

Programación para Física y Astronomía

Departamento de Física.

Profesores Felipe Moreno / Fabián Gómez-Villafañe

Segundo Semestre / 2022

Universidad Andrés Bello



Gráficos

- Gráficos de líneas

- Gráficos de Dispersión

- Gráficos de Densidad

Semillas y generación de números al azar

- Generación de números al azar

- Semillas

- Usando números al azar

Gráficos

Hasta el momento las herramientas de programación que conocemos nos permiten imprimir números y string en pantalla. Sin embargo, un aspecto muy relevante de la programación es poder visualizar los resultados de un programa mediante algún tipo de gráfico o imagen.

“If I can't picture it, I can't understand it.”

— — Albert Einstein

Python tiene a disposición distintos paquetes con funcionalidades gráficas, nosotros nos enfocaremos en *matplotlib*. Para poder instalar ésta librería es necesario contar con permisos de administrador.

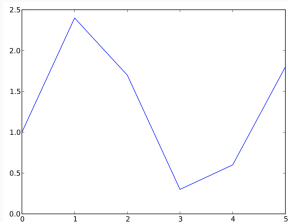
Instrucciones de instalación en Ubuntu

```
sudo apt-get install python3-matplotlib
```

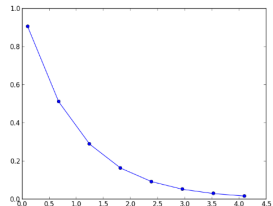
Gráficos de líneas

Nos concentraremos en tres tipos de gráficos más comunes: **gráficos de líneas, dispersión y densidad**. Los cuales se construyen usando *plot* y *show* del subpaquete *matplotlib.pyplot* como en los siguientes ejemplo:

```
1 import matplotlib.pyplot as plt
2 y = [ 1.0, 2.4, 1.7, 0.3, 0.6, 1.8 ]
3 plt.plot(y)
4 plt.show()
```



```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 x = np.arange(0.1, 4, 0.5)
4 y = np.exp(-x)
5 plt.plot(x, y, '-o')
6 plt.show()
```



Gráficos de líneas y más

- La función ***plot*** es la encargada de crear el gráfico en la memoria del computador dependiendo de los argumentos pasados a dicha función:

```
1 plot(x, y)           #plot x e y usando estilos por defecto
2 plot(x, y, 'bo')     #plot x e y círculos azules
3 plot(y)              #plot y usando x como un arreglo 0...N-1
4 plot(y, 'r+')        #idem, usando '+' rojo
```

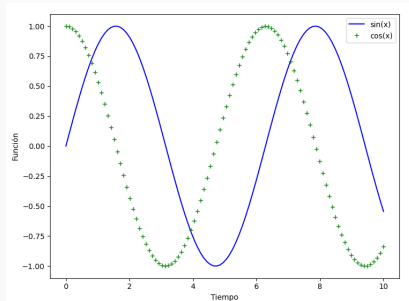
Para más detalle de todas las opciones revisar la página web:
<https://matplotlib.org/stable/>

- La función ***show*** es la encargada de desplegar el gráfico en pantalla.

Personalizando Gráficos

Función	Descripción
xlabel(), ylabel()	Etiquetas en los ejes
xlim(), ylim()	Límites de los ejes
xticks(), yticks()	Localización y etiquetas de los marcadores en los ejes
xscale(), yscale()	Escala de los ejes
legend()	Muestra la leyenda
subplot()	Crea múltiple gráficos en una sola figura
etc...	Muchas más funciones

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 x = np.linspace(0,10,100)
4 S = np.sin(x)
5 C = np.cos(x)
6 plt.plot(t,S,"b-",label="sin(x)")
7 plt.plot(t,C,"g+",label="cos(x)")
8 plt.ylim(-1.1,1.1)
9 plt.xlabel("Tiempo")
10 plt.ylabel("Función")
11 plt.legend()
12 plt.show()
```



Aplicando lo aprendido

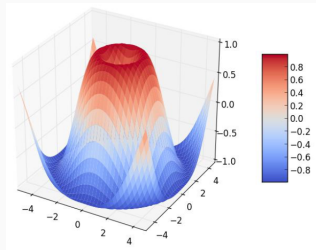
```
1 # Graficando desde archivo de datos
2 import matplotlib.pyplot as plt
3 import numpy as np
4 datos=np.loadtxt("values.txt",float)
5 plt.plot(datos[:,0],datos[:,1])
6 plt.show()
```

```
1 # Graficando múltiples paneles
2 import numpy as np
3 import matplotlib.pyplot as plt
4 x1 = np.linspace(0.0, 5.0)
5 x2 = np.linspace(0.0, 2.0)
6 y1 = np.cos(2 * np.pi * x1) * np.exp(-x1)
7 y2 = np.cos(2 * np.pi * x2)
8 plt.subplot(2, 1, 1)
9 plt.plot(x1, y1, 'yo-')
10 plt.title('A tale of 2 subplots')
11 plt.ylabel('Oscilación Amortiguada')
12 plt.subplot(2, 1, 2)
13 plt.plot(x2, y2, 'r.-')
14 plt.xlabel('time (s)')
15 plt.ylabel('Oscilación')
16 plt.show()
```

A modo de actividad, ejecute estos códigos en su terminal.

También es posible crear gráficos en tres dimensiones para poder visualizar superficies o volúmenes.

