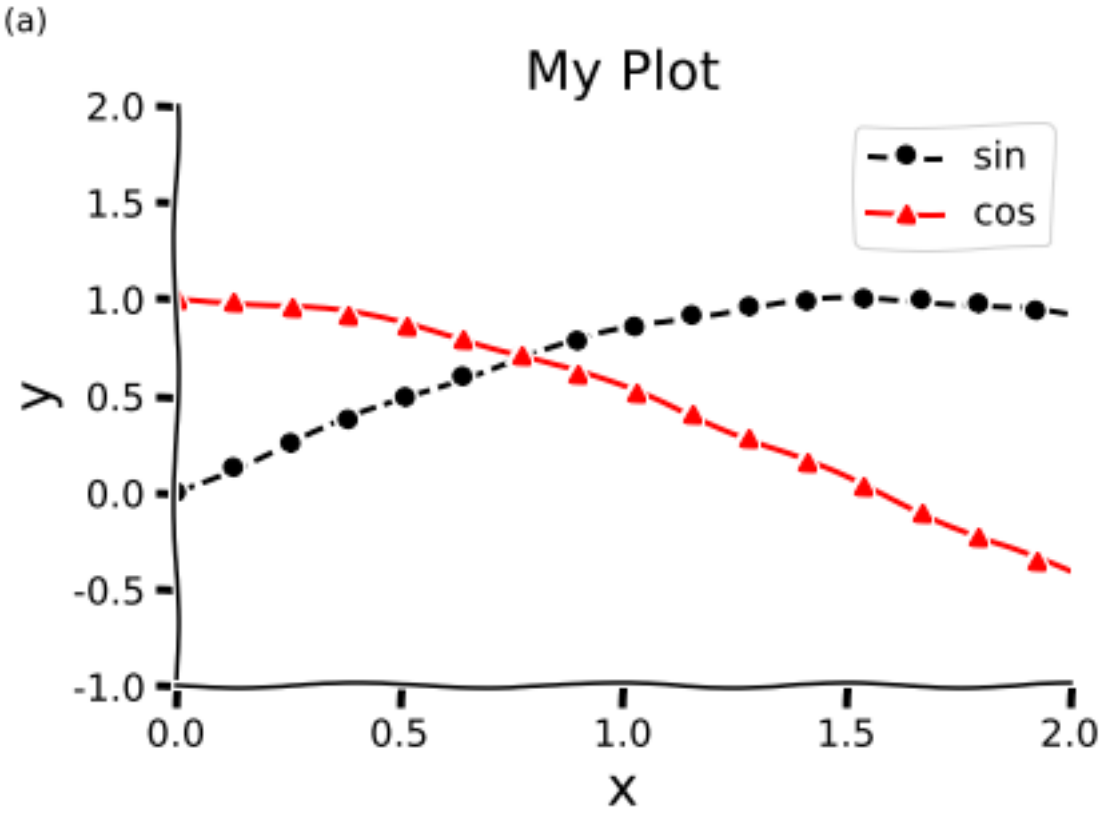
```
findfont: Font family ['xkcd', 'xkcd Script', 'Humor Sans', 'Comic Neue', 'Comic
Sans MS'] not found. Falling back to DejaVu Sans.
findfont: Font family ['xkcd', 'xkcd Script', 'Humor Sans', 'Comic Neue', 'Comic
Sans MS'] not found. Falling back to DejaVu Sans.
```



