Trabajo Práctico Nº 0: Introducción a Python/Numpy/Scipy/Matplotlib

Aprendizaje Profundo y Redes Neuronales Artificiales

21 de agosto de 2020

Alumnos :

Tomás LIENDRO tomas.liendro@ib.edu.ar

Docentes :

Ariel CURIALE
Germán MATO
Lucca Dellazoppa

San Carlos de Bariloche, Argentina

Índice

1. Ejercicio 1 **1**

2. Ejercicio 2 **1**

3. Ejercicio 3 **2**

4. Ejercicio 4 **2**

5. Ejercicio 5 **3**

6. Ejercicio 6 **3**

7. Ejercicio 7 **4**

8. Ejercicio 8 **4**

9. Ejercicio 9 **4**

10.Ejercicio 10 **5**

11.Ejercicio 11 **6**

12.Ejercicio 12 **7**

13.Ejercicio 13 **8**

14.Ejercicio 14 **9**

15.Ejercicio 15 **10**

En este trabajo se realizará una breve mención de los ejercicios correspondientes a la Guía de Trabajos Prácticos N°0 y se expondrán los resultados obtenidos. El objetivo de este trabajo práctico es familiarizarse con las librerías de Python Numpy, Matplotlib y Scipy.

1. Ejercicio 1

El primer ejercicio consiste en desarrollar un código en Python para resolver el sistema de ecuaciones lineales (SEL):

$$\begin{aligned} x + z &= -2 \\ 2x - y + z &= 1 \\ -3x + 2y - 2z &= -1 \end{aligned}$$

El SEL anterior, puede ser expresado matricialmente como:

$$A \cdot \vec{x} = \vec{b},$$

donde $\vec{x}$ es el vector de incógnitas, A es la matriz de coeficientes y $\vec{b}$ es el vector de términos independientes. Este sistema puede resolverse mediante la operación:

$$\vec{x} = A^{-1} \cdot \vec{b}.$$

Utilizando la librería Numpy se pueden definir estas matrices y el resultado obtenido es:

$$\vec{x} = \begin{bmatrix} -4 \\ -\frac{5}{2} \\ +\frac{9}{2} \end{bmatrix}$$

2. Ejercicio 2

En el segundo ejercicio se solicita calcular el histograma, la media (μ) y la desviación estándar (σ) de un conjunto de datos generados de manera aleatoria bajo una distribución de tipo Gamma con los valores de los parámetros de 3 para el parámetro de forma k, 2 para el parámetro de escala θ y con un total de 1000 puntos según lo indica la línea de comando:

```
data = np.random.gamma(3,2,1000).
```

Para hallar el valor medio y la desviación estándar se utilizaron las funciones de la librería de Numpy: **np.mean(array)** y **np.std(array)**. Estas funciones retornaron los valores de $\mu = 5{,}86$ y $\sigma = 3{,}44$. Para graficar el histograma se utilizó la librería matplotlib y la función **matplotlib.pyplot.hist(data)**. El histograma resultante fue:

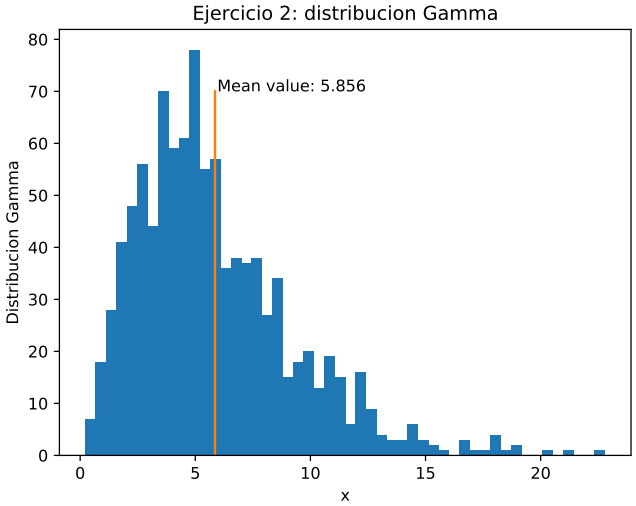


Figura 1: Ejercicio 2: distribución Gamma

Finalmente, para verificar los resultados anteriores se calculó el valor teórico y se obtuvo:

$$\mu_{teo} = k\theta = 6$$

$$\sigma_{teo} = \sqrt{k\theta^2} = 3{,}46$$

3. Ejercicio 3

El tercer ejercicio consiste en definir una función para encontrar las raíces de una ecuación de segundo orden utilizando la expresión de Bhaskara:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}. \tag{3.1}$$

