Universidad Del Norte

Estructuras Discretas

Grupo 01

Informe Proyecto Computacional PC-AGUACATE-202310

Apresa Echeverria Rubens Andre, 200161783
Gómez Rosales Laura Sofía, 200161861
Espinel Luna Luis, 200149985

Prof. Alfonso Mancilla

22 de mayo de 2023

Índice

1. Abstract

This report explores the study and application of ordinary generating functions (OGFs) and symbolic method in combinatorial analysis and counting problems. The first section covers two methods for obtaining OGFs for both sequences and recursive functions, demonstrating their practical use for problem-solving in Python. In the second section, OGFs are used to solve counting problems, providing closed-form expressions for recursive functions and simplifying the calculation of possible cases. The third section introduces a Telegram bot developed in Python to solve recurrence relations and to make requests related to stars and constellations. The bot provides the option to display all stars, a graph of all stars and a chosen constellation, or all stars and constellations. Additionally, the third delivery includes the design of recursive functions in R2, graphing tensors of size (nf, nc, 3) with an initial content of nf $\times$ nc $\times$ 3 ones. Finally, a video presentation showcases the entire project, highlighting the usefulness and versatility of OGFs and symbolic method in combinatorial analysis and problem-solvin

2. Resumen

Las funciones generadoras ordinarias es una serie de potencias cuyos coeficientes corresponden a los términos de una secuencia, lo que permite transformar problemas con secuencias en problemas con funciones. Por su parte, el método simbólico proporciona definiciones básicas que permiten definir estructuras combinatorias. Por lo tanto, el trabajo consistió en el estudio de las funciones generadoras ordinarias y su aplicación en problemas de conteo y análisis combinatorio.

En la primera entrega se presentaron dos métodos para obtener la FGO, sea mediante una sucesión o una función recurrente, así mismo para situaciones problema como los de método simbólico. También se resolvieron distintos problemas con el lenguaje de programación Python.

En la segunda entrega, se aplicó FGO para resolver situaciones problema de conteo. Se mostró que estas funciones permiten obtener expresiones cerradas para funciones recurrentes, y dado un valor para n objetos, es posible obtener la cantidad de posibles casos, lo que facilita la solución de problemas de conteo mediante operaciones sencillas como la suma y multiplicación de polinomios.

Por último, en la tercera entrega se desarrolló un bot de Telegram haciendo uso de Python que permite obtener la función no recurrente de una sucesión, función recurrente o una FGO en pocos pasos junto con la realización de gráficos de estrellas y constelaciones. Por otro lado, el diseño de funciones recursivas en R^2 graficando tensores tamaño (nf, nc, 3).

3. Introducción

La teoría de las Funciones Generadoras Ordinarias y el Método Simbólico son herramientas fundamentales en el análisis combinatorio y la teoría de la probabilidad. En este trabajo se aborda de manera integral el estudio de estas técnicas a través de tres entregas que contienen ejercicios y aplicaciones prácticas.

En la primera entrega, se profundiza en la obtención de Funciones Generadoras Ordinarias y funciones no recurrentes a partir de sucesiones y funciones recurrentes. Además, se aborda el Método Simbólico para la construcción de estructuras combinatorias, resolviendo situaciones problema de cadenas binarias, ternarias y cuaternarias. Se demuestra su aplicación en la resolución de problemas utilizando el lenguaje de programación Python, así como se resulven otro tipos de problemas, como: gráficar el tablero de ajedrez, mostrar el triangulo de pascal, crear fractales polinomicos, graficar la sucesión de FGOs, graficar en 3D coordenadas rectangulares, polares y cilindricas y en 2d el gradiente decendente, solucionar funcion recurrentes homogeneas y no homogeneas, hallar la sucesion de FGO y FGE, y por ultimo Tensores de 1 a 6 dimensiones

La segunda entrega se centra en la aplicación de las FGO en la resolución de problemas de conteo y probabilidad, utilizando la suma de polinomios y la multiplicación. También se desarrolló un programa en Python para simular la compra de láminas de un álbum del mundial, utilizando librerías de Numpy y Pandas para el análisis de datos de acuerdo resultado arrojado por las simulaciones, graficando asi las frecuencias, la media, la moda y la mediana.

En la tercera entrega se realizó un bot de Telegram en Python para resolver funciones recurrentes, facilitando la solución de problemas de conteo y análisis combinatorio previamente planteados como función o sucesión, tambien hacer peticiones relacionadas con las estrellas y constelaciones. Por otro lado, diseñaron funciones recursivas en R^2 graficando tensores tamaño (nf, nc, 3) de acuerdo a una función. Por último, se realizó un video sobre el proyecto completo, el cual muestra los objetivos alcanzados en cada una de ellas.

4. Objetivos

1. Profundizar en el estudio y mostrar la aplicabilidad de las Funciones Generadoras Ordinarias y el Método Simbólico, dos herramientas fundamentales en el análisis combinatorio y la teoría de la probabilidad.

2. Abordar de manera integral el estudio de estas técnicas a través de tres entregas que contienen ejercicios y aplicaciones prácticas.

3. Resolver distintas situaciones, problemas mediante el lenguaje de programación Python

4. Aplicar análisis de datos a partir de simulaciones y gráficas generadas

5. Diseñar un bot de Telegram en Python para resolver funciones recurrentes, y permitir peticiones relacionadas con las estrellas y constelaciones.

6. Determinar la utilidad de ChatGPT en el desarrollo del proyecto

7. Realizar un video sobre el proyecto completo para demostrar el conocimiento y habilidades adquiridos en el estudio y aplicación de las Funciones Generadoras Ordinarias y el Método Simbólico en la resolución de problemas de conteo y análisis combinatorio.

5. Problema de investigación

El problema de investigación que se aborda en este trabajo es el análisis combinatorio y la teoría de la probabilidad, específicamente la aplicación de las Funciones Generadoras Ordinarias y el Método Simbólico para la resolución de problemas de conteo y análisis combinatorio. Además, se busca explorar la implementación de estas técnicas en el lenguaje de programación Python para resolver situaciones problema concretas y complejas de manera práctica y accesible. Asimismo, se plantea el desafío de diseñar y desarrollar un bot de Telegram que permita resolver funciones recurrentes y FGO, facilitando la solución de estos problemas. Por último, se busca ampliar la perspectiva de la investigación mediante la exploración de la astronomía, mediante el análisis y visualización de datos de estrellas y constelaciones. Así mismo, desarrollar y analizar datos estadísticos a partir de múltiples simulaciones de probabilidad.

6. Justificación de la investigación

La justificación de esta investigación es demostrar la utilidad y aplicabilidad de las FGO y el Método Simbólico en la resolución de problemas de conteo y probabilidad. Además, se busca fomentar la comprensión y el interés en la teoría de la probabilidad, al proporcionar herramientas prácticas, como un bot de Telegram y que puedan ser utilizadas por estudiantes y profesionales en el campo de las matemáticas y la informática.

Esta investigación es relevante, ya que el análisis combinatorio y la teoría de la probabilidad son áreas fundamentales en la matemática aplicada y en la resolución de problemas en diversos campos como la ingeniería, la economía, la física, entre otros.

Por otro lado, la investigación permite conocer los alcances de ChatGPT para generar ideas iniciales que permitan al programador conocer librerías y métodos, que ayuden a optimizar tanto la forma de pensar, como de resolver programando un problema.

7. Marco Teórico-Conceptual

A partir de los antecedentes y aplicaciones de las teorías y herramientas utilizadas, se plantean los conceptos para comprender lo que se ha implementado en este proyecto.

7.1. Antecedentes

Las **Funciones Generadoras**, según Lehman, Leighton y Meyer [12], es una serie de potencias que se usa para codificar una secuencia de números. Estas funciones se utilizan comúnmente en la teoría de la probabilidad, el análisis combinatorio y la teoría de la complejidad algorítmica, pueden, pueden ser utilizadas para contar objetos discretos. Esta herramienta matemática es útil para el análisis de la complejidad de los algoritmos y puede ser relevante para la resolución de funciones recurrentes mediante la técnica de FGO.

El **método simbólico** ha sido ampliamente utilizado en el análisis de algoritmos y en la teoría de la combinatoria. Según Sedgewick y Flajolet [1], este método se basa en la generación de funciones, las cuales permiten representar una secuencia de números en términos de una fórmula explícita. A partir de estas funciones, es posible analizar la complejidad de algoritmos y obtener información sobre diversas propiedades combinatorias.

En particular, el método simbólico ha sido utilizado en la solución de problemas recurrentes en la teoría de la combinatoria. Como mencionan Graham, Knuth y Patashnik [2], este método permite obtener fórmulas cerradas para el número de estructuras combinatorias de interés, como los árboles, grafos, permutaciones, entre otros.

El **análisis combinatorio** se encarga de estudiar las propiedades de los conjuntos finitos y sus elementos, según Rosen, el análisis combinatorio es una herramienta fundamental para el estudio de algoritmos, ya que permite determinar la complejidad de los mismos.

Las **funciones no recurrentes**, Graham, Knuth y Patashnik [2] explican que estas funciones son utilizadas para resolver ecuaciones lineales no homogéneas con coeficientes constantes, que se presentan en problemas de análisis combinatorio y probabilidades.

Numpy es una biblioteca para la computación numérica en Python. Proporciona una gran cantidad de funciones matemáticas para realizar operaciones de matriz y álgebra lineal en matrices y matrices multidimensionales [10].

Pandas es una biblioteca que proporciona estructuras de datos de alto nivel y herramientas de análisis de datos para Python. Permite la manipulación y el análisis de datos tabulares y series de tiempo [9].

Numpy y Pandas, junto con las estadísticas básicas como la media, moda y mediana, son herramientas importantes en el análisis de datos y en la investigación en general.

En el contexto de la informática, el **ChatGPT** se refiere a un modelo de lenguaje basado en la arquitectura GPT (Transformador Generativo Preentrenado), desarrollado por OpenAI. Este modelo utiliza técnicas de aprendizaje automático y procesamiento de lenguaje natural para generar respuestas coherentes y contextualmente relevantes en conversaciones escritas. Su capacidad para comprender y generar texto humano-like ha hecho del ChatGPT una herramienta valiosa para mejorar la experiencia del usuario y automatizar tareas en múltiples dominios[20].

Por otro lado, en el ámbito de la astronomía, la **identificación y estudio de estrellas y constelaciones** ha sido un tema de interés desde hace siglos. En la actualidad, existen diversas herramientas y recursos para la visualización de estrellas y constelaciones, como los atlas y mapas estelares [3]. Con el avance de la tecnología, se han desarrollado aplicaciones y herramientas en línea para la visualización y estudio de objetos celestes. En particular, Telegram ha ofrecido una plataforma para el desarrollo de bots que permiten realizar diversas tareas y consultas.

Un **tensor** es un arreglo multidimensional, es decir, una matriz generalizada con un número arbitrario de dimensiones. En el contexto de aprendizaje automático y aprendizaje profundo, los tensores a menudo se utilizan para representar datos, como imágenes, que tienen múltiples dimensiones, como la altura, el ancho y los canales de color [18].

7.2. Aplicaciones

Algunas aplicaciones a las temáticas tratadas son:
En el campo de la informática, las funciones generadoras se utilizan para analizar la complejidad de los algoritmos y para contar el número de estructuras de datos. Por ejemplo, en el trabajo de investigación de T. Hagerup y J. Katajainen, "Succinct representations of some combinatorial objects,"se utiliza FGO para representar de forma eficiente las estructuras de datos utilizadas en la programación dinámica.

En el ámbito de la estadística, las funciones generadoras se utilizan para modelar la distribución de probabilidad de una variable aleatoria. En el artículo de D. Aldous y P. Diaconis, "Shuffling cards and stopping times,"se utiliza FGO para analizar la distribución de probabilidad de un mazo de cartas después de que se ha barajado varias veces.

En biología molecular, el análisis combinatorio se utiliza para estudiar la estructura de las moléculas de ADN y proteínas. En el trabajo de investigación de A. Frieze y B. Pittel, .ºn the number of perfect matchings in random graphs,"se utilizan técnicas combinatorias para estimar el número de estructuras secundarias posibles en una molécula de ARN.

En la investigación científica, se han utilizado tanto Numpy como Pandas para el análisis de datos de imágenes médicas [19].

En el campo financiero, Numpy y Pandas se han utilizado en la gestión de carteras y la evaluación de riesgos [11].

Para el procesamiento de lenguaje natural (NLP, por sus siglas en inglés), los tensores se utilizan para representar palabras y frases en un modelo de procesamiento de lenguaje natural, como en el modelo de lenguaje GPT-3. En este sentido, los tensores pueden ayudar a comprender la semántica y el contexto de los textos[15].

El ChatGPT ha demostrado ser una herramienta versátil en diversos campos de la informática, como el procesamiento de lenguaje natural, la generación de texto y el desarrollo de asistentes virtuales. En la vida diaria, el ChatGPT tiene aplicaciones prácticas, como brindar soporte en línea, asistir en la búsqueda de información, y facilitar la comunicación y la interacción en plataformas digitales[20]

En el área de visión por computadora, los tensores se utilizan para representar imágenes y videos, como en la red neuronal convolución VGG-16. Los tensores pueden ayudar a detectar objetos, clasificar imágenes y reconocer patrones en imágenes[16].

8. Metodología

La metodología utilizada en este proyecto consistió en trabajar en equipo de 3 personas, liderado por Laura Gómez, quien estableció las fechas de entrega para cada subpunto asignado a cada miembro del grupo. Cada integrante se enfocó en investigar y resolver el punto asignado. Una vez que se tuvo una comprensión clara del trabajo de cada integrante, se procedió a la compilación de los subpuntos de acuerdo a cada entrega.

Se destaca que para el desarrollo de este trabajo, se tuvo en cuenta la habilidad de cada integrante:
Rubens Apresa como programador principal empleando métodos de las librerías NumPy, Pandas, Sympy, etc. De forma que optimiza el código y lo hace fácil de comprender, así mismo transfiere los conocimientos de los métodos y funciones implementados en pro de nuestro desarrollo como programadores
Laura Gómez como la líder, organizadora, compiladora, asistente de programación y encargada de perfeccionar los resultados de cada entrega.
Luis Espinel comprometido con el desarrollo de los puntos asignados.

El proyecto se dividió en tres entregas, en las cuales se abordaron diferentes temas relacionados con el análisis combinatorio, las funciones generadoras ordinarias, el método simbólico, el cálculo de probabilidades, el uso de librerías como NumPy y Pandas, y la implementación de un bot de Telegram. Se utilizó ChatGPT como base para ampliar conocimientos y tener sus aportes como un punto de partida para los problemas, incluso para la bibliografía de este informe.

Para la primera entrega, se realizaron ocho ejercicios relacionados con funciones generadoras ordinarias, donde se tuvo que hallar la sucesión generada por cada FGO encontrada, así como la función no recurrente para algunos puntos. Para ello, se utilizaron herramientas como Symbolab y Wolfram, además de un código hecho por el equipo que permitió verificar los resultados hallados manualmente.

En la parte del método simbólico, se trabajó en seis ejercicios que involucraron situaciones problemas relacionados con números binarios, ternarios y cuaternarios, donde se aplicaron los pasos: 1. Definir clase combinatoria, 2. identificar los átomos, 3. se planteó la combinatoria, 4. se transformó la clase combinatoria a función generadora y 5. se halló la sucesión y función recurrente. Se programó cada enunciado en Python para que el usuario pudiera generar cadenas de acuerdo a la restricción elegida.

En la última parte de la entrega 1, se resolvieron nueve ejercicios en Python que involucraron graficar el tablero de ajedrez, mostrar el triángulo de Pascal, crear fractales polinómicos, graficar FGO, graficar en 3D coordenadas rectangulares, polares y cilíndricas, graficar en 2D el gradiente descendente, resolver funciones recurrentes homogéneas y no homogéneas, hallar la sucesión de FGO y FGE y tensores de 1 a 6 dimensiones.

En la entrega 2, se trabajó en siete ejercicios donde se aplicó análisis combinatorio haciendo uso de las FGO conocidas y suma o multiplicación de polinomios. En dos de estos puntos se tuvo que hallar la probabilidad de que el evento ocurra dado n objetos, y se desarrolló un código en Python para simular la compra de láminas del álbum del mundial, generando histogramas con la media, mediana y moda de acuerdo a cada situación y cantidad de simulaciones.

Para la entrega 3, se implementó un bot de Telegram para resolver funciones recurrentes y FGO que permitió hallar la función no recurrente y realizar peticiones relacionadas con estrellas y constelaciones, el bot es muy claro con sus comandos, cualquier problema te hará saber utilizar el comando ayuda para que te ayude más que nada sobre la relación de recurrencia, describiendo cada comando y como se debe implementar. El bot se desarrolló con la API de Telegram y su funcionamiento se hizo con Python, y para que el bot este activo siempre subió en una máquina virtual que nos brinda Google Cloud.

Además, se diseñaron funciones recursivas en R^2 con base en un tensor de tamaño (nf, nc, 3) cuyo contenido inicial era nf $\times$ nc $\times$ 3 unos, con el propósito de generar una dupla aleatoria (I, J), que indica que en ese punto se va a resaltar la fila I, la columna J, también la diagonal principal y secundaria que pasan por el punto, por último. dibujar las 4 ramas con (I, J) como vértice; esto también fue posible aplicar a al tensor de una imagen PNG. Para la segunda parte, se presentaron varios inconvenientes con respecto a la recursión máxima permitida por Google Colab y Python, ya que el rango esta entre 3000 a 5000 ejecuciones, es decir, si se pasa una imagen de 1200×1200 y se ejecuta triángulo rectángulo, con un subtensor de 100 filas, el k máximo será 10000, lo que equivale a la cantidad de llamados, es por esto que se prefirió trabajar con imágenes entre 32 a 64 píxeles, las cuales se subieron a GitHub, se obtuvo su raw link y se colocó en el código de una forma en que se pudiesen acceder a ellas, seguidamente, al obtener sus tensores, se ejecutaron sobre las imágenes las funciones de las figuras, teniendo en cuenta la intensidad RGB de cada pixel para poder interpretar la dirección de recursividad en la figura.

Por último, se realizó un video sobre el proyecto completo que sintetizó algunas partes de lo trabajado en las entregas anteriores y la redacción del informe final.

9. Resultados

9.1. Entrega 01:

9.1.1. FGO:

- Esto es una explicación sobre los resultados que se obtuvo en este punto
  - Secuencia $\langle 0, 1, -1, 0, 1, -1, 0, 1, -1, 0, 1, -1, 0, 1, -1, \ldots \rangle_{n \geq 2}$: Utilizando la función recursiva $f(n) = -f(n-1) - f(n-2)$, donde $f(0) = 0$ y $f(1) = 1$, se llega a la siguiente ecuación generatriz ordinaria (EGO): $F(z) = \frac{z}{1+z+z^2}$.
  - Secuencia $\langle 0, 1, 5, 14, 30, 55, 91, 140, 204, 285, 385, 506, 650, 819, 1015, \ldots \rangle$: Utilizando la función recursiva $f(n) = f(n-1) + n^2$, donde $f(0) = 0$, se llega a la siguiente EGO: $F(z) = \frac{z^2+z}{(1-z)^4}$. La fórmula cerrada para esta secuencia es $f(n) = \frac{2n^3+3n^2+n}{6}$.
  - Secuencia $\langle f_n \rangle$ con $f_n = 2^{n-1} f_{n-1} + (-1)^n$: Mediante manipulaciones algebraicas, se obtiene la siguiente EGO: $F(z) = z(F(2z) - 1) + \frac{1}{1+z}$.
  - Secuencia $\langle f_n \rangle$ con $f_n = (n-1)(f_{n-1} + f_{n-2})$ y $f(1) = 0$, $f(2) = 1$: Mediante cálculos con ayuda de Wolfram Alpha, se obtiene la solución cerrada: $F(z) = \frac{c_1 \cdot e^{-\frac{1}{z}}}{z} + \frac{e^{-\frac{1}{z}-1} Ei\left(1+\frac{1}{z}\right)}{z} - 1$.
  - Secuencia $\langle f(n) \rangle$ con $f(n) = 16 \cdot f(n-4) - 5 \cdot (-1)^n$ y $f(0) = f(1) = 1$, $f(2) = 3$, $f(3) = 5$: Se llega a la siguiente EGO: $F(z) = \frac{8z^3+4z^2+2z+1}{1+z} \cdot \frac{1}{(1-16z^4)}$. La fórmula cerrada para esta secuencia es $f(n) = \frac{(-1)^n+2^{n+1}}{3}$.
  - Secuencia de los números de Catalán: Siguiendo el enfoque de Euler para hallar la función generatriz, se llega a la siguiente EGO: $F(z) = \frac{1-\sqrt{1-4z}}{2z}$. Esta es la función generatriz de los números de Catalán.
  - La fórmula generadora $F_6(z) = \dfrac{z+z^2}{(1-z)^4}$ se descompone en fracciones parciales para obtener los coeficientes. Al realizar la descomposición, se obtiene $\dfrac{z+z^2}{(z-1)^4} = \dfrac{1}{(z-1)^2} + 3 \cdot \dfrac{1}{(z-1)^3} + 2 \cdot \dfrac{1}{(z-1)^4}$.
    Cada fracción parcial se convierte en una secuencia individual.
    - La primera fracción $\dfrac{1}{(z-1)^2}$ genera la secuencia de los números naturales con un corrimiento, es decir, $f(n) = n + 1$ para $n \geq 0$.

