Práctica 0: Introducción a Python, Numpy, Matplotlib y Scipy.

Evelyn G. Coronel
Aprendizaje Profundo y Redes Neuronales Artificiales
Instituto Balseiro

(21 de agosto de 2020)

EJERCICIO 1

Para este ejercicio se utiliza la librería `numpy.linealg`. Se inicializa una matriz A de los coeficientes de las ecuaciones, así como también el vector b para trabajar con $A \cdot x = b$.

Para solucionar este sistema de ecuaciones, se utiliza la expresión $x = A^{-1}b$. Primero debe verificarse la existencia de una solución única, para esto calculamos el determinante de A. Si la matriz A no es invertible, entonces $\det(A) = 0$, por lo que el programa termina ahí. Caso contrario, se usa la función

```
x= np.linalg.solve(A, b)
```

EJERCICIO 2

En este ejercicio se inicializa un conjunto de datos aleatorios de 1000 puntos, dados según la función de la distribución Γ. Los parámetros de forma para la función son: `shape=3, scale=2`. Se calcula la media y desviación estándar mediante la librería `numpy`.

Con los parámetros `shape, scale` y la librería `scipy.stats` se grafica la distribución esperada. Posteriormente se grafica y se obtiene el histograma de la Fig. 1 usando la librería `matplotlib.pyplot`, cambiando los parámetros del gráfico mediante `matplotlib.rcParams.update(...)`.

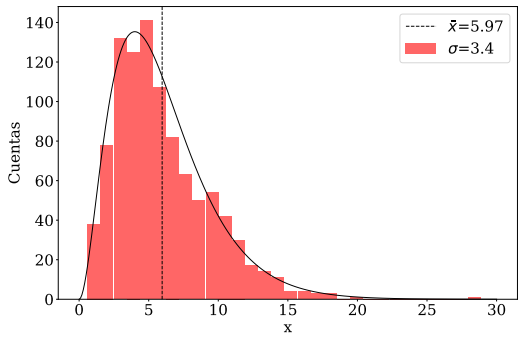

Fig. 1: Ejercicio 2

EJERCICIO 3 Y 4

El ejercicio 3 consiste en resolver un ecuación cuadrática $ax^2 + bx + c = 0$ dada la fórmula, donde los parámetros b y c son opcionales. El ejercicio 4 consiste en graficar la función y ubicar sus raíces en el mismo gráfico.

Para realizar estos ejercicios, se implementaron las siguientes funciones para garantizar la validez de los parámetros ingresados por el usuario.

- `parametros()`:

  Esta función verifica que los parámetros a, b y c ingresados por el usuario sean válidos. Para el caso de b y c, al no ser inicializados, toman un valor predeterminado nulo.

- `cuadratica(a,b,c)`:

  Verifica que la ecuación tenga al menos una solución real. Si existen devuelve un `array` con las soluciones, caso contrario devuelve `None`

Una vez que el usuario ingresó los parámetros correctamente y se verificó que la ecuación tiene al menos una solución real, se puede graficar la cuadrática y sus raíces. En la Fig.2 se muestra $f(x) = 3x^2 + 5x - 1$ y sus raíces. Para graficar las mismas se utilizó la función `scatter`, para las flechas `annotate` y las anotaciones `text`. No se utiliza `annotate` para el texto porque la función `text` deja un mejor aspecto en el caja que contiene al texto.

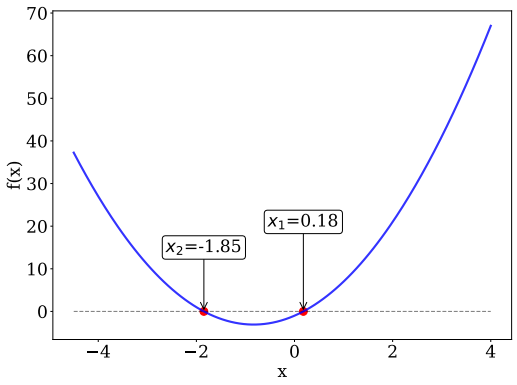

Fig. 2: Ejercicio 4

EJERCICIO 5

En este ejercicio se inicializa la clase `Lineal` que busca implementar una función lineal $f(x) = ax + b$. Dentro de esta clase se definen las siguientes funciones:

- `solucion(y)`: Dado los parámetros a y b y un valor de $f(x) = y$, se calcula el valor de x. Este programa tiene una "flag" cuando el parámetro $a = 0$, donde la solución no está definida.
- `f(x)`: Evalúa la expresión dado un x.
- `__str__`: Imprime la expresión $f(x) = ax + b$ a la salida.

EJERCICIO 6

Este ejercicio es análogo al anterior, salvo que la clase es `Exponencial` y la expresión es $f(x) = ax^b$.

EJERCICIO 7

Para este ejercicio se creo un librería llamada `circunferencia.py`, donde se definen la constante `PI`, que equivale a π, y la función `area(r)` que acepta el valor del radio y devuelve al área.

En la implementación se importan la constante y la función del área de dos formas distintas:

- `import circunferencia as circle`
- `from circunferencia import area, PI`

Se observa que al ejecutar el comando `circle.PI is PI` se obtiene `True`, análogamente para `circle.area is area`, lo que indicarían que las formas de llamar a estas entidades devuelven el mismo tipo de constante o función.

EJERCICIO 8

Se creó un archivo llamado `geometria.py`, donde se llamaron los módulos `circunferencia` y `rectangulo`.

EJERCICIO 9

Para generar la carpeta con la librería `p0_lib`, primeramente se creo una carpeta con ese nombre. Se agregó un archivo en blanco llamado `__init__.py` para que Python inicialice la carpeta como librería.

Luego en el código principal de la práctica 0 `practica_0_DNN_main.py` se llamó a esta librería, como indica la práctica, de manera exitosa.

EJERCICIO 10

Para graficar la Fig.3 se ejecutó el comando `meshgrid` con los vectores `x` e `y` inicializados en el código de la práctica.

Al comando `imshow` se agregó el parámetro de color `bone`, además de agregar la barra de colores y quitar los números de los ejes.

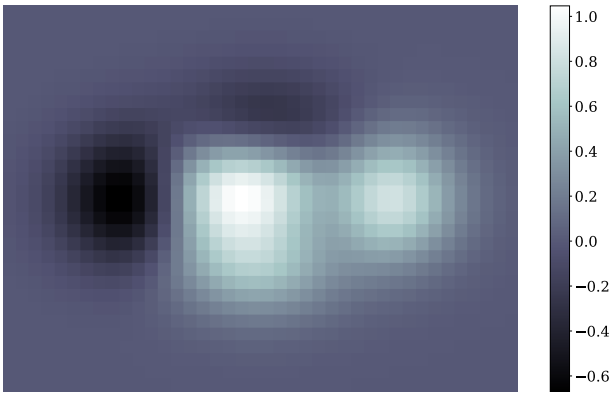

Fig. 3: Ejercicio 10

EJERCICIO 11

Continuando con la función del ejercicio anterior, se usaron las funciones:

- `contourf`: para separar el gráfico en 6 zonas según el valor de la función.
- `contour`: para dibujar las líneas y separar las zonas.
- `clabel`: para agregar el valor que representan estas líneas en el gráfico.

De esta forma se obtuvo el gráfico presentado en la Fig. 4.

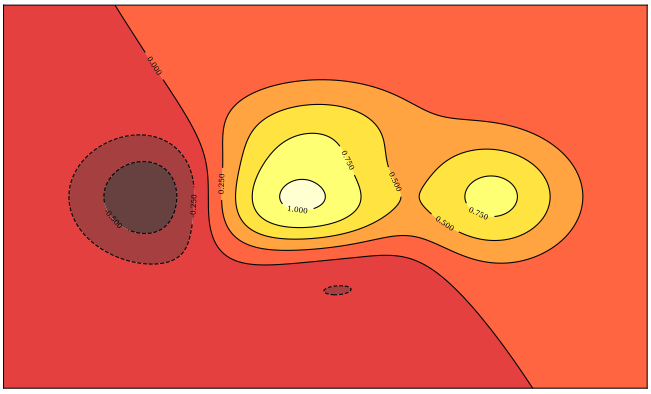

Fig. 4: Ejercicio 11

EJERCICIO 12

Para realizar este gráfico, se tomaron 1024 puntos en el plano XY para graficarlos con la función `scatter`. Usando