La función debe recibir como parámetros de entrada los coeficientes a (obligatorio), b y c (opcionales). Para ello, cuando se define la función que resuleve la ecuación, se fija el valor por defecto de 0 para los parámetros de entrada b y c. La función definida, calcula en un primer instante el discriminante arg de la raíz cuadrada de la expresión **??**. Si arg es mayor o igual a cero, se calculan las raíces x_1 y x_2 y se retorna un Numpy array que contiene ambas raíces. Si el discriminante arg es negativo, se definen las raíces x_1 y x_2 como variables complejas cuya parte real es $\frac{-b}{2a}$ y la parte imaginaria es $\frac{\sqrt{|arg|}}{2a}$. Finalmente, se retornan las raíces como un Numpy array.

Los resultados obtenidos para el caso $a = 22$ es $(x_1, x2) = (0, 0)$. Para $a = 22,\ b = 22$ el resultado es $(x_1, x2) = (-1, 0)$. Y para $a = 22,\ b = 22,\ c = 22$ el resultado es: $(x_1, x2) = (-0{,}5 + 0{,}866j, -0{,}5 - 0{,}866j)$.

4. Ejercicio 4

El ejercicio 4 solicita definir una función que grafique una parábola en el rango de valores $x\ in[-4{,}5, 4]$ a partir de los coeficientes de la parábola $a,\ b,\ c$. Esta función también debe marcar las raíces de la parábola con puntos rojos y textos que indiquen sus valores con tres decimales utilizando Latex.

Para este ejercicio se importó la función Bhaskara definida en el ejercicio 3, que permite encontrar las raíces de un polinomio de segundo orden. Lo primero que se hizo fue definir, dentro de la función, el conjunto x que contiene los valores de -4.5 a 4 utilizando la función **np.linspace(-4.5,4,1000)**, donde 1000 es la cantidad de

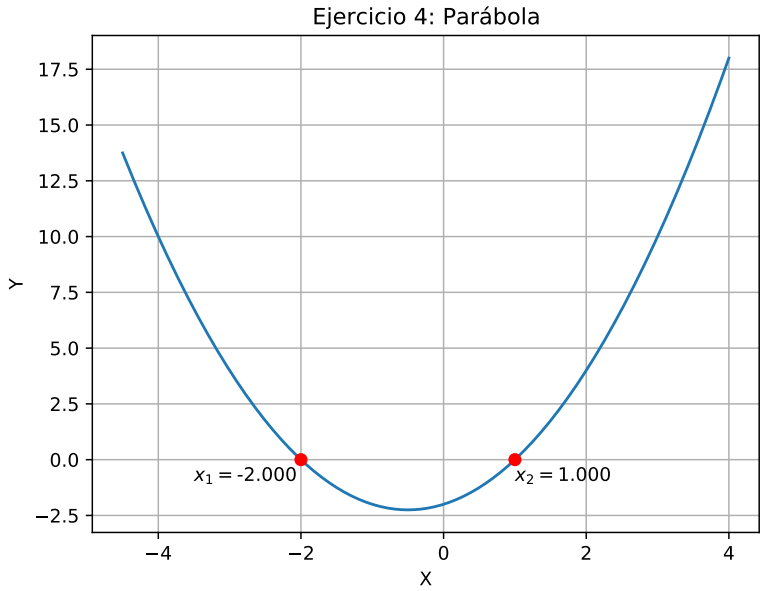


Figura 2: Ejercicio 4

elementos dentro del arreglo x. Luego, se calculó el arreglo y mediante la función $y[i] = a * x[i]^2 + b * x[i] + c$, donde a, b, c son parámetros de entrada de la función.

