Taller de Física Computacional

NumPy - (a vuelo rasante)

Cristián G. Sánchez y Carlos J. Ruestes

2020

Numpy

Listas

- Las listas son contenedores abstractos para secuencias de objetos de cualquier tipo.
- Las listas son versátiles.
- En computación científica, en general, se trabaja con secuencias ordenadas de números del mismo tipo.
- La versatilidad se transforma en un obstáculo, las operaciones sobre listas son *lentas*

Numpy

`ndarray`

Para facilitar las operaciones sobre secuencias ordenadas de números Numpy implementa una nueva clase (objeto) llamado `ndarray` junto con una serie de métodos y funciones que aceleran estas operaciones.

- Un `ndarray` representa un arreglo multidimensional (vector, matriz, tensor, etc.) **homogéneo**
- Con homogéneo nos referimos a que un `ndarray` contiene elementos que son **todos del mismo tipo.**
- Los tipos que pueden formar parte de un `ndarray` deben poder ser descriptos por un número fijo de bytes.
- Los elementos de un `ndarray` se almacenan (físicamente) en la RAM en un único bloque de memoria.

numpy.dtype

Los elementos de un `ndarray` son `dtypes`. NumPy implementa una serie de tipos numéricos mucho más amplia que Python genérico. Algunos ejemplos son:

`np.bool`	`np.csingle`	`np.int_`
`np.single`	`np.cdouble`	`np.long`
`np.double`	`np.clogdouble`	`np.longlong`
`np.longdouble`	`np.intc`	`np.byte`

Los usuarios pueden definir nuevos `dtypes`.

Creación de un ndarray

Antes que nada importar Numpy. La forma usual es `import numpy as np`

Algunas formas de crear ndarrays

- `A = np.empty((5,5))`
- `A = np.zeros((5,5))`
- `A = np.ones((5,5))`
- `A = np.zeros_like(B)`
- `A = np.array([[0,1],[2,3]])`
- `A = np.zeros((1000,100),dtype=np.cdouble)`

Útiles para crear arreglos con grillas

grillas en $\mathbb{R}$

- `A = np.arrange(0,10,0.1)` Vector de números equiespaciados en 0.1 entre 0 y 10
- `A = np.linspace(0,2*np.pi,1000)` Vector de 1000 números equiespaciados entre 0 y 2π

grillas en $\mathbb{R}^n$

```
xs = np.linspace(0,2*np.pi,100)
xy = np.linspace(0,2*np.pi,100)
xm,ym = np.meshgrid(xs,xy)
```

Indexación

Indexación

- Los elementos de un `ndarray` se indexan como una secuencia pero con una *tupla* de la forma `A[1,2,3]`.
- Para cada índice de la tupla se pueden utilizar todas las formas de indexación que vimos para las secuencias.
- Ejemplos:

```
A[0,0]
A[-1,-1]
A[0:3,4:10:1]
A[:,2]
A[2,:]
C[0:10:2,2,:]
```

Vectorización y *broadcasting*

Los loops sobre elementos de un `ndarray` son lentos. Esta es una limitación intrínseca de Python (que ya hemos discutido). Numpy implementa para los objetos del tipo `ndarray` métodos que permiten hacer operaciones que uno normalmente haría en un loop de otra forma:

broadcasting

Las operaciones entre arreglos y escalares (o arreglos compatibles) se hacen elemento a elemento para todos los elementos del/los arreglo/s.†

vectorización

Una función (vectorizada) aplicada a un arreglo opera sobre todos los elementos, elemento a elemento.

† Para comprender los detalles de la compatibilidad leer el manual.

broadcasting

Sean A, B y C arreglos de tres índices con dimensiones L, M y N. Queremos hacer:

```
for i in range(L):
    for j in range(M):
        for k in range(N):
            # OJO! mismos índices
            C[i,j,k] = (3*(A[i,j,k]**2) + 1.0) * B[i,j,k]
```

En NumPy esto se hace de la forma:

```
C = (3*(A**2) + 1.0) * B # 300 veces más rápido con L=M=N=100
```

vectorización

Sean A, C arreglos de tres índices con dimensiones L, M y N. Queremos hacer:

```
for i in range(L):
    for j in range(M):
        for k in range(N):
            # OJO! mismos índices
            C[i,j,k] = m.exp(m.sin(m.cos(A[i,j,k])))
```

En NumPy esto se hace de la forma:

```
C = np.exp(np.sin(np.cos(A))) # 26 veces más rápido
```

vectorización

Para una función definida por el usuario:

```
def f(x):
    return x**2 + 1

for i in range(L):
    for j in range(M):
        for k in range(N):
        # OJO! mismos índices
            C[i,j,k] = f(A[i,j,k])
```

En NumPy esto se hace de la forma:

```
f_vec = np.vectorize(f)
C = f_vec(A) # 4 veces más rápido
```

Síntesis y recursos:

- Documentación de NumPy
- Objetos `ndarray`
- Bradcasting basics
- Broadcasting theory
- Machete de NumPy que viene en el material de esta clase.