- La segunda fracción $3 \cdot \dfrac{1}{(z-1)^3}$ genera la secuencia de los números triangulares negativos con un corrimiento, es decir, $f(n) = -\dfrac{3n \cdot (n+3)}{2} + 3$ para $n \geq 0$.
- La tercera fracción $2 \cdot \dfrac{1}{(z-1)^4}$ genera una secuencia que se puede expresar de dos formas: $2 \cdot \binom{n}{3}$ o mediante la recurrencia $f(n) = 4f(n-1) - 6f(n-2) + 4f(n-3) - f(n-4)$. En ambos casos, para $n \geq 0$, se tiene $f(n) = \frac{n^3+11n}{3} + 2n^2 + 2$.

Por lo tanto, al sumar estas secuencias generadas por las fracciones parciales, se obtiene la secuencia completa representada por la fórmula generadora $\dfrac{z^2+z}{(z-1)^4}$.
y su función $f(n)$ está dada por: $f(n) = \dfrac{2n^3+3n^2+n}{6}$

- La fórmula generadora $F_7(z) = \dfrac{(z+1)^4}{(z-1)^3}$ se simplifica a $z + 7 + \dfrac{24}{z-1} + \dfrac{32}{(z-1)^2} + \dfrac{16}{(z-1)^3}$.
Cada fracción parcial se convierte en una secuencia individual.
  - La primera fracción $\dfrac{24}{z-1}$ genera la secuencia constante $f(n) = -24$.
  - La segunda fracción $\dfrac{32}{(z-1)^2}$ genera una secuencia lineal $f(n) = 32n + 32$.
  - La tercera fracción $\dfrac{16}{(z-1)^3}$ genera una secuencia cuadrática $f(n) = -8n^2 + 8n - 1$.

  Por lo tanto, al sumar estas secuencias generadas por las fracciones parciales, se obtiene la secuencia completa representada por la fórmula generadora $F_7(z)$. En forma no recurrente, la secuencia se puede expresar como $f(n) = \langle 0, 1 \rangle - 8n^2 + 8n - 1$.

9.1.2. Método Simbólico

- Explicación sobre como se obtuvo los resultados del Método simbólico

  - La secuencia que cuenta las cadenas binarias de longitud n que no contienen las subcadenas "11z "10"puede ser representada por la función generadora $f(n)$.
    La construcción de la combinatoria para esta secuencia es la siguiente:
    Clase combinatoria $\mathcal{B}$: Cadenas binarias que no contienen la subcadena "10z "11".
    Átomos: z_0 y z_1.
    Construcción de la combinatoria: $\mathcal{B}_{11,10} = \epsilon + z_1 + z_0 \times \mathcal{B}$, donde ϵ representa la cadena vacía.
    Transformación a funciones generadoras:
    $B(z) = 1 + z + z \cdot B(z)$
    $B(z) \cdot (1 - z) = 1 + z$
    $B(z) = \frac{1+z}{1-z}$
    $f(n) = \frac{1+z}{1-z} = \langle 1, \langle 1 \rangle_{n \geq 1} \rangle + \langle 0, \langle 1 \rangle_{n \geq 1} \rangle$

  - La secuencia que cuenta las cadenas ternarias de longitud n que no contienen la subcadena "00"puede ser representada por la función generadora $f(n)$.

    La construcción de la combinatoria para esta secuencia es la siguiente:

    Clase combinatoria $\mathcal{T}$: Cadenas ternarias que no contienen la subcadena "00".
    Átomos: z_0, z_1, z_2.
    Construcción de la combinatoria:

$$\mathcal{T}00 = \epsilon + (z_0 + z_1 + z_2) + (z_0 z_1 + z_0 z_2 + z_1 + z_2) \cdot \mathcal{T}$$
$$\mathcal{T}00 = \epsilon + z_0 + (z_1 + z_2 + z_0 \times (z_1 + z_2)) \times \mathcal{T}$$

    Transformación a funciones generadoras:

$$\begin{aligned} T(z) &= 1 + z + (z + z + z \cdot (z + z)) \cdot T(z) \\ T(z) &= 1 + z + (2z + 2z^2) \cdot T(z) \\ T(z) - (2z + 2z^2) \cdot T(z) &= 1 + z \\ T(z) \cdot (1 - 2z - 2z^2) &= 1 + z \\ T(z) &= \frac{1 + z}{1 - 2z - 2z^2} \end{aligned}$$

    Por lo tanto, la función generadora $T(z)$ para la secuencia es $\frac{1+z}{1-2z-2z^2}$
    y el resultado de la $f(n)$ sería:
    $\langle 1, 3 \langle 2f(n-1) + 2f(n-2) \rangle_{n \geq 2} \rangle$

- La secuencia que cuenta las cadenas cuaternarias de longitud n que tienen sus caracteres en orden estrictamente creciente puede ser representada por la función generadora $C(z)$.

  La construcción de la combinatoria para esta secuencia se realiza considerando todos los casos posibles, ya que son finitos. La construcción es la siguiente:

  Clase combinatoria $\mathcal{C}$: Cadenas cuaternarias en orden estrictamente creciente.
  Átomos: z_0, z_1, z_2, z_3.
  Construcción de la combinatoria:

$$\begin{aligned}\mathcal{C}_{0123} = \epsilon &+ (z_0 + z_1 + z_2 + z_3) + (z_0 \times (z_1 + z_2 + z_3)) + (z_1 \times (z_2 + z_3)) + (z_2 \times z_3) \\ &+ (z_0 z_1 \times (z_2 + z_3)) + (z_1 \times z_2 \times z_3) + (z_0 z_1 z_2 z_3)\end{aligned}$$

  Transformación a funciones generadoras:

$$\begin{aligned}C(z) &= 1 + 4z + 3z^2 + 2z^2 + z^2 + 2z^3 + z^3 + z^4 \\ C(z) &= z^4 + 3z^3 + 6z^2 + 4z + 1\end{aligned}$$

  Por lo tanto, la función generadora $C(z)$ para la secuencia de cadenas cuaternarias en orden estrictamente creciente es $z^4 + 3z^3 + 6z^2 + 4z + 1$. Esto representa la cantidad de cadenas válidas de longitud n.

- La función generadora $T(z)$ representa la secuencia de cadenas ternarias de longitud n que contienen un número par de unos.

  La construcción de la combinatoria para esta secuencia se realiza considerando las posibles posiciones en las que puede haber un par de unos. Se aplican las reglas de recursividad para generar las cadenas siguientes. La construcción es la siguiente:

  Clase combinatoria $\mathcal{T}$: Cadenas ternarias que contienen un número par de unos.
  Átomos: z_0, z_1, z_2.
  Construcción de la combinatoria:

$$\begin{aligned}\mathcal{T} = z_0 \times \mathcal{T} &+ z_2 \times \mathcal{T} + (z_1 \times z_1) \times \mathcal{T} + (z_1 \times (z_0 + z_2) \times z_1) \times \mathcal{T} \\ &+ (z_1 \times z_0 \times z_0 \times z_1) \times \mathcal{T} + (z_1 \times z_0 \times z_2 \times z_1) \times \mathcal{T} + (z_1 \times z_2 \times z_0 \times z_1) \times \mathcal{T} \\ &+ (z_1 \times z_2 \times z_2 \times z_1) \times \mathcal{T} + \epsilon\end{aligned}$$

  Transformación a funciones generadoras:

$$
\begin{aligned}
T(z) &= zT(z) + zT(z) + z^2T(z) + 2z^3T(z) + z^4T(z) + z^4T(z) + z^4T(z) + z^4T(z) + 1 \\
&= 2zT(z) + z^2T(z) + 2z^3T(z) + 4z^4T(z) + 1 \\
T(z) &= \frac{1}{-2z - z^2 - 2z^3 - 4z^4 + 1}
\end{aligned}
$$

Por lo tanto, la función generadora $T(z)$ para la secuencia de cadenas ternarias de longitud n que contienen un número par de unos es $\frac{1}{-2z-z^2-2z^3-4z^4+1}$. Esto representa la cantidad de cadenas válidas de longitud n.

- La función generadora $B(z)$ representa la secuencia de cadenas binarias de longitud n que no contienen la subcadena `000` ni la subcadena `010`.

  La construcción de la combinatoria para esta secuencia se realiza considerando las restricciones de no tener las subcadenas mencionadas. Se aplican las reglas de recursividad para generar las cadenas siguientes. La construcción es la siguiente:

  Clase combinatoria $\mathcal{B}$: Cadenas binarias que no contienen la subcadena `000` ni la subcadena `010`.
  Átomos: z_0, z_1.
  Construcción de la combinatoria:

$$
\begin{aligned}
\mathcal{B}_{000,010} &= \epsilon + (z_0 + z_1 \times \mathcal{B}) + (z_0 \times z_0 + z_1 \times z_0 + z_0 \times z_1 + z_1 \times z_1) \\
&\quad + (z_0 \times z_0 \times z_1 + z_0 \times z_1 \times z_1) \times \mathcal{B} \\
\mathcal{B}_{000\,010} &= \epsilon + z_0 + (z_0 \times z_0 + z_0 \times z_1) + (z_1 + z_0 \times z_0 \times z_1 + z_0 \times z_1 \times z_1) \times \mathcal{B}
\end{aligned}
$$

  Transformación a funciones generadoras:

$$
\begin{aligned}
B(z) &= 1 + z + 2z^2 + (z + 2z^3)B(z) \\
B(z) - (z + 2z^3)B(z) &= 1 + z + 2z^2 \\
B(z) &= \frac{1 + z + 2z^2}{1 - z - 2z^3} \\
f(n) &= \frac{1}{1 - z - 2z^3} + \frac{z}{1 - z - 2z^3} + \frac{2z^2}{1 - z - 2z^3} \\
&= \langle \langle 1 \rangle_{0 \le n \le 2}, \langle f(n-1) + 2f(n-3) \rangle_{n \ge 3} \rangle \\
&+ \langle 0, \langle 1 \rangle_{1 \le n \le 3}, \langle f(n-1) + 2f(n-3) \rangle_{n \ge 4} \\
&+ \langle \langle 0 \rangle_{n=0, n=1}, \langle 2 \rangle_{2 \le n \le 4}, \langle f(n-1) + 2f(n-3) \rangle_{n \ge 5} \rangle
\end{aligned}
$$

  Por lo tanto, la función generadora $B(z)$ para la secuencia de cadenas binarias de longitud n que no contienen la subcadena `000` ni la subcadena `010` es $\frac{1+z+2z^2}{1-z-2z^3}$.

- La función generadora $T(z)$ representa la secuencia de cadenas ternarias de longitud n que no contienen la subcadena `000`.

  La construcción de la combinatoria se realiza considerando las restricciones de no tener la subcadena mencionada. Se aplican las reglas de recursividad para generar las cadenas siguientes. La construcción es la siguiente:

  Clase combinatoria $\mathcal{T}$: Cadenas ternarias que no contienen la subcadena `000`.
  Átomos: z_0, z_1, z_2.
  Construcción de la combinatoria:

  $$\mathcal{T} = \epsilon + z_0 + z_1 \times \mathcal{T} + z_2 \times \mathcal{T} + z_0 \times z_0 + (z_0 \times z_0 + z_0) \times (z_1 + z_2 + z_1 \times z_0 + z_2 \times z_0) \times \mathcal{T}$$

  Transformación a funciones generadoras:

  $$\begin{aligned} T(z) &= 1 + z + zT(z) + zT(z) + z^2 + (z^2 + z) \cdot (2z + z^2) \cdot T(z) \\ &= 1 + z + 2zT(z) + z^2 + (z^4 + 3z^3 + 2z^2)T(z) \\ &= 1 + z + (z^4 + 3z^3 + 2z^2 + 2z) \\ T(z) &= \frac{1 + z + z^2}{1 - z^4 - 3z^3 - 2z^2 - 2z} \end{aligned}$$

  $$\begin{aligned} f(n) &= \frac{1}{1 - z^4 - 3z^3 - 2z^2 - 2z} + \frac{z}{1 - z^4 - 3z^3 - 2z^2 - 2z} + \frac{z^2}{1 - z^4 - 3z^3 - 2z^2 - 2z} \\ &= \langle 1, 2, 6, 19, \langle 2f(n-1) + f(n-2) + f(n-3) + f(n-4) \rangle_{n \geq 4} \rangle \\ &+ \langle 0, 1, 2, 6, \langle 2f(n-1) + f(n-2) + f(n-3) + f(n-4) \rangle_{n \geq 4} \rangle \\ &+ \langle \langle 0 \rangle_{n=0, n=1}, 2, 6, \langle 2f(n-1) + f(n-2) + f(n-3) + f(n-4) \rangle_{n \geq 4} \rangle \end{aligned}$$

  Por lo tanto, la función generadora $T(z)$ para la secuencia de cadenas ternarias de longitud n que no contienen la subcadena `000` es $\frac{1+z+z^2}{1-z^4-3z^3-2z^2-2z}$.

- A partir de lo teórico se generó un código de Python que nos dio un resultado interesante para cada n cadenas.

9.1.3. Python

- Tablero de Ajedrez: crea un tablero de ajedrez utilizando la biblioteca NumPy y lo visualiza utilizando la biblioteca Matplotlib.
  El resultado será una matriz impresa de ceros en la consola y una representación visual del tablero de ajedrez con casillas negras y blancas.

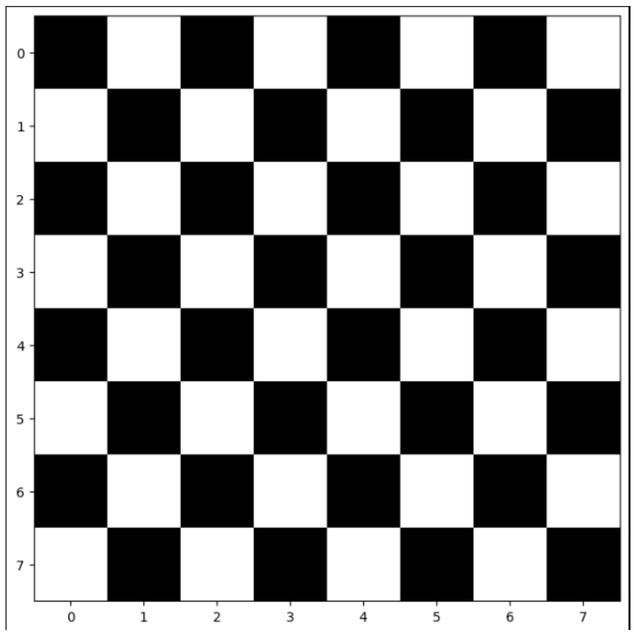
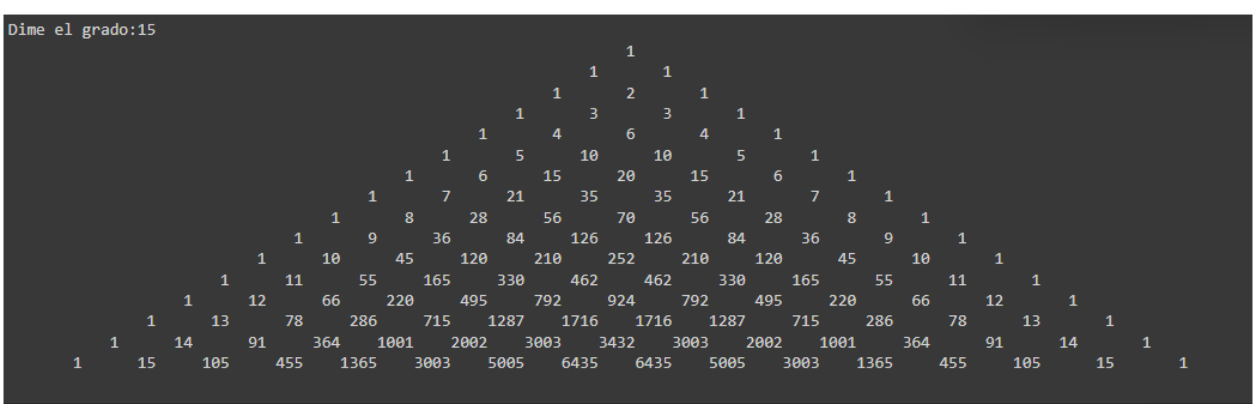


Fig. 1: Resultado Tablero de ajedrez

- Triángulo Pascal:
  El resultado del código del triángulo de pascal debe mostrar de forma de consola el triángulo dependiendo del tamaño n que se le proponga, un resultado para $n = 15$ sería:



```
Dime el grado:15
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
1 10 45 120 210 252 210 120 45 10 1
1 11 55 165 330 462 462 330 165 55 11 1
1 12 66 220 495 792 924 792 495 220 66 12 1
1 13 78 286 715 1287 1716 1716 1287 715 286 78 13 1
1 14 91 364 1001 2002 3003 3432 3003 2002 1001 364 91 14 1
1 15 105 455 1365 3003 5005 6435 6435 5005 3003 1365 455 105 15 1
```

Fig. 2: Resultado del triángulo pascal

- Fractal Polinómico: Python genera un fractal polinómico utilizando la biblioteca Matplotlib.

  El valor de n se decrementa en cada iteración del bucle for para generar un efecto de fractal con polígonos cada vez más pequeños, el resultado será una representación visual del fractal polinómico con polígonos regulares y círculos concéntricos en blanco y negro.
  Resultado de $n = 15$ fractales:

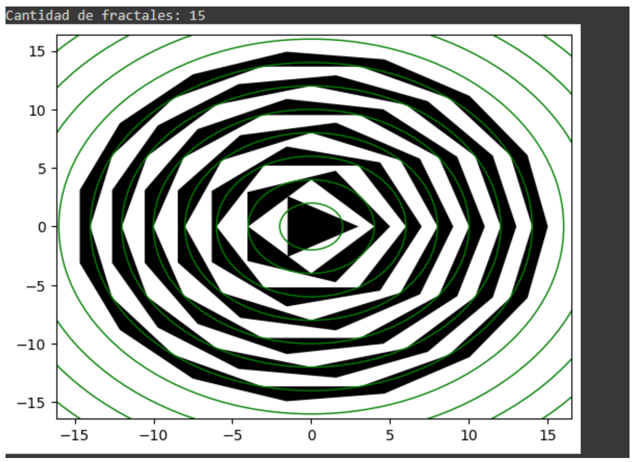
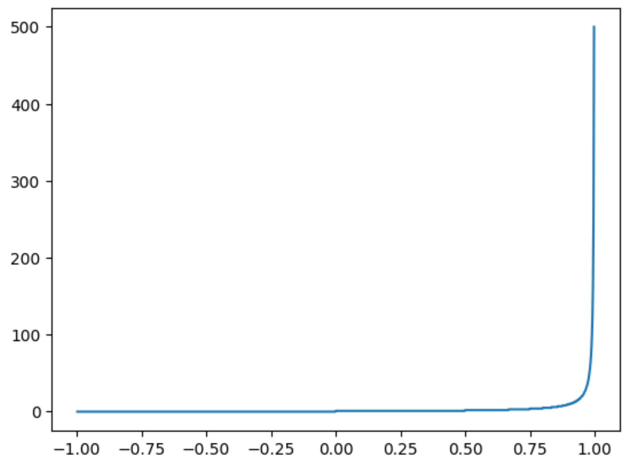


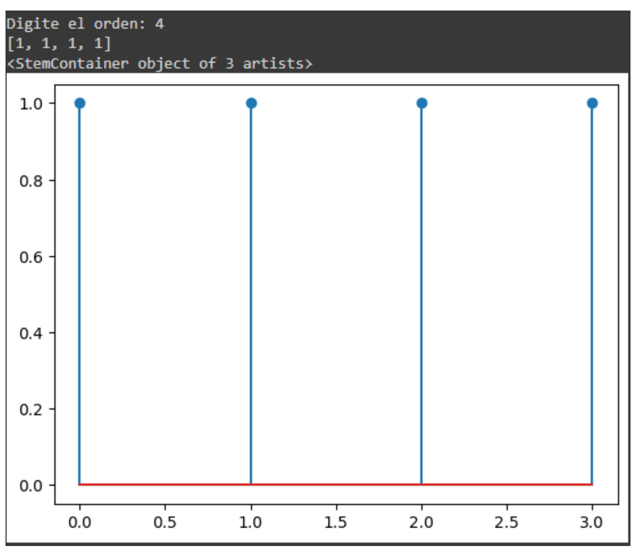
Fig. 3: Resultado del fractal

- Gráfica de las FGO y secuencias de la Tabla 1.