De la librería **Matplotlib.pyplot**, utilizamos la función **plot** para graficar la parábola y llamando la función Bhaskara con los mismos parámetros de la parábola se puede encontrar y graficar las raíces de la parábola. El resultado de ejecutar la función definida para los parámetros de la parábola $a = 1,\ b = 1,\ c - 2 =$ es:

5. Ejercicio 5

Para el ejercicio 5 se solicita crear una clase $'Lineal'$ que reciba los parámetros a, b y un método que resuelva la ecuación lineal $ax + b$. El método se define con el método $\_\_call\_\_(self, x)$. De esta manera se puede llamar la clase como una función haciendo:

```
a = 1
b = 2
x = 3
f = Lineal(a,b)
solution = f(x)
```

El resultado de llamar el método de la clase Lineal da 5, que es el resultado correcto.

6. Ejercicio 6

En este ejercicio la idea es heredar la clase $'Lineal'$ desde una nueva clase definida $'Exponencial'$. Al heredar la clase $'Lineal'$, la nueva clase tiene acceso a las variables a y b y se define un método $\_\_call\_\_(self, x)$ dentro de la clase $'Exponencial'$ para que retorne el resultado de ax^b. Si se toma $a = 1, b = 2$ y $x = 3$ el resultado obtenido es 9, que es el resultado correcto.

7. Ejercicio 7

En el ejercicio 7 hay que crear un módulo çircunferencia.py"que contenga la constante PI y la función **area(r)**, donde r es el radio de la circunferencia. El valor de PI se lo toma de la librería Numpy importando el valor de PI y la función **area(r)** retorna el valor de πr^2.

En otro archivo, se importa este módulo con dos comandos distintos:

```
import circunferencia as circle
```

y

```
from circunferencia import PI, area
```

Se importó además la constante PI de la librería Numpy para verificar que al importar la constante desde el módulo circunferencia el valor sea el correcto. Si los valores de PI se verifican, se calcula el área utilizando los dos métodos para importar haciendo:

```
area1 = circle.area(10)
```

y

```
area2 = area(10).
```

Tanto area1 como area2 valen 314, sin embargo son distintos objetos. Esto se debe a que se importan de distinta manera por lo que se almacenan en distintos espacios de memoria. Con el comando **id(area1)** y **id(area2)**, se ve que los ID son distintos, por lo que son objetos distintos.

8. Ejercicio 8

En este ejercicio se debe crear un directorio (paquete) llamado 'geometria', el cual contenga el módulo **circunferencia** creado en el ejercicio anterior y un módulo **rectangulo** que contenga la función **area** para calcular el área de un rectángulo de lados a y b.

Para esto creamos una carpeta que contiene ambos módulos (archivos de formato .py). Si querémos usar una función contenida en alguno de estos módulos debemos importar el paquete y el módulo haciendo por ejemplo:

```
from geometria.rectangulo import area
from geometria.circunferencia import area, PI
```

o bien,

```
import geometria.rectangulo as rect
import geometria.circunferencia as circle
```

9. Ejercicio 9

Para que los comandos:

```
import p0_lib
from p0_lib import rectangulo
from p0_lib.circunferencia import PI, area
from p0_lib.rectangulo import area as area_rect
```

funcionen correctamente se creó un directorio (paquete) llamado **p0_lib** con los módulos **rectangulo** y **circunferencia**. Dentro del módulo **rectangulo** se define la función **area** que calcula el área de un rectángulo

de lados a y b. Dentro del módulo **circunferencia** se define la función **area** que retorna el valor del área de un círculo de radio r y además se define la constante PI. En un caso, se importa la función área del archivo **rectangulo.py** con el alias **area_rect** y la función **area** queda reservada para la función con el mismo nombre definida en el archivo **circunferencia.py**. Otra manera de acceder a la función **area** del módulo **rectangulo** sería haciendo: **p0_lib.rectangulo.area** o bien **rectangulo.area**. Y otra manera para acceder a la función **area** del módulo circunferencia sería mediante la línea: **p0_lib.circunferencia.area**. No se puede llamar mediante **circunferencia.area** ya que no está importado explícitamente este módulo.

