Escalona_Joaquín_9

Octubre 2018

Ejercicio 1

Gracias al gran aporte de **Diego Salvador** pude realizar este ejercicio.
La distribución Gausseana queda representada como:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Apartado A)

```
#-*- coding:utf-8 -*-

import numpy as np
import matplotlib.pyplot as plt

#Se configuran los ejes
plt.axis([-10,10,-10,10])
#Se añade el punto con 'gravedad'
plt.plot(0,0,marker='*',c='r')
n=10
pos=(20*np.random.sample(n*2)-10).reshape(n,2)
vel=(0.1*np.random.normal(size=n*2)).reshape(n,2)
sizes=100*np.random.sample(n)+100
#propiedades de bolas
colors=np.random.sample([n,4])
circles=plt.scatter(pos[:,0], pos[:,1], marker="o", s=sizes, c=colors)


def P(x):
        '''
        Entrega aceleración cuya X esté con distribución normal.
        Las variables son
        mu,sigma
        '''
        mu=(sum(x)/n)                       #Promedio de las posiciones
        sig=10                                    #sigma.
        ra=sig*np.sqrt(2*np.pi)    #raiz de la función
        num=x-mu                            #numerador de la función
        S=10                                        #desface
        pot=-0.5*(num/sig)**2      #exponente
        X=(1/ra)*np.exp(pot)       #distribución gausseana a x
        dist=np.sqrt(X**2+pos[:,1]**2) #Se crea una variable de distancia.
        a=(-1/(S+dist**2))         #Nuevo factor de aceleracion.
        ax=(a/2)*pos[:,0]          #Nueva aceleracion en X.
        return ax
```

```
for i in range(1000):
        x=pos[:,0]    #Se crea una variable con las posiciones originales de x.
        vel[:,0]=vel[:,0]+P(x) #Se añade la aceleracion a la velocidad de x.
        pos=pos+vel
        bounce=abs(pos) > 10
        vel[bounce]=-vel[bounce]
        circles.set_offsets(pos)
        plt.pause(0.05)
```

Ver video aquí.

Apartado B)

```
#-*-coding: utf-8

import numpy as np
import matplotlib.pyplot as plt

#Configuración de ejes

plt.axis([-10,10,-10,10])
#Se añadirá "punto con gravedad"
plt.plot(0,0, marker ='*',c='r')
#Para mayor elegancia (no caos), elegimos 1 bolita
n = 10
pos = (20*np.random.sample(n*2) - 10).reshape(n,2)
vel = (0.1*np.random.normal(size=n*2)).reshape(n,2)
sizes = 100*np.random.sample(n)+100
circles = plt.scatter(pos[:,0],pos[:,1],marker = 'o', s=sizes, c='g')
for i in range(1000):
        #distancia al centro
        dis = np.sqrt(pos[:,0]**2 + pos[:,1]**2) #Distancia al centro
        #aceleracion
        a_x = ((1/(15. + dis**2))/dis) * pos[:,0] #De aquí quitamos el - que antes tenía
        a_y = ((1/(15. + dis**2))/dis) * pos[:,1] #De aquí igual.
        #velocidades
        vel[:,0] +=  a_x
        vel[:,1] +=  a_y
        pos = pos + vel
        bounce = abs(pos)>10
        vel[bounce] = -vel[bounce]
        circles.set_offsets(pos)
        plt.pause(0.05)
```

Ejercicio 2

a) Función Universal

Una función universal es una función que opera en **ndarrays** elemento por elemento. Es un wrapper (contenedor) 'vectorizado' de una función que toma un número fijo de entradas escalares y produce un número fijo de salidas escalares. (fuente: aquí)

b) Es numpy.exp() una funcion universal?

Sí, lo es. Al hacer `type(numpy.exp()` nos dice que este pertenece a **numpy.ufunc**

c) Qué es 'broadcasting'?

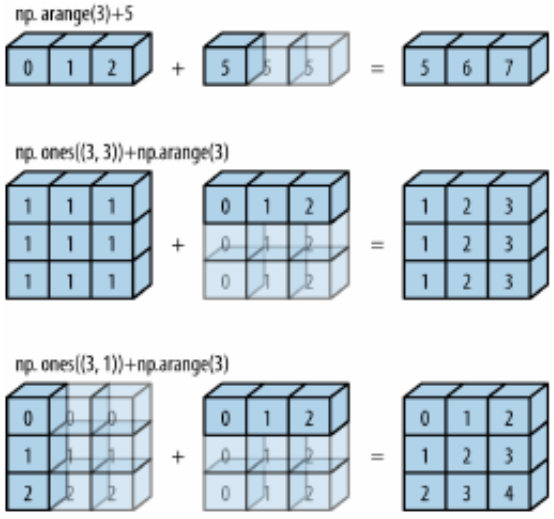
Básicamente *broadcasting* es un conjunto de reglas para aplicar a las **funciones universales binarias** (adición,resta,mutiplicación, etc.). La funcionalidad broadcasting de Numpy se utiliza cuando estamos frente a arrays cuyas dimensiones son distintas.

