Python a la velocidad de C

C

~~Python~~ a la velocidad de C

Por qué?

- Python es mucho más cómodo de usar
- No tenemos que compilar cada vez que cambiamos algo

- En general no usamos Python porque es lento, no suele haber otros motivos.

Cómo?

Usualmente sólo una parte del código lo hace pesado (En ising metrópolis, en MD verlet).

Entonces sería ideal poder programar la parte pesada en C para que corra rápido, y todo el resto del programa hacerlo en Python, donde no nos interesa tanto la velocidad. (Poblar la red, condiciones iniciales de MD, promedios, desviaciones estándar, correlación, etc.) Para las últimas tenemos Numpy y SciPy!

Ctypes

Librería ya instalada en Python. Nos permite convertir los tipos de variables de Python a variables de C, para poder ejecutar funciones de C dentro de python

Esto es todo lo que necesitamos!

1º Paso:

Escribir las funciones en c

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
float Suma(float a, float b)
{
    return a + b;
}
```

2º Paso:

Compilar y generar un archivo .so (.dll en windows)

```
gcc -fPIC -c my_math.c
gcc -shared -o libmy_math.so my_math.o
nm -n libmy_math.so
```

Nota: .so es por Shared Objects, son librerías dinámicas. Carga todo lo necesario a la hora de ejecutar el programa, a diferencia de las librerías estáticas que pegan todas las funciones externas en el código cuando se compila.

De acá en adelante es todo Python

En Python:

```
>>> import ctypes as C
>>> my_math = C.CDLL ("Path/./libmy_math.so")
>>> my_math.Suma.restype = C.c_float
>>> x = my_math.Suma(C.c_float(4.3), C.c_float(2.1))
>>> print(x)
6.400000095367432
```

Importo las funciones de C

Declaro el tipo del return

Ejecuto la función casteando las variables al tipo correspondiente de C

Funciona... Pero es horrible, nadie quiere usar Python así

Solución: Envolvemos la función

```
def Suma(a, b):

    my_math.Suma.restype = C.c_float
    x = my_math.Suma(C.c_float(a), C.c_float(b))
    return x
```

```
>>> Suma(1.1, 3.2)
4.300000190734863
```

Acá ya es indistinguible de una función de Python. Pero se ejecuta en C!!!

Repaso:

- Conseguimos tener una función de C dentro de nuestro programa de Python, que es lo que queríamos.

- Cuando llamamos a esa función, se ejecuta en C, a la velocidad de C. No pasa por el intérprete de Python.

- Con esto podemos armar archivos de Python donde importamos las funciones de C y las envolvemos. Estos archivos los importamos como librerías en nuestros programas, al igual que importamos Numpy.

Punteros:

¿Qué pasa si mi función trabaja con punteros?

```c
float Producto_interno(float *A, float *B, int dim)
{
    float x = 0;
    for(int i = 0; i < dim; i++)
    {
        x += *(A + i) * *(B + i);
    }
return x;
}
```

```python
def Producto_interno(A, B):
    dim = C.c_int(len(A))
    my_math.Producto_interno.restype = C.c_float
    in1 = np.array(A, dtype = C.c_float)
    in2 = np.array(B, dtype = C.c_float)
    intp = C.POINTER(C.c_float)
    point1 = in1.ctypes.data_as(intp)
    point2 = in2.ctypes.data_as(intp)
    x = my_math.Producto_interno(point1, point2, dim)
    return x
```

```
>>> A = [1,2,3]
>>> B = [1,1,1]
>>> x = Producto_interno(A, B)
>>> print(x)
6.0
```

Funciona perfecto para listas y arrays de Python!

Agregamos punteros en el Output

```
void Producto_matrices(float *C, float *A, float *B, int rows_A, int cols_A, int rows_B, int cols_B)
{
    for(int i = 0; i < rows_A; i++)
    {
        for(int j = 0; j < cols_B; j++)
        {
            for(int k = 0; k < cols_A; k++)
            {
                *(C + i*rows_A + j) += *(A + cols_A*i + k) * *(B + cols_B * k + j);
            }

        }
    }
}
```

En Python:

```python
def Producto_matrices(A, B):
    rows_A, cols_A = np.shape(A)
    rows_B, cols_B = np.shape(B)
    c_rows_A = C.c_int(rows_A)
    c_cols_A = C.c_int(cols_A)
    c_rows_B = C.c_int(rows_B)
    c_cols_B = C.c_int(cols_B)
    in1 = np.array(A, dtype = C.c_float)
    in2 = np.array(B, dtype = C.c_float)
    out = np.zeros(rows_A*cols_B, dtype = C.c_float)
    floatp = C.POINTER(C.c_float)
    point1 = in1.ctypes.data_as(floatp)
    point2 = in2.ctypes.data_as(floatp)
    pointout = out.ctypes.data_as(floatp)
    my_math.Producto_matrices(pointout, point1, point2, c_rows_A, c_cols_A, c_rows_B, c_cols_B)
    c = np.zeros([rows_A, cols_B])
    for i in range(rows_A):
        for j in range(cols_B):
            c[i,j] = out[cols_B * i + j]
    return c
```

- Agrego dentro de la función de Python un array para que sea el puntero output.
- Luego de ejecutar la función de C copio el output a un array de Python.

El resultado de usar la función:

```
>>> A = [[1,2],[3,4],[5,6]]
>>> B = [[0,1],[1,0]]
>>> C = Producto_matrices(A, B)
>>> print(C)
[[2. 1.]
 [4. 3.]
 [6. 5.]]
```

Es interesante notar que a las funciones de Python le tenemos que pasar muchos menos parámetros que a las funciones de C, ya que los parámetros que faltan los incluímos cuando definimos la función de Python.

En resumen:

- Podemos usar funciones de C en python, con la misma facilidad que usamos a las funciones de Python

- Las funciones hechas en C van a la velocidad de C, no pasan por el intérprete de Python

---

- Podemos esconder todos los parámetros molestos para que las funciones queden simples de usar

- Ahora sólo tenemos que traducir las funciones una vez, y podemos usar código de Python que corra rápido

Beneficios:

- En general sólo una porción pequeña del código es la que lleva la mayoría del tiempo. Por lo tanto sólo es necesario hacer una pequeña porción del código en C

- Si ya tengo un código lento en Python, no hace falta hacer un código desde cero en C para hacerlo correr rápido

- No tenemos que compilar cada vez que modificamos algo, y es más fácil modificar las cosas en Python (por ejemplo si quiero cambiar una matriz)

Beneficios:

- Ya no tenemos que exportar los datos en txt para luego importarlos en Python y graficarlos. Ya están en Python!

- Podemos usar programación orientada a objetos (Python), que no se puede hacer en C.

Muchas Gracias!