[10]:
```
# Make some data to plot
x = np.linspace(0, 2*np.pi)
y1 = np.sin(x)
y2 = np.cos(x)

# First, create an empty figure with 1 subplot
fig, ax1 = plt.subplots(1, 1)

# Add title and labels
ax1.set_title('My Plot',fontsize=20)
ax1.set_xlabel('x',fontsize=20)
ax1.set_ylabel('y',fontsize=20)

# Change axis limits
ax1.set_xlim([0,2])
ax1.set_ylim([-1, 2])

# Add the lines, changing their color, style, and marker
ax1.plot(x, y1, 'k--o', label='sin') # Black line, dashed, with 'o' markers
ax1.plot(x, y2, 'r-^', label='cos') # Red line, solid, with triangle-up markers

# Adjust tick marks and get rid of 'box'
ax1.tick_params(direction='out', top=False, right=False) # Turn ticks out
ax1.spines['top'].set_visible(False) # Get rid of top axis line
ax1.spines['right'].set_visible(False) #  Get rid of bottom axis line
```

```
# Add subplot letter
ax1.annotate('(a)', (0.01, 0.96), size=12, xycoords='figure fraction')

# Add legend
ax1.legend()

# Finally, save the figure as a png file
fig.savefig('Salidas-matplotlib/myfig-formatted.png')
```

```
findfont: Font family ['xkcd', 'xkcd Script', 'Humor Sans', 'Comic Neue', 'Comic
Sans MS'] not found. Falling back to DejaVu Sans.
findfont: Font family ['xkcd', 'xkcd Script', 'Humor Sans', 'Comic Neue', 'Comic
Sans MS'] not found. Falling back to DejaVu Sans.
```



