Módulo de adaptación

Master en Business Intelligence y Big Data

PROFESOR/A
Antonio Sarasa Cabezuelo

Escuela de organización industrial

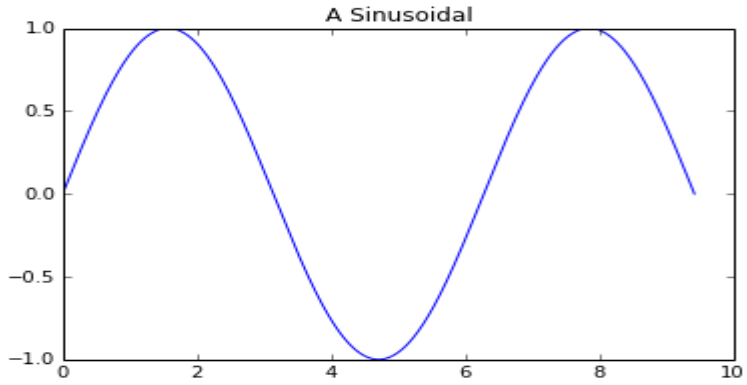
www.eoi.es

Matplotlib

Matplotlib: Introducción

- Matplotlib es un módulo de Python que proporciona un conjunto de rutinas gráficas para dibujar.
- Por ejemplo:

```
In [1]: import numpy as np
        import matplotlib
        import matplotlib.pyplot as plt

        %matplotlib inline

        x = np.linspace(0, 3*np.pi, 500)
        plt.plot(x, np.sin(x))
        plt.title('A Sinusoidal')
        plt.show()
```



Matplotlib : Introducción

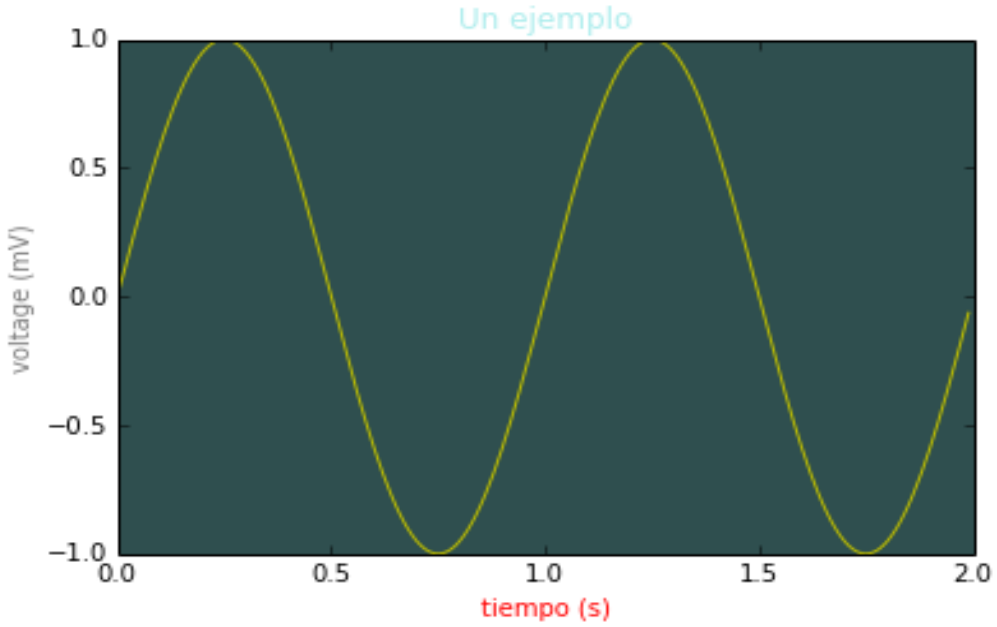
- Observar:
  - Las líneas del "import" indican que se quieren usar las funciones definidas en numpy y matplotlib.
  - La línea %matplotlib inline es un comando especial de iPython que indica que el gráfico se debe incrustar en el documento en vez de mostrarlo en otra ventana.
  - La linea `x = np.linspace(0, 3np.pi, 500)crea un numpy array de la misma forma que range()` crea listas de números. Concretamente se crea un array de 500 números reales entre 0 y 3*pi.
  - Las tres líneas plt crean un gráfico en x, y, le añaden título y muestran el dibujo.

