



INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE COMPUTO



2024

# SESIÓN DE LABORATORIO 1: INTRODUCCIÓN A PYTHON

## ALGORITMOS BIOINSPIRADOS

**Elaborado por:** Flores Estopier Rodrigo

**Profesor:** Rosas Trigueros Jorge Luis

**Grupo:** 6CV2

**Fecha de realización:** 3/09/2024

**Fecha de entrega:** 10/09/2024

**Semestre:** 25-1

## Contenido

1. Marco Teórico .....	2
2. Material y Equipo.....	3
3. Desarrollo .....	3
Lista de 100 números aleatorios de punto flotante .....	4
Extremos de una lista .....	4
Valores de una función.....	5
4. Conclusiones .....	8
5. Bibliografía .....	8

## 1. MARCO TEÓRICO

Los algoritmos bioinspirados son una clase de técnicas computacionales que toman inspiración en procesos biológicos para resolver problemas de optimización y búsqueda. Ejemplos notables incluyen algoritmos genéticos, optimización por enjambre de partículas y redes neuronales artificiales. Estos algoritmos son utilizados en una variedad de aplicaciones, como la ingeniería, economía, biomedicina, entre otros.

Para implementar algoritmos bioinspirados, es esencial tener una base sólida en lenguajes de programación que permitan realizar cálculos numéricos y manipular datos de manera eficiente. En este contexto, Python se ha convertido en una herramienta crucial debido a su simplicidad, amplia comunidad y bibliotecas especializadas como NumPy y Matplotlib.

Se hará uso de las siguientes técnicas para el desarrollo de la presente práctica.

- Generación de números aleatorios.
  - En muchos algoritmos, es común generar números aleatorios para representar poblaciones iniciales o muestras.
- Búsqueda de valores mínimos y máximos de una lista.
  - En la búsqueda de soluciones o en la optimización, encontrar los valores extremos de una función o conjunto de datos es fundamental.
- Generación de graficas de funciones periódicas.
  - Los algoritmos bioinspirados a menudo requieren la evaluación de funciones matemáticas continuas. La función seno es una de las más básicas y representa un fenómeno periódico, siendo útil para modelar oscilaciones y ciclos en sistemas biológicos.
- Calculo de la tangente y derivadas numéricas.
  - La tangente de una función en un punto es una línea recta que representa la pendiente de la función en ese punto. El cálculo de pendientes o derivadas es fundamental para algoritmos de optimización que buscan puntos donde la tasa de cambio es mínima o nula (puntos de máximo, mínimo o puntos de inflexión).

En esta práctica, se utilizarán técnicas numéricas básicas y visualización de datos para introducir conceptos esenciales de programación en Python que serán útiles en el desarrollo de algoritmos bioinspirados.

## 2. MATERIAL Y EQUIPO

Para la realización de la práctica se necesitará:

- Computadora o Laptop con acceso a internet
- Cuenta de Google con acceso a Google Colaboratory

## 3. DESARROLLO

Comenzamos creando un archivo nuevo del Google Colab en Google Drive, la opción para crear dicho archivo es presentada en la Ilustración 1.

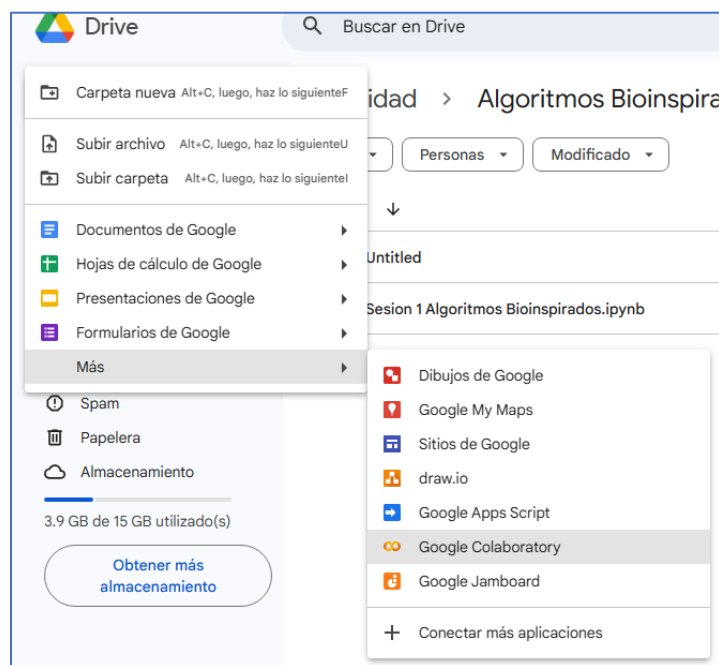


Ilustración 1. Ubicación de la opción para crear archivo de Google Colab

Una vez creado el archivo podemos comenzar a realizar las actividades propuestas en la presente práctica, el entorno de Colab nos permite hacer anotaciones y bloques de código independientes que nos permitirán entender mejor el funcionamiento del código que estamos realizando.