el mapa de colores `jet`, un tamaño de puntos 70 y que cada punto se le asigne un color por su ángulo mediante la función `arctan2(x,y)` se obtiene la Fig. 5.

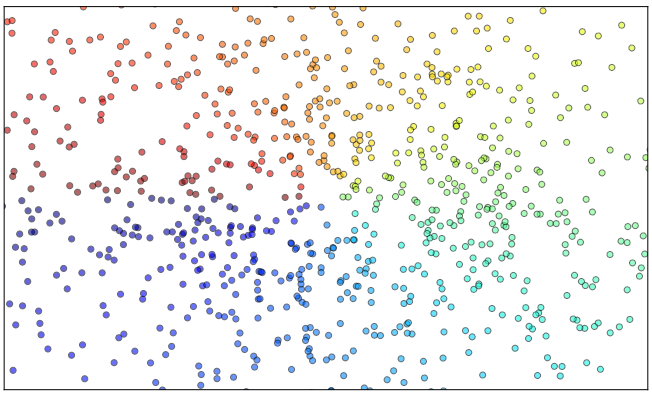

Fig. 5: Ejercicio 12

EJERCICIO 13

En este ejercicio se simula un modelo de juguete para el movimiento de un cardumen. El mismo consiste en 3 reglas, mencionadas en la práctica, que limitan la velocidad que debe tener cada pez.

Esta simulación se creo una clase `R2` que intenta recrear las operaciones usuales en vectores de dos dimensiones. Para ello, se utilizo el *overloading* de funciones del tipo suma, resta, multiplicación y división por escalares.

También se creó una clase `Pez`, que guarda la posición y velocidad de un pez. Finalmente se creó también la clase `Cardumen`, que guarda información sobre todos los peces y a su vez contiene métodos para modificar al cardumen.

Para la inicialización de los peces, se tomaron posiciones aleatorias dentro de un cuadrado de 40×40 y las velocidades aleatorias con un módulo menor a una velocidad límite `maxVel=4`. Durante la ejecución del programa, para la actualización de la velocidad de un pez según la regla 2, se tuvieron en cuenta los peces que están a una distancia de `d=3`.

Durante la ejecución del programa, se actualiza una ventana de `matplotlib` en cada iteración, además se toma un tiempo muerto de 80 ms entre iteración para dar la impresión de ver una simulación en vivo. De esta forma se facilita ver la evolución del cardumen pero no es útil para guardar la animación en un archivo o para terminar la ejecución del programa sin correr todas las iteraciones.

Condiciones de contorno

El enunciado del ejercicio no habla de las condiciones de contorno, por lo que en este trabajo se simularon con dos condiciones: paredes duras o sin paredes. La primera simularía a un cardumen en un estanque, con la condición de que si llega a la pared, el sentido de la velocidad cambia. En cambio sin paredes, esta condición no se aplica

Sin paredes duras

En este caso el cardumen puede moverse en todo el espacio. Durante la ejecución del código, la figura que se esta imprimiendo sigue a los peces en caso de salirse del área inicial, como se ve en la Fig. 6