```
1 from mpl_toolkits.mplot3d import Axes3D
2 import matplotlib.pyplot as plt
3 from matplotlib import cm
4 import numpy as np
5
6 fig = plt.figure()
7 ax = fig.gca(projection='3d') # Necesita Axes3D
8 X = np.arange(-5, 5, 0.25)
9 Y = np.arange(-5, 5, 0.25)
10 X, Y = np.meshgrid(X, Y)
11 R = np.sqrt(X**2 + Y**2)
12 Z = np.sin(R)
13 surf = ax.plot_surface(X, Y, Z,
14 rstride=1, cstride=1, cmap=cm.coolwarm,
15 linewidth=0)
16 ax.set_zlim(-1.01, 1.01)
17
18 fig.colorbar(surf, shrink=0.5, aspect=5)
19
20 plt.show()
```



- Son útiles para representar dos variables independientes. Un ejemplo clásico es la temperatura y luminosidad (llamada también magnitud) de las estrellas.
- El gráfico de dispersión me permite visualizar cómo estas dos cantidades están relacionadas.
- Podemos crear gráficos de dispersión usando dos métodos:
 1. Usando gráficos ordinarios con pequeños puntos:
`plot(x,y,"k.")`.
 2. Usando la función `scatter(x,y)`.

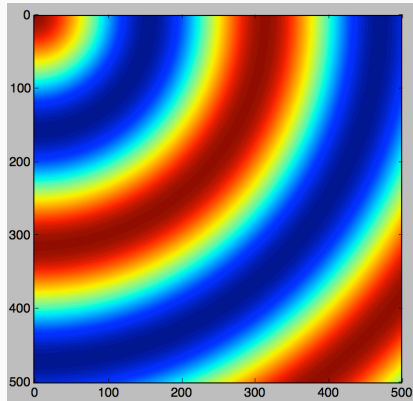
Temperatura y luminosidad de un conjunto de estrellas.

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 data = np.loadtxt("stars.txt",float)
4 x=data[:,0]
5 y=data[:,1]
6 plt.scatter(x,y)
7 plt.xlabel("Temperature")
8 plt.ylabel("Magnitude")
9 plt.xlim(0,13000)
10 plt.ylim(-5,20)
11 plt.show()
```

- Gráficos bi-dimensionales donde el color o luminosidad es usado para indicar otro valor (se agrega una 3ra dimensión).
- En Python, los gráficos de densidad se obtienen usando la función *imshow* del paquete pylab.

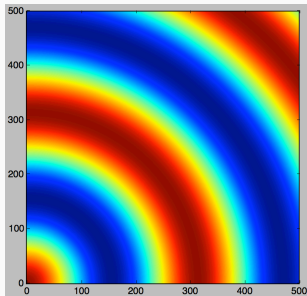
```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 data = np.loadtxt("circular.txt", float)
4 plt.imshow(data)
5 plt.show()
```

Resultado del programa anterior:



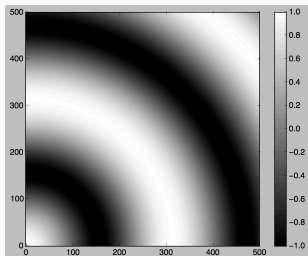
Variaciones de aspecto en gráficos de densidad

- Origen del arreglo en la esquina izquierda inferior:
`imshow(data, origin="lower")`



Variaciones de aspecto en gráficos de densidad

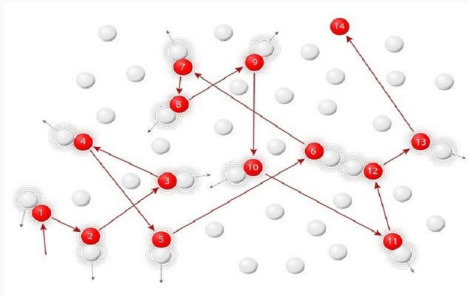
- Escala de color y barra con legenda de color:
jet(defecto), *gray*(escala de grises), *hot*(otra escala de temperatura), etc. Seguido de: *colorbar*().



