Graficas con Pitón

Andrew Reeve
School of Earth and Climate Sciences
University of Maine

Introducción

- matplotlib
  - más popular y más viejo
  - muchos tipos de tramas
  - bien desarrollado y documentado
- seaborn
  - realce a matplotlib
  - funciona bien con Pandas
  - tramas estadísticas (como R)
- otros
  - bokeh
  - plotly
  - HoloViews (añada a matplotlib)
  - and others!

- figure: pagina virtual
- subplots/axes
  - área virtual en la pagina para una trama
  - posible tener más que uno en una pagina
  - hay diferentes plano de deseño
  - espinas (spines) de x, y or z ejes (axis)
  - leyenda de la figura
  - anotación

Hacer una Trama Sencillo

- Necesita unos datos en un envase (estructura)
- Importa `matplotlib`
  - `from matplotlib import pyplot`
- plotea procesalmente
  - `pl.plot([1,2,3],[1,4,9])`
- plotea usando objetos
  - crea una figura y subplot
    - `fig,sps = pl.subplots(1,1)`
    - `fig` papel/pagina virtual
    - `sps` es subplot (o tramas)
  - Alternativamente:
    - `fig=pl.figure(1)`
    - `sps=fig.add_subplot(1,1,1)`
  - formatea como aparece con los metodos
    - sps.plot([1,2,3],[1,4,9])
    - sps.set_xlabel('Xx (m)')
    - sps.set_xticks([2*i for i in range(10)])

- mostrar en la pantalla: `pl.show()`
- hacer un archivo: `pl.savefig('aname.png')`

```
from matplolib import pyplot as pl
import math

## make some simple data to plot
x = [2*math.pi*i/40 for i in range(41)]
y1 = [math.sin(i) for i in x]
y2 = [math.cos(i) for i in x]

## plot data
pl.plot(x, y1)
pl.plot(x, y2)

## create screen image
pl.show()

## save to file
pl.savefig('simple_plot.png')
```

Ejemplos

```
import matplotlib.pyplot as pl
import math

## make some simple data to plot
x = [2*math.pi*i/40 for i in range(41)]
y1 = [math.sin(i) for i in x]
y2 = [math.cos(i) for i in x]

## make subplot
pl.subplot(2,1,1)
pl.plot(x, y1)
pl.xlabel(r'$\theta$')
pl.ylabel(r'$sin(\theta)$')

pl.subplot(2,1,2)
pl.plot(x, y2)
pl.xlabel(r'$\theta$',fontsize=20)
pl.ylabel(r'$cos(\theta)$',fontsize=20)
## create screen image
pl.show()
```

```
from matplotlib import pyplot as pl
import numpy as np

## make some simple data to plot
x = 2*np.pi/40*np.arange(0,41)
y1 = np.sin(x)
y2 = np.cos(x)
fig = pl.figure(0, figsize = (8, 6))
sp = fig.add_subplot(1,1,1)
## plot data
p1, = sp.plot(x, y1, linestyle='--',color
    =(.5,0,0), linewidth=1, marker='D', label='
    sin')
p2, = sp.plot(x, y2, color='.6', label='cos')
p2.set_linewidth(4.5)
sp.legend()
sp.grid()
# set sp background color
sp.set_facecolor('green')
## to update changes in ipython ...
## fig.canvas.draw()
pl.show()
```

Tramas Univariantes

- Tramas de histograms, dot/vline, box & whisker, violin plots
- Comunica la distribución del muestreo
- ¿qué es el problema con promedia y varianza?
  - Asume la distribución de datos
  - pierde información de los valores extremas
  - incluye parte aislada
  - ninguna realimentación visual

```
import matplotlib.pyplot as pl
import json  # to load data

data = json.load(open("../examples/blue.
    json"))

# make data bins
temp_values = sorted(set(data["tmpt"]))
temp_values = {val: 0 for val in
    temp_values}
# count freq. in each bin
for val in data["tmpt"]:
    temp_values[val] += 1

pl.vlines(list(temp_values.keys()), 0,
    list(temp_values.values()))
pl.ylim([0, 150])
pl.xlabel("Temp. ($^\circ$C)")
pl.ylabel("# of occurances")
pl.show()
```