1.5 Reiniciar

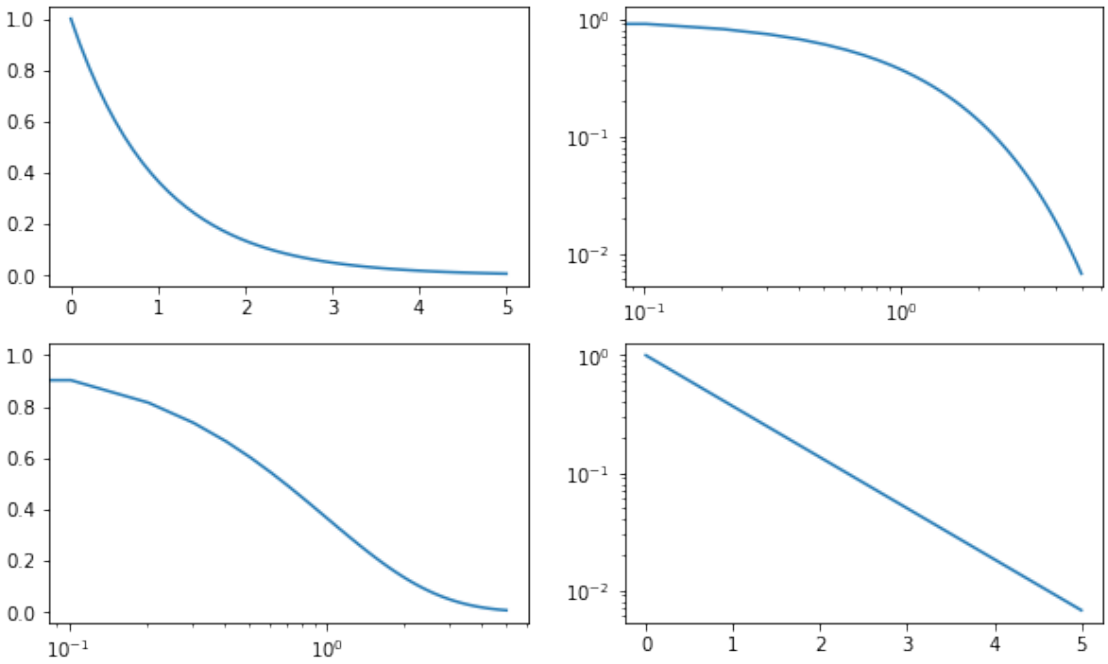
[11]:
```
plt.rcdefaults()
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

Ejercicio Ejecuta el código de la última gráfica ¿Qué sucede?

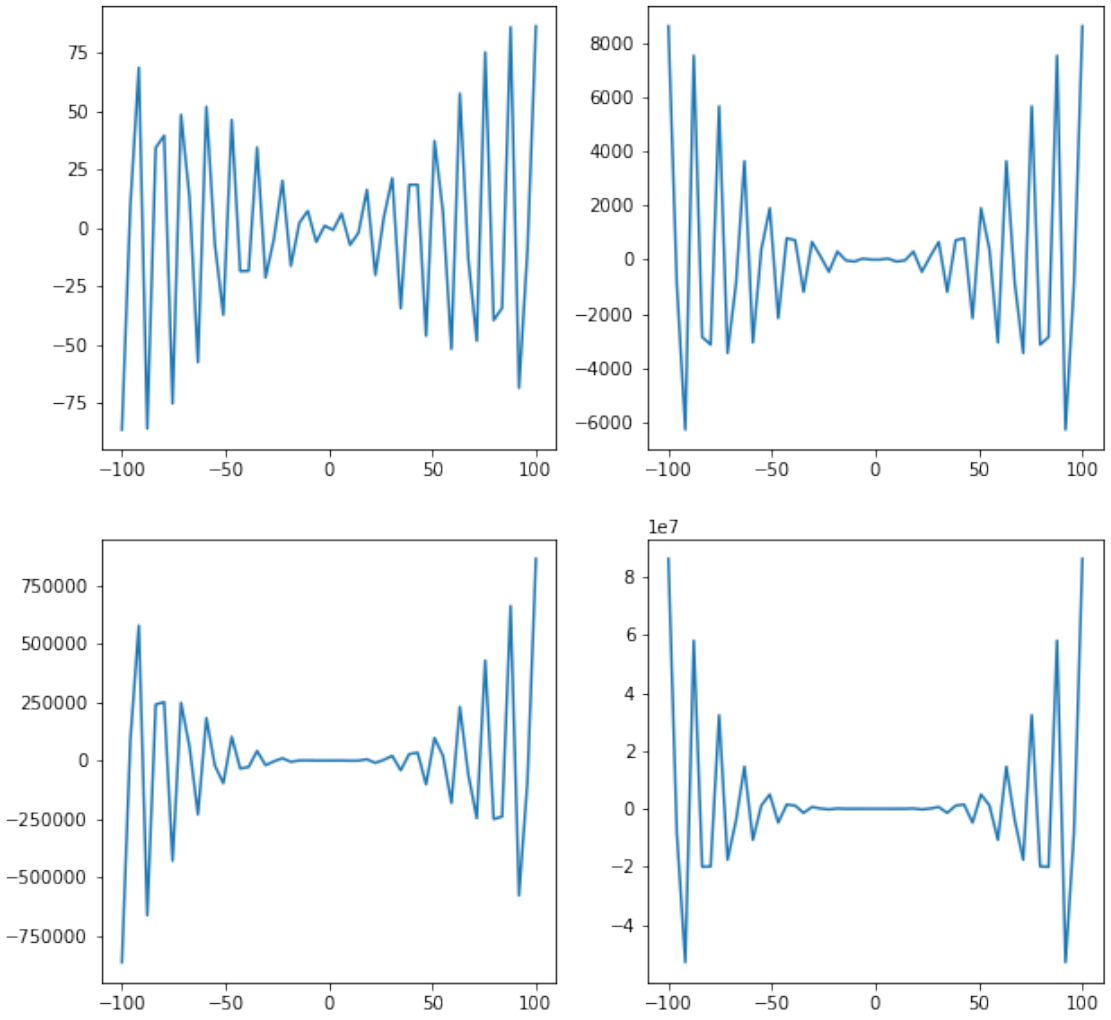
1.6 Subplots

[12]:
```
x = np.linspace(0., 5.)
y = np.exp(-x)

# Una figura con cuatro subgráficas
fig, ax = plt.subplots(2,2, figsize=(10,6))

# Dibuja cada eje con diferentes escalas
#[0,0] son las coordenadas
ax[0,0].plot(x,y) # Normal
ax[0,1].loglog(x,y) #Log en x, Log en y
ax[1,0].semilogx(x,y) # Sólo log en x
ax[1,1].semilogy(x,y); # Sólo log en y
```



[13]:
```
x = np.linspace(-100., 100.)
y1 = np.cos(x)

