Temario

2do Parcial dia 5 de Noviembre.

- Repasos varios
- Graficación usando matplotlib.

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.

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 nost
integracion(abcos,-1.,1.,100)
integracion(m.cos,-1.,1.,100) # funcion del math
```

Guardado de arrays. Numpy Ascii

En formato ascii:

```
>>> a = np.array([1, 2, 3, 4])
>>> np.savetxt('test1.txt', a, fmt='%d')
>>> b = np.loadtxt('test1.txt', dtype=int)
```

Para multiples arrays:

```
>>> np.savetxt('test1.txt', [a,b], fmt='%d')
```

```
with open('data.txt', 'w') as f:
    np.savetxt(f, a,fmt='%d')
    np.savetxt(f, b,fmt='%d')
    np.savetxt(f, c,fmt='%d')
```

Cambiar 'w' por 'a' si es un archivo que ya fue creado y quieren agregar datos.

Guardado de arrays. Numpy Binario

En formato binario (recomendado):

```
>>> np.save('test3.npy', a)
>>> d = np.load('test3.npy')
```

Multiple arrays:

```
m1 = np.arange(9).reshape(3, 3)
m2 = np.arange(10).reshape(2, 5)
np.save('matrices.npy', matriz1=m1, matriz2=m2)
```

Para leerlos

```
data = np.load('matrices.npy')
m1=data['matriz1']
m2=data['matriz2']
print m1q
```

Con np.savez se guarda comprimido. Usar '.npz'. Leer con load.

Guardado con estructura

pickle es un serializador (pone los datos en linea seguidos) y un de-serializador (lee la cadena y la vuelve al formato original).

```
import cPickle as pickle
import numpy as np

data = [np.arange(8).reshape(2, 4), np.arange(10).reshape

with open('mat.pkl', 'wb') as outfile:
    pickle.dump(data, outfile, pickle.HIGHEST_PROTOCOL)

with open('mat.pkl', 'rb') as infile:
    result = pickle.load(infile)
```

Guardado de datos científicos

Formato para grandes bases de datos cientificas (supercomputadoras): Netcdf: Network Common Data Format (formato de guardado para matrices multiple plataforma). HDF5: Hierarchical Data Format

Graficación en python

La librería de graficación se llama matplotlib. Dentro de esta tenemos dos opciones: pylab y pyplot.
Por el momento usemos la opción mas sencilla pylab.
plot es para graficar curvas.
show para mostrar la figura.

```
import pylab
import numpy as np

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

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

Límites de los ejes

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

En el caso anterior el plot acomoda los ejes a los máximos. Definamos nosotros lo 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).

```
>>> subplot(2, 2, 1)
>>> plot(x, sin(x))
>>> subplot(2, 2, 2)
>>> plot(x, cos(x))
>>> subplot(2, 1, 2)
>>> plot(x, x**2-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 cunado se cambia el tamaño).

Graficos de densidades

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

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

Para cambiar el origin: `origin="lower"`
Si se quiere el grafico 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])
```

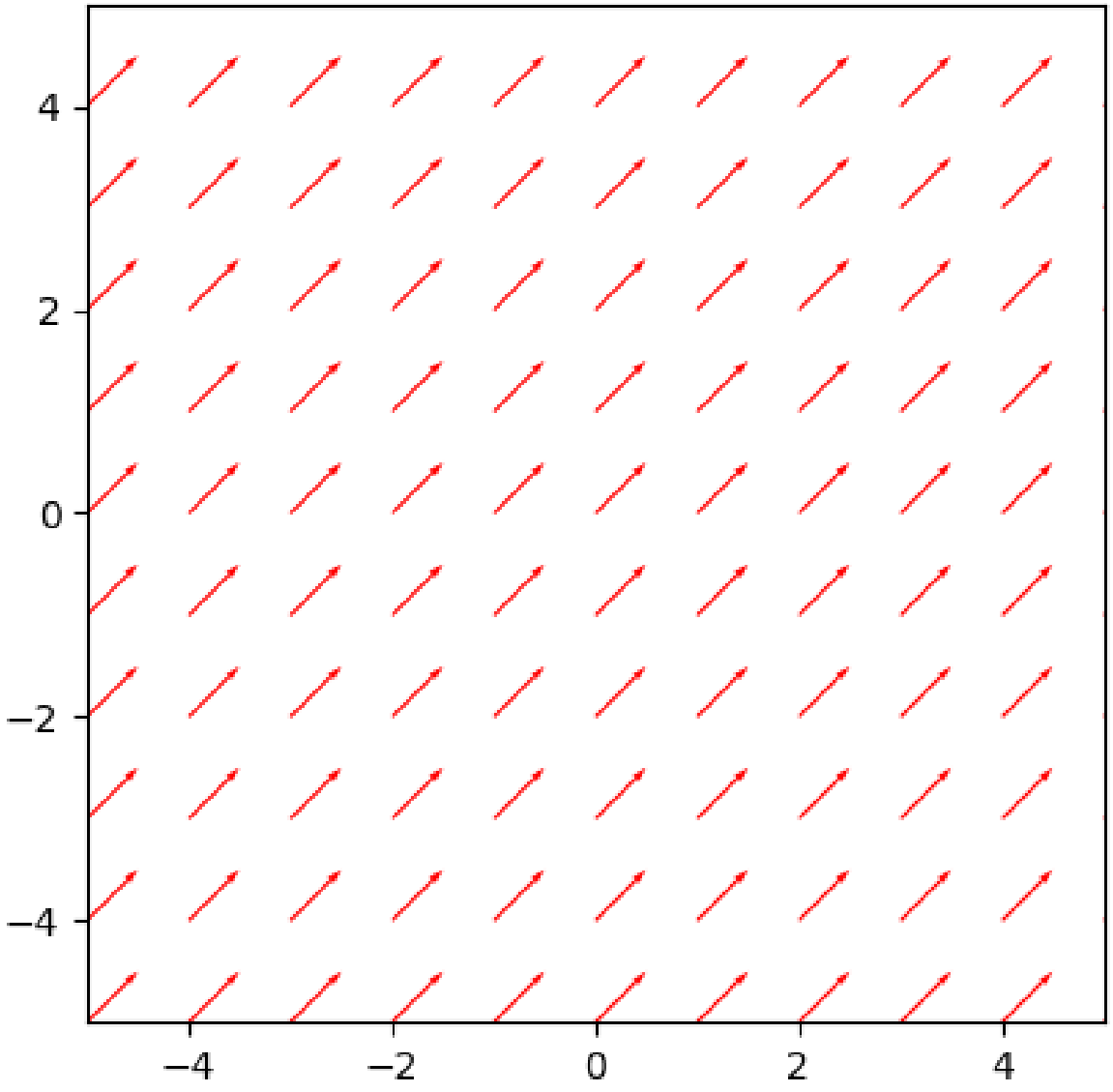
Se queremos cambiar el aspecto (la razon de distancia entre x e y)
`aspect=2`

