

Temario

- ▶ Graficación usando matplotlib.
- ▶ Método de cuadrados mínimos
- ▶ Determinación de raíces
- ▶ Ajuste de curvas
 - ▶ Método de bisección
 - ▶ Método de Newton

Miercoles 8 Noviembre. 2do Parcial.

Graficación en python

La librería de graficación se llama matplotlib. Dentro de esta tenemos dos opciones: **pylab** y **pyplot**.

- ▶ **pylab**. Interface de graficación inspirada en Matlab.
- ▶ **pyplot**. Librería de graficación basada en objetos.

Graficación en python

plot es para graficar curvas 2D.

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

x = np.linspace(0, 20, 1000)
y = np.sin(x)

plt.plot(x, y)
plt.savefig('fig.png')

plt.show() # Solo en la laptop no en el servidor
```

savefig para guardar la figura en un archivo (png,jpg,eps,etc.)

show para mostrar la figura. **OJO NO usar en el servidor**

Límites de los ejes

`xlim(xmin,xmax), ylim(ymin,ymax)`

En el caso anterior el plot acomoda los ejes a los máximos. Definamos nosotros los límites de los ejes. [xmin, xmax, ymin, ymax]

```
plt.plot(x, y)  
plt.axis((5, 15, -1.2, 1.2))
```

Títulos de gráficos

`xlabel('x'), ylabel('y'), title('Titulo')`

```
plt.plot(x, y)

plt.xlabel('this is x!')
plt.ylabel('this is y!')
plt.title('My First Plot')
```

Entiende latex para escribir fórmulas:

```
y = np.sin(2 * np.pi * x)
plt.plot(x, y)
plt.title(r'$\sin(2 \pi x)$')
```

Tipos de curvas/líneas

```
plt.plot(x, y, '-r')
```

Colores disponibles:

'r' = red - rojo

'g' = green - verde

'b' = blue - azul

'c' = cyan - celeste

'm' = magenta - violeta

'y' = yellow - amarillo

'k' = black - negro

'w' = white - blanco

Las líneas pueden tener distintos **estilos**:

'-' = línea continua

'--' = línea a trazos

'.' = línea punteada

'-.' = punteada a trazos

'.' = puntos

'o' = círculos

'^' = triángulos

Múltiple curvas y leyendas

```
x = np.linspace(0, 20, 1000)
y1 = np.sin(x)
y2 = np.cos(x)

plt.plot(x, y1, '-b', label='sine')
plt.plot(x, y2, '-r', label='cosine')
plt.legend(loc='upper right')
plt.ylim(-1.5, 2.0)
```

Hasta aca el uso de las librerias pyplot o pylab sería indistinto.

Múltiple plots

`subplot(filas, columnas, nro de plot)`

El nro empieza en 1, y sigue la numeración en orden de lectura (izq a der arriba a abajo).

Con pylab

```
pylab.subplot(2, 2, 1)
pylab.plot(x, np.sin(x))
pylab.subplot(2, 2, 2)
pylab.plot(x, np.cos(x))
pylab.subplot(2, 1, 2)
pylab.plot(x, x**2-x)
```

Con pyplot

```
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(9,3))
ax1 = fig.add_subplot(1,2,1)
ax1.plot(x, np.sin(x))
ax1 = fig.add_subplot(1,2,2)
ax1.plot(x, np.cos(x))
```


Guardar el gráfico en un archivo

Para guardar la figura en un archivo de imagen tenemos el comando:

```
savefig(fname, dpi=None, facecolor='w', edgecolor='w',  
orientation='portrait', papertype=None, format=None):
```

Ejemplo:

```
savefig('fig05.png', dpi=80)
```

Formatos recomendados: png o eps (conservan la resolución de fuentes y líneas cuando se cambia el tamaño).

Esta es la opción recomendada en el servidor (en lugar del show).

Gráficos de densidades

Si lo que queremos graficar es una “imagen” o gráfico de densidad 2d se utiliza el comando:

```
imshow(data)  
show()
```

Para cambiar el origen: `origin="lower"`

Si se quiere el gráfico en blanco y negro en lugar del “heatmap” se utiliza:

```
imshow(data)  
gray()  
show()
```

Para cambiar la escala del gráfico:

```
imshow(data, extend=[0, 10, 0, 5])
```

Si queremos cambiar el aspecto (la razón de distancia entre x e y)

`aspect=2`

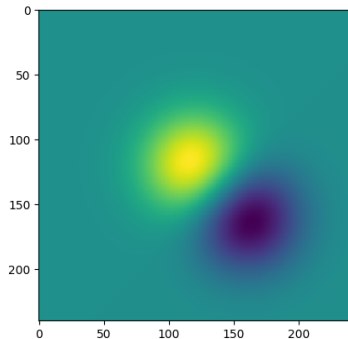
Ejemplo densidades