# Una figura con cuatro subgráficas
fig, ax = plt.subplots(2,2, figsize=(10,10))

# Dibuja cada eje con diferentes escalas
#[0,0] son las coordenadas
```

```
ax[0,0].plot(x,x*y1)
ax[0,1].plot(x,x**2*y1)
ax[1,0].plot(x,x**3*y1)
ax[1,1].plot(x,x**4*y1);
```

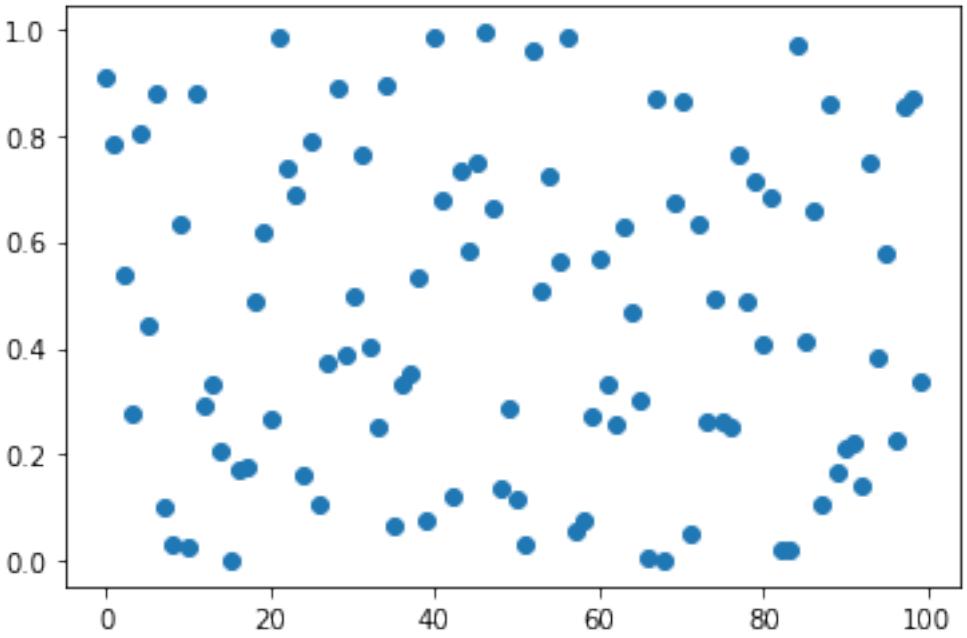


Ejercicio Crea una figura, en la que muestres en un arreglo de 2×2 los *subplots* de $x \cos x$, $x^2 \cos x$, $x^3 \cos x$ y $x^4 \cos x$. **Ejercicio** Repite las instrucciones pero ahora dibujalas todas en una misma gráfica, utilizando diferentes colores. **NOTA**: En ambas usa calidad profesional

