

Archivos. Vectores y matrices

Temario

- Lectura y escritura de archivos.
- numpy: vectores. arrays. numpy (vs listas)
- generaciones automáticas: linspace
- multiplicaciones de arrays: matmul, dot,

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 indentación, significa que python tiene que cerrar el archivo.

Lectura de archivos

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

La sintaxis de estas instrucciones es coherente con objetos (recordar que toda estructura en python es un objeto: desde la mas sencilla, como una variable, hasta las mas complejas). En el primero instanciamos el objeto `text_file`. Y luego utilizamos ciertas funciones del objeto `.read()` y `.close()`

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.

El archivo se guarda en formato ascii es decir que si lo abren con un editor de texto pueden leer el contenido.

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, tensores, etc existe una librería específica: Numerical Python “numpy”.

```
import numpy as np
```

En general a los vectores, matrices o tensores de cualquier rango, 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 matrices

Para generar una matriz de n filas por m columnas hacemos

```
mtx=np.zeros([n,m])
```

No olvidar los corchetes.

Para acceder a los elementos de un array se hace de la misma manera que en las listas. Si es un vector

```
a[5]; a[0:2]; a[3:]; a[2,-3]
```

Si es una matriz:

```
a[0,5]; a[3,5:7]
```

Las matrices también pueden definirse a través de listas:

```
a=[[1,2],[3,4]]
```

Para acceder a un elemento de la lista de listas se tiene que hacer `a[0][1]`

Matriz identidad

Si queremos generar la matriz identidad:

```
identidad=eye (N) ;  identidad2=eye (N,M)
```

Si queremos generar un vector o una matriz de unos:

```
unos=ones (N) ;  unos2=ones (N,M)
```

Si queremos extraer la diagonal de una matriz

```
A.diagonal()
```

Operaciones con matrices

Multipliación por un escalar:

```
mtx1=alpha*mtx2; mtx3=alpha*ones(N)
```

Multipliación entre vectores o matrices:

```
mtx1=np.matmul(mtx2,mtx3); alpha=np.dot(v2,v3)
```

Una nueva forma de escribir el producto matricial en python 3 es:

```
mtx1=mtx2 @ mtx3
```

Funciones matemáticas

Evaluación de funciones matemáticas que dependen de arrays:

```
v2=np.exp(-v1); v3=np.log(v1); v4=np.sin(v1)
```

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 `np.linspace` nos genera el vector automaticamente:

```
xvec= np.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?

En numpy tenemos la instrucción: `np.arange(1.5,10,2.3)` similar a `range` pero se puede utilizar con flotantes.

Operaciones con las componentes

Algunas operaciones que podemos realizar con las componentes de un array:

```
a = np.array([2, 4, 3], float)
a.sum()
a.prod()
```

Alternativa:

```
np.sum(a)
np.prod(a)
```

Operaciones estadísticas

```
a = np.array([2, 4, 3], float)
a.mean()
a.std()
a.var()
```

Ejercicio: Tenemos un array pesos donde se tienen todos los pesos medidos y se quiere sacar la media y el error de las mediciones.

Máximo y mínimo

Para obtener el **valor** máximo o mínimo de los elementos de un array se debe hacer:

```
a.min()  
a.max()
```

Si en cambio queremos determinar los índices del elemento donde se encuentra el máximo o mínimo se debe hacer:

```
a.argmin()  
a.argmax()
```

Operaciones lógicas con arrays

Comparación de dos arrays. Esto lo realiza elemento por elemento:

```
>>> a = np.array([1, 3, 0], float)
>>> b = np.array([0, 3, 2], float)
>>> c = a > b
>>> print c
array([ True, False, False], dtype=bool)
```

Igualdad:

```
>>> a == b
array([False,  True, False], dtype=bool)
```

Comparación con un valor:

```
>>> c = a > 2
>>> print c
array([ False,  True, False], dtype=bool)
```


Para todo el array

Si queremos saber si se satisface para cualquier elemento del array o para todo elemento del array. Se realiza primero la operación lógica y luego

```
>>> c = np.array([ True, False, False], bool)
>>> any(c)
True
>>> all(c)
False
```

Selección de elementos

Supongamos que queremos realizar la raíz cuadrada a los elementos positivos de un array.

```
>>> a = np.array([[4, -1, 9]], float)
>>> a >= 0
array([[ True, False,  True], dtype=bool)
>>> np.sqrt(a[a >= 0])
array([ 2.,  3.]
```

De lo contrario se puede guardar en un array lógico:

```
>> sel = (a >= 0)
a = np.array([[4, -1, 9]], float)
>>> np.sqrt(a[sel])
array([ 2.,  3.]
```

Guardado de arrays

En formato ascii:

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

En formato binario (recomendado):

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

La extensión *npy* es la que se utiliza para datos binarios en python. Si uds no la agregan python la tomará por default.

Librería para guardar datos estructurados

En data science existe una librería muy utilizada “pandas” que permite leer y procesar datos estructurados de muy diversa índole.

Un ejemplo, en el GICA se esta realizando un proyecto CORR01-COVID-Federal para el monitoreo y la predicción de la pandemia COVID-19 en Argentina que utiliza pandas para las bases de datos del Ministerio de Salud.

<http://covid19.unne.edu.ar>

Este proyecto lee y procesa los datos (con pandas) en tiempo real y luego utiliza técnicas de machine learning para aprender parametros y la propagación del virus SARS-Cov-2.