Matplotlib : Introducción

- En el siguiente ejemplo, se ilustra como usar colores

In [8]:
```
#!/usr/bin/env python
"""
matplotlib ofrece 4 caminos para especificar colores

    1) como una simple letra en forma de cadena.

    2) como una cadena de estilo hexadecimal de html o un nombre de color html.

    3) como una tupla R, G, B donde R,G,B están en el rango entre 0 y 1

    4) como una cadena representando un número en coma flotante entre 0 y 1, correspondiente a las tonalidades de grises.


Ver help(colors) para más información
"""
from pylab import *

subplot(111, axisbg='darkslategray')
#subplot(111, axisbg='#ababab')
t = arange(0.0, 2.0, 0.01)
s = sin(2*pi*t)
plot(t, s, 'y')
xlabel('tiempo (s)', color='r')
ylabel('voltage (mV)', color='0.5') # color escala de grises
title('Un ejemplo', color='#afeeee')
show()
```

Matplotlib : Introducción



Matplotlib: Ajuste de curvas

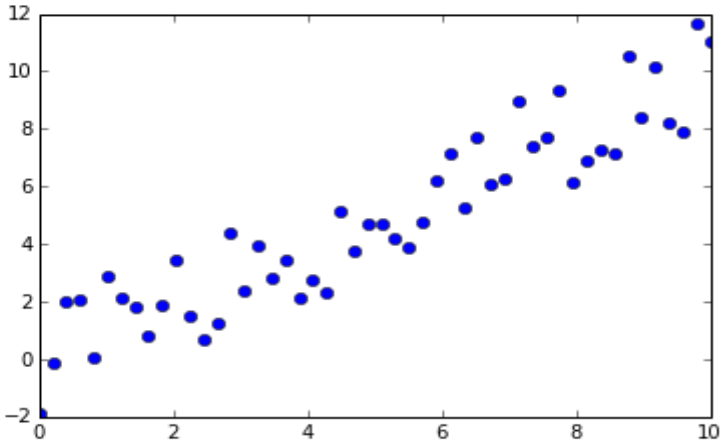
- En el siguiente ejemplo se van a crear datos usando una función conocida, y a continuación se les va añadir ruido. Entonces se intentará recuperar la función original con las que se generaron los datos.
- En primer lugar importamos las librerías necesarias:

```
In [10]: #Código necesario para programar.
         %matplotlib inline
         import numpy as np
         import matplotlib.pyplot as plt
         import matplotlib.mlab as mlab
         from scipy import optimize, polyfit
```

Matplotlib : Ajuste de curvas

- Se crean datos de la función y=mx+b, que es una función lineal, y a continuación se le añade ruido.

```
In [11]: n = 50
         ruido_amp = 4.0
         x = np.linspace(0,10,n)
         ruido = ruido_amp * (np.random.random(n)-.5)
         y = x+ruido
         plt.plot(x,y,'o')
Out[11]: [<matplotlib.lines.Line2D at 0xa5fd320>]
```



Matplotlib : Ajuste de curvas

- Se va a intentar ajustar los datos usando la función polyfit del módulo numpy. Esta función fija a un polinomio los datos x,y. El polinomio de primer orden es un simple ajuste lineal. Los primeros dos argumentos de polyfity son los valores x e y,y el tercer argumento es el orden(en este caso es 1).