1.7 Otros tipos de gráficas

1.7.1 Scatterplot

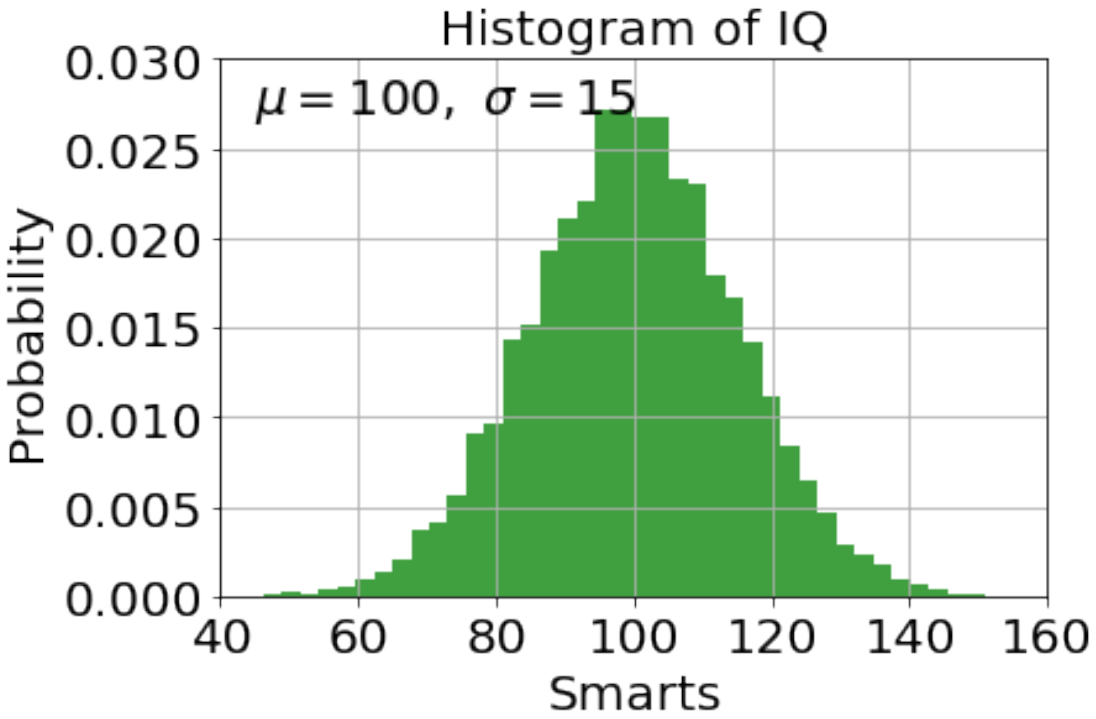
[14]:
```
# Inventamos datos al azar
x = np.arange(0, 100)
y = np.random.rand(100)
plt.scatter(x,y);
```



1.8 Histograma

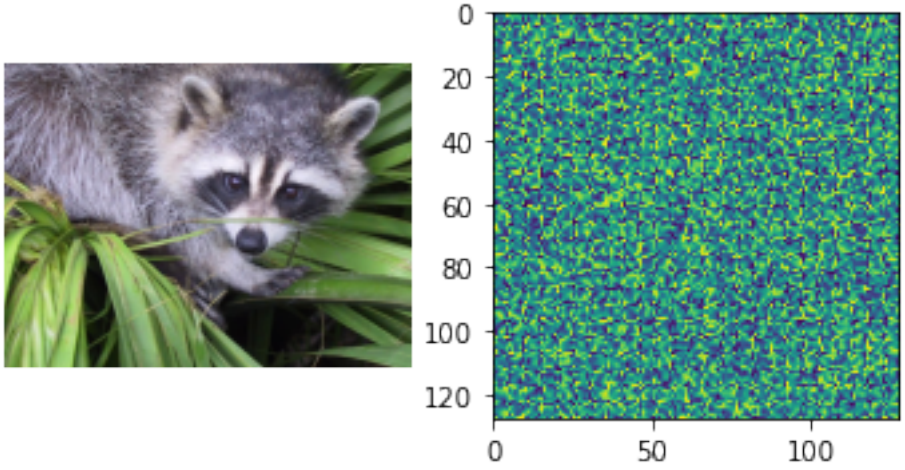
[15]:
```
#Estamos viendo la distribucion de una variable
mu, sigma = 100, 15
x = mu + sigma * np.random.randn(10000)

# Histograma de los datos
n, bins, patches = plt.hist(x, 50, density=True, facecolor='g', alpha=0.75)

plt.xlabel('Smarts',fontsize=20)
plt.ylabel('Probability',fontsize=20)
plt.title('Histogram of IQ',fontsize=20)