Campos de vectores

Función `quiver`. pyplot!

```
import matplotlib.pyplot as plt 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')
plt.xlim(-5,5)
plt.ylim(-5,5)
plt.show()
```

Campos de vectores



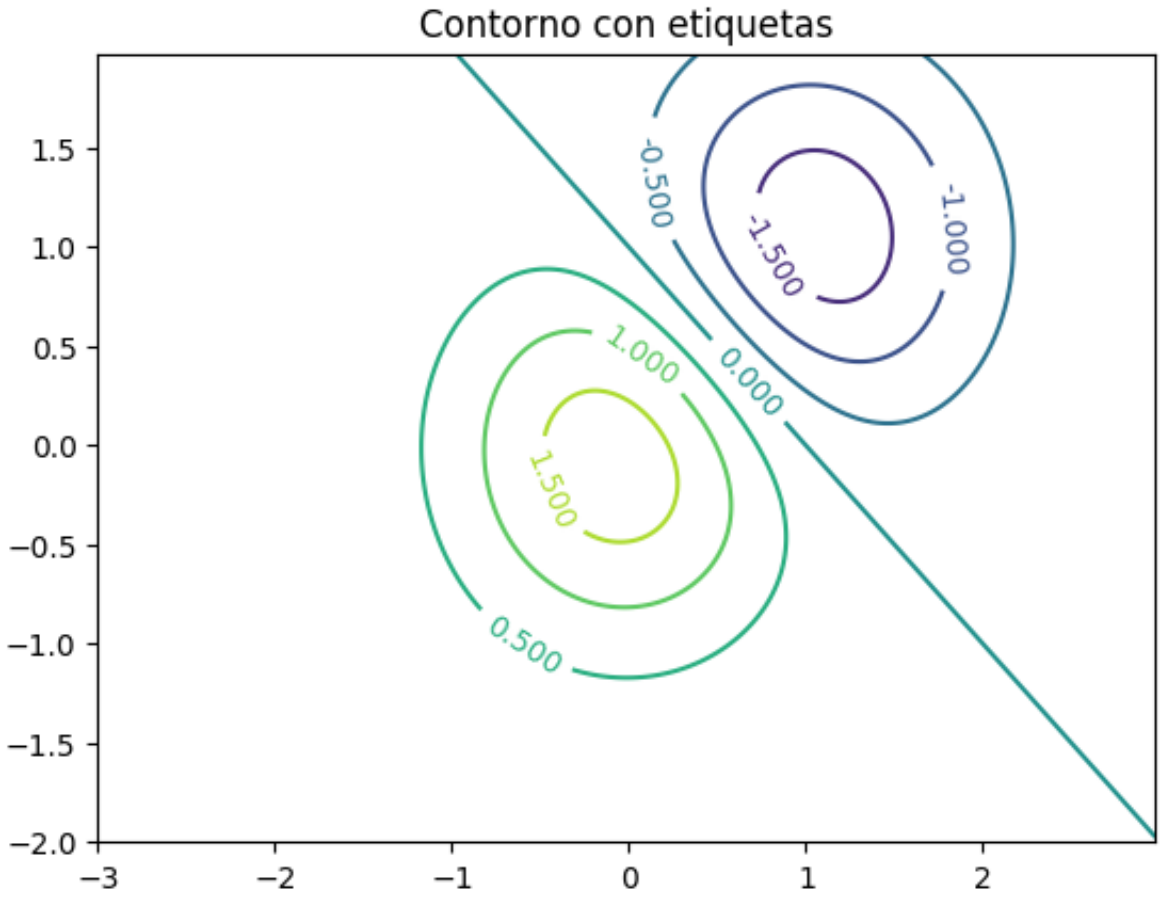
Contornos

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


Contorno con etiquetas
-0.500
-1.000
-1.500
0.000
1.000
1.500
0.500
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−3
−2
−1
0
1
2

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/3.3.1/
```