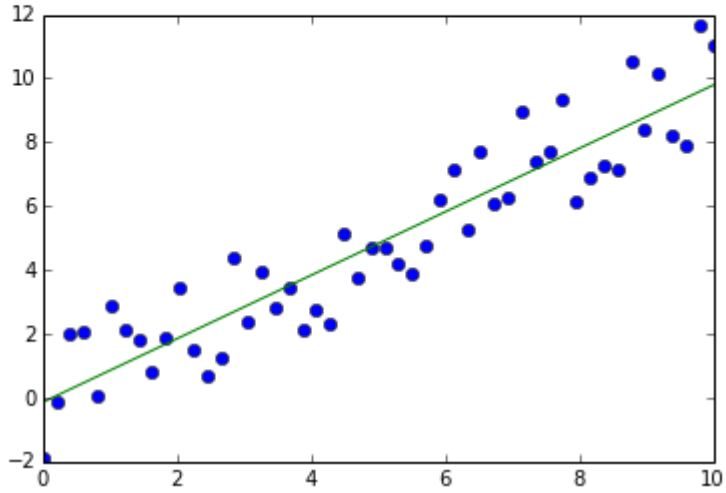
```
In [12]: m,b = polyfit(x,y,1)
         [m,b]

Out[12]: [0.99292448685937962, -0.13982083726584954]
```

- Recordar que el parámetro m corresponde a la relación (Δy/Δx). El parámetro b es el desplazamiento. Para estos datos, nuestros datos proceden de la función, por lo que la pendiente deber ser 1 y el desplazamiento es 0.

Matplotlib : Ajuste de curvas

- Ahora, se añadirá una linea de ajuste a los datos. Para ello se usará el comando plt.plot con la opción hold=True.

```
In [13]: plt.plot(x,y,'o', hold=True)
         t = np.array([0,10])
         plt.plot(t,m*t+b)

Out[13]: [<matplotlib.lines.Line2D at 0x1301d978>]
```



Matplotlib : Ajuste de curvas

- Ahora vamos a definir nuestra propia función de ajuste y se usará la función optimize.curve_fit.

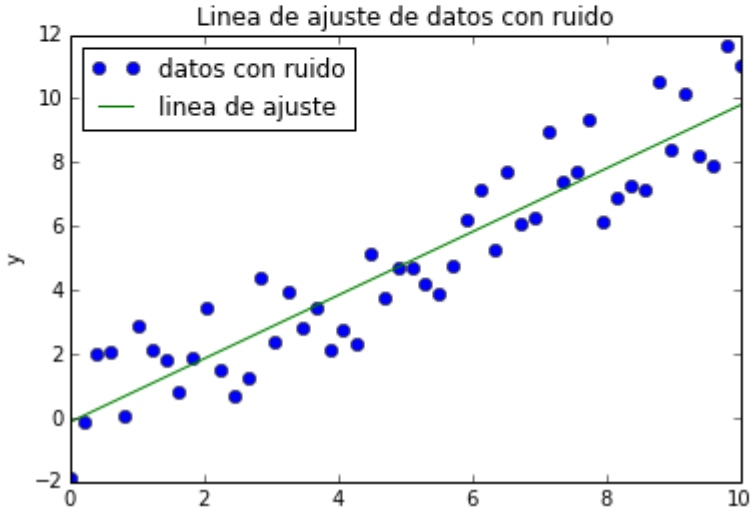
```
In [14]: def funcion_ajuste(t, m, b):
             return m*t+b

In [15]: p, cov = optimize.curve_fit(funcion_ajuste, x, y)
         p

Out[15]: array([ 0.99292449, -0.13982084])
```

- Ahora se dibuja nuevamente el gráfico, pero con leyenda y etiquetas.

Matplotlib : Ajuste de curvas

```
In [17]: plt.plot(x,y,'o', hold=True)
         t = np.array([0,10])
         plt.plot(t,p[0]*t+p[1])
         plt.xlabel('x')
         plt.ylabel('y')
         plt.title('Linea de ajuste de datos con ruido')
         plt.legend(['datos con ruido','linea de ajuste'], loc=0)
```

```
Out[17]: <matplotlib.legend.Legend at 0x13484b38>
```