Histograms

- Histrogramas agrupen los datos
- Muestra la frecuencia de intervalos de los datos
- Usualmente un grafico de barras

```
import matplotlib.pyplot as pl
import json

data=json.load(open('examples/blue.json'))

pl.hist(data['tmpt'],
        bins=[10+i*.5 for i in range(20)],
        cumulative=False)
pl.grid()

## Obj. Orient. way
# sp = pl.subplot(1,1,1)
# hg=sp.hist(data['tmpt'])
## hg is a list with 3 items in it, the last
     is the graphic object
# hg[2][0].set_color('k')
# hg[2][0].set_linewidth(5)
# hg[2][0].set_edgecolor('r')

pl.show()
```

Graficas de Box y Whisker

- Una metido compacto para mostrar la distribución de datos
- La caja contorno 50% de los datos, desde primera a tercera cuartilla (IQR: interquartile range)
- Whiskers (bigotes?) extienden a los datos más lejos pero adentro $1.5 \cdot IQR$
- Los datos alrededor de este rango están ploteado como puntos
- Muesca (opcional) indica información sobre numero de muestras
  - Muesca extiende encima y por debajo de la mediana
  - $\pm 1.57 \frac{IQR}{\sqrt{n}}$

```
import matplotlib.pyplot as pl
import json

blue=json.load(open("../examples/blue.json"))
red=json.load(open("../examples/red.json"))

datasets=[blue['tmpt'],red['tmpt']]
pl.boxplot(datasets, labels=['blue','red'],
    notch=1)
pl.ylabel('Temp. (C)')
pl.title('Caribou Bog Temperture data from two
    depths')
pl.show()
```

Graficas de Violín

- 'Alisa' las graficas de 'box y whisker'
- Mejor vista de la distribución
- Se falta información sobre las cuartillas

```
import matplotlib.pyplot as pl
import json

blue=json.load(open("./eamples/blue.
    json"))
red=json.load(open("../examples/red.
    json"))

datasets=[blue['tmpt'],red['tmpt']]
labels=['blue','red']
pl.violinplot(datasets, positions
    =[1,2], showmedians=True,
    showextrema=False)
pl.xticks([1,2],labels)
pl.ylabel('Temp. (C)')
pl.title('Caribou Bog Temperture data
    from two depths')
pl.show()
```

Multivariate plots

- Plotear dos o más variables en 2-D
- Representa la tercera (o mayor) valor con:
  - color
  - tipo de glyph
  - tamaño de símbolos
  - ejes hermanados
- Gráficas trianguladas mostrar tres variables (datos cerrados)
- Gráficas de contornos
- Gráficas de imágenes

Asignado el color

- nombre: 'green'
- rgba tupla: (0,1,0,0)
- html o hex rgb: #ffaa00
- escala de grises: 0.5
- mapa de colores: pl.cm.flag(254)
  - listar mapas disponible: `pyplot.colormaps()`

Graficas de xy plots y los pasos básicos

- Importa matplotlib
- Hace una figura y defina subplots
- Hace graficas en subplots
- Embellece la grafica
  - título
  - ejes etiquetas
  - limites
  - tick etiquetas
  - reja

```
import numpy as np
import matplotlib.pyplot as pl
import string
# make some data to plot, uses numpy
x = np.random.uniform(0, 100, 20)
noise = np.random.normal(0, 5.0, 20)
y = 10 + 0.5 * x + noise
# more recent  method for making a subplot
fig1, sp1 = pl.subplots(nrows=1, ncols=1)
sp1.plot(x, y)
fig2, sp2 = pl.subplots(nrows=2, ncols=1)
## grayscale color
sp2[0].plot(x, y, linestyle="None", marker="o", markeredgecolor="blue",
    markerfacecolor="0.5")
## rgb colors:tuple w/ values from 0 to 1, alpha sets transparency
sp2[1].plot( x, y,
    linestyle="None", marker="o",
    markeredgecolor="blue",
    markerfacecolor=(0.8, 0.1, 0.9),
    markersize=10.0, alpha=0.7,
    markeredgewidth=3.0,)
sp2[0].set_xlim([0, 100])
sp2[0].set_ylim([0, 100])
sp2[0].grid()  ##add grid lines

sp2[1].set_xticks([i * 5 for i in range(21)])  # should be done before defining labels
sp2[1].set_xticklabels([string.ascii_uppercase[i] for i in range(21)],rotation=45)

pl.show()
```