Semillas y generación de números al azar

Números al azar

Existen procesos en física que ocurren al azar, como el decaimiento radiactivo de un átomo. Sin embargo, existen otros que no son realmente al azar, como el movimiento Browniano, pero que pueden ser modelados adecuadamente asumiendo un proceso al azar.



Generación de números al azar

- Para imitar los procesos azarosos, necesitamos números al azar. Técnicamente, usamos pseudo-azar debido a que son generados por una fórmula determinista. Por ejemplo:

Pseudo-Random

$$s_{i+1} = (a \times s_i + c) \bmod M$$

$$i = 0, 1, 2, 3 \dots N$$

donde a , c y M son constantes y s_0 es el valor inicial o *semilla*.

```
1 import matplotlib.pyplot as plt
2 N, s = 100, 1
3 a, c, m = 1664525, 1013904223, 4294967296
4 results = []
5 for i in range(N):
6     s = (a*s+c)%m
7     results.append(s)
8 plt.plot(results, "o")
9 plt.show()
```

Generación de números al azar

- El programa anterior es un generador de números al azar con congruencia lineal:
 1. Uno de los generadores de números al azar más famosos.
 2. Obviamente no azaroso, es determinista.
 3. Los números son siempre > 0 o cero, y menores que M . El rango de valores es $[0, M - 1)$. Para generar otro rango es necesario escalar.
 4. Las constantes a , c y M fueron escogidas con cuidado. Estos valores se han probado y se sabe que funcionan bien.
 5. Para valores particulares de a , c y m , y escogiendo un valor de partida, s_0 , distinto se pueden generar secuencias de números al azar completamente diferentes.
- Este generador es adecuado sólo para sencillos cálculos en física.
- Principal defecto: existe una correlación entre los valores de números sucesivos (los números al azar no presentan ningún tipo de correlación.)

Generador de números al azar en Python

- El generador por defecto en Python es el algoritmo de *Mersenne Twister*.
- El paquete que implementa dicho algoritmo es *random*.

Función	Descripción
<code>random()</code>	número <i>float</i> uniformemente distribuido $\in [0, 1)$.
<code>randrange(n)</code>	número <i>int</i> uniformemente distribuido $\in [0, n - 1)$.
<code>randrange(m, n)</code>	número <i>int</i> uniformemente distribuido $\in [m, n - 1)$.
<code>randrange(m, n, k)</code>	número <i>int</i> uniformemente distribuido $\in [m, n - 1)$ en pasos de <i>k</i> .

- Todas estas funciones retornan un nuevo número al azar cada vez que se invoca la función.

Semillas y números al azar

- Del mismo modo que el generador de números al azar con congruencia lineal, las funciones del paquete *random* pueden usar una semilla para comenzar una secuencia.
- Las semillas para el generador *Mersenne Twister* se obtienen usando la función *seed()* del paquete *random*:

```
1 from random import randrange, seed
2 seed(42)
3 for i in range(10):
4     print(randrange(100))
```

Este programa siempre genera la misma secuencia de números.

- Simulación de eventos al azar con probabilidad p :
En Python es sencillo generar este tipo de eventos usando las funciones del paquete *random*. Por ejemplo simular una partícula que se mueve con probabilidad $p = 0.2$

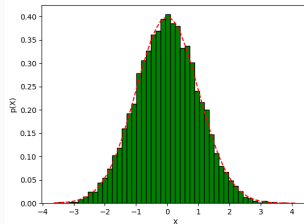
```
1 from random import random
2 if random()<0.2:
3     print("Movimiento")
4 else:
5     print("No se mueve")
```

- Este comportamiento es posible debido a que *random* genera números uniformemente distribuidos, de hecho todas las funciones revisadas funcionan igual.

Histograma

Una distribución de frecuencias muestra con qué frecuencia ocurre cada valor diferente en un conjunto de datos. Un histograma es el gráfico más utilizado para mostrar distribuciones de frecuencia. Es similar a un gráfico de barras, pero con diferencias importantes.

```
1 import numpy as np
2 from math import sqrt, exp, pi
3 import matplotlib.pyplot as plt
4 x = np.random.randn(10000)
5 def G(z, mu, sigma):
6     gauss = (1.0/(sqrt(2.0*pi)*sigma))*\
7         exp(-(0.5/(sigma**2.0))\
8             *(z-mu)**2.0)
9     return gauss
10 num_bins = 50
11 n, bins, patches = plt.hist(x,
12     num_bins, density=1,
13     facecolor='green',
14     edgecolor='black')
15 y = [G(z, 0.0, 1.0) for z in bins]
16 plt.plot(bins, y, 'r--')
17 plt.xlabel('X')
18 plt.ylabel('p(X)')
19 plt.show()
```



Fin