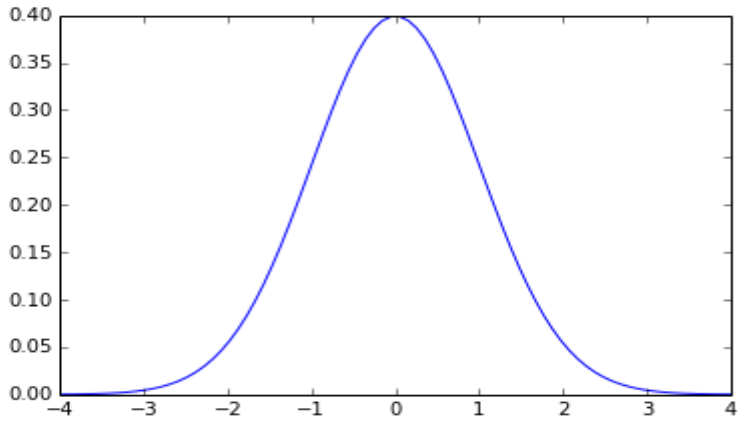
Matplotlib : Ajuste de curvas

- Se puede consultar la documentación acerca de la función optimize.curve_fit mediante el comando help(optimize.curve_fit)

Matplotlib : Ajuste de curvas

- Ahora se ajustará a una función que no es lineal. Para ello se elige una distribución normal. En este sentido se utilizará la función normpdf de matplotlib.mlab. En primer lugar se generan los datos.

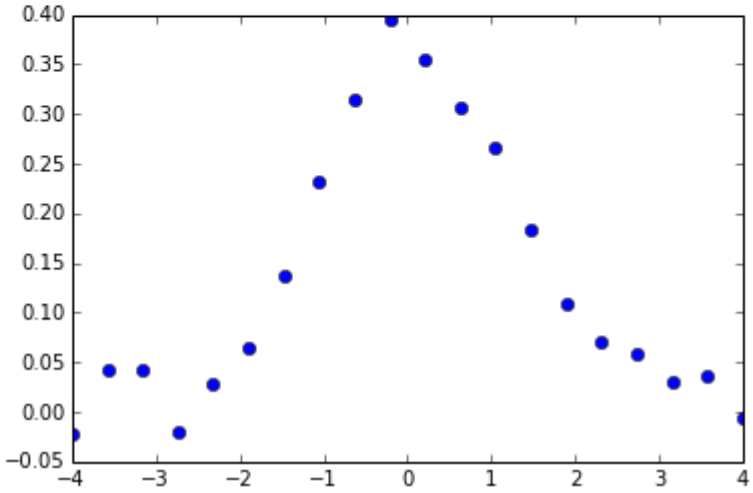
```
In [19]: media = 0.0
         var = 1.0
         sigma = np.sqrt(var)
         x = np.linspace(-4,4,80)
         plt.plot(x,mlab.normpdf(x,media,sigma))
Out[19]: [<matplotlib.lines.Line2D at 0x13509cf8>]
```



Matplotlib : Ajuste de curvas

- Se le añade ruido a los datos.

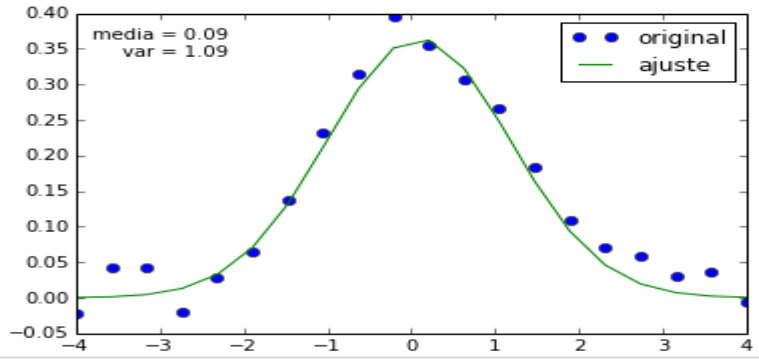
```
In [20]: n = 20
         x = np.linspace(-4,4,n)
         ruido_amp = .1
         ruido = ruido_amp * (np.random.random(n)-.5)
         y = mlab.normpdf(x,media,sigma)+ruido
         plt.plot(x,y,'o')
Out[20]: [<matplotlib.lines.Line2D at 0x13703048>]
```



