Estructuras de los programas

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 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 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:

```
a[5],a[5:7]
```

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:

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

Si queremos extraer la diagonal de una matriz

```
A.diagonal()
```

Operaciones con matrices

Multiplicación por un escalar:

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

Multiplicación entre vectores o matrices:

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

Funciones matemáticas

Evaluacion de funciones matematicas 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 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?

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 maximo o minimo de los elementos de un array se debe hacer:

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

Si en cambio queremos determinar los indices del elemento donde se encuentra el maximo o minimo 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 operacion logica y luego

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

Selección de elementos

Supongamos que queremos realizar la raiz 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 extension *npy* es la que se utiliza para datos binarios en python. Si uds no la agregan python la tomara por default.