# Texto en la posición pedida
plt.text(45, .027, r'$\mu=100,\ \sigma=15$', fontsize=20)
plt.axis([40, 160, 0, 0.03])
plt.xticks(fontsize=20)
```

```
plt.yticks(fontsize=20)
plt.grid();
```



1.9 Imágenes y matrices

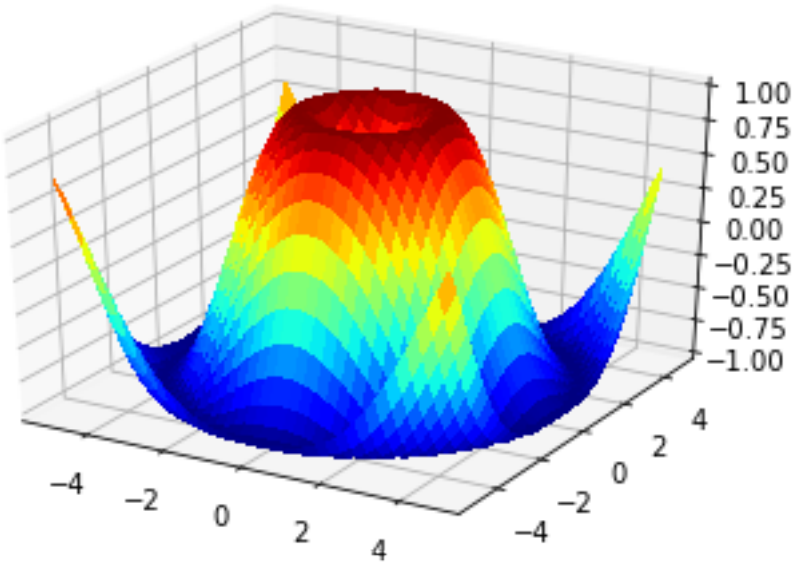
[16]:
```
# Una imagen y un arreglo al azar
#from scipy import misc
import scipy.misc
img1 = scipy.misc.face()
img2 = np.random.rand(128, 128)

# Realizamos la figura
fig, ax = plt.subplots(1, 2)
ax[0].imshow(img1)
ax[1].imshow(img2)

ax[0].set_axis_off()  # Quitamos los ejes de la primera figura (es una imagen)
```



1.10 Graficar en 3D (Básico)

[17]:
```
from mpl_toolkits.mplot3d import Axes3D
```

[18]:
```
from mpl_toolkits.mplot3d.axes3d import Axes3D
from matplotlib import cm

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1, projection='3d')
X = np.arange(-5, 5, 0.25)
Y = np.arange(-5, 5, 0.25)