10. Ejercicio 10

La idea de este ejercicio es replicar la siguiente imagen:

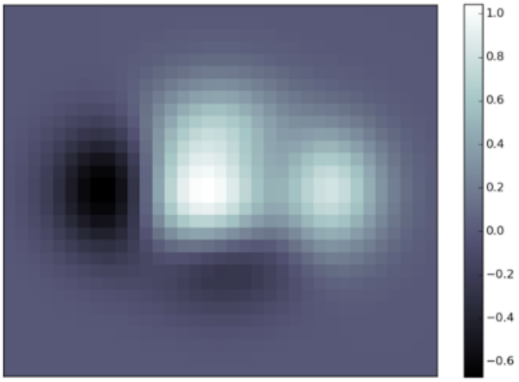


Figura 3: Ejercicio 10: imagen a replicar

utilizando la siguiente ecuación:

$$\left(1 - \frac{x}{2} + x^5 + y^3\right) \cdot e^{-x^2-y^2} \tag{10.1}$$

Para ello se definieron los vectores x e y con la función **np.linspace** y se definió el conjunto de puntos del espacio **(X,Y)** definido por los vectores x e y utilizando la función **np.meshgrid**. Graficando con **plt.imshow(f(X,-Y)), cmap='bone'** donde **f** es la función definida anteriormente y **'bone'** es el tipo de *colormap*, se obtiene la siguiente gráfica:

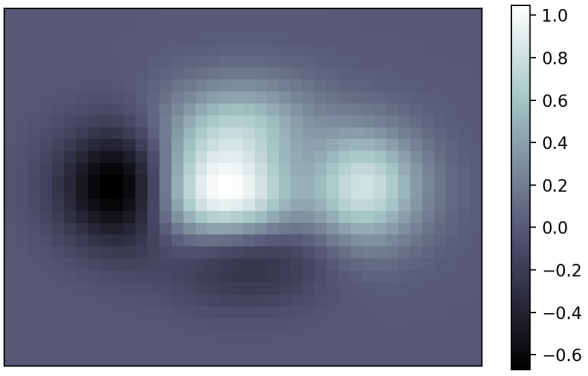


Figura 4: Ejercicio 10: resultado

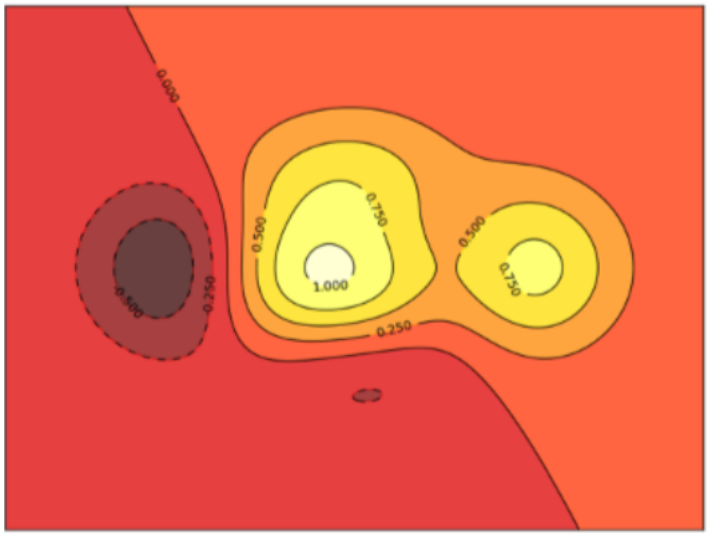


Figura 5: Ejercicio 11: imagen a replicar

Para lograr este resultado se utilizaron las funciones **plt.xticks()** y **plt.yticks()** para quitar los ejes x e y, y la función **plt.colorbar(shrink=0.82)** para fijar la barra de referencia a la derecha escaleado en 0.82 para que replique la imagen original.