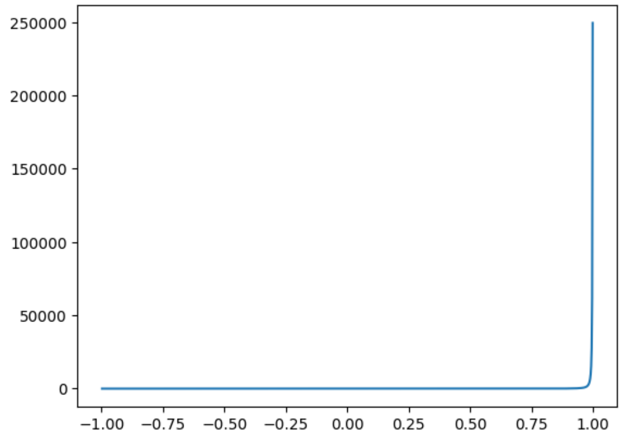
  En esta parte se graficó cada función de la tabla 1 de las FGO con ayuda de Matplotlib, estos son los resultados:

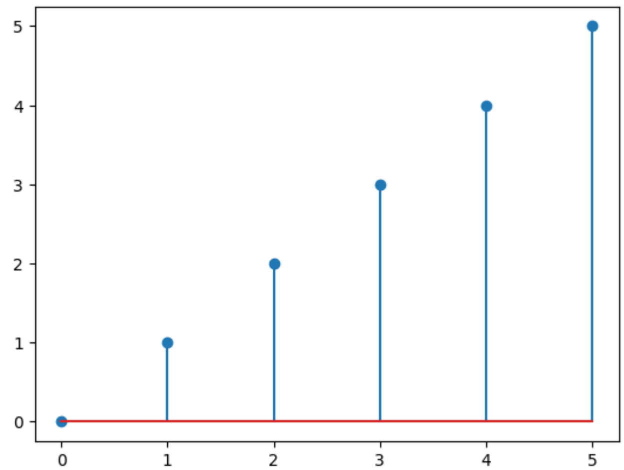
  - $F(z) = \frac{1}{1-z}$



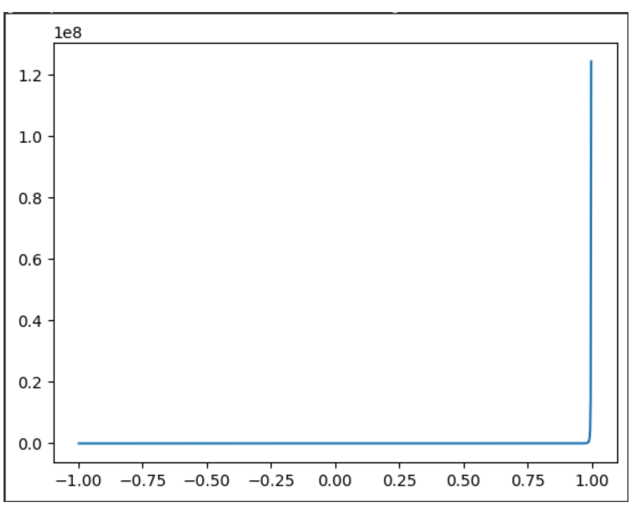


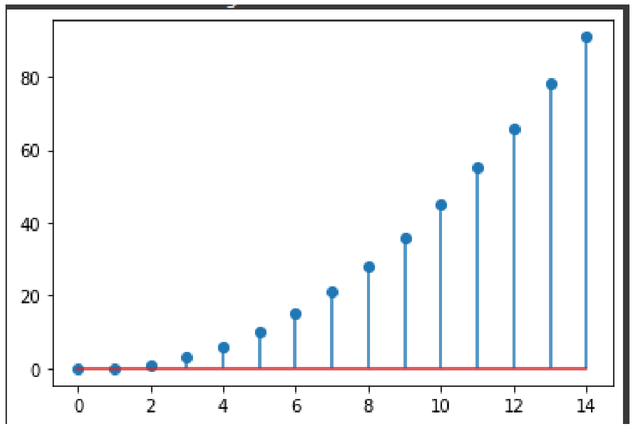
- $F(z) = \frac{z}{(z-1)^2}$



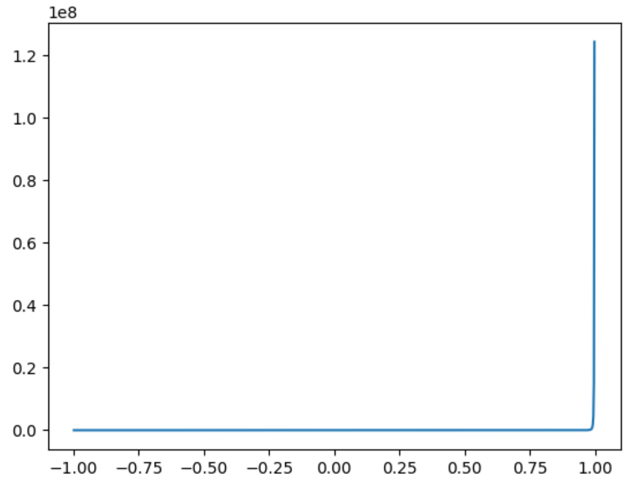


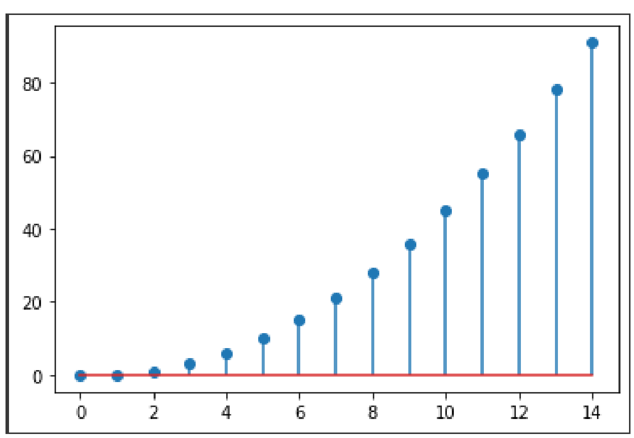
- $F(z) = \frac{z^2}{(1-z)^3}$



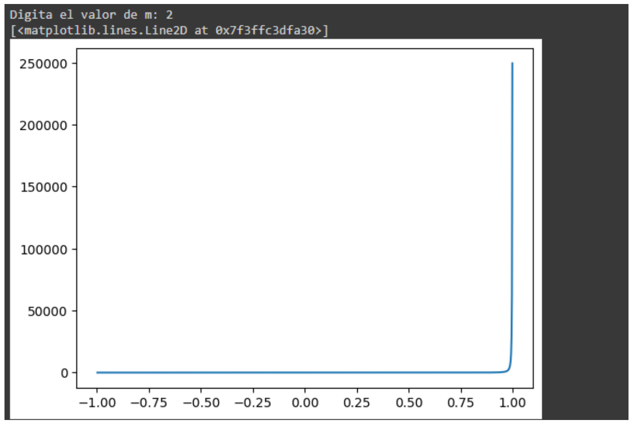


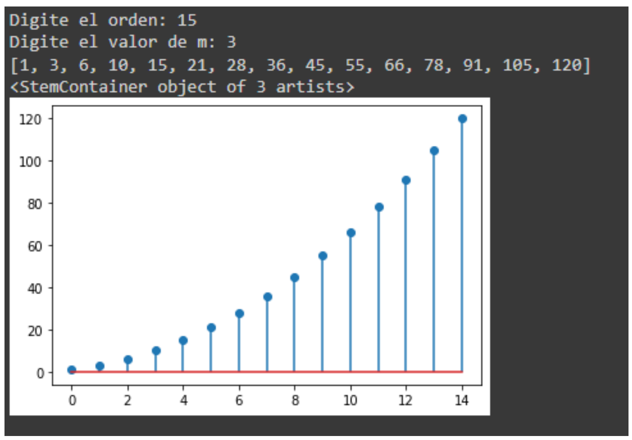
- $F(z) = \frac{z^m}{(1-z)^{m+1}}$



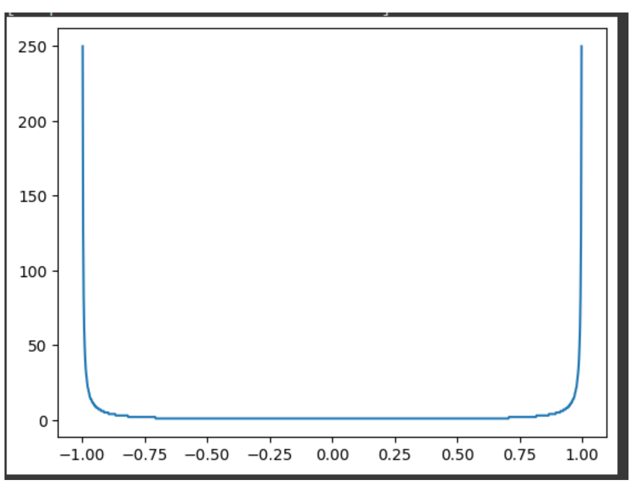


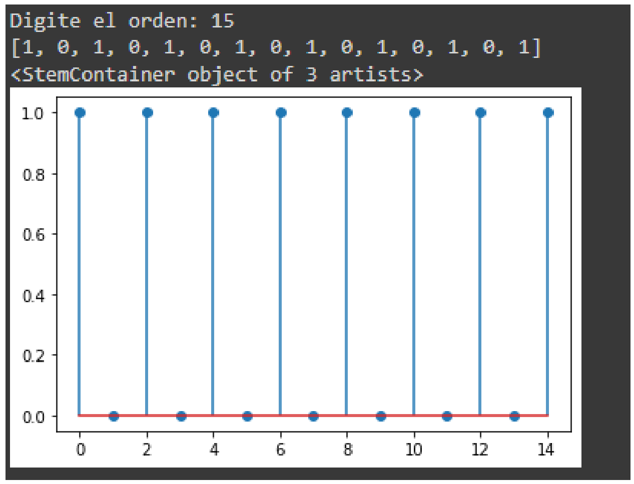
- $F(z) = \frac{1}{(1-z)^m}$


Digita el valor de m: 2
[<matplotlib.lines.Line2D at 0x7f3ffc3dfa30>]
250000
200000
150000
100000
50000
0
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

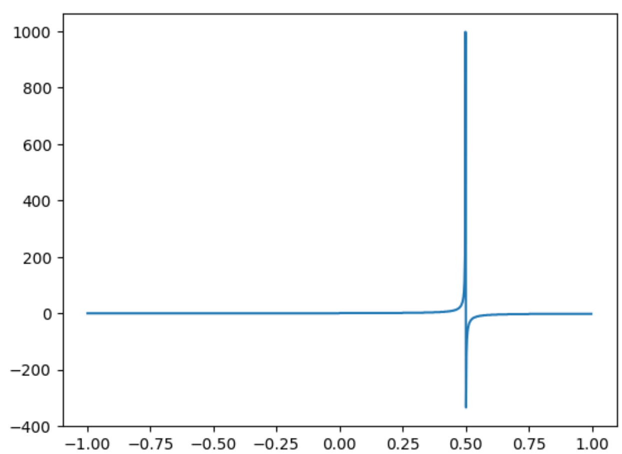

Digite el orden: 15
Digite el valor de m: 3
[1, 3, 6, 10, 15, 21, 28, 36, 45, 55, 66, 78, 91, 105, 120]
<StemContainer object of 3 artists>
120
100
80
60
40
20
0
0 2 4 6 8 10 12 14

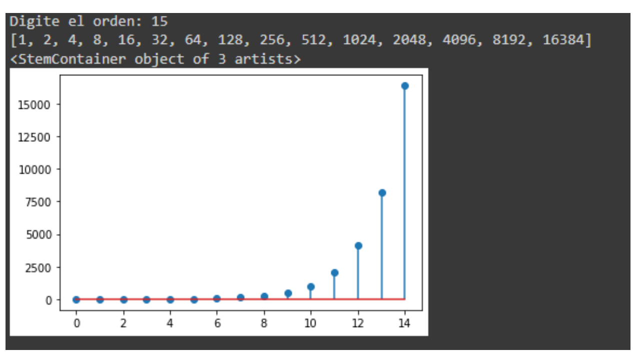
- $F(z) = \frac{1}{1-z^2}$


250
200
150
100
50
0
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

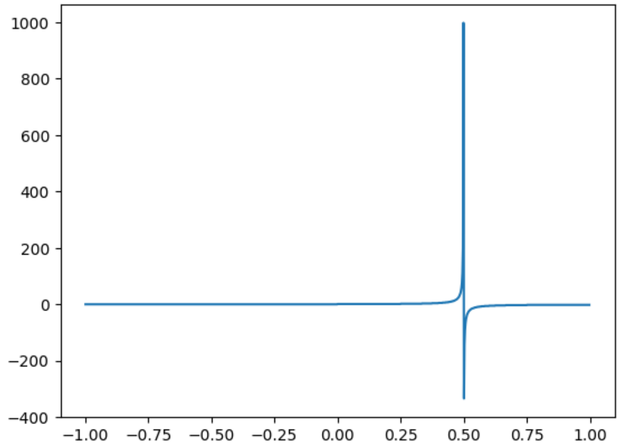


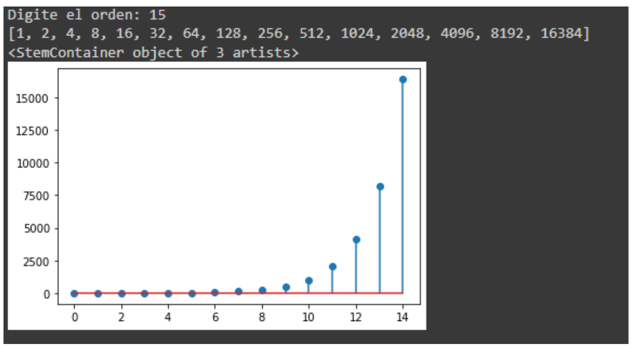
- $F(z) = \frac{1}{1-2z}$



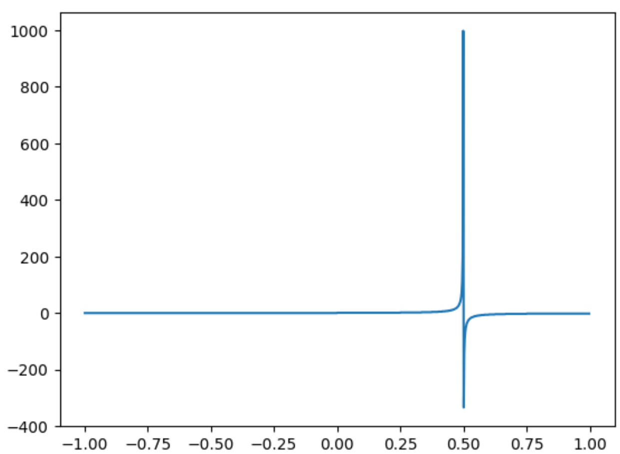


- $F(z) = \frac{1}{1-cz}$

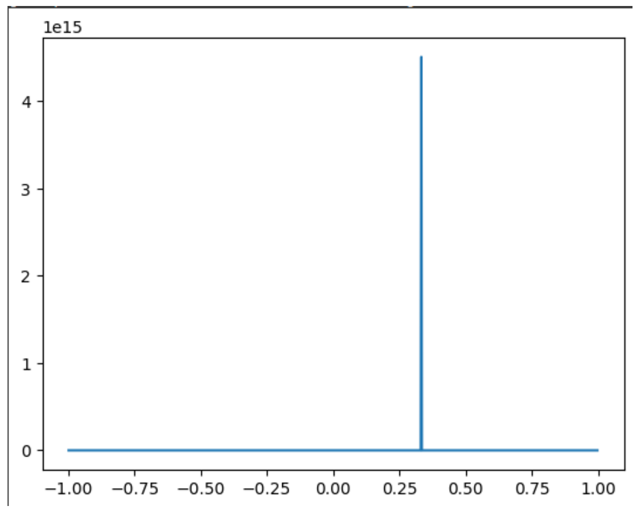




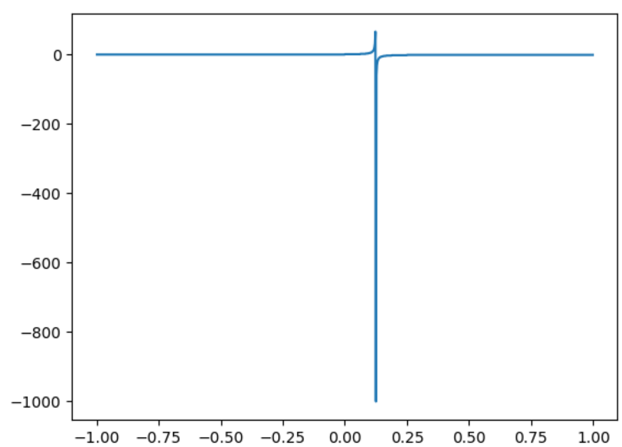
- $F(z) = \frac{1}{1-cz}$
  - si $C = 2$

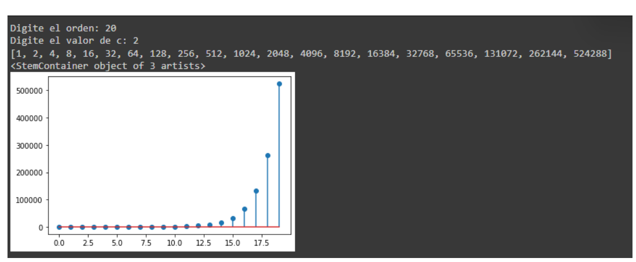


  - si $C = 3$

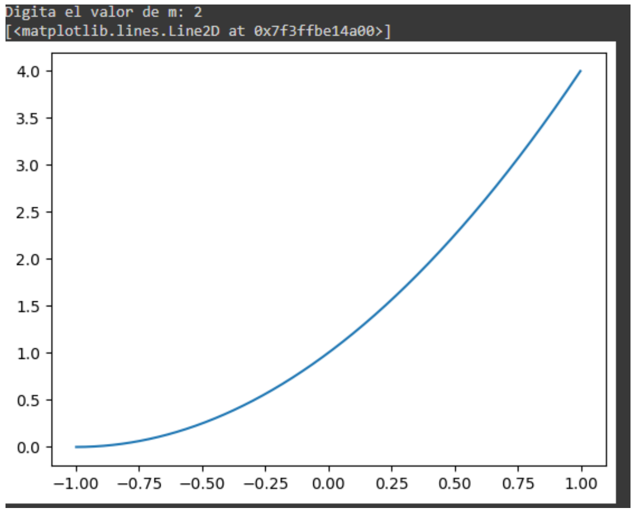

1e15
4
3
2
1
0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

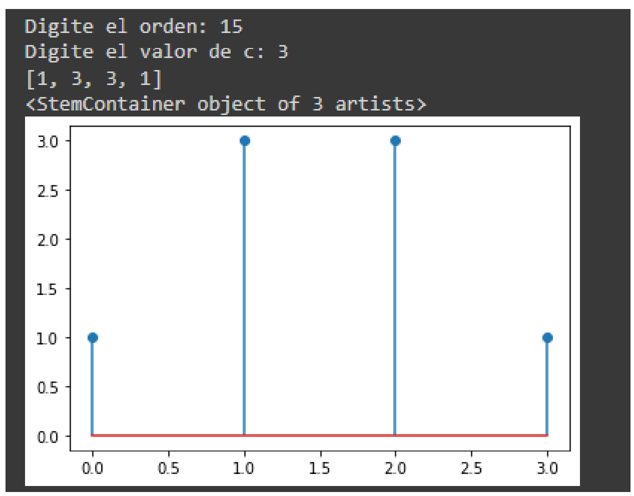
- si $C = 8$


0
−200
−400
−600
−800
−1000
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

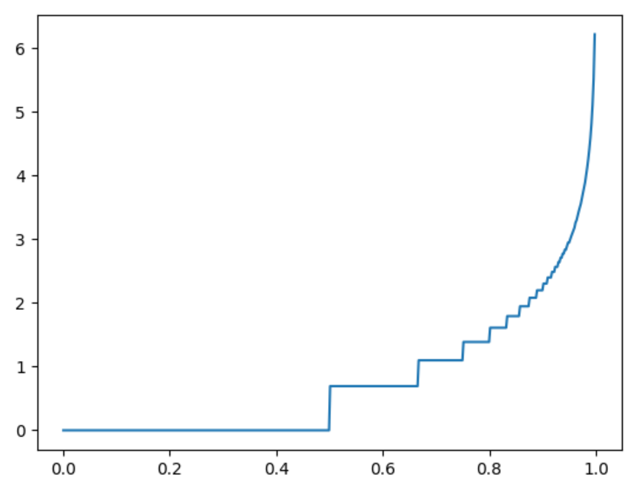

Digite el orden: 20
Digite el valor de c: 2
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768, 65536, 131072, 262144, 524288]
<StemContainer object of 3 artists>
500000
400000
300000
200000
100000
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

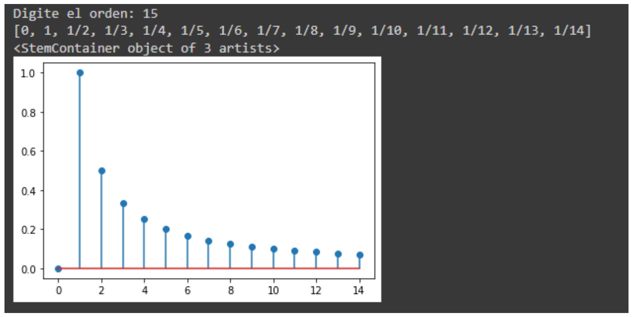
- $F(z) = (1 + z)^m$



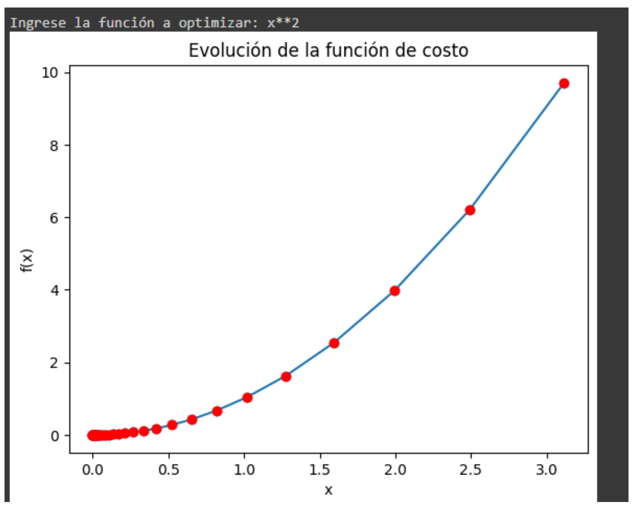


- $F(z) = ln\left(\frac{1}{1-z}\right)$





- Gráfica de gradiente 2D: ilustra el proceso de gradiente descendente en 2D para optimizar una función dada. El resultado es una visualización de cómo el valor de xy la función de costo $f(x)$ evolucionan a medida que se aplica el gradiente descendente en la función proporcionada por el usuario. Esto permite observar cómo el algoritmo busca minimizar la función moviéndose en la dirección del gradiente descendente.
  Este sería el resultado:



- Solución de RRLHCCC y RRLNHCCC:
  El código nos dará como resultado dado una función recurrente y los valores iniciales, la función no recurrente de esta.
  Este será nuestro resultado:

```
Ingresa la relación de recurrencia en términos de f(n): 2*f(n-1)+f(n-2)
Valores iniciales (Ejemplo: si es f(0)=1 y f(1)=2, ponga 1,2): 1,2
```

$$\left(\frac{1}{2}-\frac{\sqrt{2}}{4}\right)\left(1-\sqrt{2}\right)^{n}+\left(1+\sqrt{2}\right)^{n}\left(\frac{\sqrt{2}}{4}+\frac{1}{2}\right)$$

- FGO y FGE:
  FGO, generará dado una función generadora ordinaria la secuencia y FGE dado una

función generadora exponencial la secuencial
Resultado (FGO):

```
Digita la FGO f(z)=1/(1-z)
  1
─────
1 - z
None
Sucesión, o secuencia, generada
 [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

Resultado (FGE):

```
Dame el n: 15
Digita la FGO F(z)=1/(1-z)
  1
─────
1 - z
None
Sucesión, o secuencia, generada por la FGO, F(z)
 [1 1 2 6 24 120 720 5040 40320 362880 3628800 39916800 479001600
 6227020800 87178291200]
```

- Tensores permite al usuario generar tensores de diferentes rangos y dimensiones. Ejemplo de tensor rango 3

```
1. Tensor vector
2. Tensor matriz
3. Tensor rango 3
4. Tensor rango 4
5. Tensor rango 5
6. Tensor rango 6
Que tipo de tensor quieres: 3


Ingrese el tamaño (ej: 2x3x4): 2x3x4
[[[2 7 0 2]
  [4 6 1 2]
  [3 0 5 8]]

 [[4 9 6 6]
  [5 2 1 6]
  [7 0 3 8]]]
```

9.2. Entrega 02:

9.2.1. Análisis Combinatorio:

- La cantidad de formas de seleccionar n pelotas rojas y azules, donde cada color debe tener una cantidad impar, se puede obtener utilizando funciones generadoras.

  Para ello, consideramos S_1 como la cantidad de pelotas rojas y S_2 como la cantidad de pelotas azules. Queremos encontrar la función generadora para la cantidad total de formas de seleccionar n pelotas rojas y azules con una cantidad impar de cada color.

  Utilizando la técnica de funciones generadoras, multiplicamos las funciones generadoras individuales para S_1 y S_2 para obtener la función generadora $F(z)$.

  Luego, aplicamos la descomposición en fracciones parciales a $F(z)$ para simplificarla y obtener una expresión más manejable.

  Al simplificar la función generadora, obtenemos una fórmula que nos permite calcular la cantidad de formas de seleccionar las pelotas rojas y azules en función de n. Esta fórmula es $f(n) = \frac{n(1+(-1)^n)}{4}$.

  En resumen, utilizamos funciones generadoras para contar la cantidad de formas de seleccionar n pelotas rojas y azules, con una cantidad impar de cada color. Luego, simplificamos la función generadora y obtenemos la fórmula $f(n) = \frac{n(1+(-1)^n)}{4}$ como resultado.

- El número de maneras de distribuir n pelotas en 2 cajas, de modo que la primera caja tenga una cantidad par de pelotas y la segunda caja tenga una cantidad impar de pelotas, se puede representar utilizando funciones generadoras.

  Consideramos S_1 como la cantidad de pelotas en la primera caja y S_2 como la cantidad de pelotas en la segunda caja. Queremos encontrar el número de formas en las que $S_1 + S_2 = n$, donde $S_1, mod, 2 = 0$ (par) y $S_2, mod, 2 = 1$ (impar).

  Definimos el polinomio generador para los posibles resultados como $\left(\sum_{S_1,mod,2=0} z^n\right) \cdot \left(\sum_{S_2,mod,2!=0} z^n\right)$. Desarrollamos este polinomio utilizando técnicas de álgebra de series.

  Obtenemos $F(z) = \frac{z}{(1-z^2)^2}$ como el polinomio generador.

  Luego, aplicamos fracciones parciales para descomponer $F(z)$ en fracciones más simples. Obtenemos $F(z) = -\frac{1}{4(z+1)^2} + \frac{1}{4(z-1)^2}$.

  A partir de esta descomposición, podemos determinar la expresión para $f(n)$ utilizando funciones generadoras ya conocidas y simplificando la fórmula resultante. Finalmente, llegamos a $f(n) = \frac{(n+1)(1-(-1)^n)}{4}$.

En resumen, utilizamos funciones generadoras y fracciones parciales para obtener la expresión $f(n) = \frac{(n+1)(1-(-1)^n)}{4}$, que representa el número de maneras de distribuir n pelotas en 2 cajas, de manera que la primera caja tenga una cantidad par de pelotas y la segunda caja tenga una cantidad impar de pelotas.

- La cantidad de formas de repartir n objetos idénticos en dos cajas, con al menos 2 objetos en cada caja, se puede obtener utilizando funciones generadoras.

  Consideramos S_1 como la cantidad de objetos en la primera caja y S_2 como la cantidad de objetos en la segunda caja. Queremos encontrar la función generadora para la cantidad total de formas de repartir los objetos.

  Utilizando la técnica de funciones generadoras, multiplicamos las funciones generadoras individuales para S_1 y S_2 para obtener la función generadora $F(z)$.

  Luego, simplificamos la expresión de $F(z)$ utilizando propiedades algebraicas y obtenemos la función generadora $F(z) = \frac{z^4}{(1-z)^2}$.

  A partir de la función generadora, podemos observar que los coeficientes de la expansión de $F(z)$ coinciden con la secuencia generada por Python FGO [0,0,0,0,1,2,3,4,5,6,7,8,9].

  Por lo tanto, podemos concluir que la cantidad de formas de repartir n objetos en dos cajas, con al menos 2 objetos en cada caja, es $f(n) = n - 3$ para $n \geq 3$.

  En resumen, utilizamos funciones generadoras para contar la cantidad de formas de repartir n objetos en dos cajas, con al menos 2 objetos en cada caja. Mediante la simplificación de la función generadora, obtuvimos la fórmula $f(n) = n - 3$ para $n \geq 3$.

- Para distribuir n caramelos a 2 niños de tal manera que cada uno obtenga una cantidad par de caramelos, podemos utilizar funciones generadoras.

  Definimos S_1 como la cantidad de caramelos que recibe el primer niño y S_2 como la cantidad de caramelos que recibe el segundo niño. Queremos encontrar la función generadora $F(z)$ para contar la cantidad total de formas de distribuir los caramelos.

  Utilizando la técnica de funciones generadoras, multiplicamos las funciones generadoras individuales para S_1 y S_2 para obtener $F(z)$.

  Luego, aplicamos las fórmulas de las funciones generadoras básicas, ya que buscamos distribuir una cantidad par de caramelos a cada niño. Esto nos lleva a la función generadora $F(z) = \frac{1}{(1-z)^2 \cdot (1+z)^2}$.

A continuación, aplicamos la descomposición en fracciones parciales a $F(z)$ para simplificarla y obtener una expresión más manejable.

Al simplificar la función generadora, obtenemos la fórmula $f(n) = \frac{(-1)^n+1}{2} + \frac{n+(-1)^n \cdot n}{4}$, que nos da la cantidad de formas de distribuir los caramelos entre los dos niños, de manera que cada uno tenga una cantidad par.

En resumen, utilizamos funciones generadoras para contar la cantidad de formas de distribuir n caramelos a 2 niños, donde cada uno obtiene una cantidad par de caramelos. Simplificamos la función generadora y obtenemos la fórmula $f(n) = \frac{(-1)^n+1}{2} + \frac{n+(-1)^n \cdot n}{4}$ como resultado.

- Podemos calcular el número de formas en las que la suma de 5 dados puede dar como resultado n utilizando funciones generadoras.

  Definimos el polinomio generador como el delimitador de los posibles resultados al lanzar los 5 dados.

  Desarrollamos el polinomio y simplificamos, obteniendo $[z^n]\left(\frac{(1-z^6)}{1-z}\right)^5$.

  A continuación, utilizamos la fórmula del binomio de Newton elevado a una potencia para expandir $(1-z^6)^5$.

  Obtenemos $[z^{n-5}]\left(\sum_{k=0}^{5}\binom{5}{k}(-1)^k z^{6k}\right)\cdot\left(\sum_{l\geq 0}\binom{l+5-1}{l}z^l\right)$.

  Simplificamos y obtenemos la expresión $[z^{n-5}]\left(\sum_{k=0}^{5}\binom{5}{k}(-1)^k z^{6k}\right)\cdot\left(\sum_{l\geq 0}\binom{l+5-1}{l}z^l\right)$.

  Otra forma de obtener el mismo resultado es utilizando la función generadora $\left(\frac{1}{1-z}-1\right)^5$. Desarrollamos esta función generadora y obtenemos $-1+\binom{4+n}{n}-5\binom{3+n}{n}+10\binom{2+n}{n}-10\binom{1+n}{n}+5\binom{n}{n}$.

  Simplificamos la expresión obtenida y llegamos a la fórmula final $f(n) = \frac{n^4-10n^3+35n^2-50n}{24}$, que representa el número de formas en las que la suma de 5 dados puede dar como resultado n.

  En resumen, utilizamos funciones generadoras y la fórmula del binomio de Newton para obtener la expresión $f(n) = \frac{n^4-10n^3+35n^2-50n}{24}$, que representa el número de formas en las que la suma de 5 dados puede dar como resultado n.

- La probabilidad de escoger una cantidad impar de pelotas azules al seleccionar n pelotas de entre pelotas rojas y azules se puede calcular utilizando funciones generadoras.

Definimos el polinomio generador $F(z) = \frac{z}{(1-z)^2(1+z)}$ como el delimitador de los posibles resultados al seleccionar las pelotas.

Aplicamos fracciones parciales para descomponer el polinomio en fracciones simples y obtenemos $F(z) = \frac{1}{4(1-z)} + \frac{1}{2(1-z)^2} - \frac{1}{4(1+z)}$. Hallamos los coeficientes utilizando Symbolab: $A = \frac{1}{4}$, $B = \frac{1}{2}$, $C = -\frac{1}{4}$.

Luego, utilizando las fórmulas conocidas, simplificamos la expresión y obtenemos $f(n) = \frac{2n+1-(-1)^n}{4}$.

Finalmente, calculamos la probabilidad $p(\text{azules impares})$ dividiendo $f(n)$ por el número total de formas de seleccionar las pelotas, que es $\binom{2+n-1}{n}$. Por lo tanto, la probabilidad es $p(\text{azules impares}) = \frac{2n+1-(-1)^n}{4(2+n)}$.

En resumen, utilizamos funciones generadoras para obtener el polinomio $F(z) = \frac{z}{(1-z)^2(1+z)}$, que representa la probabilidad de seleccionar una cantidad impar de pelotas azules al seleccionar n pelotas de entre pelotas rojas y azules. Después de descomponer el polinomio en fracciones simples y simplificar, llegamos a la fórmula cerrada $f(n) = \frac{2n+1-(-1)^n}{4}$ para el número de formas de seleccionar una cantidad impar de pelotas azules. Finalmente, dividimos esta fórmula por el número total de formas de seleccionar las pelotas para obtener la probabilidad $p(\text{azules impares}) = \frac{2n+1-(-1)^n}{4(2+n)}$.

- La probabilidad de escoger una cantidad par de pelotas rojas al seleccionar n pelotas de entre pelotas rojas y azules se puede calcular utilizando funciones generadoras.

  Definimos el polinomio generador $F(z) = \frac{1}{(1-z)^2(1+z)}$ como el delimitador de los posibles resultados al seleccionar las pelotas.

  Aplicamos fracciones parciales para descomponer el polinomio en fracciones simples y obtenemos $F(z) = -\frac{1}{4(z-1)} + \frac{1}{2(z-1)^2} + \frac{1}{4(z+1)}$. Hallamos los coeficientes A, B, y C utilizando los valores $z = -1$ y $z = 1$.

  Simplificamos la expresión y obtenemos $f(n) = \frac{2n+3+(-1)^n}{4}$ como la fórmula cerrada para el número de formas de seleccionar una cantidad par de pelotas rojas.

  Finalmente, calculamos la probabilidad $p(\text{rojas pares})$ dividiendo $f(n)$ por el número total de formas de seleccionar las pelotas, que es $\binom{2+n-1}{n}$. Por lo tanto, la probabilidad es $p(\text{rojas pares}) = \frac{2n+3+(-1)^n}{4(n+1)}$.

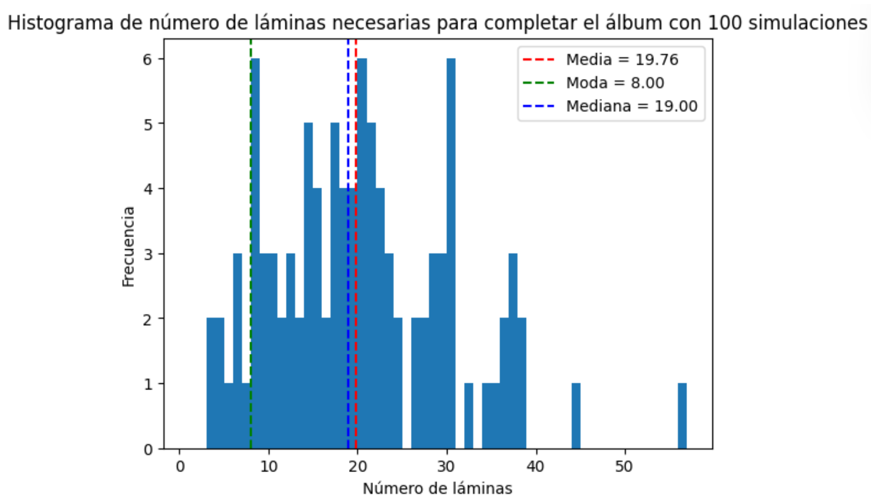
  En resumen, utilizamos funciones generadoras para obtener el polinomio $F(z) = \frac{1}{(1-z)^2(1+z)}$, que representa la probabilidad de seleccionar una cantidad par de pelotas rojas al seleccionar n pelotas de entre pelotas rojas y azules. Después de descomponer el polinomio en fracciones simples y simplificar, llegamos a la fórmula cerrada $f(n) = \frac{2n+3+(-1)^n}{4}$

para el número de formas de seleccionar una cantidad par de pelotas rojas. Finalmente, dividimos esta fórmula por el número total de formas de seleccionar las pelotas para obtener la probabilidad $p(\text{rojas pares}) = \frac{2n+3+(-1)^n}{4(n+1)}$.

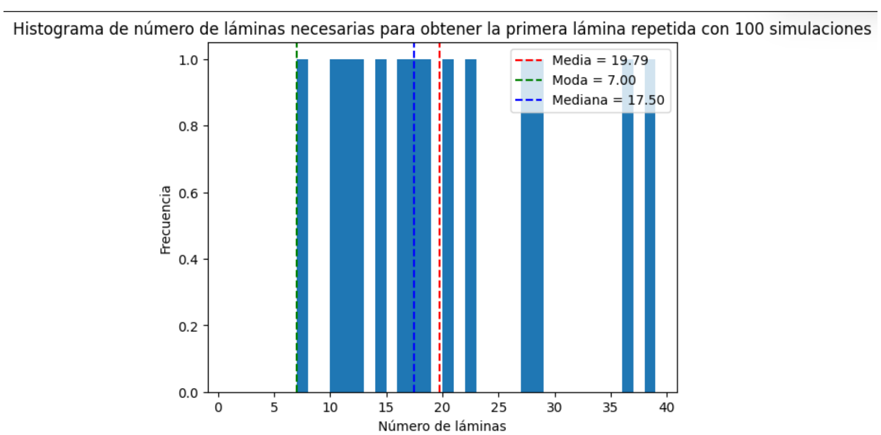
9.2.2. Coleccionista:

La parte del coleccionista se tenía que mostrar unos resultados dado sus puntos y para diferentes simulaciones, mostraremos los resultados para $n = 100$ en el primer punto

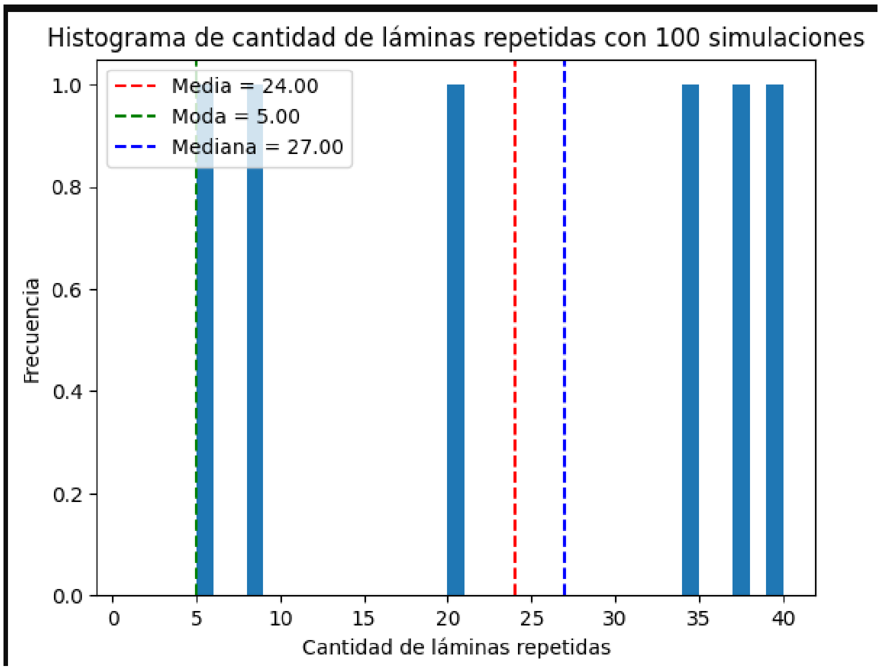
- Punto 1
  - La frecuencia del número de láminas que debe comprar.



  - La frecuencia de a las cuantas láminas sale la primera lámina repetida.



  - La frecuencia de la cantidad de láminas repetidas.



- Punto 2 (Para cada simulacion hacer lo siguiente:
  - Top 10 de láminas con mas repeticion

Top 10 de laminas con mas repeticion

	Nombre Lámina	Frecuencia
0	Bailey Wright	37
1	Garang Kuol	34
2	Youstin Salas	32
3	Kendall Waston	31
4	Matthew Smith	31
5	Thomas Meunier	30
6	Thomas Muller	30
7	Kye Rowles	30
8	Kai Havertz	29
9	Mateusz Wieteska	29

  - Top 10 de menos láminas repetidas

```
Top 10 de menos laminas repetidas
              nombre  repeticiones
0    Brandon Aguilera             1
1       Neco Williams             1
2        Thilo Kehrer             1
3         Nouhou Tolo             1
4   Youssoufa Moukoko             1
5           Ao Tanaka             1
6   Eduardo Camavinga             1
7         Martin Boyle            1
8       Joel Campbell             1
9        Chris Mepham             1
```

- Total de láminas repetidas

```
Total de laminas repetidas: 3981
```

- láminas que aparecieron solo una vez

```
Hubo 19 láminas que aparecieron solo una vez.
```

- Listado de selecciones de las que se obtuvieron todas las láminas

lista de selecciones:

	nombre
0	Costa Rica
1	Wales
2	Brazil
3	France
4	Japan
5	Belgium
6	Cameroon
7	USA
8	Germany
9	Australia
10	Poland
11	Denmark

- Los nombres de las 5 selecciones de las que se obtuvieron menos láminas

Los nombres de las 5 selecciones de las que se obtuvieron menos láminas.

	Seleccion	Num Laminas
0	France	264
1	Wales	270
2	Denmark	273
3	USA	301
4	Belgium	342

- Selecciones que obtuvieron más láminas repetidas

Selecciones que obtuvieron mas laminas repetidas:

	nombre	repeticiones
0	Costa Rica	377
1	Australia	373
2	Japan	365
3	Germany	364
4	Poland	362
5	Cameroon	358
6	Brazil	351
7	Belgium	342
8	USA	301
9	Denmark	273

- Selección con menos repetidas

```
seleccion con menos repetidas
```

	nombre	repeticiones
0	France	264
1	Wales	270
2	Denmark	273
3	USA	301
4	Belgium	342

- Primera y última selección en ser completada