Matplotlib : Ajuste de curvas

- Se ajustan los datos

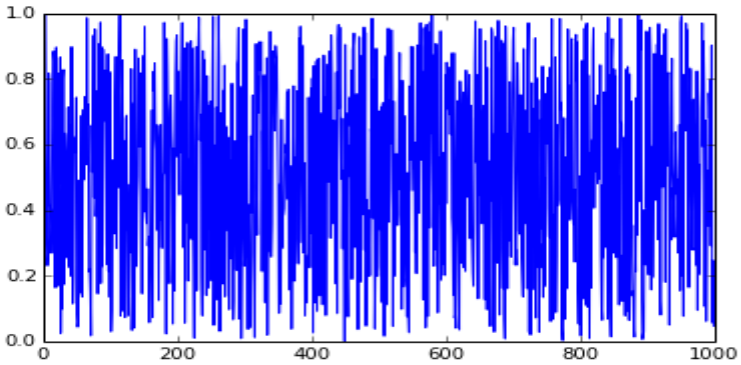
```
In [21]: def ajuste_normal(t, media, var):
             return mlab.normpdf(t,media,var)

         p, cov = optimize.curve_fit(ajuste_normal, x, y)
         p
Out[21]: array([ 0.08946055,  1.09442167])

In [24]: plt.plot(x,y,'o',hold=True)
         plt.plot(x,ajuste_normal(x,p[0],p[1]),'-')
         plt.legend(['original', 'ajuste'])
         plt.text(-2.2,.34, 'media = %.2f\nvar = %.2f' % (p[0],p[1]), ha='right')
Out[24]: <matplotlib.text.Text at 0x13c1e6d8>
```



Matplotlib : Histogramas

- En el siguiente ejemplo se va a ilustrar como dibujar histogramas. En primer lugar generamos un conjunto de datos aleatorios y los dibujamos

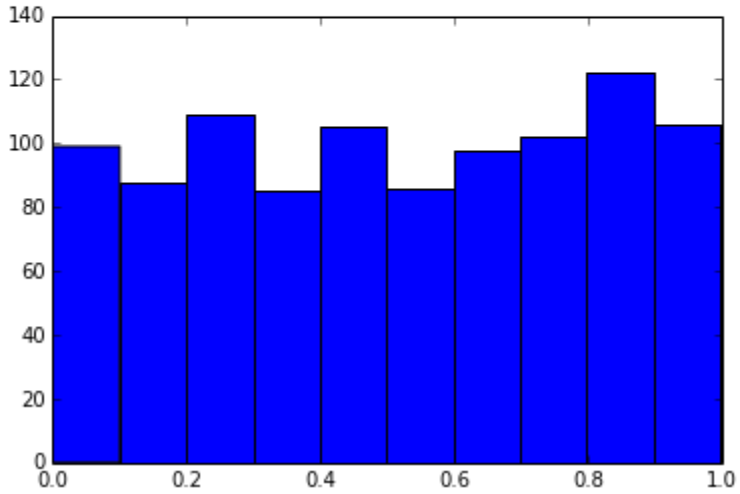
```
In [25]: #Código necesario
         %matplotlib inline
         import numpy as np
         import matplotlib.pyplot as p

In [29]: datos = np.random.rand(1000)

In [30]: p.plot(datos)
Out[30]: [<matplotlib.lines.Line2D at 0x143f8748>]
```



Matplotlib : Histogramas

- A continuación dibujamos el histograma usando hist()(help(hist))

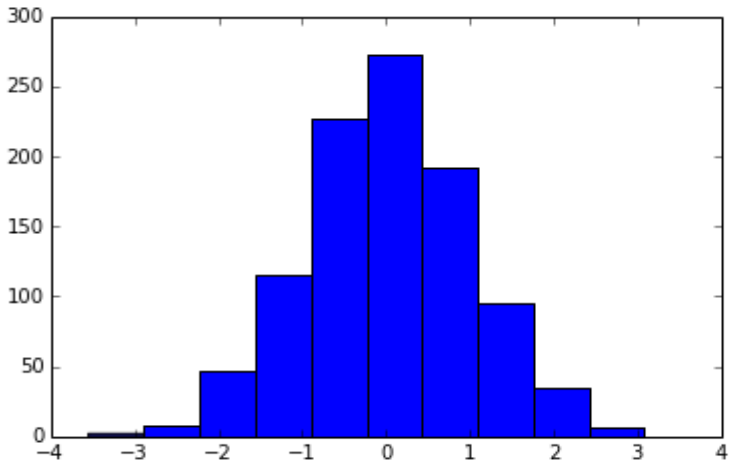
```
In [31]: p.hist(datos)

Out[31]: (array([  99.,   88.,  109.,   85.,  105.,   86.,   98.,  102.,  122.,  106.]),
          array([  2.49677915e-04,   9.98717585e-02,   1.99493839e-01,
                   2.99115920e-01,   3.98738000e-01,   4.98360081e-01,
                   5.97982161e-01,   6.97604242e-01,   7.97226322e-01,
                   8.96848403e-01,   9.96470484e-01]),
          <a list of 10 Patch objects>)
```