Por ejemplo,

```
import numpy as np

a = np.array([0,1,2])
a + 5
>>> array([5,6,7])
```

Aquí lo que python hizo fue, por así decirlo, transformar el escalar 5 en un `array[5,5,5]` y sumarlo elemento a elemento con el array a. Esto mismo ocurre a dimensiones más altas. Podemos explicar esto mediante la siguiente imagen.



fuente

Ejercicio 3

```
#-*- coding:utf-8

import numpy as np

f = open('data1.txt','r') #abrimos el archivo

#Leemos e ignoramos las lineas de encabezado
header1 = f.readline()
header2 = f.readline()
header3 = f.readline()

name1 = [] #lista vacia con los futuros datos de name
jmag1 = [] #lista vacia con los futuros datos de jmag

#Hacemos un loop y sobre cada línea y extraimos los datos que queremos
for line in f:
        line = line.strip()
```

```
        columns = line.split()
        name = name1.append(columns[2]) #agregar a name1
        jmag = jmag1.append(float(columns[3])) #agregar a jmag1

np.savetxt('datos.txt',np.column_stack((name1,jmag1)),fmt='%5s',delimiter='\t')
```

Si abrimos el archivo de texto nos econtramos con lo siguiente:

```
00424433+4116085        9.453
00424403+4116069        9.321
00424455+4116103        10.773
00424464+4116092        9.299
00424403+4116108        11.507
00424464+4116106        9.399
00424446+4116016        12.07
```

Ejercicio 4

Apartado 1

```
# por defecto es base=10, realiza espaciados logarítimos.
# En este caso son 11 valores entre -20 y -10.

print(np.logspace(-20,-10,11))
```

```
    [1.e-20 1.e-19 1.e-18 1.e-17 1.e-16 1.e-15 1.e-14 1.e-13 1.e-12 1.e-11
 1.e-10]
```

```
#Aquí repetimos el 2, 10 veces.
print(np.repeat(2,10))
```

```
[2 2 2 2 2 2 2 2 2 2]
```

```
#Creamos un array de numeros cercanos al 0.
print np.empty(10)
```

```
[1.e-323 1.e-323 1.e-323 1.e-323 1.e-323 1.e-323 1.e-323 1.e-323 1.e-323
 1.e-323]
```

```
#Creamos un array de 5 ceros.
print np.zeros(5, dtype=np.float32)
```

```
    [0. 0. 0. 0. 0.]
```

Apartado 2

```
#Creamos un array de diez números al azar.
x = np.random.random(10)
print x
```

```
[0.85773005 0.03399547 0.38807368 0.05452955 0.46931283 0.84700727
 0.87602007 0.54999978 0.77823639 0.10187733]
```

```
#Se imprime la diferencia entre el primero y el último
print x[1:] - x[:-1]
```

```
[-0.82373458  0.35407821 -0.33354414  0.41478328  0.37769444  0.0290128
 -0.32602028  0.2282366  -0.67635906]
```

```
#Se imprime la diferencia entre el primer y el último
print np.diff(x)
```

```
[-0.82373458  0.35407821 -0.33354414  0.41478328  0.37769444  0.0290128
 -0.32602028  0.2282366  -0.67635906]
```

Apartado 3

```
#Importamos el archivo con los datos de las temperaturas
temp = 'munich_temperatures_average_with_bad_data.txt'
#Asignamos variables a las columnas del archivo
date, temperature = np.loadtxt(temp, unpack=True)
#Creamos un filtro para datos menores a 50
keep = np.abs(temperature) < 50
#Aplicamos el filtro a date y a temperature
date1 = date[keep]
temp1 = temperature[keep]
#Imprimimos los valores
print date1, temp1
```

```
[1995.00274 1995.00548 1995.00821 ... 2013.27926 2013.282   2013.28474] [ 0.944444
   -1.61111  -3.55556  ... 10.5556    8.94444  11.1667  ]
```