### LISTA DE 100 NÚMEROS ALEATORIOS DE PUNTO FLOTANTE

Se sugiere utilizar la librería numpy para realizar esta tarea. Por lo tanto, debemos importar la librería para poder usar sus métodos, entre las cuales incluye la función **numpy.random.uniform(low, high, tamaño)**, que permite generar números aleatorios dentro de un intervalo especificado, donde los primeros dos argumentos son el límite inferior y superior respectivamente, mientras que el ultimo argumento es la cantidad de números que deseamos generar.

Se generará una lista de 100 números aleatorios flotantes entre -1000 y 1000, en la Ilustración 2 se muestran los resultados obtenidos.

```
import numpy as np

random_list = np.random.uniform(-1000,1000,100)
random_list
```

```
array([-989.86540398,  316.99490352, -340.70440212,  449.70190037,
       -83.52246391, -378.18212566, -582.57510051,  441.62784841,
        567.0024227 ,  474.81065974,  373.28305645,  691.26272158,
        340.6727378 ,  264.20852412,  -74.44541776,  -92.24291754,
       -797.31326835,  799.6960888 , -996.30660662,  857.08173679,
       -149.83891482,  335.06109144, -135.20612038,  601.72330175,
        886.0993937 ,  804.11041536,  369.92003394,  635.57734007,
       -823.96513543,  838.00171793,  773.09933943, -269.49696279,
       -677.61779517, -451.50704643,  111.51268519,  357.57305069,
        934.82302581,  453.53948017,  -77.94789606, -463.52707779,
       -883.37148601, -284.4650472 ,  474.49797561,  721.2448418 ,
        361.27962874,  312.69409445, -272.52221669,  592.80189561,
         17.3851816 ,  408.92189536,  774.90833369,  171.18878949,
        632.56811636,  40.9027399 , -594.14217934,  985.60247538,
        137.67422558, -146.80026696,  223.02386911, -223.14409296,
       -97.29239623,  20.38433325, -317.00750016,  121.59752714,
        961.05821331,  85.02675166, -124.91747843,  511.68703453,
        274.96840995,  748.0675841 ,  725.22481946,  712.94868004,
         89.10750873,  -8.05284902,  -58.00984416, -801.07656182,
       -322.39944926,  179.82365844,  295.73291666, -242.06454947,
        402.29279904, -136.28948218,  186.20030359,  320.02039598,
       -670.73808129,  789.60952696,  943.24075864, -156.0988265 ,
        174.40978592,  676.00868544, -808.3951824 , -270.12696246,
         67.55439273,  712.48992958,  501.71081128, -545.83616896,
        919.31017985, -309.93437551,  655.39959194,  282.8782341 ])
```

Ilustración 2. Código y ejecución del primer ejercicio.

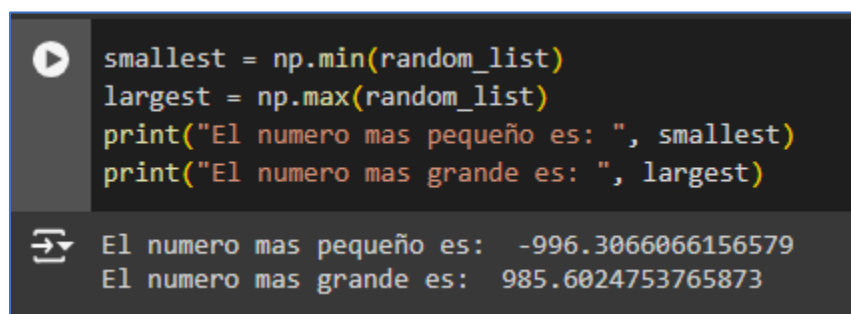
### EXTREMOS DE UNA LISTA

Ahora debemos identificar el valor más grande y el más pequeño de la lista generada. De nuevo, se hará uso de los métodos proporcionados por numpy.

Tenemos dos formas de realizar este tipo de búsqueda, el uso de una u otra forma dependerá de las necesidades que tengamos al desarrollar un programa.

- La función **min()** y **max()** pueden emplearse para identificar los valores más extremos.
- Para encontrar los dos mínimos y máximos, puede utilizarse **sorted()** junto con rebanadas (**slicing**).

