Programación para Física y Astronomía

Departamento de Física.

May 3, 2022

Universidad Andrés Bello®

python powered

Resumen

Gráficos

- Gráficos de líneas
- Gráficos de Dispersión
- Gráficos de Densidad

Animación

Semillas y generación de números al azar

- Generación de números al azar
- Semillas
- Usando números al azar

Gráficos

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

En este capítulo revisaremos dos tipos principales de gráficos computacionales:

- Gráficos: representación de datos numéricos desplegados en ejes calibrados.
- Diagramas y animaciones: representación de arreglos o movimientos de un sistema físico.

Gráficos de líneas

En Python existen varios paquetes con funcionalidades gráficas. Nosotros usaremos *matplotlib*. Esta instrucción de instalación es para BASH.

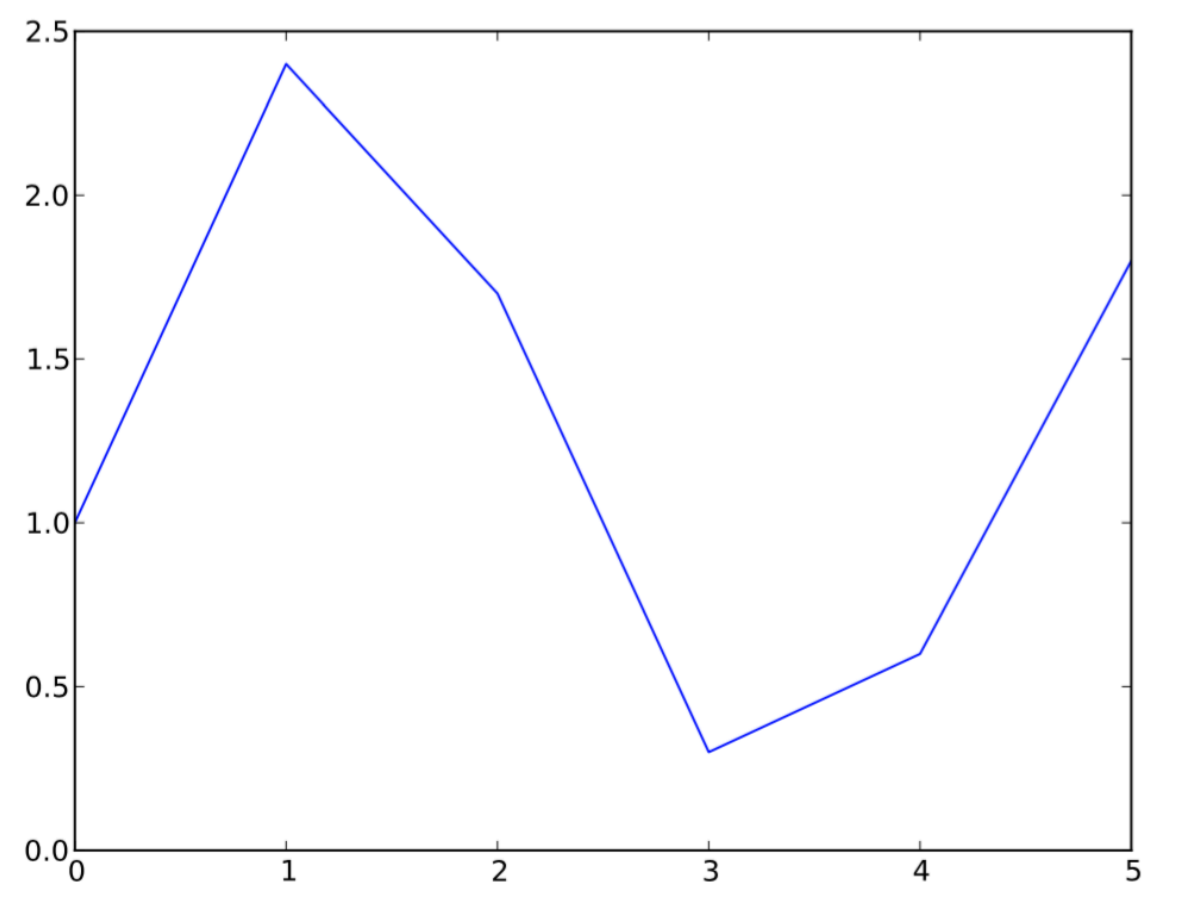
sudo apt-get install python3-matplotlib

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 el siguiente 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()
```

Gráficos de líneas

El resultado del programa anterior es:



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:

```
plot(x, y)        #plot x e y usando estilos por defecto
plot(x, y, 'bo') #plot x e y círculos azules
plot(y)           #plot y usando x como un arreglo 0...N-1
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.

Aplicando lo aprendido

```
#Graficando sin(x)
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0,10,100)
y = np.sin(x)
plt.plot(x,y)
plt.show()
```

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

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
etc...	