```
primera seleccion: Costa Rica
Ultima seleccion: Germany
```

9.3. Entrega 3

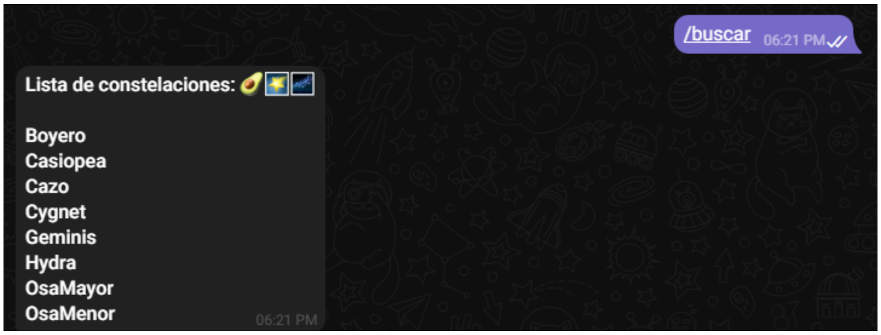
Link para acceder al bot: Bot Recurrente del Espacio

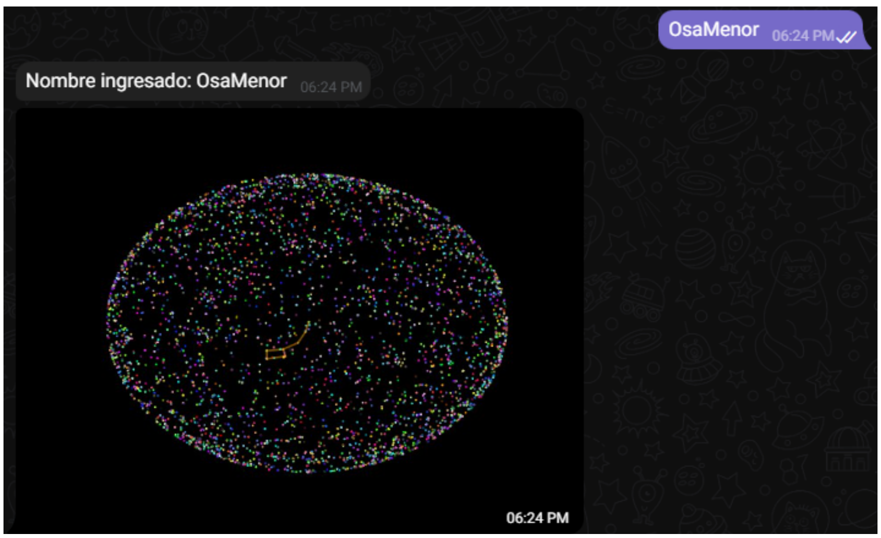
9.3.1. Problema 1: Estrellas y Constelaciones

- Mostrar todas las estrellas: Por medio del comando 'Estrellas' el bot dara el gráfico de todas las estrellas

- Mostrar todas las constelaciones: Con el comando 'TODAS' mostrará todas las constelaciones y estrellas

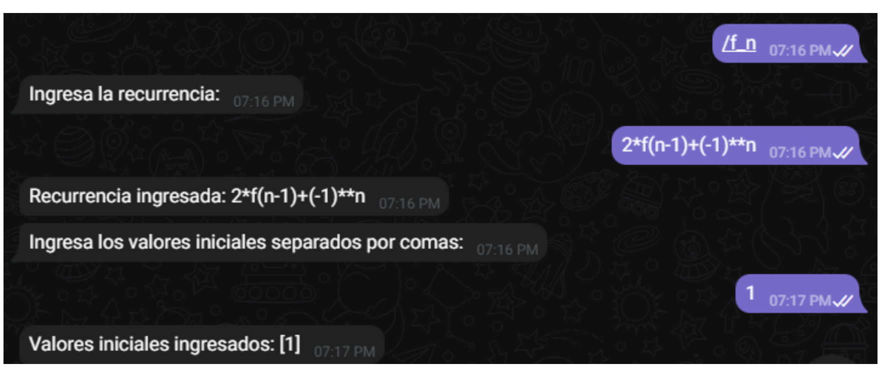
- Constelaciones: con el comando 'buscar' te mostrará la lista de constelaciones y al final te pedirá una (Ejemplo con OsaMenor)

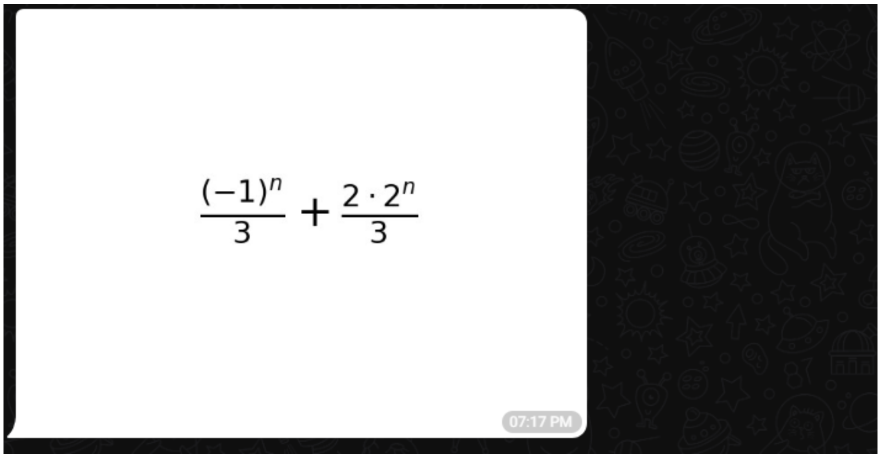




9.3.2. Función de recurrencia

- El bot pedirá una función de recurrencia y después sus valores iniciales al final mostrará una imagen con la función no recurrente (Ejemplo con: $2 \cdot f(n-1) + (-1)^n$, $f(0) = 1$)





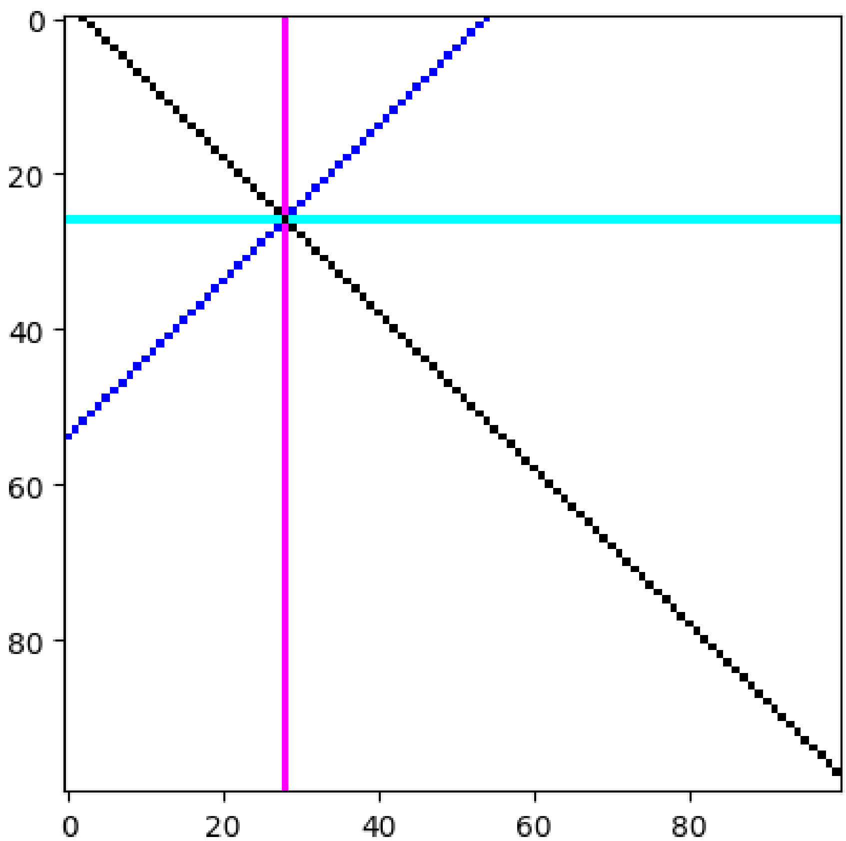
$\frac{(-1)^n}{3} + \frac{2 \cdot 2^n}{3}$
07:17 PM

9.3.3. Diseño de Funciones recursivas en R^2

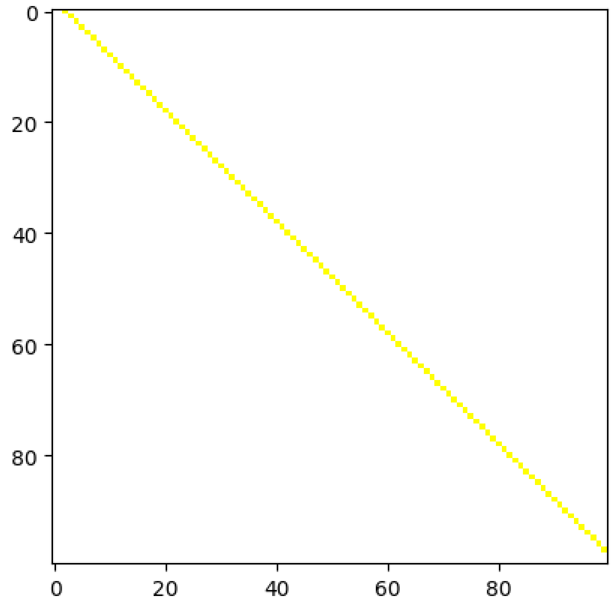
9.3.4. Filas, columnas, diagonales y ramas en un tensor

- Considerando un tensor de tamaño $(nf, nc, 3)$ cuyo contenido inicial es $nf \times nc \times 3$ unos, Se procede a graficar sus filas, columnas, diagonales y ramas a partir de una dupla aleatoria (I, J)

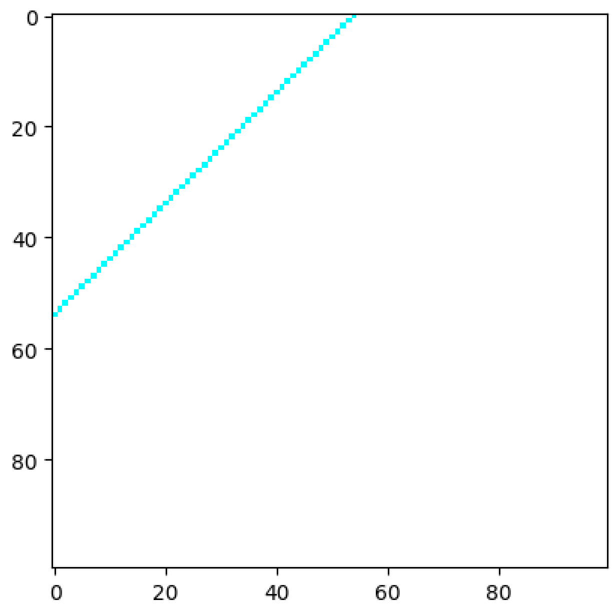
  - Todo en uno



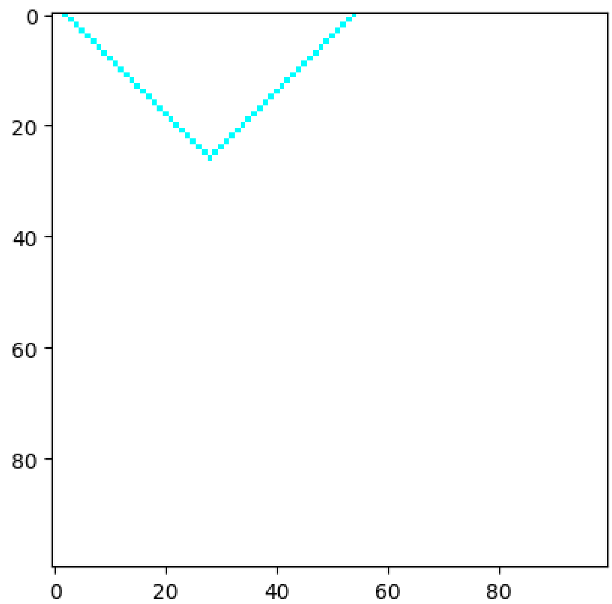
- Diagonal Principal



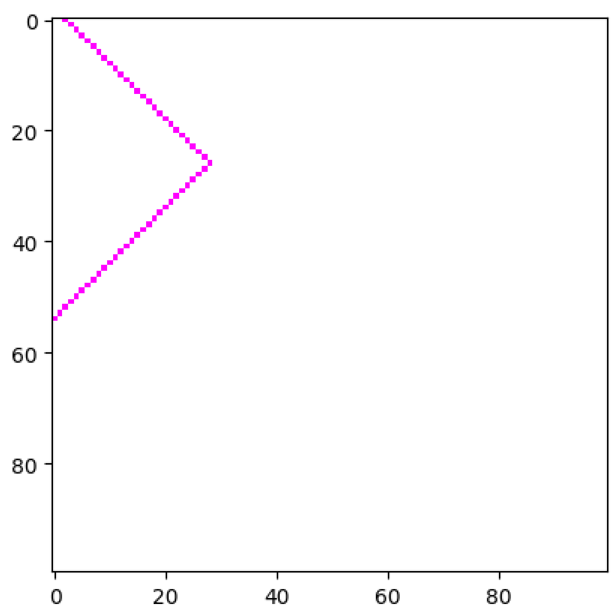
- Diagonal Secundaria



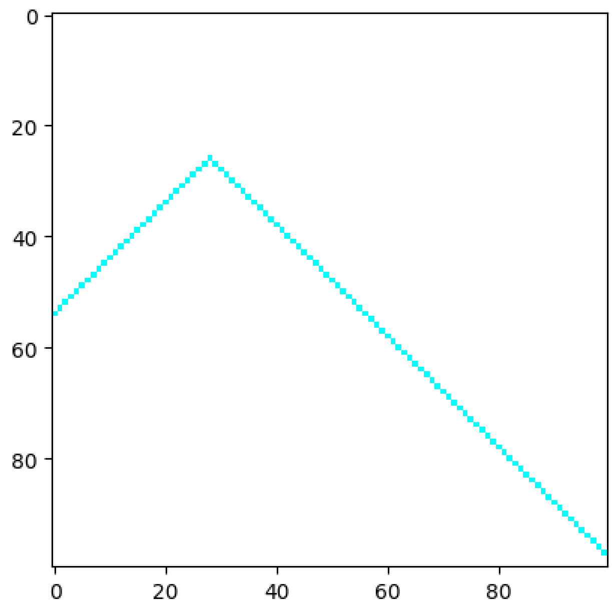
- Rama 12



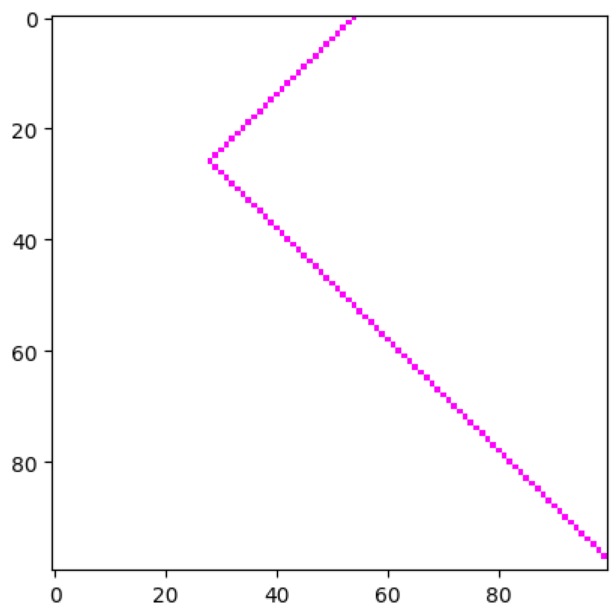
- Rama 23



- Rama 34



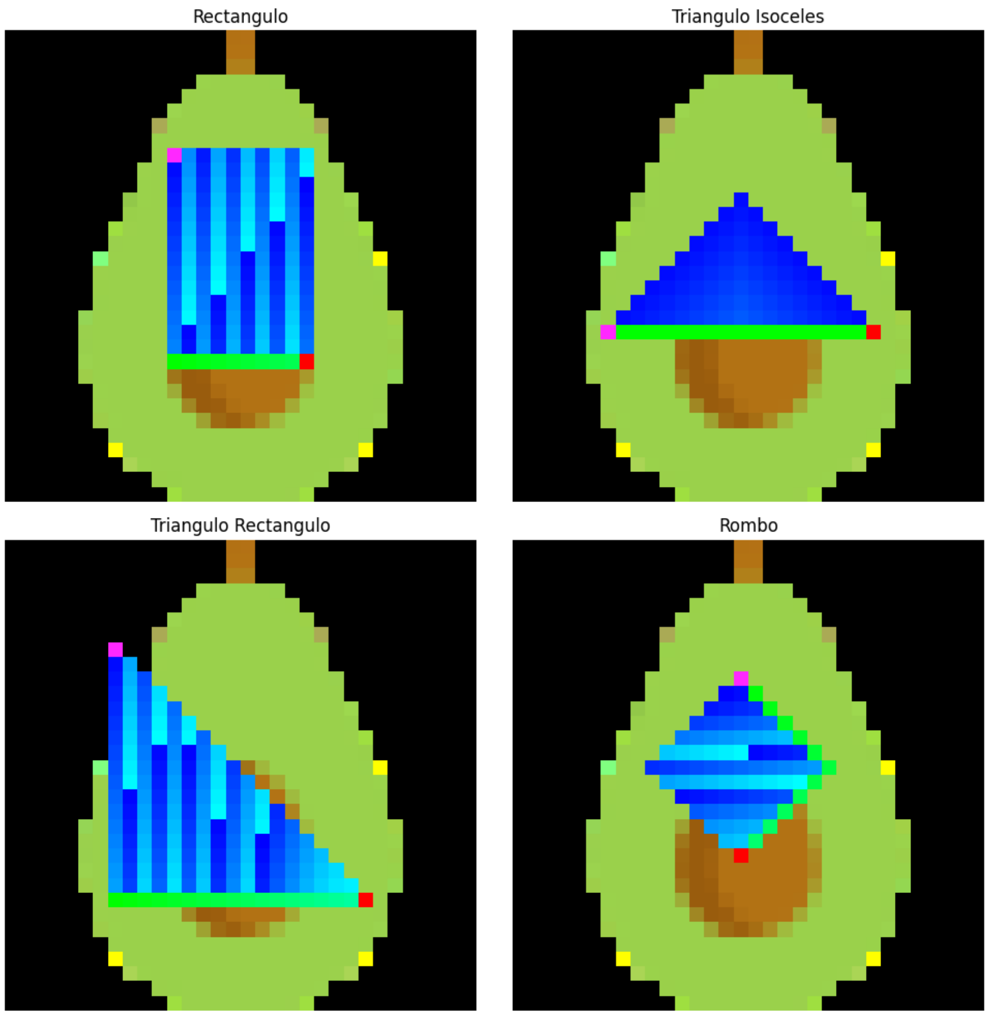
- Rama 14



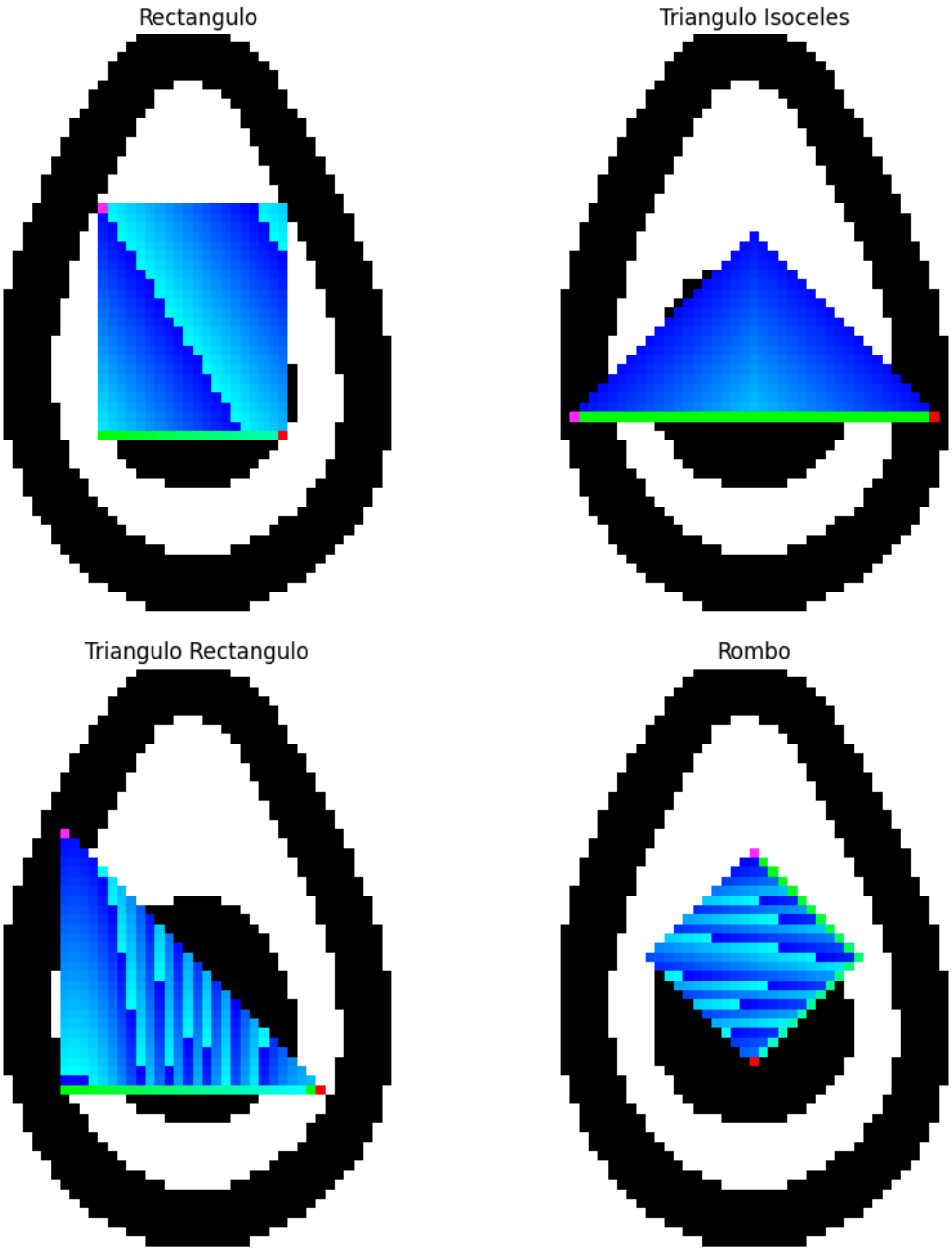
- A partir del tensor de una imagen

- Tensores y gráficas Tomando de la web tres imágenes, una de cada formato, y aplicando las funciones vistas en clase, se le aplicó a las imágenes la modificación de píxeles:
  - Formato 1: .png

- Formato 2: .svg



- Formato 3: .eps



Funcionamiento:

- El color magenta dice donde empieza la recursión
- La intensidad del color azul muestra el desplazamiento por la figura, va de más intensidad a menor intensidad y cuando llega a 0, vuelve a iniciar desde 255
- La intensidad del color verde muestra las ejecuciones de que ocurrirá una singularidad, también va de mayor intensidad a menor intensidad
- El color rojo simboliza donde termina la recursión

Se recomienda el uso de imágenes menores o iguales a 64, dado que al tomar un subtensor mayor a 100 píxeles, se puede alcanzar la máxima recursividad que puede soportar Google Colab y Python, por lo tanto, se desconectará de la plataforma automáticamente

10. Conclusión

En conclusión, el proyecto computacional ha logrado cumplir satisfactoriamente los objetivos establecidos al profundizar en el estudio y mostrar la aplicabilidad de las Funciones Generadoras Ordinarias y el Método Simbólico en el análisis combinatorio y la teoría de la probabilidad. A lo largo de tres entregas, hemos abordado de manera integral estas técnicas, desarrollando ejercicios y aplicaciones prácticas que han permitido demostrar su eficacia y utilidad; los resultados demuestran el dominio adquirido en el análisis de problemas complejos.

También ha permitido identificar diversas aplicaciones prácticas en diferentes áreas. A través del análisis combinatorio y la teoría de la probabilidad, hemos explorado su relevancia en campos como la informática, la estadística, la biología molecular, la medicina, las finanzas, la visión por computadora y el procesamiento de lenguaje natural.

Gracias a nuestro estudio, hemos comprendido cómo las FGO y el Método Simbólico pueden ser utilizados en el análisis de algoritmos, la modelización de distribuciones de probabilidad, el análisis de imágenes como tensores y como estos pueden ser mutados para generar una nueva imagen, en el campo estadístico para predecir sucesos a partir de simulaciones con base en la probabilidad, entre otros. Estas aplicaciones demuestran la versatilidad y la importancia de estas herramientas matemáticas en la resolución de problemas en diversas disciplinas.

En cuanto a la programación, destacamos la relevancia de utilizar el lenguaje de programación Python y las librerías como Numpy, Pandas y Sympy, además del uso de ChatGPT que nos ha permitido optimizar el código y hacerlo más comprensible, a pesar de tener un mediano margen de error en sus respuestas mediano, se tiene en cuenta que, es tan solo una herramienta para dar ideas al programador de lo que podría implementar y con su ingenio, mejorar, porque permite aprender nuevos metodos y funciones que simplifican los procesos para no tener que hacer los programas *a pedal*.

El desarrollo del código para resolver funciones recurrentes es una excelente prueba de ello, ya que no es necesario gastar minutos, incluso horas resolviendo un problema a lápiz y papel que involucre: funciones recurrentes, FGO y/o sucesiones, porque el programa te da la respuesta en segundos. Es por esto, que es importante resaltar la relevancia del bot de Telegram **BotRecurrente del espacio**. El desarrollo de este bot ha demostrado ser una herramienta práctica y accesible para resolver funciones recurrentes y permite la identificación y estudio de estrellas y constelaciones.

Estas referencias fueron buscadas y citadas con ayuda de ChatGPT, además de contener las referencias citadas por el profesor, ya que fue nuestra base para el contexto de este proyecto.

Referencias

[1] R. Sedgewick and P. Flajolet, *An introduction to the analysis of algorithms*, 2nd ed. Addison-Wesley, 2013, pp. 91-97, 219-229.

[2] R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete mathematics: a foundation for computer science*, 2nd ed. Addison-Wesley, 1994.

[3] Star charts and constellations. [Online]. Available: http://nifty.stanford.edu/2009/reid-starmap/starmap.html

[4] T. Hagerup and J. Katajainen, "Succinct representations of some combinatorial objects," *Journal of Algorithms*, vol. 35, no. 2, pp. 192-217, Nov. 2000.

[5] D. Aldous and P. Diaconis, "Shuffling cards and stopping times," *American Mathematical Monthly*, vol. 93, no. 5, pp. 333-348, May 1986.

[6] A. Frieze and B. Pittel, .°n the number of perfect matchings in random graphs," *Journal of Combinatorial Theory, Series A*, vol. 54, no. 1, pp. 215-224, Jan. 1990.

[7] W. McKinney, "Data structures for statistical computing in Python,ïn *Proceedings of the 9th Python in Science Conference*, 2010, pp. 51-56.

[8] T. Oliphant, *A Guide to NumPy*, USA: Trelgol Publishing, 2006.

[9] "Pandas: powerful Python data analysis toolkit,"2021. [Online]. Available: https://pandas.pydata.org/docs/. [Accessed: 08-May-2023].

[10] "NumPy,"[Online]. Available: https://numpy.org. [Accessed: May 18, 2023].

[11] T. C. Atkinson, *Python for Finance: Analyze Big Financial Data*. Apress, 2018.

[12] E. Lehman, F. Leighton, and A. Meyer, "Mathematics for Computer Science,"2018. [Online]. Available: https://www.mathematics4cs.com/. [Accessed: 08-May-2023].

[13] M. Abadi et al., "TensorFlow: A system for large-scale machine learning,ïn *Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation*, 2016, pp. 265-283.

[14] A. Paszke et al., "PyTorch: An Imperative Style, High-Performance Deep Learning Library,ïn *Advances in Neural Information Processing Systems 32*, 2019, pp. 8024-8035.

[15] A. Radford et al., "Language models are unsupervised multitask learners," *OpenAI Blog*, 2019. [Online]. Available: https://openai.com/blog/better-language-models/.

[16] K. Simonyan and A. Zisserman, ”Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.

[17] R. Platt et al., ”Baxter: A robot for the real world,ïn *Proceedings of the 8th ACM/IEEE International Conference on Human-Robot Interaction*, 2013, pp. 59-60.

[18] I. Goodfellow, Y. Bengio y A. Courville, ”Deep Learning”. MIT Press, 2016.

[19] S. van der Walt, S. C. Colbert, and G. Varoquaux, ”The NumPy array: a structure for efficient numerical computation,” *Computing in Science & Engineering*, vol. 13, no. 2, pp. 22–30, Mar. 2011.

[20] OpenAI. (2021). GPT-3: Language Models are Few-Shot Learners.

11. Anexos (Código)

Cuaderno de Google Colab
Bot Recurrente del Espacio
GitHub Bot Recurrente del Espacio

11.1. Entrega 01

11.1.1. Método Simbólico