X, Y = np.meshgrid(X, Y) #Eso es un producto cruz entre 2 vectores
R = np.sqrt(X**2 + Y**2)
Z = np.sin(R)
surf = ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap=cm.jet,
        linewidth=0, antialiased=False)
ax.set_zlim3d(-1.01, 1.01);
```



Un ejemplo de Electrostática: Podemos utilizar gráficos 3D para visualizar el potencial electrico de un grupo de cargas. El caso particular del dipolo eléctrico es simplemente la suma del postencial de dos cargas separadas por una distancia. Considera un par de cargas puntuales $\pm q$ localizadas en $x = 0$ y $y = \pm d$.

El potencial será entonces:

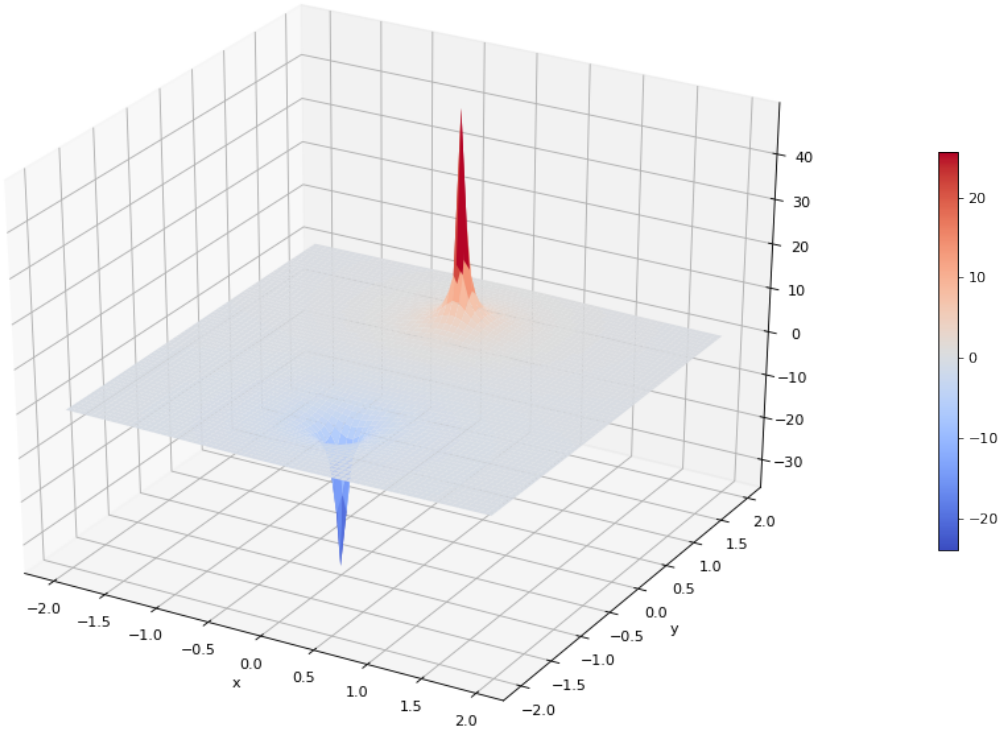
$$V(x, y) = V_1 + V_2 = \frac{k(q)}{\sqrt{x^2 + (y - d)^2}} + \frac{k(-q)}{\sqrt{x^2 + (y + d)^2}}$$

Donde $k = 1/4\pi\epsilon_0$. Para ver el potencial, da los valores como k=1, q=1, and d=1.

Primero, haremos el grafico de la función $V(x, y)$. Las funciones 2D son más complicadas que las funciones 1D. Se necesita un arreglo de bidimensional para (x, y) de puntos para cubrir el plano. La función `meshgrid` crea los arreglos x y y.

[19]:
```
from mpl_toolkits.mplot3d import Axes3D

# Hacemos una malla x-y entre -2<x<2 and -2<y<2 en steps de 0.06
dx = 0.06
dy = 0.06
x = np.arange(-2, 2, dx)
y = np.arange(-2, 2, dy)
x, y = np.meshgrid(x, y)

# La fucnión V(x,y)
k = 1; q = 1; d = 1
V=(k*q/(x**2+(y-d)**2)**(1/2.)) - (k*q/(x**2+(y+d)**2)**(1/2.))
```

```
# Para usar los gráficos 3D en matplotlib, debemos crear una instancia de la␣
 ↪clase Axes3D.
fig = plt.figure(figsize=(15, 10), dpi=80)

# Usemos el método gca ("get current axis") en la figura para obtener la␣
 ↪proyección 3D
ax = fig.gca(projection='3d')
p = ax.plot_surface(x, y, V, rstride=1, cstride=1, linewidth=0,cmap=cm.coolwarm)

#¿Qué significan los argumentos?
#help(Axes3D.plot_surface).
cb = fig.colorbar(p, shrink=0.5)

plt.xlabel("x")
plt.ylabel("y")

plt.savefig("Salidas-matplotlib/Dipole.png")
```



1.11 Gráficas de contorno

[20]:
```
# Representa gráficos 3D pero en contornos...
levels = array([-8, -4, -2, -1, -0.5, -0.25, 0, 0.25, 0.5, 1, 2, 4, 8])
plt.contour(x,y,V,levels ,linewidths=4,cmap=cm.coolwarm);
```