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3  x = np.linspace(0,10,100)
4  y1 = np.sin(x)
5  y2 = np.cos(x)
6  plt.plot(x,y1,"k-",label="sin(x)")
7  plt.plot(x,y2,"k+",label="cos(x)")
8  plt.ylim(-1.1,1.1)
9  plt.xlabel("x axis")
10 plt.ylabel("y axis")
11 plt.legend()
12 plt.show()
```

Gráficos de Dispersión

- Son útiles para representar dos variables dependientes. 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)`.

Gráficos de Dispersión

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()
```

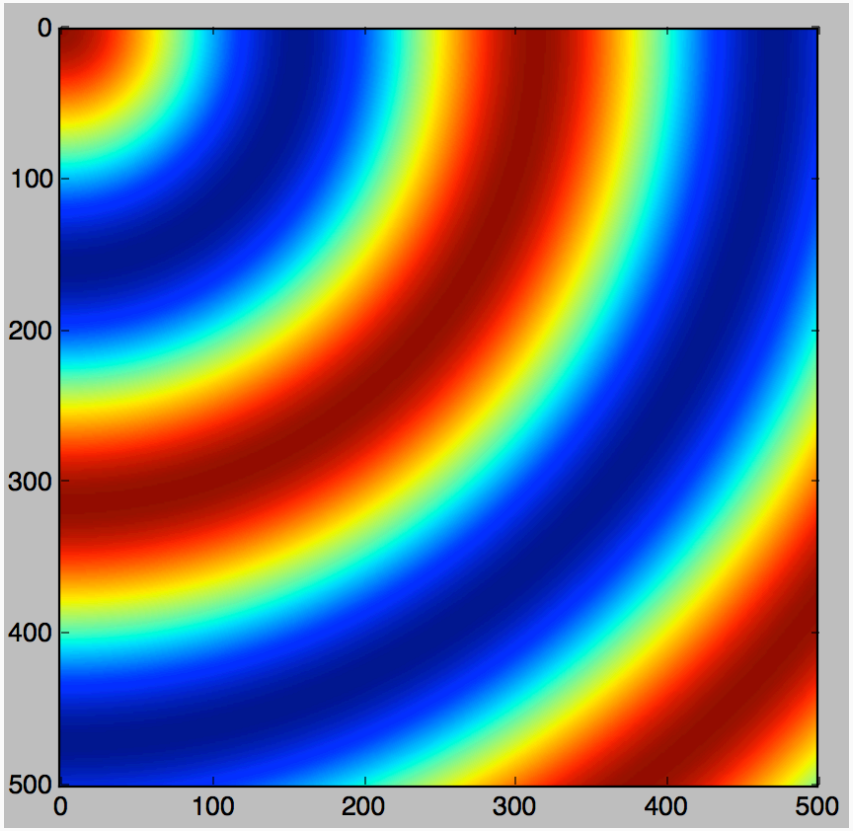
Gráficos de Densidad

- 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()
```

Gráficos de Densidad

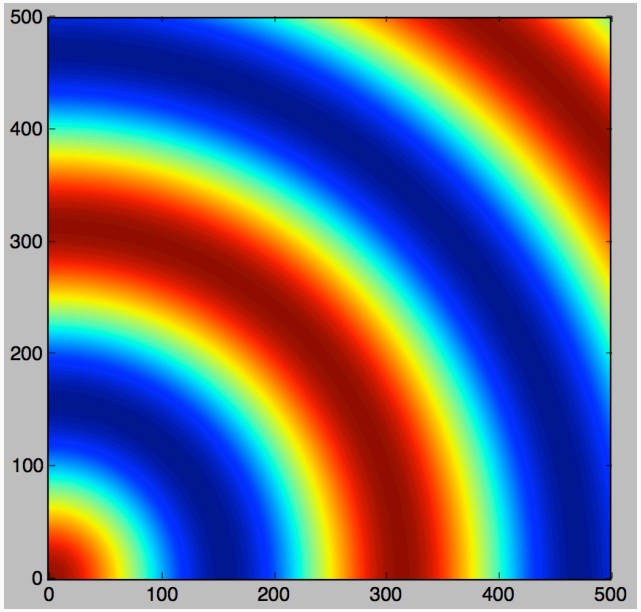
Resultado del programa anterior:



Gráficos de Densidad

Variaciones de aspecto en gráficos de densidad

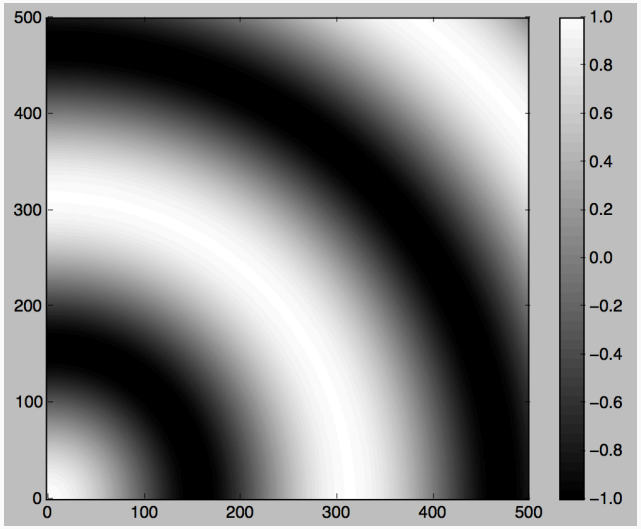
- Origen del arreglo en la esquina izquierda inferior:
  imshow(data,origin="lower")



Gráficos de Densidad

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()*.



Animación

Gráficos Animados

La manera más simple de generar animaciones en *matplotlib* es mediante el uso de la clase Animate.

- Es importante mantener una referencia a la instancia.
- La animación es avanzada mediante un *timer*.
- Si no se mantiene una referencia al objeto, entonces la animación se termina y el *garbage collector* se encarga de eliminar todo.

En el siguiente slide usted verá un código de animación en *matplotlib*. Escriba el código, ejecútelo, y comprenda su funcionamiento.

