

# Temario

- ▶ Derivadas numéricas
- ▶ Repasos varios
- ▶ Graficación usando matplotlib.

Miercoles 9 Noviembre. 2do Parcial.

Faltan 2 semanas!!!

# Diferenciación numérica

Supongamos que conocemos y podemos evaluar a una función arbitraria  $f(x)$ .

Esta función puede ser:

- ▶ una función matemática de una línea,
- ▶ una función/mapa computacional de 100 líneas de código (con for, if etc), que no se puede traducir en una función analítica.

¿Como calculamos computacionalmente la derivada conociendo solo la función?

## Serie de Taylor

Aplicamos como en el caso de aproximación de funciones la serie de Taylos alrededor de  $h$ ,

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2!}f''(x) + \frac{h^3}{3!}f'''(x) + \cdots$$

Donde asumimos  $h$  pequeño.

Ahora no conocemos  $f'(x)$  y lo queremos calcular a través de la función.

Despejando:

$$f'(x) = \frac{f(x+h) - f(x)}{h} - \frac{h}{2!}f''(x) + \frac{h^2}{3!}f'''(x)$$

Error de la aproximación:  $\sigma(h) \propto h$  proporcional a  $h$ .

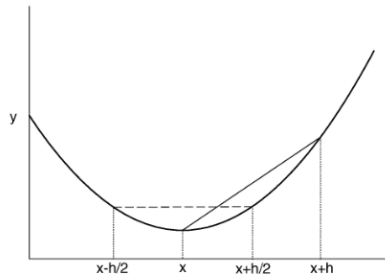
## Diferencias centradas

Si miro la serie de Taylor para  $-h$ :

$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2!}f''(x) - \frac{h^3}{3!}f'''(x) + \dots$$

Positiva:

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2!}f''(x) + \frac{h^3}{3!}f'''(x) + \dots$$



$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{h^2}{6}f'''(x) + \dots$$

Se eliminó el término de la derivada segunda  $\rightarrow$  Aproximación de segundo orden:  $\sigma(h) \propto h^2$

## Derivada segunda

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2!}f''(x) + \frac{h^3}{3!}f'''(x) + \dots$$

$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2!}f''(x) - \frac{h^3}{3!}f'''(x) + \dots$$

Si sumamos el resultado es:

$$f(x+h) + f(x-h) = 2f(x) + 2\frac{h^2}{2!}f''(x) + \sigma(h^3)$$

Despejando:

$$f''(x) = \frac{f(x+h) + f(x-h) - 2f(x)}{h^2} + \sigma(h^2)$$

# Variables acumuladores

$$y = \sum_i^N i^2$$

En formato secuencial esto es:

$$y_N = y_{N-1} + i^2|_{i=N}$$

En términos computacionales:

```
yacum=0.0
for i in range(N+1):
    yacum = yacum + i**2
```

En el caso del factorial  $N!$ :

```
facum=1.0
for i in range(2,N+1):
    facum = facum * i
```

## Funciones como argumentos de entrada de funciones

Supongamos que tenemos una rutina de integración, que queremos nos sirva para integrar **cualquier función matemática**.

Los lenguajes (incluido python) permiten que se pueda ingresar como argumento de entrada al nombre de una función:

```
def integracion(fn,a,b,n):
```

donde  $a, b$  son los límites inferior y superior.  $n$  la cantidad de evaluaciones y  $fn$  es el nombre de la función a evaluar.

Las funciones que tienen como argumento otras funciones son funciones de alto-orden.

# Funciones como argumentos de entrada de funciones

```
def integracion(fn,a,b,n):  
    delta=(b-a)/n  
    x=a  
    integ=0.0  
    for i in range(n):  
        integ=integ+ fn(x) * delta  
        x=x+delta  
    return integ
```

Se llama con

```
def absin(x):  
    return m.sin(abs(x))  
def abcos(x):  
    return m.cos(abs(x))  
integracion(absin,-1.,1.,100) # funcion definida por nosotros  
integracion(abcos,-1.,1.,100)  
integracion(m.cos,-1.,1.,100) # funcion del math
```



# 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.

Por el momento usemos la opción mas sencilla **pylab**.

El que este interesado en hacer gráficos mas “cool” (personalizados) es conveniente la **pyplot** pero de todas maneras los conceptos son similares.

# Graficación en python

**plot** es para graficar curvas 2D.

```
import pylab
import numpy as np

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

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

pylab.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.

```
pylab.plot(x, y)
pylab.xlim(5, 15)
pylab.ylim(-1.2, 1.2)
```

# Títulos de gráficos

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

```
pylab.plot(x, y)
```

```
pylab.xlabel('this is x!')
```

```
pylab.ylabel('this is y!')
```

```
pylab.title('My First Plot')
```

Entiende latex para escribir fórmulas:

```
y = np.sin(2 * np.pi * x)
```

```
pylab.plot(x, y)
```

```
pylab.title(r'$\sin(2 \pi x)$')
```

# Tipos de curvas/líneas

```
pylab.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)

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

# 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()

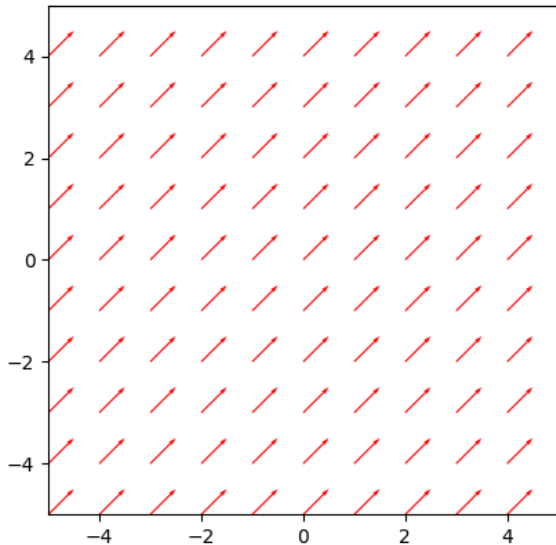
im = ax.imshow(Z, interpolation='
                bilinear', cmap=
                cm.RdYlGn,
                origin='lower',
                extent
                =[
                -3
                ,
                3,
                -
```

# 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



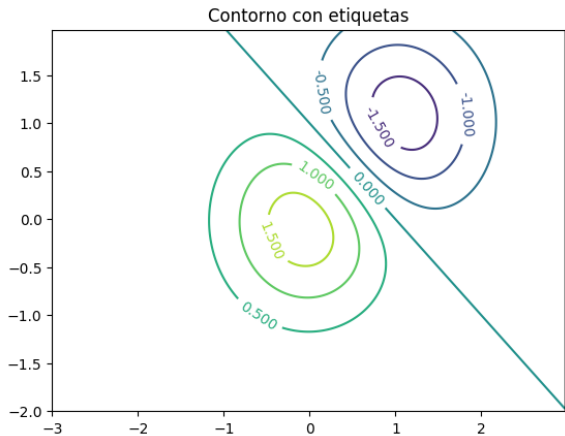
# Contornos

## 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



## 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/>