Matplotlib : Histogramas

- En el siguiente ejemplo se genera un conjunto de datos aleatorios de carácter normalizado, y lo dibujamos.

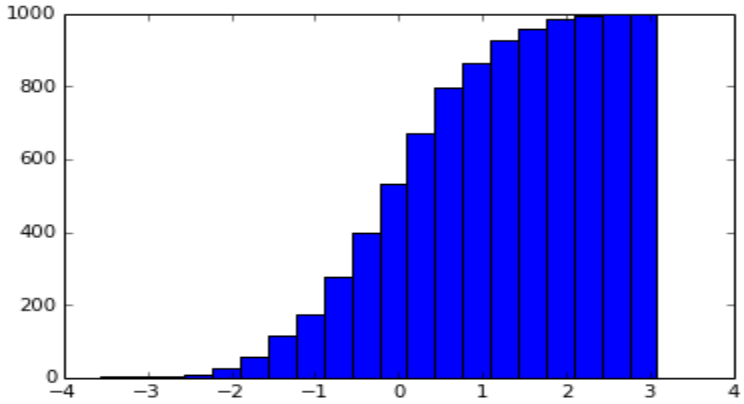
```
In [32]: datos= np.random.randn(1000)
         p.hist(datos)

Out[32]: (array([   2.,    8.,   47.,  115.,  227.,  273.,  192.,   95.,   35.,    6.]),
          array([-3.55155571, -2.88765271, -2.2237497 , -1.55984669, -0.89594368,
                 -0.23204068,  0.43186233,  1.09576534,  1.75966835,  2.42357135,
                  3.08747436]),
          <a list of 10 Patch objects>)
```



Matplotlib : Histogramas

- Usando los datos anteriores, se va a dibujar un histograma con el parámetro cumulative a cierto.

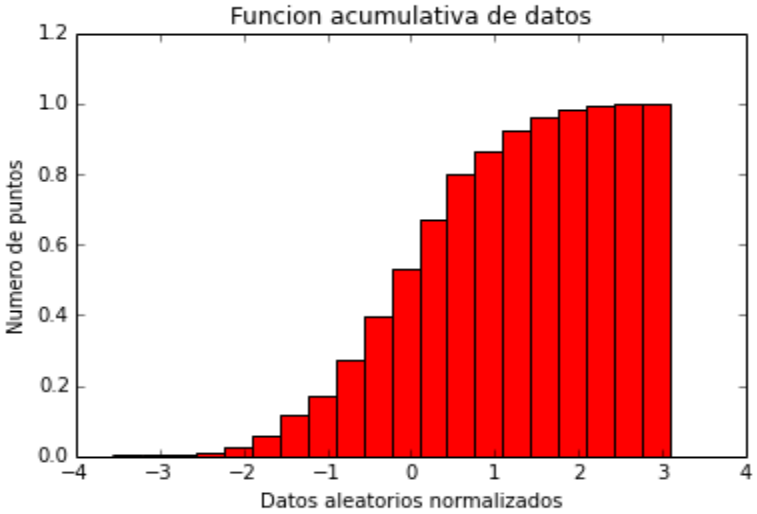
```
In [34]: p.hist(datos, 20,cumulative=True)

Out[34]: (array([    1.,     2.,     5.,    10.,    27.,    57.,   116.,   172.,
                   275.,   399.,   533.,   672.,   798.,   864.,   926.,   959.,
                   983.,   994.,   998.,  1000.]),
          array([-3.55155571, -3.21960421, -2.88765271, -2.5557012 , -2.2237497 ,
                 -1.8917982 , -1.55984669, -1.22789519, -0.89594368, -0.56399218,
                 -0.23204068,  0.09991083,  0.43186233,  0.76381384,  1.09576534,
                  1.42771684,  1.75966835,  2.09161985,  2.42357135,  2.75552286,
                  3.08747436]),
          <a list of 20 Patch objects>)
```



Matplotlib : Histogramas

- En el siguiente ejemplo se combinan las características anteriores.