```
delta = 0.025
x = y = np.arange(-3.0, 3.0, delta)
X, Y = np.meshgrid(x, y)
Z1 = np.exp(-X**2 - Y**2)
Z2 = np.exp(-(X - 1)**2 - (Y - 1)**2)
Z = (Z1 - Z2) * 2

fig, ax = plt.subplots()
im = ax.imshow(Z)
plt.show()

im = ax.imshow(Z,
               interpolation='bilinear',
               cmap=cm.RdYlGn, origin='lower',
               extent=[-3, 3, -3, 3],
               vmax=abs(Z).max(),
               vmin=-abs(Z).max())

plt.show()
```

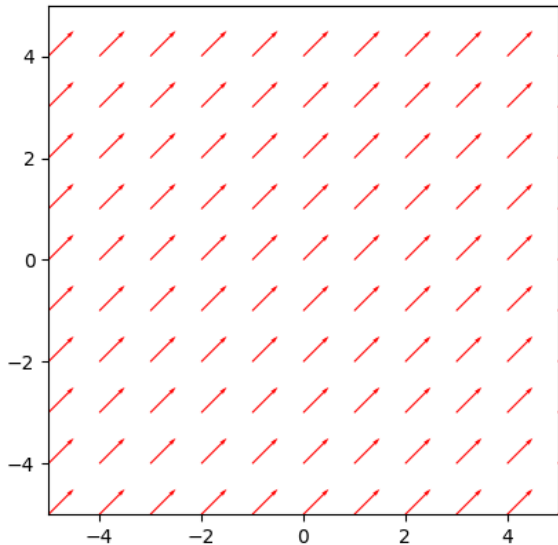


Campos de vectores

Función `quiver`. `pyplot`! Grafica vectores en un plano.

```
import matplotlib.pyplot as plt
import numpy as np
X, Y = np.meshgrid(np.arange(-10, 10, 1), np.arange(-10, 10, 1))
x_shape = X.shape
U=np.ones(x_shape)
V=np.ones(x_shape)
fig, ax = plt.subplots()
q = ax.quiver(X, Y, U, V, units='xy' ,scale=2, color='red')
ax.set_aspect('equal')
ax.set_xlim(-5,5)
ax.set_ylim(-5,5)
plt.show()
```

Campos de vectores



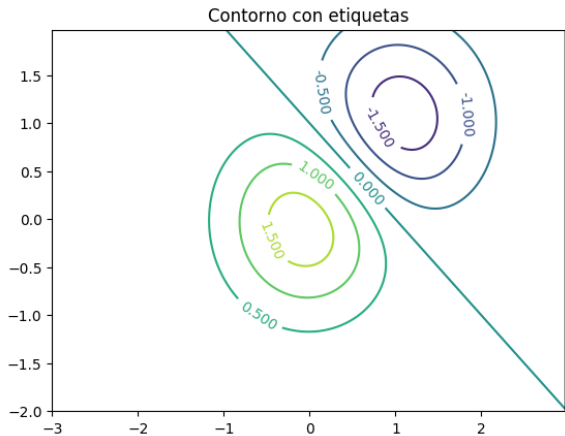
Contornos

Función `contour`. `pyplot`!

```
import matplotlib.pyplot as plt

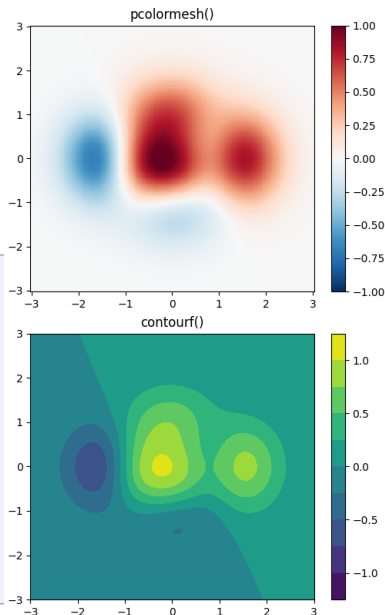
fig, ax = plt.subplots()
CS = ax.contour(X, Y, Z)
ax.clabel(CS, inline=True, fontsize=10)
ax.set_title('Contorno con etiquetas')
plt.savefig('contorno.png', bbox_inches='tight')
plt.show()
plt.close()
```

Contorno



Contornos: barra de colores y pcolormesh

```
fig, axs = plt.subplots(1, 2,  
    layout='constrained')  
pc = axs[0].pcolormesh(X, Y, Z,  
    vmin=-1, vmax=1, cmap='RdBu_r')  
fig.colorbar(pc, ax=axs[0])  
axs[0].set_title('pcolormesh()')  
co = axs[1].contourf(X, Y, Z,  
    levels=np.linspace(-1.25, 1.25  
        , 11))  
fig.colorbar(co, ax=axs[1])  
axs[1].set_title('contourf()')  
  
fig.savefig('contorno2.png')
```



Mas alla!

Hay un montón de formas de graficar.

- ▶ Se pueden hacer animaciones. Ver `animation`
- ▶ Se pueden hacer gráficos interactivos.

Widgets adaptativos.

Muy bueno para cambiar los parametros del gráfico y lo veo

```
from matplotlib.widgets import Slider, Button,  
RadioButtons
```

```
https://matplotlib.org/3.4.2/
```