Temario

- Lectura y escritura de archivos.
- numpy: vectores. arrays. numpy
- generaciones automáticas: linspace
- matplotlib: generación de gráficos
- curvas simples.
- múltiples curvas.
- parámetros que controlan los gráficos

Lectura de archivos

Supongamos que queremos leer datos de un archivo.

Para esto tenemos que abrir el archivo, open, leer read y cerrar el archivo close.

```
text_file = open("Salida.txt", "r")
data=text_file.read()
text_file.close()
```

En forma mas pythonica:

```
with open('data.txt', 'r') as myfile:
    data=myfile.read()
```

Cuando deja de haber identación, significa que python tiene que cerrar el archivo.

Escritura de archivos

El formato para escritura de un archivo es similar. Para abrir aclaramos es para escritura: “w”

```
text_file = open("Saluda.txt", "w")
text_file.write("Precio: %s" % Cantidad)
text_file.close()
```

```
with open("Output.txt", "w") as text_file:
    text_file.write("Precio: %s" % Cantidad)
```

El formato es equivalente a lo que usamos en el print. ej. %6.2f.

Listas para manejo de funciones matemáticas

Supongamos que queremos trabajar con los puntos de la función $f(x) = x * sin(x^2)$ en el intervalo $[0, \sqrt{2\pi}]$. Resolución de 100 puntos.

```
import math as m
n=100
dx=m.sqrt(2*m.pi)/(n-1.0)
x=[] #crea una lista vacia
y=[]
for i in range(n):
    x1=i*dx
    x.append(x1)
    y1=funcion(x1)
    y.append(y1)
```

Listas para manejo de funciones matemáticas

La función viene dada por:

```
def funcion(x):
    f=x*m.sin(x**2)
    return f
```

Para generar listas en forma pythonica es:

```
x=[i*dx for i in range(n)]
y=[funcion(x1) for x1 in x]
```

NumPy: Vectores

Para realizar cálculos científicos y trabajar con vectores, matrices, etc existe una librería específica: Numerical Python “numpy”.

```
import numpy  as np
```

En general a los vectores/matrices se les llama “arrays”.

Para convertir una lista en un array, array:

```
ar=np.array([5.0,2.3,7.2])
```

```
ar=np.array(lista)
```

Para generar un array, zeros, es decir lo llenamos de 0s para generarlo.

```
ar=np.zeros(n)
```

Esto genera un vector de n componentes.

Generación de vectores uniformes

Cuando hacemos una tabla para graficación en general necesitamos generar puntos equiespaciados entre un valor mínimo y un máximo.

La funcion linspace nos genera el vector automaticamente:

```
xvec= linspace(xmin,xmax,1000)
```

Esto nos genera un vector que comienza con el valor xmin y determina con el valor xmax y tiene 1000 puntos.

Cual es la resolución que tienen los puntos?

$$dx = (xmax - xmin)/(nptos - 1)$$

Entonces si hacemos:

```
xvec=[xmin+i*dx for i in range(nptos)]
```

Cual es la diferencia?

Graficación en python

Uno de los grandes atractivos de python para sus aplicaciones científicas es el hecho que podemos realizar cálculos y luego inmediatamente ver el resultado en un gráfico.

Esto permite la graficación directa a partir de datos en memoria, sin tener guardar en archivos y releer en otros entornos de programación.

Para el debugging, la graficación de datos intermedios, y una directa interacción con el usuario, es un entorno óptimo.

Graficación en fortran-python

Existen algunas librerías que permiten realizar esto en fortran, sin embargo el procedimiento termina siendo bastante engorroso y poco flexible. (existen librerías de graficación python para fortran, NCAR graphics, gnuplot, etc).

Dada la dificultad de fortran para interactuar con otros lenguajes en general las librerías de graficación graban los datos en formatos universales y luego lo leen en entornos de graficación.

Una de las formas con mayor auge actual, es la integración, utilizar python como integrador de librerías de cálculo en C y/o fortran. Esto permite el control de las entradas y salidas para luego la graficación. Se utiliza un compilador fortran-python f2py.

Esto es un poco mas complejo que para una introducción a python.

Cálculos en fortran-Graficación en python

Supongamos que realizaron un cálculo en fortran y los resultados los guardaron en un archivo “ASCII” con datos en forma de columna, todo lo que tienen que hacer en python es leer (o cargar) el archivo en una tabla:

```
tabla = np.loadtxt(archivo)
```

Para el caso en que los datos en fortran fueron guardados en formato binario (grandes bases de datos) es bastante mas complejo, nosotros utilizamos una librería personalizada: fortranfile (disponible pagina gica). unformatted

Archivos de datos con formatos portables: netcdf (network common 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).