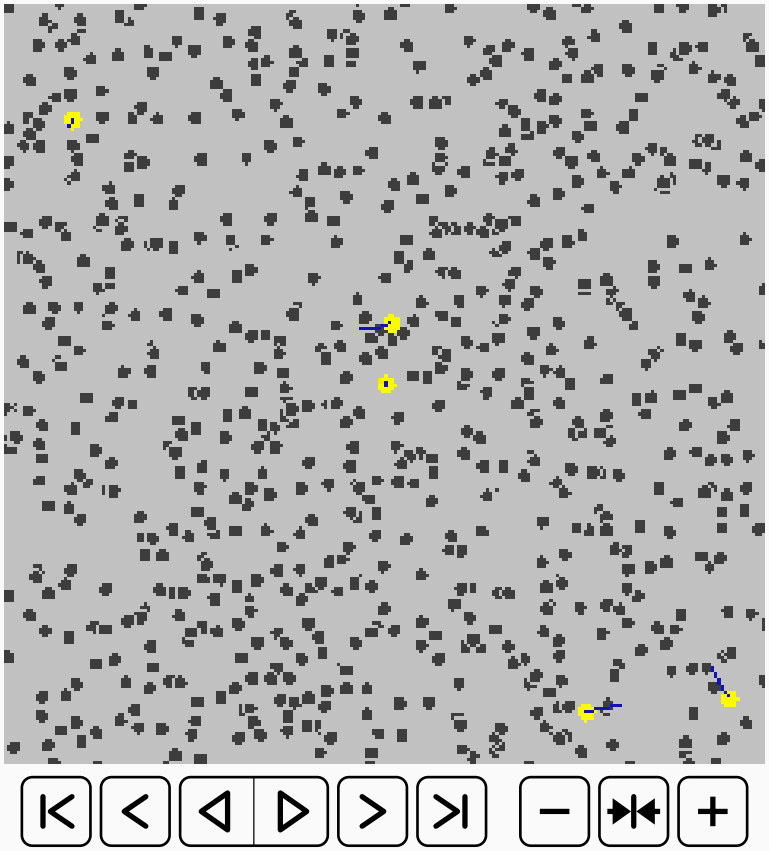
Gráficos Animados

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  from matplotlib.animation import FuncAnimation
4  from IPython.display import HTML #Solo Jupyter
5
6  fig, ax = plt.subplots()
7  xdata, ydata = [], []
8  ln, = plt.plot([], [], 'ro', animated=True)
9
10 def init():
11     ax.set_xlim(0, 2*np.pi)
12     ax.set_ylim(-1, 1)
13     return ln,
14
15 def update(frame):
16     xdata.append(frame)
17     ydata.append(np.sin(frame))
18     ln.set_data(xdata, ydata)
19     return ln,
20
21 ani = FuncAnimation(fig, update, frames=np.linspace(0, 2*np.pi, 128),
22                     init_func=init, blit=True)
23 HTML(ani.to_html5_video()) #Only Jupyter otherwise plt.show()
```

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

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

$$i = 0, 1, 2, 3....N$$

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
`random()`	número `float` uniformemente distribuido $\in [0, 1)$.
`randrange(n)`	número `int` uniformemente distribuido $\in [0, n-1)$.
`randrange(m,n)`	número `int` uniformemente distribuido $\in [m, n-1)$.
`randrange(m,n,k)`	número `int` uniformemente distribuido $\in [m, n-1)$ en pasos de k.

- 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.

Eventos probabilísticos

- Simulación de eventos al azar con probabilidad p:
  En Pyhton 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.

Fin