11. Ejercicio 11

En el ejercicio 11 se debe replicar la imagen de la Fig. 5 utilizando la función definida por la ecuación 10.1 y el **colormap** denominado **hot**. En este caso, la cantidad de elementos en el vector **x** es 1024 y en el eje **y** es 768.

Al igual que en el ejercicio anterior, se definen los vectores X e Y como

```
X,Y = np.meshgrid(x,y)
```

para generar el espacio bidimensional con los valores de x e y.

Con la función **plt.xticks(())** y **plt.yticks(())** se quitan los ejes de la figura. Se utiliza la función de Matplotlib **plt.contourf** para graficar contornos pintados por niveles, contour. Esta función recibe como parámetros: un array Z que resulta de evaluar la ecuación 10.1 cada punto del espacio (X,Y), el colormap *'hot'* y un factor de transparencia *alpha* de 0.8. Con la función **plt.contour** se pueden graficar los contornos de las superficies y se le pasan como parámetros: Z, el color definido como negro y el ancho de la linea *'linewidths'* definido en 0.8 puntos. Finalmente, con la función **plt.clabel** se escriben los label (etiquetas) con los valores en cada nivel. Esta función recibe como parámetros: el conjunto de contornos **cnt** obtenidos con la función:

```
cnt = plt.contour(Z, colors='black', linewidths=0.8),
```

un vector que contiene los valores de cada nivel de interés **[-0.5, -0.25, 0, 0.25, 0.5, 0.75, 1]**, el color definido como *'black'* y el tamaño de la fuente (*'Fontsize'*) definido en 7 puntos.

El resultado obtenido es:

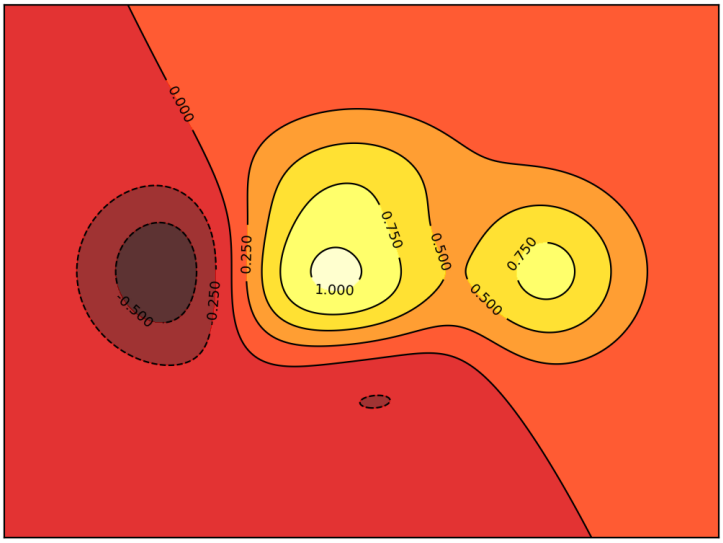


Figura 6: Ejercicio 11: resultado

12. Ejercicio 12

En este ejercicio se solicita replicar la siguiente imagen:

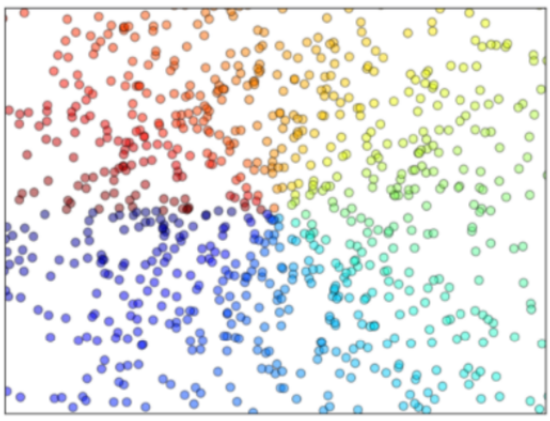


Figura 7: Ejercicio 12: imagen a replicar

Esta imagen se genera a partir de puntos cuya posición se obtiene de manera aleatoria con la función de Numpy **np.random.normal(0,1,n)** que genera n (1024 en este caso) valores aleatorios con valor medio 0 y desviación estándar 1. Se utiliza la función **plt.scatter** para graficar los valores generados como puntos en el espacio. A la función **scatter** se le pasa como argumento los vectores **X**,**Y** que contienen los valores aleatorios, **c** que es un arreglo que define los colores de cada punto, el colormap del tipo *'rainbow'*, un factor de transparencia de **alpha** con un valor de 0.5, el color del bode (**edgecolors**) definido como *'black'* y el ancho de las líneas con un valor de 0.5 puntos. El arreglo **c** surge de calcular la arcotangente (**np.arctan2(Y,X)**) del cociente de la ordenada de un punto y su abscisa.

El resultado obtenido es:

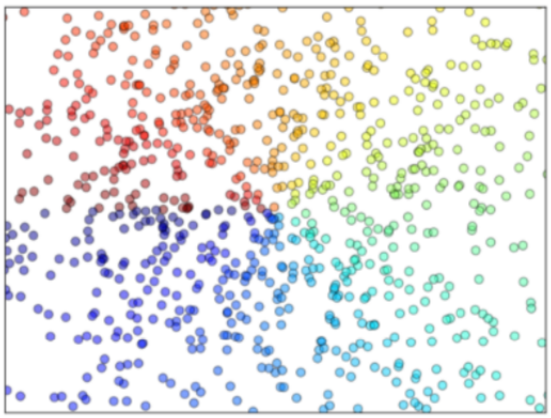


Figura 8: Ejercicio 12: resultado

13. Ejercicio 13

La idea de este ejercicio es simular el comportamiento de peces cuando interatúan con otros peces formando un cardúmen en un espacio bidimensional.

Inicialmente, se define la clase **R2** que contiene los atributos **x** e **y**. Esta clase permite representar un objeto que se encuentra en el espacio bidimensional. Luego, se define la clase **Pez** con los atributos **pos**, **vel** y **maxVelPez** que se refieren a la posición, velocidad y máxima velocidad que puede tener un pez. Los atributos **pos** y **vel** se definen objetos como de la clase **R2** y la velocidad máxima es un parámetro inicializado con un valor entero, en este caso 30.

Dentro de la clase Cardumen, se definen los atributos **size** para limitar las posiciones iniciales de los peces, **_maxVel** como un atributo privado para limitar las velocidades iniciales de los peces, **_maxDist** como un atributo privado para indicar la mínima distancia admisible entre los peces, **dt** para indicar un paso de tiempo (de 0.1), **npeces** para indicar la cantidad de peces en el cardumen (i.e. 16), **pez** que es un arreglo (inicialmente vacío) donde cada elemento se incializa como un objeto de la clase **Pez** y **centro cardumen** que es de la clase Pez y que representa la posición y velocidad del centro del cardumen.

Luego se define el método **initialize**, cuyos argumentos de entrada son **size**, **maxVel** y **maxDist**. En este método se actualizan los valores de **_maxVel**, **_maxDist** y **size**. Este método crea un arreglo **pez** donde cada elemento es un objeto de la clase *Pez* y se le asigna una posición y velocidad de manera aleatoria entre los valores límites de la posición (**size**) y velocidad (**maxVel**).

A continuación se define un método **calcular_centro** para calcular la posición y velocidad del centro de masa del cardumen. Para ello se calcula el promedio de la posición y velocidad de todos los peces.

Un tercer método **calc_deltaVel** inicializa un objeto local **dv** de la clase *R2*, cuyo valor se calcula sumando (por separado para la coordenada **x** e **y**) los resultados de tres ecuaciones que modelan el comportamiento de los peces en un cardúmen (ver guía de Trabajos Prácticos N°0). Este método retorna el valor de **dv**.