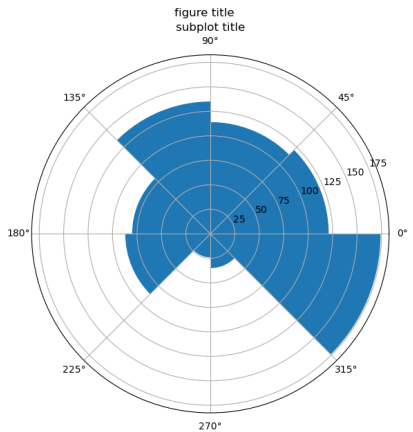
Graficas de 'Scatter'

- Similar al mando 'plot'
  - tamaño de símbolo está basado en variable
  - colores de símbolo está basado en variable

```
import numpy as np
import matplotlib.pyplot as pl

x=np.random.uniform(0,100,20)
noise=np.random.normal(0,5.,20)
y=10+.5*x+noise

fig,sp=pl.subplots()
sp.scatter(x,y,s=np.absolute(noise)*20,c=noise,cmap='
    gist_rainbow')
#colors based on colormap value
# use 'pl.colormaps()' to see list of colormaps

pl.grid()##add grid lines
pl.show()
```

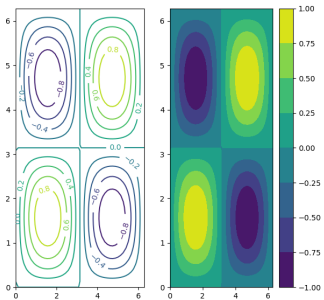
Relación de Aspecto y Graficas de Polares

```
from dateutil.parser import parse
import math
import matplotlib.pyplot as pl
datafile=open('../examples/WgScreenscrape.csv','r')
date=[]
direction=[]
windspeed=[]
for line in datafile:
        try:
        words=line.split(',')
        date.append(parse(words[0]))
        direction.append(float(words[5]))
        windspeed.append(float(words[6]))
        except ValueError:
        pass
#remove 'bad' data
idx=[i for i,d in enumerate(direction) if d>0]
windspeed = [w for i,w in enumerate(windspeed) if i in idx]
##need to convert to radians
direction = [math.radians(d) for i,d in enumerate(direction) if i
        in idx]

fig1,sp1=pl.subplots(subplot_kw={'projection': 'polar'})
fig1.suptitle("figure title")
sp1.set_title("subplot title")
bins=[i*2*math.pi/8. for i in range(9)]
h_data=sp1.hist(direction,bins=bins)

fig2,sp2=pl.subplots(subplot_kw={'aspect': 1})
sp2.plot(direction,windspeed,ls='None',marker='*')

pl.show()
```



Graficas de Contorno

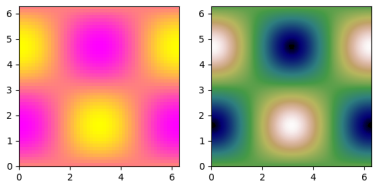
- Normal y Lleno
- Necesita datos en un formato estructuro (e.g. array)



```
import numpy as np
import matplotlib.pyplot as pl

#make a grid of x and y coords
grid = np.mgrid[0:50,0:50]
grid=grid*(2.*np.pi/49)
##make up some data
z=np.sin(grid[0])*np.sin(grid[1])

fig1,sp = pl.subplots(nrows=1, ncols=2)

cp0=sp[0].contour(grid[0],grid[1],z)
sp[0].clabel(cp0) #add contour labels
cp1=sp[1].contourf(grid[0],grid[1],z)
pl.colorbar(cp1,ax=sp[1]) ## add colorbar
pl.show()
```