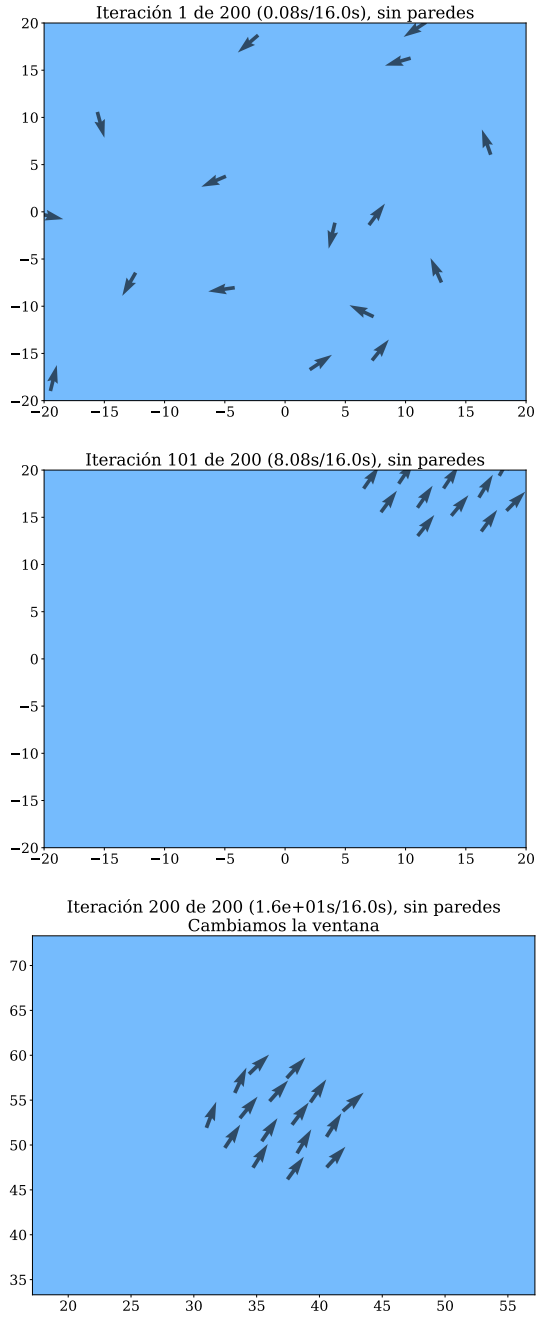

Fig. 6: La simulación del ejercicio 13 sin considerar condiciones de contorno.

Con paredes duras

Para este caso el comportamiento de los peces se ven afectado cuando llegan a la pared. En la Fig. 7 se muestra que durante la mitad de la simulación, los peces se acercan con la pared, y ya para el final de la simulación se alejan de la pared que encontraron.