El método **doStep** permite calcular las nuevas posiciones y velocidades de los peces y actualizarlas. Para esto, debe primero llamar al método **calcular_centro** y actualizar la posición y velocidad del centro del cardumen y en función de estas coordenadas y las posiciones y velocidades de cada pez se puede calcular las variaciones (deltas) de velocidad de cada pez y usando la fórmula:

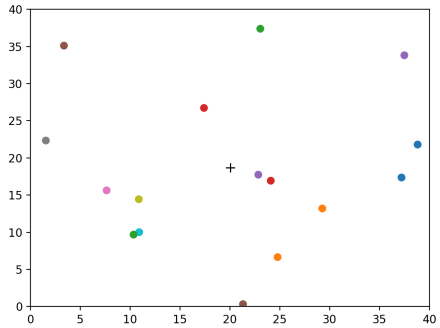
$$r_i(t + d_t) = r_i(t) + v_i(t) * \delta_t$$

donde r_i es la posición del i-ésimo pez y $v_i(t) = v_i(t-1) + \delta v$. Cuando se calcula el nuevo valor de la velocidad, debe tenerse en cuenta que la velocidad resultante no sea mayor a la máxima velocidad. En caso de que se

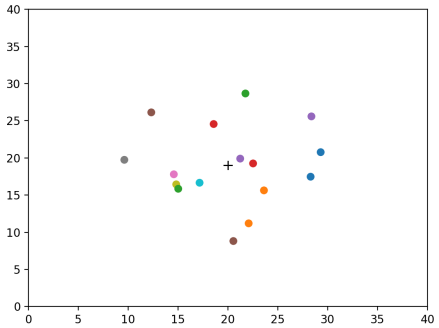
supere la máxima velocidad luego de calcular la nueva velocidad, se debe mantener la velocidad que se tenía inicialmente.

Finalmente, un método **print** permite graficar la evolución en cada instante de tiempo de los peces interactuando en el cardúmen.

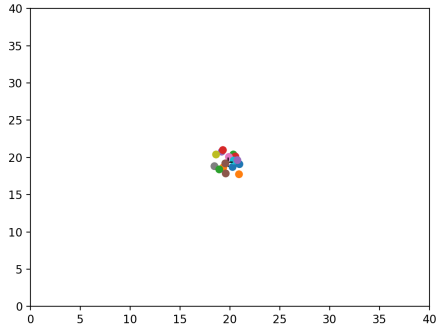
El resultado del comportamiento de los peces se muestra en las siguientes imágenes:



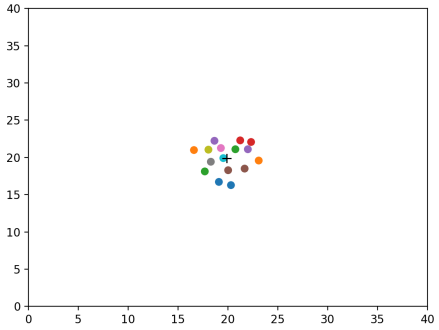
(a) Condición inicial del cardumen



(b) t=1



(c) t=2



(d) t=3

Figura 9: Evolución de la posición de los peces en el cardúmen

14. Ejercicio 14

Para este ejercicio se solicita encontrar la probabilidad de que al menos dos personas cumplan años el mismo día en grupos de distintos tamaños.

Lo primero que se hizo fue crear una clase *Persona* que contiene los atributos de **mes** y **día** de cumpleaños asignados de manera aleatoria. Por otro lado, se define una clase *Grupo* que recibe como parámetro de entrada un arreglo con los posibles tamaños. Dentro esta clase, se elige de manera aleatoria un elemento del arreglo y queda definido así un tamaño del grupo. Se inicializa un arreglo **grupo** dentro de la clase *Grupo*. A través de un método **iniciar** dentro de la clase, se define cada elemento del arreglo **grupo** como un objeto de la clase *Persona*, por lo que a cada elemento del grupo le queda asignado un mes y día de cumpleaños. Finalmente, se llama a un método **verificar** que detecta si en el grupo se repitió alguna fecha de cumpleaños. En caso afirmativo, el método retorna un 1, en caso negativo un 0.

En el *main* del programa, dentro de un ciclo *for*, se simulan 1000 experimentos donde se inicializa un objeto de la clase *Grupo* y se almacena en un .txt la información del tamaño del grupo y el *flag* que vale 1 cuando se repitió algún cumpleaños o 0 si no se repitió ninguno.