En la Ilustración 3 observamos los resultados obtenidos.



```
smallest = np.min(random_list)
largest = np.max(random_list)
print("El numero mas pequeño es: ", smallest)
print("El numero mas grande es: ", largest)
```

```
El numero mas pequeño es: -996.3066066156579
El numero mas grande es: 985.6024753765873
```

Ilustración 3. Código y ejecución del segundo ejercicio.

### VALORES DE UNA FUNCIÓN

Ahora debemos obtener los valores de los primeros dos periodos de la función  $f(x) = \sin\left(\frac{2\pi t}{5} + 0.2\right)$ , una función sinusoidal modificada, y guardarlos en una lista.

Para producir dos periodos de esta función, es necesario calcular sus valores en un rango de puntos **t**, y luego graficar **f(t)** contra **t**.

La biblioteca **Matplotlib** será utilizada para la visualización de la gráfica. Visualizar funciones es clave para entender mejor su comportamiento.

El período (T) de una función trigonométrica, donde (P) es el período, se puede calcular directamente a partir de la fórmula de la función. En este caso, el período (P) es el valor que acompaña a (t) en el denominador dentro del argumento del seno.

Para la función dada, el período (P) es 5, ya que es el valor que multiplica a (t) en el denominador del argumento del seno. Esto significa que la función completa un ciclo completo de sus valores cada 5 unidades de (t).

También se utilizará numpy para generar los vectores de tiempo y la función senoidal.

Para generar el vector tiempo utilizaremos **np.linspace(inicio,final,muestras)** donde estableceremos un vector de 0 a 10 (2 periodos de la función) con 300 muestras.

Para generar los valores de la función utilizaremos **np.sin(funcion)** donde utilizaremos el vector **t** generado anteriormente.

Por último, para generar la gráfica utilizaremos **plt.plot(x, y)**, y posteriormente **plt.show()** para mostrarla en pantalla.

Los resultados obtenidos se visualizan en la Ilustración 4.

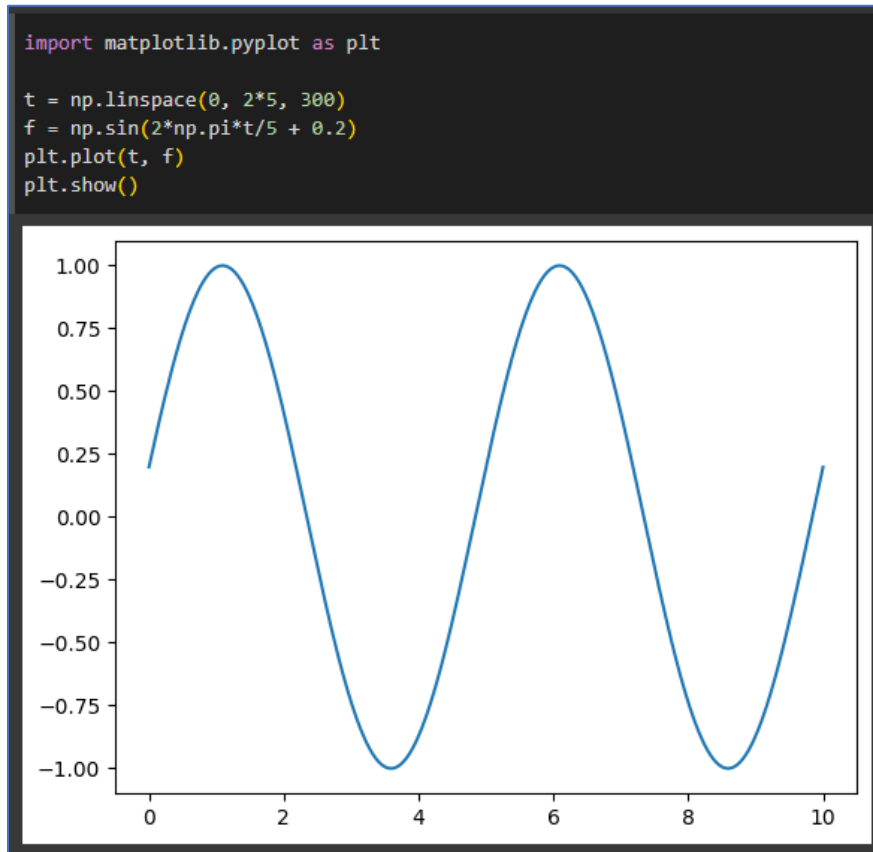


Ilustración 4. Código y ejecución del ejercicio 3.

Por último, debemos encontrar los puntos de la función donde la tangente es cercana a cero, utilizando la secante.