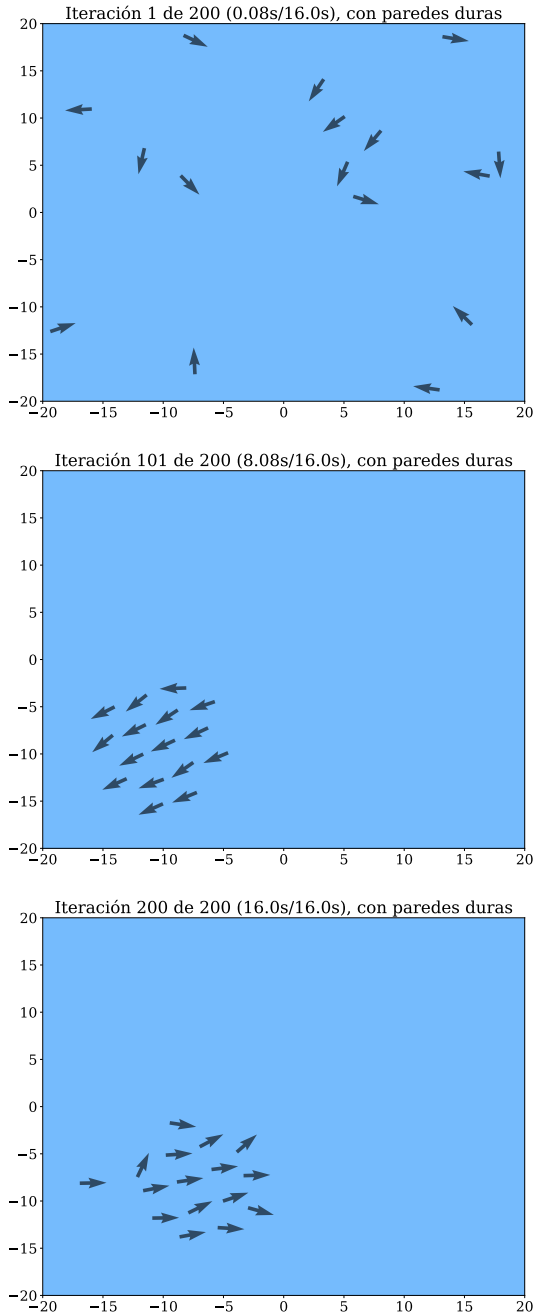

Fig. 7: La simulación del ejercicio 13 considerando paredes duras como condiciones de contorno.

EJERCICIO 14

Utilizando la librería `numpy.random` se generaron números entre el 1 y 365, excluyendo así al posible caso donde alguien haya nacido el 29 de Febrero, para ser asignados a las personas dentro de los grupos.

Una vez que cada persona se le asigna un día, se recorre grupo por grupo para verificar si existen por lo menos dos personas con el mismo número asignado, es decir, con el mismo cumpleaños.

La simulación arrojó las valores presentados en la Tabla I. Como se observa en los datos, así como también en la Fig. 8, la probabilidad va convergiendo al 100 % a medida que se aumenta la cantidad de personas por grupo.

Personas	Probabilidad
10	9.8 %
20	39.3 %
30	71.5 %
40	88.9 %
50	97.6 %
60	99.1 %

Tabla I: Tabla de la cantidad de personas por grupo y la probabilidad que en ese grupo, dos personas hayan nacido el mismo día.

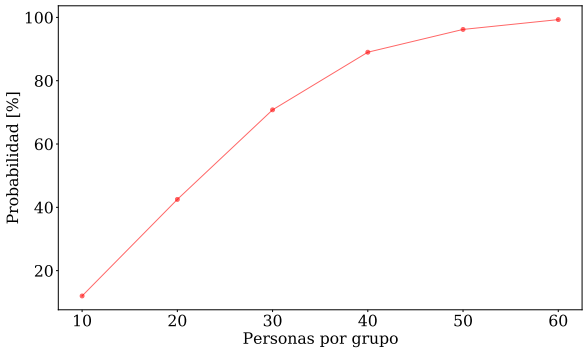

Fig. 8: Ejercicio 14

EJERCICIO 15

Para este ejercicio se crearon dos clases, `functor` y `Noiser`, donde el último hereda las propiedades del primero. En la inicialización de `functor` definimos los atributos `minV` y `maxV`, que en la función `__call__` se utilizan para devolver un número aleatorio entre estos dos valores.

Para la clase `Noiser` se hereda la inicialización y para la función `__call__` acepta un argumento. Para la salida ahora se agrega la función `__call__` de functor y se suma el argumento.

Para poder usar estas clases y sus métodos sobre alguna señal almacenada en un `array`, se utiliza la función

`np.vectorize`. En la Fig.9 se muestra una señal sinusoidal que pasó por el Noiser, con un ruido entre $[-0.2, 0.2]$

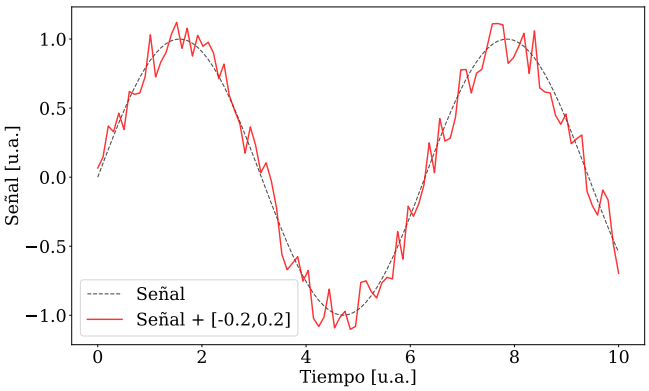

Fig. 9: Ejercicio 15