```python
# n es la longitud
# tipo es tipo de cadena
# a es el 1er subcadena a eliminar
# b es la 2da subcadena a eliminar

def eliminar_cadena(n, tipo, a, b):
    # Funcion para crear la cadena:
    def metodo_simbolico(cadena):

        # Se verifica si las subcadenas ingresadas estan en la cadena
        if a in cadena or b in cadena:
            return
         # si la cadena tiene la misma longitud que el input n, se agrega
        if len(cadena) == n:
            cadenas_permitidas.append(cadena)
            return

        # Agregamos los atomos de acuerdo al tipo de cadena
        metodo_simbolico(cadena + "0")
        metodo_simbolico(cadena + "1")
        if tipo == 3 or tipo == 4:
          metodo_simbolico(cadena + "2")
          if tipo == 4:
            metodo_simbolico(cadena + "3")



    # Lista para almacenar las cadenas binarias validas
    cadenas_permitidas = []

    metodo_simbolico("")
    # Para que no salga cadenas repetidas
    cadenas_sin_repetidos = list(set(cadenas_permitidas))

    return cadenas_sin_repetidos, len(cadenas_sin_repetidos)

```

```python
# n es la longitud
def par_unos(n):
    # Funcion para crear la cadena:
    def metodo_simbolic(cadena):
```

```python


        # si la cadena tiene la misma longitud que el input n, se agrega
        if len(cadena) == n:
            count_ones = 0
            for i in range(len(cadena)):
                if cadena[i] == '1':
                    count_ones += 1
            if count_ones % 2 == 0:
                cadenas_permitidas.append(cadena)
            return


        # Agregamos z_0 y Z_1 y Z_2
        metodo_simbolic(cadena + "0")
        metodo_simbolic(cadena + "1")
        metodo_simbolic(cadena + "2")



    # Lista para almacenar las cadenas binarias validas
    cadenas_permitidas = []

    metodo_simbolic("")
    # Para que no salga cadenas repetidas
    cadenas_sin_repetidos = list(set(cadenas_permitidas))

    return cadenas_sin_repetidos, len(cadenas_sin_repetidos)

```

```python
# n es la longitud
def estrictamente_creciente(n):
  if n == 0:
    return [""]
  elif n == 1:
    return ["0", "1", "2", "3"]
  elif n== 2:
    return ["01", "02", "03", "12", "13", "23"]
  elif n == 3:
    return ["012", "023", "123"]
  elif n == 4:
    return ["0123"]
```

Menú

```python
print("Que desea hacer")
print("1. Eliminar subcadenas (max 2)")
print("2. Imprimir una cadena cuaternaria estrictamente creciente")
print("3. Imprimir cadenas con un numero par de unos")
res = int(input())
while (res < 1 or res >3):
  res = int(input("valor incorrecto, intente nuevamente: "))

```

```python
print("\n")
if res == 1:
  n = int(input("Longitud de la cadena: "))
  print("\nTipo de Cadena:\n 2. Binaria\n 3. Ternaria\n 4. Cuaternaria")
  tipo = int(input())
  while (tipo < 2 or res >3):
    tipo = int(input("valor incorrecto, intente nuevamente: "))

  cadena = input("\nIngrese maximo 2 subcadenas que desee eliminar (ej: 000, 01): ")
  coma = cadena.find(',')

  if coma == -1:
    a = cadena
    b = a
  else:
    a,b = cadena.split(", ")
  print("\n")
  p,x = eliminar_cadena(n,tipo, a, b)
  p = '\n'.join(str(elemento) for elemento in p)
  print("Cadenas generadas\n", p, "\nTotal: ", x)

elif res == 2:
  n = int(input("Longitud de la cadena entre 0 y 4: "))
  while (n < 0 or n > 4):
    n = int(input("valor incorrecto, intente nuevamente: "))

  p =  estrictamente_creciente(n)
  p = '\n'.join(str(elemento) for elemento in p)
  print("Cadenas generadas\n", p, "\nTotal: ", len(p))
elif res == 3:
  n = int(input("Longitud de la cadena: "))
  p,x = par_unos(n)
  p = '\n'.join(str(elemento) for elemento in p)
  print("Cadenas generadas\n", p, "\nTotal: ", x)
```

11.1.2. Ejercicios resueltos con Python

1. [Profe] Tablero de Ajedrez

```
T = np.zeros((8,8))
print(T)

```

```
#[vi: cf: step]
#T[filas pares, columnas pares]
T[::2, 1::2]=1

#T[filas impares, columnas pares]
T[1::2, ::2]=1

plt.figure(figsize = (16,8))
plt.imshow(T, cmap='gray')

```

2. [Profe] Triángulo de Pascal

```
n=int(input("Dime el grado:"))
n=n+1
TP = np.zeros((n+1,2*n+1))
TP[0, (2*n+1)//2]=1

for i in range(1,n):
  for j in range(n-i, n+i+1,2):
    TP[i,j] = TP[i-1, j-1]+TP[i-1,j+1]
print('\n'.join([
    ''.join(['{:4}'.format(int(item) if item != 0 else '') for
item in row])
    for row in TP
]))

```

```
def polinomio(n):
  x = n * np.cos(np.linspace(0, 2 * np.pi, n + 1))
  y = n * np.sin(np.linspace(0, 2 * np.pi, n + 1))
  return x, y

def circunferencia(d, factor_escala):
  circle = plt.Circle((0, 0), d * factor_escala, color='g', fill=
False)
  return circle

fig, ax = plt.subplots()

for i in range(n):
  x, y = polinomio(n)

  if i % 2 == 0:
    ax.fill(x, y, "black")
  else:
    ax.fill(x, y, "white")

  circle = circunferencia(n, 2)
  ax.add_artist(circle)

  n = n - 1


```

3. [Profe] Fractal Polinómico

```
#n = numero de segmentos
#linspace(vi, area a dividir, vf)
n = int(input("Cantidad de fractales"))

def polinomio(n):
  x = n*np.cos(np.linspace(0, 2*np.pi, n+1))
  y = n*np.sin(np.linspace(0, 2*np.pi, n+1))
  return x,y

for i in range (n):
  x,y = polinomio(n)

  if (i)%2 == 0:
    plt.fill(x,y, "black")
  else:
    plt.fill(x,y, "white")
  n=n-1

```

Versión 2 con circunferencias

```
def polinomio(n):
  x = n * np.cos(np.linspace(0, 2 * np.pi, n + 1))
  y = n * np.sin(np.linspace(0, 2 * np.pi, n + 1))
  return x, y

def circunferencia(d, factor_escala):
  circle = plt.Circle((0, 0), d * factor_escala, color='g', fill=False)
  return circle

fig, ax = plt.subplots()

for i in range(n):
  x, y = polinomio(n)

  if i % 2 == 0:
    ax.fill(x, y, "black")
  else:
    ax.fill(x, y, "white")

  circle = circunferencia(n, 2)
  ax.add_artist(circle)

  n = n - 1

```

4. [Laura] Gráfica de las FGO y secuencias de la Tabla 1. Función FGO

```
    def FGO(orden, expresion):
        z = parse_expr('z')
        f =  parse_expr(expresion)
        Fz=Poly(f.series(x=z, x0=0, n=orden).removeO())
        fn=Fz.all_coeffs()
        return fn[::-1]
```

Código base para gráficar una FGO

```
    n = np.linspace(0,1, 1000)
    f = (1)//(1-n)
    plt.plot(n, f)
```

Código base para gráicar la sucesión

```
    n = int(input("Digite el orden: "))
    expr = "(1)/(1-z)"
    array = FGO(n, expr)
    print(array)
    plt.stem(array)
```

5. [Luis] Gráfica 3D en coordenadas rectangulares, polares y cilíndricas

```
def create_esfera():
    # Definimos la formula de una esfera cerrada
    def esfera(x, y, z, r):
        return x**2 + y**2 + z**2 - r**2

    # Pedimos al usuario que ingrese el radio de la esfera
    r = float(input("Ingresa el radio de la esfera: "))

    # Creamos los valores para el eje X, Y y Z
    u = np.linspace(0, 2 * np.pi, 100)
    v = np.linspace(0, np.pi, 100)
    x = r * np.outer(np.cos(u), np.sin(v))
    y = r * np.outer(np.sin(u), np.sin(v))
    z = r * np.outer(np.ones(np.size(u)), np.cos(v))

    # Creamos el grafico 3D
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.plot_surface(x, y, z, color='blue', alpha=0.5)
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')
    ax.set_title('Coordenadas rectangulares')

    # Convertir a coordenadas polares
    R = np.sqrt(x**2 + y**2 + z**2)
    Theta = np.arctan2(y, x)
    Phi = np.arccos(z / R)

    # Crear una figura 3D
    fig = plt.figure()

    # Graficar en coordenadas polares
    ax = fig.add_subplot(111, projection='3d')
    ax.plot_surface(R*np.sin(Phi)*np.cos(Theta), R*np.sin(Phi)*np.sin
    (Theta), R*np.cos(Phi), cmap='viridis', edgecolor='none', alpha
    =0.6)
    ax.set_xlabel('R sin(Phi) cos(Theta)')
    ax.set_ylabel('R sin(Phi) sin(Theta)')
    ax.set_zlabel('R cos(Phi)')
    ax.set_title('Coordenadas polares')

    # Convertir a coordenadas cilindricas
    Rc = np.sqrt(x**2 + y**2)
    Theta = np.arctan2(y, x)

    # Crear una figura 3D
    fig = plt.figure()

    # Graficar en coordenadas cilindricas
```

```
  ax = fig.add_subplot(111, projection='3d')
  ax.plot_surface(Rc*np.cos(Theta), Rc*np.sin(Theta), z, color='red', alpha=0.5)
  ax.set_xlabel('R cos(Theta)')
  ax.set_ylabel('R sin(Theta)')
  ax.set_zlabel('Z')
  ax.set_title('Coordenadas cilindricas')

  # Pedimos al usuario que ingrese una formula
  formula = input("Ingresa una formula: ")

  if(formula == "x**2 + y**2" or
     formula == "x**2+y**2" or
     formula == "x**2 + y**2 + z**2" or
     formula == "x**2+y**2+z**2"):
    create_esfera()
  else:
    # Creamos una funcion que evalua la formula en un rango de valores
    def evaluar_formula(formula, x, y):
        z = eval(formula)
        return z

# Creamos los valores para el eje X y el eje Y
x = np.linspace(-5, 5, 100)
y = np.linspace(-5, 5, 100)
X, Y = np.meshgrid(x, y)

# Evaluamos la formula en los valores X e Y
Z = evaluar_formula(formula, X, Y)
F = Z

# Creamos el grafico 3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, Z, cmap='viridis', edgecolor='none')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
ax.set_title('Coordenadas cartesianas')

# Convertir a coordenadas polares y cilindricas
R = np.sqrt(X**2 + Y**2 + Z**2)
Rc = np.sqrt(X**2 + Y**2)
Theta = np.arctan2(Y, X)
Phi = np.arctan2(np.sqrt(X**2 + Y**2), Z)

# Crear una figura 3D
fig = plt.figure()

# Graficar en coordenadas cartesianas
ax = fig.add_subplot(131, projection='3d')
```

```
ax.plot_surface(X, Y, Z, facecolors=plt.cm.viridis(F/F.max()),
 rstride=1, cstride=1, linewidth=0, antialiased=False)
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
ax.set_title('Cartesianas')

# Graficar en coordenadas polares
ax = fig.add_subplot(132, projection='3d')
ax.plot_surface(R*np.sin(Phi)*np.cos(Theta), R*np.sin(Phi)*np.sin(
 Theta), R*np.cos(Phi), facecolors=plt.cm.viridis(F/F.max()),
 rstride=1, cstride=1, linewidth=0, antialiased=False)
ax.set_xlabel('R sin(Phi) cos(Theta)')
ax.set_ylabel('R sin(Phi) sin(Theta)')
ax.set_zlabel('R cos(Phi)')
ax.set_title('Polares')

# Graficar en coordenadas cilindricas
ax = fig.add_subplot(133, projection='3d')
ax.plot_surface(Rc*np.cos(Theta), Rc*np.sin(Theta), Z, facecolors=
 plt.cm.viridis(F/F.max()), rstride=1, cstride=1, linewidth=0,
 antialiased=False)
ax.set_xlabel('R cos(Theta)')
ax.set_ylabel('R sin(Theta)')
ax.set_zlabel('Z')
ax.set_title('Cilindricas')

```

6. [Luis] Ilustración del gradiente descendente en 2D

```python
# Pedir la funcion al usuario
funcion = input("Ingrese la funcion a optimizar: ")

# Definir la funcion de costo y su derivada
def f(x):
    return eval(funcion)

def df(x):
    h = 1e-8
    return (f(x+h) - f(x))/h

# Inicializar los parametros del algoritmo
learning_rate = 0.1
num_iterations = 50
x = np.random.uniform(-5, 5)

# Listas para almacenar los valores de x y f(x) en cada iteracion
x_values = [x]
f_values = [f(x)]
# Aplicar el metodo de gradiente descendente
for i in range(num_iterations):
    # Calcular la derivada de la funcion en el punto actual
    grad = df(x)
    # Actualizar x
    x = x - learning_rate * grad
    # Agregar los nuevos valores de x y f(x) a las listas
    x_values.append(x)
    f_values.append(f(x))

# Graficar la evolucion de x y f(x) en cada iteracion
plt.plot(x_values, f_values, '-o')
plt.plot(x_values, f_values, 'o', color='red')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Evolucion de la funcion de costo')
```

7. [Rubens] Solución de RRLHCCC y RRLNHCCC

```
 def RRLHCCC_RRLNHCCC(funcion,valores_iniciales):
     n = sp.symbols('n')
     # concatenamos "f(n)" a la relacion de recurrencia, es como pasar el f(n): a la derecha
     recurrencia_input_con_prefijo =  funcion.strip()+"-f(n)"
     f = sp.Function('f')
     # Evaluamos la relacion de recurrencia
     recurrencia = eval(recurrencia_input_con_prefijo)
     #Obtener la ecuacion general
     ecuacion_general = sp.rsolve(recurrencia, f(n))
     #hacemos el sistema de ecuaciones para encontrar la constantes
     sistema_ecuaciones = []

     #Genera el sistema de ecuaciones dependiendo de los valores iniciales f(0),f(1)...
     """
      El objeto zip se utiliza en el ciclo for para recorrer simultaneamente
      los indices y los valores iniciales de la relacion de recurrencia y
      generar el sistema de ecuaciones correspondiente.