Graficas Imágenes

- Necesita datos en un formato estructuro (e.g. array)
- Hace un imagen de color de los datos
- palabra clave 'interpolation' controla el alisado



```
import numpy as np
import matplotlib.pyplot as pl

#make a grid of x and y coords
grid = np.mgrid[0:50,0:50]
grid=grid*(2.*np.pi/49)
##make up some data
z=np.sin(grid[0])*np.cos(grid[1])

fig1,sp = pl.subplots(nrows=1, ncols=2)
sp[0].imshow(z,extent=[0,2*np.pi,0,2*np
    .pi],aspect=1.,
        interpolation='nearest', cmap='
    spring')
sp[1].imshow(z,extent=[0,2*np.pi,0,2*np
    .pi],aspect=1.,
        interpolation='bilinear', cmap=
    'gist_earth')
pl.show()
```

Animación

- use el método `animation` adentro de matplotlib.
  - necesita
  - un objecto de `figure` (para acceder información sobre lo que está mostrado)
  - una funciona que define come actualizar
- usualmente, la figura está actualizado por modificar los datos y re-dibujando la figura.

```
import numpy as np
import matplotlib.pyplot as pl
import matplotlib.animation as animation

b=1
x=np.arange(0,20,.2)
def func(x,a=1,b=b):
    return a*x**b

# set up plot
fig=pl.figure()
sp=fig.add_subplot(1,1,1)
sp1,=sp.plot([],[])
sp.set_xlim([0,50])
sp.set_ylim([0,20])

def init():
    ## defines initial plot, will be first image if not given
    sp1.set_data([0],[0])
    return sp1,

def update(*args):
    ## use global to access variables that you need to modify
    global x,b
    b=b/1.01 #change b
    x=np.concatenate((x,x[-1:]+.5)) #append value to array
    sp1.set_data(x,func(x,b=b)) #reset x and y data used in plot
    return sp1,

ani = animation.FuncAnimation(fig, update, frames=50, init_func=
      init,repeat=False, blit=False)
```

Seaborn

- Graficas estadísticas
- Añadido a matplotlib
- Graficas bonitas con menos esfuerza
- Menos flexibilidad

```
import json
import numpy as np
import matplotlib.pyplot as pl
import pandas as pd
import seaborn as sns

data = json.load(open("landfill_data.json", "r"))
# get keys
wells = list(data.keys())
params = list(data[wells[0]].keys())
dates = list(data[wells[0]][params[0]].keys())
"""
picking and setting a style
- find available styles: pl.style.available
- setting style: pl.style.use('ggplot')
"""
pl.style.use("bmh")
# make dictionary to plot data and create dataframe
df = {
    "dates": [dt for dt in dates for w in wells],
    "wells": [w for dt in dates for w in wells],
}
for p in params:
    row = np.array([data[w][p][d] for d in dates for w in wells])
    row = row.astype("float")
    not_nan = np.sum(~np.isnan(row))
    # only sace params that have enough measurements
    if not_nan > 25:
       df[p] = row
```

Seaborn Continuado

```
fig1 = pl.figure(1)
sp11 = fig1.add_subplot(1, 2, 1)
sp12 = fig1.add_subplot(1, 2, 2, sharey=sp11)

# with matplotlib, need to remove nans manually
notnan_vals = ~np.isnan(df["CHLORIDE"])
sp11.boxplot(df["CHLORIDE"][notnan_vals], vert=True)
sp12.violinplot(df["CHLORIDE"][notnan_vals], vert=True)

fig2 = pl.figure(2)
sp21 = fig2.add_subplot(1, 2, 1)
sp22 = fig2.add_subplot(1, 2, 2, sharey=sp21)
sns.set(style="whitegrid")
sns.set_context("paper")
# with seaborn, handles nans automatically
sns.boxplot(y=df["CHLORIDE"], ax=sp21, orient="h")
sns.violinplot(y=df["CHLORIDE"], ax=sp22, orient="h")

# build for use with pandas
df = pd.DataFrame.from_dict(df)
cols = df.columns[[i for i in range(1, 5)]]
sns.pairplot(df.loc[:, cols], data=df)
sns.jointplot(x="CHLORIDE", y="SODIUM", data=df)
sns.catplot(x="wells", y="CHLORIDE", data=df)
sns.lmplot(x="CALCIUM", y="ALKALINITY", data=df)

pl.show()
```