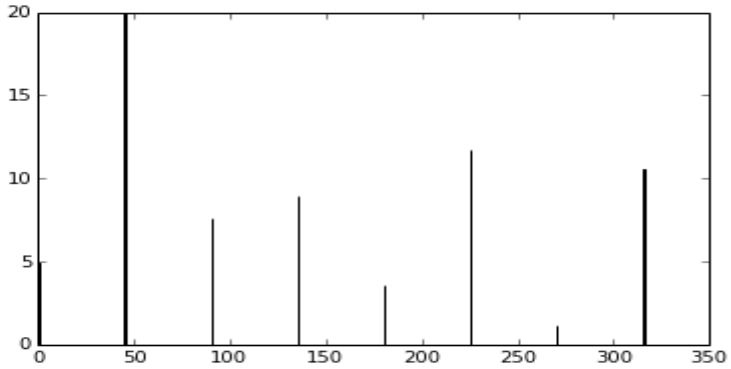
```
In [39]: p.hist(datos, 20,normed=True,cumulative=True,color='r')
         p.xlabel('Datos aleatorios normalizados')
         p.ylabel('Numero de puntos')
         p.title('Funcion acumulativa de datos')

Out[39]: <matplotlib.text.Text at 0x15830160>
```



Matplotlib : Histogramas

- En el siguiente ejemplo se va a utilizar la función para dibujar un gráfico de barras(help(p.bar))

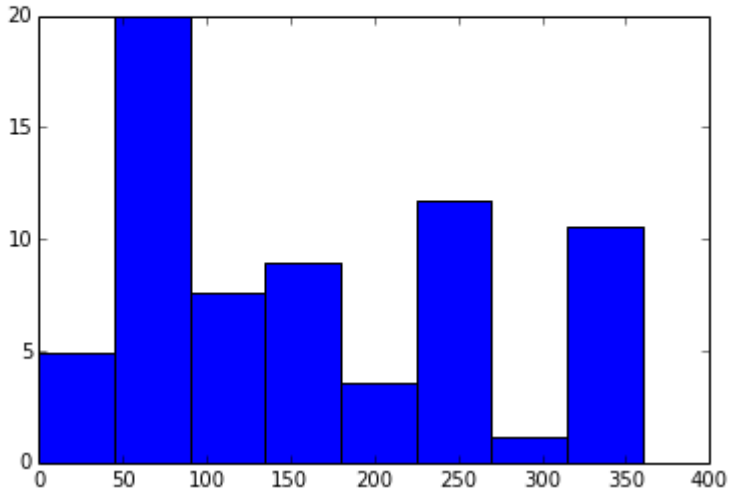
```
In [40]: x = np.arange(0, 360, 45)
         x
Out[40]: array([  0,  45,  90, 135, 180, 225, 270, 315])

In [42]: numero_picos= np.random.rand(8)*20

In [43]: p.bar(x,numero_picos)
Out[43]: <Container object of 8 artists>
```