Se muestra a continuación el código realizado y los resultados obtenidos en su ejecución.

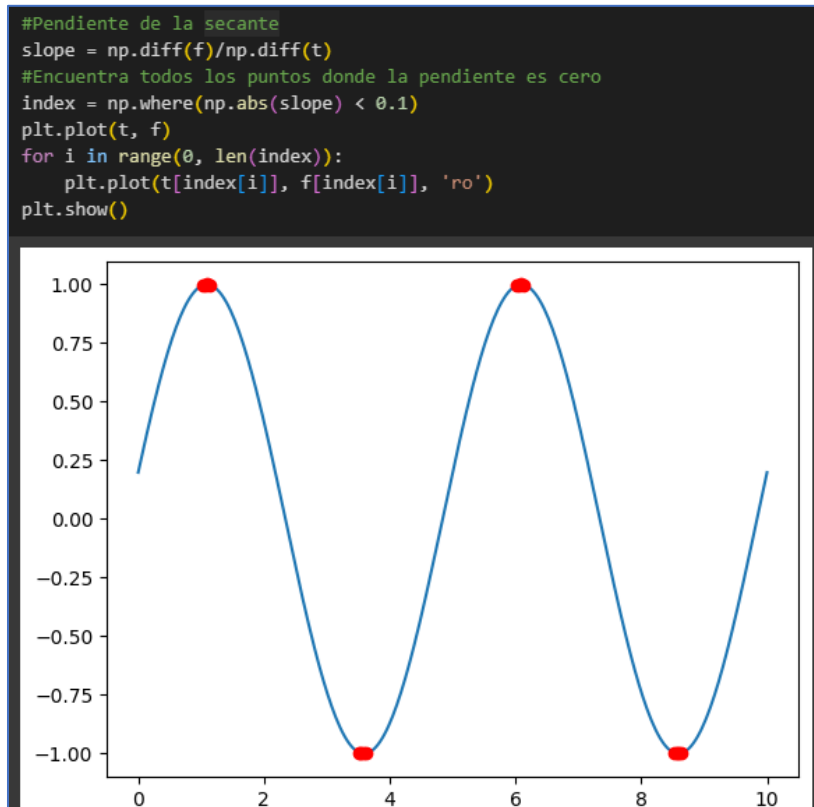


Ilustración 5. Código y ejecución de la segunda parte del ejercicio 3.

Primero calculamos la pendiente aproximada de la función  $f(t)$  usando la **secante** entre puntos consecutivos. Para hacerlo, se utiliza la función **np.diff()**, que calcula la diferencia entre cada valor de  $f(t)$  y su siguiente valor, así como entre los valores consecutivos de  $t$ . Esta operación aproxima la derivada de la función, permitiendo estimar cómo cambia  $f(t)$  en relación con  $t$ , es decir, la pendiente.

Después, buscamos todos los puntos donde la pendiente es cercana a cero. Esto se logra usando la función **np.where(condición)**, que devuelve los índices donde la condición especificada es verdadera. En este caso, se está buscando cuando la pendiente es menor que un valor usando **np.abs(valor)** para tomar el valor absoluto de la pendiente. Por último, generamos la gráfica y luego marcamos con puntos rojos ('ro') los lugares donde la pendiente es cercana a cero. Para ello, recorre todos los índices encontrados previamente con **index** y dibuja los puntos correspondientes.



## 4. CONCLUSIONES

La práctica permitió familiarizarnos con el uso de Python y sus bibliotecas más comunes para el análisis numérico, como NumPy y Matplotlib, aplicadas en la resolución de problemas matemáticos. Aprendimos a generar listas de números aleatorios, identificar valores extremos y graficar funciones trigonométricas, lo cual es fundamental en el contexto de los algoritmos bioinspirados.

Realice un experimento adicional modificando el valor de pendiente en la búsqueda de los puntos donde la función tiene pendiente cercana a cero. Al ajustar este valor, pude observar cambios en los puntos detectados, lo que muestra la importancia de seleccionar parámetros adecuados para obtener resultados precisos en los algoritmos.

## 5. BIBLIOGRAFÍA

*Introduction to NumPy.* (n.d.). W3schools.com. Retrieved September 8, 2024, from

[https://www.w3schools.com/python/numpy/numpy\\_intro.asp](https://www.w3schools.com/python/numpy/numpy_intro.asp)

*Quick start guide — Matplotlib 3.9.2 documentation.* (n.d.). Matplotlib.org. Retrieved

September 8, 2024, from

[https://matplotlib.org/stable/users/explain/quick\\_start.html](https://matplotlib.org/stable/users/explain/quick_start.html)