Con otro ciclio *for* se lee la tabla generada en .txt, y se hace un recuento de la cantidad de simulaciones

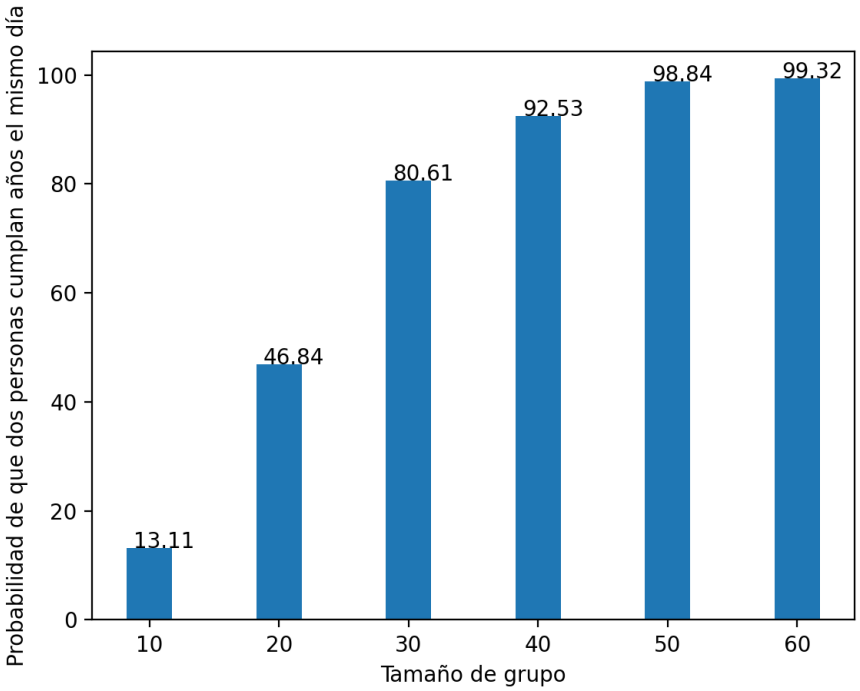


Figura 10: Probabilidad de que al menos dos cumpleaños se repitan en grupos de 10, 20, 30, 40, 50 y 60 personas.

realizadas con un tamaño de grupo y la cantidad de veces en que para ese tamaño ocurrió alguna repetición de fecha de cumpleaños. Finalmente se calcula y grafica la probabilidad de que haya ocurrido una repetición en la fecha de cumpleaños y se obtiene la siguiente gráfica:

15. Ejercicio 15

En este ejercicio se busca implementar un functor Noiser que se activa con una variable de tipo **float**. Cuando se activa, le suma un valor aleatorio contenido entre dos umbrales **minV** y **maxV**. Para ello se define la clase *Noiser* que recibe como parámetros de entrada los valores de umbral mínimo **minV** y máximo **maxV**.

Se define además un método **__call__** que recibe un argumento de entrada **x**, de manera que si el tipo de argumento es un *float* entonces se devuelve el valor de **x** sumado a un valor aleatorio dentro de los umbrales **minV** y **maxV**.

Finalmente, en el *main* del código, se incializa un objeto **noiser** de la clase *Noiser* y se define un objeto **g** como una función vectorial que transforma la función escalar **noiser** (que es llamada como función al haber implementado el método __call__) de la clase *Noiser* en una función vectorial. De esta manera, la función vectorial **g** aplica la función **noiser** al arreglo **array**, y aplica dicha función a cada elemento del arreglo.

Definiendo los umbrales como **minV=-0.5** y **maxV=0.5** se obtiene el siguiente resultado de aplicar el functor al arreglo **array=[1, 1.3, 3.4, 5]**:

```
[1.         1.13736537 3.17187554 5.        ]
```

De aquí se ve que el functor solo toma los valores de tipo *float* para sumarle un valor aleatorio entre -0.5 y 0.5 y en caso de que el elemento del arreglo sea un entero, lo ignora y devuelve el mismo elemento.