Matplotlib : Histogramas

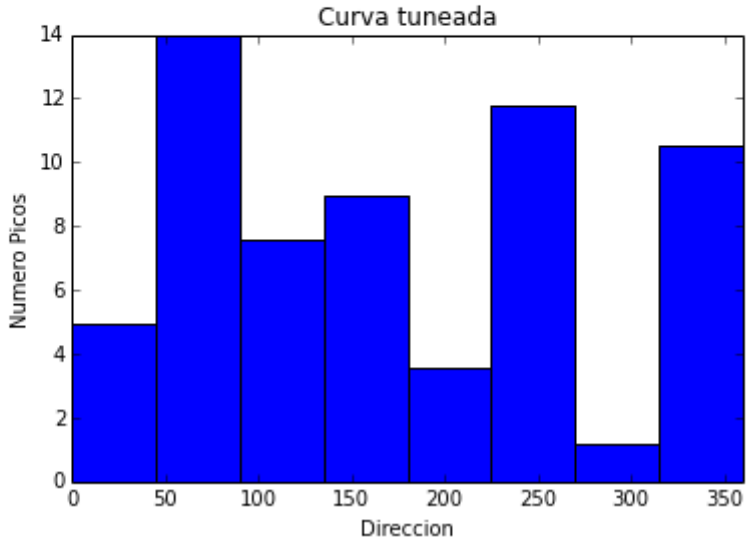
- En los siguientes ejemplos se ilustran el uso de algunos parámetros de la función bar()

```
In [44]: p.bar(x,numero_picos,width=45)

Out[44]: <Container object of 8 artists>
```



Matplotlib : Histogramas

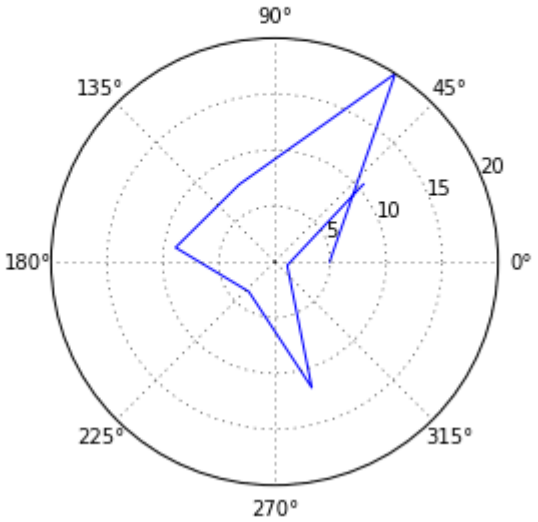
```
In [45]: p.bar(x,numero_picos,width=45)
         p.xlabel('Direccion')
         p.ylabel('Numero Picos')
         p.title('Curva tuneada')
         p.axis([0, 360, 0, 14])

Out[45]: [0, 360, 0, 14]
```



Matplotlib : Histogramas

- En los siguientes ejemplos se va a usar la función polar(help(p.polar()) para representar gráficos polares.

```
In [46]: p.polar(numero_picos)
Out[46]: [<matplotlib.lines.Line2D at 0x15ff36a0>]
```

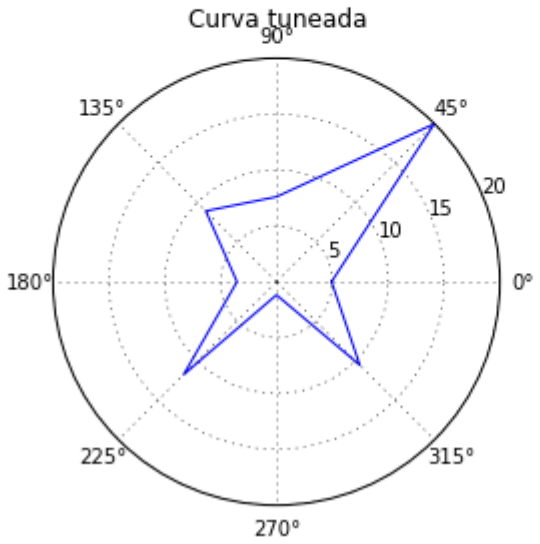


Matplotlib : Histogramas

```
In [48]: p.polar(x*np.pi/180,numero_picos)
Out[48]: [<matplotlib.lines.Line2D at 0x15fd1160>]
```

Matplotlib : Histogramas

```
In [49]: numero_picos2 = np.append(numero_picos,numero_picos[0])
         r = np.arange(0, 361, 45)*np.pi/180
         p.polar(r,numero_picos2)
         p.title('Curva tuneada')

Out[49]: <matplotlib.text.Text at 0x15e0c160>
```



Matplotlib

- Para profundizar:

http://matplotlib.org/