     """
     for k, a_k in zip(range(len(valores_iniciales)), valores_iniciales):
         """
         Se implementa la operacion subs(n, k) - a_k para generar el sistema de
         ecuaciones lineales a partir de la ecuacion general de la relacion de
         recurrencia y los valores iniciales dados.
         """
         sistema_ecuaciones.append(ecuacion_general.subs(n, k) - a_k)

     #Resolvemos el sistema de ecuaciones
     constantes = sp.solve(sistema_ecuaciones)
     #reemplazamos en la ecuacion general
     solucion_general = ecuacion_general.subs(constantes)
     #retorna la solucion general con las constantes en entorno Latex para mas comprension
     return print(display(Math(sp.latex(solucion_general))))

```

[Laura]Plus: Hallar la no recurrente a partir de una sucesion o una FGO

```python
 def sucesion(secuencia):
   n = sp.symbols('n')
   # Si la cadena ingresada es un string, se convierte en lista de enteros
   if type(secuencia) != list:
     secuencia = [int(x.strip()) for x in secuencia.split(",")]

   # Se halla los coeficientes de la funcion recurrente
   coeficientes = sp.sequence(secuencia, (n, 1, len(secuencia))).find_linear_recurrence(len(secuencia))

   cadena = ""
   valor_inicial = ""

   #Se escribe la funcion recurrente con los coeficientes encontrados
   #y se crea una lista con los valores iniciales
   for i in range(len(coeficientes)):
     if coeficientes[i] != 0:
       if i == 0:
         cadena = f"{coeficientes[i]}*f(n-{i+1})"
         valor_inicial = f"{secuencia[i]}"
       else:
         cadena = cadena + f"+{coeficientes[i]}*f(n-{i+1})"
         valor_inicial = valor_inicial + f",{secuencia[i]}"

   # Se escribe la funcion recurrente obtenida
   print(display(Math(sp.latex(cadena))))
   print("\n")
   valor_inicial = [int(val.strip()) for val in valor_inicial.split(",")]
   R = RRLHCCC_RRLNHCCC(cadena,valor_inicial)
   return R

```

```python
 def FGO2(orden, expresion):
     z = parse_expr('z')
     f =  parse_expr(expresion)
     Fz=Poly(f.series(x=z, x0=0, n=orden).removeO())
     fn=Fz.all_coeffs()
     return fn[::-1]

```

Menú

```
print("Para obtener la funcion no recurrente, Como la desea hallar")
print("1. A traves de funcion recurrente")
# Plus
print("2. A traves de una secuencia")
print("3. A traves de una FGO")

print("\n")
res = int(input())
while (res < 1 or res > 3):
  res = int(input("Error, intente nuevamente: "))

if res == 1:
  #Se le pregunta al usuario la relacion de recurrencia evaluada en f(n) por ejemplo: 2*f(n-1)+f(n-1)
  recurrencia= input("Ingresa la relacion de recurrencia en terminos de f(n): ")
  # Pedir los valores iniciales al usuario
  valores_iniciales = input("Valores iniciales (Ejemplo: si es f(0)=1 y f(1)=2, ponga 1,2): ")

  #Se crea una lista para contener los valores iniciales.
  # La funcion strip() se utiliza para eliminar los espacios en blanco alrededor de cada valor ingresado
  valores_iniciales = [int(val.strip()) for val in valores_iniciales.split(",")]

  #LLamamos la funcion con los parametros
  R=RRLHCCC_RRLNHCCC(recurrencia,valores_iniciales)

elif res == 2:
  secuencia = input("Ingrese la secuencia: ")
  R = sucesion(secuencia)

elif res == 3:
  expr = input("Digite la FGO: ")
  n = int(input("Digite el orden: "))
  array = FGO2(n, expr)
  print("\nLa secusion es: ")
  print(array)
  R = sucesion(array)
```

8. [Profe] FGO y FGE

FGO

```
    z = parse_expr('z')
    f =  parse_expr(input("Digita la FGO f(z)="))
    print(display(Math(sp.latex(f))))
    Fz=Poly(f.series(x=z, x0=0, n=15).removeO())
    fn=Fz.all_coeffs()
    print('Sucesion, o secuencia, generada\n',fn[::-1])

```

FGE

```
    fact=np.zeros(n,dtype=int)
    for x in range(n):
      fact[x]=sp.factorial(x)
    z = parse_expr('z')
    f =  parse_expr(input("Digita la FGO F(z)="))
    print(display(Math(sp.latex(f))))
    Fz=Poly(f.series(x=z, x0=0, n=15).removeO())
    fn=Fz.all_coeffs()
    fn=np.multiply(fn[::-1],fact)
    print('Sucesion, o secuencia, generada por la FGO, F(z) \n',fn)

```

9. [Laura] Tensores

```
print("1. Tensor vector")
print("2. Tensor matriz")
print("3. Tensor rango 3")
print("4. Tensor rango 4")
print("5. Tensor rango 5")
print("6. Tensor rango 6")

res = int(input("Que tipo de tensor quieres: "))
while (res <1 or res > 6):
  res = int(input("Error, intenta nuevamente: "))

print("\n")
if res == 1:
  vector = np.random.randint(low=0, high=10, size=(5,))
  vector = '\n'.join(str(elemento) for elemento in vector)
  print(vector)

elif res == 2:
  tam = input("Ingrese el tamano (ej: 3x4): ")
  n, m = tam.split("x")
  n,m = int(n,m)
  m = int(m)
  matriz = np.random.randint(low=0, high=10, size=(n, m))
  print(matriz)

elif res == 3:
  tam = input("Ingrese el tamano (ej: 2x3x4): ")
  n, m, k = tam.split("x")
  n = int(n)
  m = int(m)
  k = int(k)

  n3 = np.random.randint(low=0, high=10, size=(n, m, k))
  print(n3)

elif res == 4:
  print("Ingrese el tamano\nEjemplo: 2x3x4x5 quiere decir 2 cubos
, 3 capas, 4 filas y 5 columnas")
  tam = input()
  n, m, k,l = tam.split("x")
  n = int(n)
  m = int(m)
  k = int(k)
  l = int(l)

  n4 = np.random.randint(low=0, high=10, size=(n, m, k, l))
  print(n4)

elif res == 5:
  print("Ingrese el tamano\nEjemplo: 1x2x3x4x5 quiere decir 1
```

```
matriz de 2 cubos con 3 capas, 4 filas y 5 columnas")
        tam = input()
        n, m, k,l, r = tam.split("x")
        n = int(n)
        m = int(m)
        k = int(k)
        l = int(l)
        r = int(r)

        n5 = np.random.randint(low=0, high=10, size=(n, m, k, l, r))
        print(n5)

    elif res == 6:
        print("Ingrese el tamano\nEjemplo: 1x2x3x4x5x6 quiere decir 1 cubo de 2 matrices compuestas por 3 cubos con 4 capas, 5 filas y 6 columnas")
        tam = input()
        n, m, k,l, r, w = tam.split("x")
        n = int(n)
        m = int(m)
        k = int(k)
        l = int(l)
        r = int(r)
        w = int(w)

        n6 = np.random.randint(low=0, high=10, size=(n, m, k, l, r, w))
        print(n6)

```

11.2. Entrega 02

11.2.1. El Coleccionista

- Arreglos del dataframe

```
# Pedir al usuario el directorio del archivo CSV
csv_path = input("Ingrese el directorio del archivo CSV:\n(Ejemplo: /content/World_Cup_players_Dataset_World_Cup_players_Dataset_csv.csv)\n")

# Leer el archivo CSV
df = pd.read_csv(csv_path)

# Reemplazar los valores faltantes de 'Probabilidad' con 0 y renombrar las columnas
df['Probability'].fillna(0, inplace=True)

# Crear un nuevo DataFrame con las columnas de interes
df = df.rename(columns={'Player': 'nombre', 'Probability': 'Probabilidad', 'National team': 'Seleccion'})
df_new = df.loc[:, ['nombre', 'Probabilidad', 'Seleccion']]
print("\nLa probabilidad total es:", df['Probabilidad'].sum())
```

```

# Guardar el nuevo DataFrame como un archivo CSV en /content de Google Colab
df_new.to_csv('df_new.csv', index=False)

df=df_new
df_new.head(10)
```

- Punto 1

```

def simulacion(N,df):
  # Definimos la funcion auxiliar para comprar laminas
  def comprar_laminas(df):

    # Creamos una lista para guardar las laminas obtenidas
    laminas_obtenidas = []

    # Calculamos las probabilidades normalizadas de cada lamina
    probs_norm = df['Probabilidad'] / df['Probabilidad'].sum()

    # Hacemos un loop hasta obtener una lamina repetida
    while True:

      # Elegimos una lamina aleatoriamente con las probabilidades normalizadas
      nueva_lamina = np.random.choice(df['nombre'], p=probs_norm)

      # Si la lamina ya fue obtenida, terminamos el loop y retornamos la cantidad de laminas obtenidas
      if nueva_lamina in laminas_obtenidas:
        return len(laminas_obtenidas)

      # Si la lamina no fue obtenida antes, la agregamos a la lista y continuamos el loop
      else:
        laminas_obtenidas.append(nueva_lamina)

  # Llamamos a la funcion 'comprar_laminas' N veces y guardamos los resultados en una lista
  num_laminas = [comprar_laminas(df) for _ in range(N)]

  # Creamos un histograma con la lista de numeros de laminas
  plt.hist(num_laminas, bins=range(1, max(num_laminas)+1))
  plt.xlabel('Numero de laminas')
  plt.ylabel('Frecuencia')
  plt.title('Histograma de numero de laminas necesarias para completar el album con '+ str(N)+' simulaciones')

  # Calculamos la media, moda y mediana
  media = np.mean(num_laminas)
  moda = np.argmax(np.bincount(num_laminas))
```

```python
mediana = np.median(num_laminas)

# Mostramos las lineas verticales correspondientes en el histograma
plt.axvline(media, color='r', linestyle='--', label=f'Media = {media:.2f}')
plt.axvline(moda, color='g', linestyle='--', label=f'Moda = {moda:.2f}')
plt.axvline(mediana, color='b', linestyle='--', label=f'Mediana = {mediana:.2f}')
plt.legend()
plt.show()

# Llamamos a la funcion 'comprar_laminas' hasta obtener una lamina repetida y guardamos los resultados en una lista
laminas_obtenidas_p = []
while True:
  nueva_lamina = comprar_laminas(df)
  if nueva_lamina in laminas_obtenidas_p:
    break
  else:
    laminas_obtenidas_p.append(nueva_lamina)

# Creamos un histograma con la lista de numeros de laminas antes de  la repeticion
plt.hist(laminas_obtenidas_p, bins=range(1, max(laminas_obtenidas_p)+1))
plt.xlabel('Numero de laminas')
plt.ylabel('Frecuencia')
plt.title('Histograma de numero de laminas necesarias para obtener la primera lamina repetida con '+ str(N)+' simulaciones')
media = np.mean(laminas_obtenidas_p)
moda = np.argmax(np.bincount(laminas_obtenidas_p))
mediana = np.median(laminas_obtenidas_p)

# Mostramos los resultados en la grafica
plt.axvline(media, color='r', linestyle='--', label=f'Media = {media:.2f}')
plt.axvline(moda, color='g', linestyle='--', label=f'Moda = {moda:.2f}')
plt.axvline(mediana, color='b', linestyle='--', label=f'Mediana = {mediana:.2f}')
plt.legend()
plt.show()
# Llamamos a la funcion 'comprar_laminas' hasta obtener una lamina repetida
laminas_obtenidas = []
while True:
    nueva_lamina = comprar_laminas(df)
    if nueva_lamina in laminas_obtenidas:
        break
    else:
        laminas_obtenidas.append(nueva_lamina)
```

```python

    # Creamos un histograma con la lista de numeros de laminas repetidas
    plt.hist(laminas_obtenidas, bins=range(1, max(laminas_obtenidas)+1))
    plt.xlabel('Cantidad de laminas repetidas')
    plt.ylabel('Frecuencia')
    plt.title('Histograma de cantidad de laminas repetidas con '+  str(N)+' simulaciones')

    # Calculamos la media, moda y mediana
    media = np.mean(laminas_obtenidas)
    moda = np.argmax(np.bincount(laminas_obtenidas))
    mediana = np.median(laminas_obtenidas)

    # Mostramos los resultados en la grafica
    plt.axvline(media, color='r', linestyle='--', label=f'Media = {media:.2f}')
    plt.axvline(moda, color='g', linestyle='--', label=f'Moda = {moda:.2f}')
    plt.axvline(mediana, color='b', linestyle='--', label=f'Mediana = {mediana:.2f}')
    plt.legend()
    plt.show()
```

```python
# Mostrar Simulaciones
n_simulaciones = [1, 10, 100, 1000, 10000, 100000]

for n in n_simulaciones:
    simulacion(n,df_new)
```

- Punto 2

```python
# Esta funcion simula la compra de laminas del album y devuelve una lista con todas las laminas obtenidas
def comprar_laminas(df):
    laminas_obtenidas = []
    # Se compran 4000 laminas
    for _ in range(4000):
        # Se calcula la probabilidad normalizada de obtener cada lamina
        probs_norm =  df['Probabilidad'] / df['Probabilidad'].sum()
        # Se selecciona una lamina al azar con la probabilidad correspondiente a su probabilidad normalizada
        idx = np.random.choice(df.index, p=probs_norm)
        # Se guarda la informacion de la lamina (nombre y seleccion) en una nueva variable
        nueva_lamina = df.loc[idx, ['nombre', 'Seleccion']]
        # Se agrega la nueva lamina a la lista de laminas obtenidas
        laminas_obtenidas.append(nueva_lamina)
    # Se devuelve la lista completa de laminas obtenidas
    return laminas_obtenidas
```

```python

# Se crea un diccionario para contar cuantas veces se obtuvo cada lamina
laminas_repetidas = {}

# Se realiza la simulacion de la compra de laminas
for i in range(1):
    laminas_obtenidas = comprar_laminas(df)
    # Se guarda solo el nombre de cada lamina obtenida en una nueva lista
    laminas_obtenidas_1= [lamina['nombre'] for lamina in laminas_obtenidas]
    # Se guarda la seleccion de cada lamina obtenida en una nueva lista
    list_selecc= [lamina['Seleccion'] for lamina in laminas_obtenidas]
    # Se cuenta cuantas veces se obtuvo cada lamina y se agrega al diccionario correspondiente
    for lamina in laminas_obtenidas_1:
        if lamina in laminas_repetidas:
            laminas_repetidas[lamina] += 1
        else:
            laminas_repetidas[lamina] = 1

# Se crea un diccionario para contar cuantas laminas se obtuvieron de cada seleccion
seleccion_contador = {}
# Se cuenta cuantas laminas se obtuvieron de cada seleccion y se agrega al diccionario correspondiente
for lamina in laminas_obtenidas:
    seleccion = lamina['Seleccion']
    if seleccion in seleccion_contador:
        seleccion_contador[seleccion] += 1
    else:
        seleccion_contador[seleccion] = 1

# Se imprimen las 10 laminas mas repetidas
print('\nTop 10 de laminas con mas repeticion')
top_10_laminas = sorted(laminas_repetidas.items(), key=lambda x: x[1], reverse=True)[:10]
df_top_10 = pd.DataFrame(top_10_laminas, columns=["Nombre Lamina", "Frecuencia"])
print(df_top_10.head(10))

# Se imprimen las 10 laminas menos repetidas
print('\nTop 10 de menos laminas repetidas')
top_10_laminas = sorted(laminas_repetidas.items(), key=lambda x: x[1], reverse=False)[:10]
df_top_10_laminas = pd.DataFrame(top_10_laminas, columns=['nombre', 'repeticiones'])
print(df_top_10_laminas)
#[3]
```

```
total_repetidas = sum(count for count in laminas_repetidas.values()
    if count > 1)
print("\nTotal de laminas repetidas:", total_repetidas)
#[4]
laminas_solo_una_vez = 0
for lamina, count in laminas_repetidas.items():
    if count == 1:
        laminas_solo_una_vez += 1

print(f"Hubo {laminas_solo_una_vez} laminas que aparecieron solo una
    vez.")

#[5]
print('lista de selecciones:')
# Convertir la lista 'list_selecc' en un DataFrame
df_selecc = pd.DataFrame(list_selecc, columns=['nombre'])


# Eliminar los valores repetidos en el DataFrame
df_nombres = df_selecc.drop_duplicates().reset_index(drop=True)

print(df_nombres)

#[6]

print('\nLos nombres de las 5 selecciones de las que se obtuvieron
    menos laminas.')
selecciones_ordenadas = sorted(seleccion_contador.items(), key=lambda
     x: x[1])


df_selecciones_menos_laminas = pd.DataFrame(selecciones_ordenadas
    [:5], columns=['Seleccion', 'Num Laminas'])

# Resultado
print(df_selecciones_menos_laminas)
#[7]
print('\nSelecciones que obtuvieron mas laminas repetidas:')
top_5_selec_m = sorted(seleccion_contador.items(), key=lambda x: x
    [1], reverse=True)[:10]

df_top_5_laminas_m = pd.DataFrame(top_5_selec_m, columns=['nombre', '
    repeticiones'])

print(df_top_5_laminas_m)

#[8]
print('\nseleccion con menos repetidas')
top_5_selec = sorted(seleccion_contador.items(), key=lambda x: x[1],
    reverse=False)[:10]

df_top_5_laminas = pd.DataFrame(top_5_selec, columns=['nombre', '
```

```
    repeticiones'])

print(df_top_5_laminas)


#Primera y ultima[9]
selecciones = [lamina['Seleccion'] for lamina in laminas_obtenidas]
selecciones_unicas = set(selecciones)
selecciones_unicas = list(selecciones_unicas)
primeras_posiciones = [selecciones.index(seleccion) for seleccion in
    selecciones_unicas]
ultimas_posiciones = [len(selecciones) - 1 - selecciones[::-1].index(
    seleccion) for seleccion in selecciones_unicas]
posicion_primera_seleccion = min(primeras_posiciones)
posicion_ultima_seleccion = max(ultimas_posiciones)
primera_seleccion = selecciones[posicion_primera_seleccion]
ultima_seleccion = selecciones[posicion_ultima_seleccion]

print(f'primera seleccion: {primera_seleccion}')

print(f'Ultima seleccion:{ultima_seleccion}')
```

- Top 10 de laminas con mas repeticion

```
# Se imprimen las 10 laminas mas repetidas
print('Top 10 de laminas con mas repeticion')
top_10_laminas = sorted(laminas_repetidas.items(), key=lambda x:
    x[1], reverse=True)[:10]
df_top_10 = pd.DataFrame(top_10_laminas, columns=["Nombre Lamina"
    , "Frecuencia"])
df_top_10.head(10)
```

- Top 10 de menos laminas repetidas

```
# Se imprimen las 10 laminas menos repetidas
print('Top 10 de menos laminas repetidas')
top_10_laminas = sorted(laminas_repetidas.items(), key=lambda x:
    x[1], reverse=False)[:10]
df_top_10_laminas = pd.DataFrame(top_10_laminas, columns=['nombre
    ', 'repeticiones'])
print(df_top_10_laminas)
```

- Total de laminas repetidas

```
total_repetidas = sum(count for count in laminas_repetidas.values
    () if count > 1)
print("Total de laminas repetidas:", total_repetidas)
```

- láminas que aparecieron solo una vez

```
laminas_solo_una_vez = 0
for lamina, count in laminas_repetidas.items():
    if count == 1:
```

```
        laminas_solo_una_vez += 1

print(f"Hubo {laminas_solo_una_vez} laminas que aparecieron solo
    una vez.")
```

- Listado de selecciones de las que se obtuvieron todas las láminas

```
print('lista de selecciones:')
# Convertir la lista 'list_selecc' en un DataFrame
df_selecc = pd.DataFrame(list_selecc, columns=['nombre'])


# Eliminar los valores repetidos en el DataFrame
df_nombres = df_selecc.drop_duplicates().reset_index(drop=True)

df_nombres
```

- Los nombres de las 5 selecciones de las que se obtuvieron menos láminas

```
print('Los nombres de las 5 selecciones de las que se obtuvieron
    menos laminas.')
selecciones_ordenadas = sorted(seleccion_contador.items(), key=
    lambda x: x[1])
df_selecciones_menos_laminas = pd.DataFrame(selecciones_ordenadas
    [:5], columns=['Seleccion', 'Num Laminas'])

df_selecciones_menos_laminas
```

- Selecciones que obtuvieron mas laminas repetidas:

```
print('Selecciones que obtuvieron mas laminas repetidas:')
top_5_selec_m = sorted(seleccion_contador.items(), key=lambda x:
    x[1], reverse=True)[:10]

df_top_5_laminas_m = pd.DataFrame(top_5_selec_m, columns=['nombre
    ', 'repeticiones'])

df_top_5_laminas_m
```

- Seleccion con menos repetidas

```
print('seleccion con menos repetidas')
top_5_selec = sorted(seleccion_contador.items(), key=lambda x: x
    [1], reverse=False)[:10]

df_top_5_laminas = pd.DataFrame(top_5_selec, columns=['nombre', '
    repeticiones'])

df_top_5_laminas.head(5)
```

- Primera y última selección en ser completada

```
#Primera y ultima[9]
selecciones = [lamina['Seleccion'] for lamina in
    laminas_obtenidas]
```

```python
selecciones_unicas = set(selecciones)
selecciones_unicas = list(selecciones_unicas)
primeras_posiciones = [selecciones.index(seleccion) for seleccion
     in selecciones_unicas]
ultimas_posiciones = [len(selecciones) - 1 - selecciones[::-1].
    index(seleccion) for seleccion in selecciones_unicas]
posicion_primera_seleccion = min(primeras_posiciones)
posicion_ultima_seleccion = max(ultimas_posiciones)
primera_seleccion = selecciones[posicion_primera_seleccion]
ultima_seleccion = selecciones[posicion_ultima_seleccion]

print(f'primera seleccion: {primera_seleccion}')

print(f'Ultima seleccion: {ultima_seleccion}')
```

11.3. Entrega 03

11.3.1. Bot de Telegram

```python
# -*- coding: utf-8 -*-
"""Bot

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1qxYjjk2VzIG-6IWGR3-MWHhCEPIveSec
"""

#Librerias
import matplotlib
import telebot
from sympy.parsing.sympy_parser import parse_expr
import sympy as sp
from sympy import Poly
import io
import matplotlib.pyplot as plt
import numpy as np
# Permite que matplotlib se ejecute en segundo plano
matplotlib.use('Agg')

# Token del bot
bot = telebot.TeleBot("6148003608:AAFwM91O7NqWUznSlHIFLoEGzMxIa0a7eCA")
# Leer archivo y almacenar datos en una lista (Para cargar las estrellas)
with open("constellations/stars.txt") as f:

    lines = f.readlines()

# Eliminar saltos de linea y separar elementos de cada linea
lines = [line.strip().split() for line in lines]

for line in lines:
    # Verificar si hay mas de 6 elementos en la linea
    if len(line) > 6:
        name = ' '.join(line[6:])
        # Verificar si el nombre contiene el caracter ";"
        if ";" in name:
            name_parts = name.split(';')
            line[6:] = name_parts
        else:
            line[6:] = [name]

# Crear matriz NumPy bidimensional
num_rows = len(lines)
num_cols = max(len(line) for line in lines)
data_2d = np.empty((num_rows, num_cols), dtype=object)

```

```python
# Copiar datos de lista de listas a matriz NumPy
for i in range(num_rows):
    for j in range(num_cols):
        if j < len(lines[i]):
            data_2d[i, j] = lines[i][j]
        else:
            data_2d[i, j] = ""


def buscar(constelacion,chat_id):
    try:
        with open("constellations/" +constelacion + ".txt") as f:
            lines = f.readlines()
        # Eliminar saltos de linea y separar elementos de cada linea
        lines = [line.strip().split(',') for line in lines]
        #Diccionario donde guardara las coordenadas para las estrellas que tenga nombres
        star_coords = {}
        for row in data_2d:
            if row[6]:
                if row[7]:
                    star_name = row[7].strip()  # Eliminar espacios en nombre de estrella
                    x = float(row[0])
                    y = float(row[1])
                    star_coords[star_name] = (x, y)
                star_name = row[6].strip()  # Eliminar espacios en nombre de estrella
                x = float(row[0])
                y = float(row[1])
                star_coords[star_name] = (x, y)
        #Guarda las coordenadas de las estrellas sin nombre, agregando un nombre i
        star_coords2 = {}
        counter = 1
        for row in data_2d:
            if row[6]:
                pass
            else:
                if row[7]:
                    star_name = row[7].strip()
                    x = float(row[0])
                    y = float(row[1])
                    star_coords2[star_name] = (x, y)
                else:
                    star_name = f"Estrella {counter}"
                    counter += 1
                    x = float(row[0])
                    y = float(row[1])
                    star_coords2[star_name] = (x, y)

        # Crear un diccionario para almacenar los colores de cada estrella
```

```python
        star_colors = {}
        for name in star_coords:
            star_colors[name] = np.random.rand(3)
        star_colors2 = {}
        for name in star_coords2:
            star_colors2[name] = np.random.rand(3)
        # Crear un arreglo de todas las coordenadas de las estrellas
        star_points = np.array([star_coords[name] for name in star_coords])
        star_points2 = np.array([star_coords2[name] for name in star_coords2])
        # Crear una figura con fondo negro
        fig = plt.figure(facecolor='black')
        ax = fig.add_subplot(1, 1, 1, facecolor='black')

        # Crear un scatter plot con todas las estrellas
        ax.scatter(star_points[:, 0], star_points[:, 1], s=0.9, marker='*', c=list(star_colors.values()))
        ax.scatter(star_points2[:, 0], star_points2[:, 1], s=0.9, marker='*', c=list(star_colors2.values()))

        for line in lines:
            if line[0] in star_coords and line[1] in star_coords:
                x1, y1 = star_coords[line[0]]
                x2, y2 = star_coords[line[1]]
                ax.plot([x1, x2], [y1, y2], color='orange', alpha=0.5)

            # Ajustar los limites del grafico
        ax.set_xlim([-1.1, 1.1])
        ax.set_ylim([-1.1, 1.1])

        # Ocultar los ejes
        ax.set_xticks([])
        ax.set_yticks([])

        # Guardar la imagen en un buffer
        buf = io.BytesIO()
        plt.savefig(buf, format="png")
        buf.seek(0)

        # Enviar la imagen al chat
        bot.send_photo(chat_id, buf)

        # Limpiar el buffer y el plot
        buf.truncate(0)
        buf.seek(0)
        plt.clf()
    except FileNotFoundError:
     bot.send_message(chat_id, f"No se encontro la constelacion: {constelacion}")

def mostrarEstrellas(chat_id):
```

```python
# Crear un diccionario para almacenar las coordenadas de las estrellas


star_coords = {}
for row in data_2d:
    if row[6]:
        if row[7]:
            star_name = row[7].strip()  # Eliminar espacios en nombre de
    estrella
            x = float(row[0])
            y = float(row[1])
            star_coords[star_name] = (x, y)
        star_name = row[6].strip()  # Eliminar espacios en nombre de
   estrella
        x = float(row[0])
        y = float(row[1])
        star_coords[star_name] = (x, y)

star_coords2 = {}
counter = 1
for row in data_2d:
    if row[6]:
      pass
    else:
        if row[7]:
            star_name = row[7].strip()
            x = float(row[0])
            y = float(row[1])
            star_coords2[star_name] = (x, y)
        else:
            star_name = f"Estrella {counter}"
            counter += 1
            x = float(row[0])
            y = float(row[1])
            star_coords2[star_name] = (x, y)

# Crear un diccionario para almacenar los colores de cada estrella
star_colors = {}
for name in star_coords:
    star_colors[name] = np.random.rand(3)
star_colors2 = {}
for name in star_coords2:
    star_colors2[name] = np.random.rand(3)
# Crear un arreglo de todas las coordenadas de las estrellas
star_points = np.array([star_coords[name] for name in star_coords])
star_points2 = np.array([star_coords2[name] for name in star_coords2])
# Crear una figura con fondo negro
fig = plt.figure(facecolor='black')
ax = fig.add_subplot(1, 1, 1, facecolor='black')

# Crear un scatter plot con todas las estrellas
ax.scatter(star_points[:, 0], star_points[:, 1], s=0.9, marker='*', c=
```

```
     list(star_colors.values()))
    ax.scatter(star_points2[:, 0], star_points2[:, 1], s=0.9, marker='*', c=
     list(star_colors2.values()))
        # Ajustar los limites del grafico
    ax.set_xlim([-1.1, 1.1])
    ax.set_ylim([-1.1, 1.1])

    # Ocultar los ejes
    ax.set_xticks([])
    ax.set_yticks([])


    # Guardar la imagen en un buffer
    buf = io.BytesIO()
    plt.savefig(buf, format="png")
    buf.seek(0)

    # Enviar la imagen al chat
    bot.send_photo(chat_id, buf)
    # Limpiar el buffer y el plot
    buf.truncate(0)
    buf.seek(0)
    plt.clf()

def mostrar_constelaciones(chat_id):
    # Leer archivo y almacenar datos en una lista
    with open('constellations/Cygnet.txt') as f:
        lines = f.readlines()
    with open('constellations/Boyero.txt') as f:
        lines1 = f.readlines()
    with open('constellations/Casiopea.txt') as f:
        lines2 = f.readlines()
    with open('constellations/Cazo.txt') as f:
        lines3 = f.readlines()
    with open('constellations/Geminis.txt') as f:
        lines4 = f.readlines()
    with open('constellations/Hydra.txt') as f:
        lines5 = f.readlines()
    with open('constellations/OsaMayor.txt') as f:
        lines6 = f.readlines()
    with open('constellations/OsaMenor.txt') as f:
        lines7 = f.readlines()

    # Eliminar saltos de linea y separar elementos de cada linea
    lines = [line.strip().split(',') for line in lines]
    lines1 = [line.strip().split(',') for line in lines1]
    lines2 = [line.strip().split(',') for line in lines2]
    lines3 = [line.strip().split(',') for line in lines3]
    lines4 = [line.strip().split(',') for line in lines4]
    lines5 = [line.strip().split(',') for line in lines5]
    lines6 = [line.strip().split(',') for line in lines6]
    lines7=  [line.strip().split(',') for line in lines7]
```

```
star_coords = {}

for row in data_2d:
    if row[6]:
        if row[7]:
          star_name = row[7].strip()  # utiliza strip() para eliminar los espacios
          x = float(row[0])
          y = float(row[1])
          star_coords[star_name] = (x, y)
        star_name = row[6].strip()  # utiliza strip() para eliminar los espacios
        x = float(row[0])
        y = float(row[1])
        star_coords[star_name] = (x, y)




star_coords2 = {}
counter = 1
for row in data_2d:
    if row[6]:
      pass
    else:
        if row[7]:
            star_name = row[7].strip()
            x = float(row[0])
            y = float(row[1])
            star_coords2[star_name] = (x, y)
        else:
            star_name = f"Estrella {counter}"
            counter += 1
            x = float(row[0])
            y = float(row[1])
            star_coords2[star_name] = (x, y)

# Crear un diccionario para almacenar los colores de cada estrella
star_colors = {}
for name in star_coords:
    star_colors[name] = np.random.rand(3)
star_colors2 = {}
for name in star_coords2:
    star_colors2[name] = np.random.rand(3)
# Crear un arreglo de todas las coordenadas de las estrellas
star_points = np.array([star_coords[name] for name in star_coords])
star_points2 = np.array([star_coords2[name] for name in star_coords2])
# Crear una figura con fondo negro
fig = plt.figure(facecolor='black')
ax = fig.add_subplot(1, 1, 1, facecolor='black')

# Crear un scatter plot con todas las estrellas
```

```
ax.scatter(star_points[:, 0], star_points[:, 1], s=0.9, marker='*', c=
 list(star_colors.values()))
ax.scatter(star_points2[:, 0], star_points2[:, 1], s=0.9, marker='*', c=
 list(star_colors2.values()))

for line in lines:
    if line[0] in star_coords and line[1] in star_coords:
        x1, y1 = star_coords[line[0]]
        x2, y2 = star_coords[line[1]]
        ax.plot([x1, x2], [y1, y2], color='orange', alpha=0.5)
for line1 in lines1:
    if line1[0] in star_coords and line1[1] in star_coords:
        x1, y1 = star_coords[line1[0]]
        x2, y2 = star_coords[line1[1]]
        ax.plot([x1, x2], [y1, y2], color='orange', alpha=0.5)
for line2 in lines2:
    if line2[0] in star_coords and line2[1] in star_coords:
        x1, y1 = star_coords[line2[0]]
        x2, y2 = star_coords[line2[1]]
        ax.plot([x1, x2], [y1, y2], color='orange', alpha=0.5)

for line3 in lines3:
    if line3[0] in star_coords and line3[1] in star_coords:
        x1, y1 = star_coords[line3[0]]
        x2, y2 = star_coords[line3[1]]
        ax.plot([x1, x2], [y1, y2], color='orange', alpha=0.5)
for line4 in lines4:
    if line4[0] in star_coords and line4[1] in star_coords:
        x1, y1 = star_coords[line4[0]]
        x2, y2 = star_coords[line4[1]]
        ax.plot([x1, x2], [y1, y2], color='orange', alpha=0.5)

for line5 in lines5:
    if line5[0] in star_coords and line5[1] in star_coords:
        x1, y1 = star_coords[line5[0]]
        x2, y2 = star_coords[line5[1]]
        ax.plot([x1, x2], [y1, y2], color='orange', alpha=0.5)

for line6 in lines6:
    if line6[0] in star_coords and line6[1] in star_coords:
        x1, y1 = star_coords[line6[0]]
        x2, y2 = star_coords[line6[1]]
        ax.plot([x1, x2], [y1, y2], color='orange', alpha=0.5)

for line7 in lines7:
    if line7[0] in star_coords and line7[1] in star_coords:
        x1, y1 = star_coords[line7[0]]
        x2, y2 = star_coords[line7[1]]
        ax.plot([x1, x2], [y1, y2], color='orange', alpha=0.5)

# Ajustar los limites del grafico
ax.set_xlim([-1.1, 1.1])
```

```
    ax.set_ylim([-1.1, 1.1])

    # Ocultar los ejes
    ax.set_xticks([])
    ax.set_yticks([])

     # Guardar la imagen en un buffer
    buf = io.BytesIO()
    plt.savefig(buf, format="png")
    buf.seek(0)

      # Enviar la imagen al chat
    bot.send_photo(chat_id, buf)

      # Limpiar el buffer y el plot
    buf.truncate(0)
    buf.seek(0)
    plt.clf()
def send_latex_code(latex_text, chat_id):
    try:
        # Configurar el fondo de la figura en blanco
        plt.figure(facecolor='white')
        # Configurar el texto de Latex en la imagen
        plt.text(0.5, 0.5, f"${latex_text}$", fontsize=35,
 horizontalalignment='center')
        plt.axis("off")

        # Guardar la imagen en un buffer
        buf = io.BytesIO()
        plt.savefig(buf, format="png")
        buf.seek(0)

        # Enviar la imagen al chat
        bot.send_photo(chat_id, buf)

        # Limpiar el buffer y el plot
        buf.truncate(0)
        buf.seek(0)
        plt.clf()
    except Exception as e:
        bot.send_message(chat_id, f"Ha ocurrido un error al enviar la
 imagen: {e}")
        return


def RRLHCCC_RRLNHCCC(funcion, valores_iniciales, chat_id):
    try:
        n = sp.symbols('n')
        recurrencia_input_con_prefijo = funcion.strip() + "-f(n)"
        f = sp.Function('f')
        recurrencia = eval(recurrencia_input_con_prefijo)
        ecuacion_general = sp.rsolve(recurrencia, f(n))
```

```python
        sistema_ecuaciones = []
        for k, a_k in zip(range(len(valores_iniciales)), valores_iniciales):
            sistema_ecuaciones.append(ecuacion_general.subs(n, k) - a_k)
        constantes = sp.solve(sistema_ecuaciones)
        solucion_general = ecuacion_general.subs(constantes)
        latex_solucion_general = sp.latex(solucion_general)


        send_latex_code(latex_solucion_general, chat_id)


    except:
        bot.send_message(chat_id, f"Ha ocurrido un error, estamos trabajando para resolver mas posibles casos")
        return


def sucesion(secuencia, chat_id):
    try:
        n = sp.symbols('n')
        # Si la cadena ingresada es un string, se convierte en lista de enteros
        if type(secuencia) != list:
            secuencia = [int(x.strip()) for x in secuencia.split(",")]

        # Se halla los coeficientes de la funcion recurrente
        coeficientes = sp.sequence(secuencia, (n, 1, len(secuencia))).find_linear_recurrence(len(secuencia))

        cadena = ""
        valor_inicial = ""

        # Se escribe la funcion recurrente con los coeficientes encontrados
        # y se crea una lista con los valores iniciales
        for i in range(len(coeficientes)):
            if coeficientes[i] != 0:
                if i == 0:
                    cadena = f"{coeficientes[i]}*f(n-{i + 1})"
                    valor_inicial = f"{secuencia[i]}"
                else:
                    if coeficientes[i] < 0:
                        cadena = cadena + f"{coeficientes[i]}*f(n-{i + 1})"
                    else:
                        cadena = cadena + f"+{coeficientes[i]}*f(n-{i + 1})"

                    valor_inicial = valor_inicial + f",{secuencia[i]}"

        bot.send_message(chat_id, f"\nFuncion recurrente: {cadena}\n")
```

```python

        # Se envian los valores iniciales por mensaje
        bot.send_message(chat_id, f"\nValores iniciales: {valor_inicial}\n")
        valor_inicial = [int(val.strip()) for val in valor_inicial.split(",")]

        RRLHCCC_RRLNHCCC(cadena, valor_inicial, chat_id)
    except Exception:
        bot.send_message(chat_id,
                         "\nHa ocurrido un error, verifica que si estes escribiendo la sucesion correctamente\n Ejemplo: 1,2,3,4,5\n")
        return


def FG02(orden, expresion, chat_id):
    z = parse_expr('z')
    aux = [1, 0]
    try:
        f = parse_expr(expresion)
        Fz = Poly(f.series(x=z, x0=0, n=orden).removeO())
        fn = Fz.all_coeffs()

        if fn == None or fn == aux:
            bot.send_message(chat_id, "La expresion ingresada no es valida.")
            return
    except Exception as e:
        bot.send_message(chat_id, f"Error: {e}")
        return
    bot.send_message(chat_id, f"La secuencia generada es {fn[::-1]}")
    sucesion(fn[::-1], chat_id)

@bot.message_handler(commands=['start'])
def enviar(message):
    texto_html = '<b>BIENVENIDO, SOY BOT-RECURRENTE o BOT-ESTELAR</b> ' + '\n'
    texto_html += ' ' + '\n'
    texto_html += 'A continuacion aqui te dare mis dos personalidades cual quieres que yo sea?:' + '\n'
    texto_html += ' ' + '\n'
    texto_html += 'Comandos:' + '\n'
    texto_html += ' ' + '\n'
    texto_html += '<b>/Recurrente:</b> Modo recurrente' + '\n'
    texto_html += ' ' + '\n'
    texto_html += '<b>/Constelacion :</b> Modo constelaciones' + '\n'
    texto_html += ' ' + '\n'
    texto_html += '<b>/ayuda</b> Instrucciones de como utilizar esta eleccion ' + '\n'
    bot.reply_to(message, texto_html, parse_mode="html")

@bot.message_handler(commands=['recurrente','Recurrente'])
```

```python
def enviar(message):
    texto_html = '<b>BIENVENIDO, SOY BOT-RECURRENTE</b> ' + '\n'
    texto_html += ' ' + '\n'
    texto_html += 'A continuacion aqui te dare el menu de comandos que puedes hacer ' + '\n'
    texto_html += ' ' + '\n'
    texto_html += 'Comandos:' + '\n'
    texto_html += ' ' + '\n'
    texto_html += '<b>/FGO:</b> De la funcion generadora te doy la Funcion no recurrente' + '\n'
    texto_html += ' ' + '\n'
    texto_html += '<b>/f_n :</b> De la funcion recurrente te doy la Funcion no recurrente' + '\n'
    texto_html += ' ' + '\n'
    texto_html += '<b>/sec:</b> De una secuencia te doy la funcion no recurrente' + '\n'
    texto_html += ' ' + '\n'
    texto_html += '<b>/ayuda:</b> Instrucciones de como utilizar constelaciones ' + '\n'
    bot.reply_to(message, texto_html, parse_mode="html")

@bot.message_handler(commands=['Constelacion','constelacion'])
def enviar(message):
    chat_id = message.chat.id
    texto_html = '<b>BIENVENIDO, SOY BOT-CONSTELACIONES</b>' + '\n'
    texto_html += ' ' + '\n'
    texto_html += 'A continuacion aqui te dare el menu de comandos que puedes hacer ' + '\n'
    texto_html += ' ' + '\n'
    texto_html += 'Comandos:' + '\n'
    texto_html += ' ' + '\n'
    texto_html += '<b>/buscar:</b> Te pedire que me des el nombre de la constelacion que quieres ver' + '\n'
    texto_html += ' ' + '\n'
    texto_html += '<b>/Estrellas:</b> Te dare todas las estrellas que tengo ' + '\n'
    texto_html += ' ' + '\n'
    texto_html += '<b>/TODAS:</b> Te dare TODAS, TOOOODAS las constelaciones que tengo ' + '\n'
    texto_html += ' ' + '\n'
    texto_html += '<b>/ayuda:</b> Instrucciones de como utilizar constelaciones' + '\n'
    bot.reply_to(message, texto_html, parse_mode="html")


@bot.message_handler(commands=['ayuda', 'Ayuda'])
def enviar_ayuda(message):
    texto_help_html = 'Tranquilo aqui te dejo las instrucciones de como utilizarme ' + '\n'
    texto_help_html += ' ' + '\n'
    texto_help_html += '<b>INSTRUCCIONES BOT-RECURRENTE:</b> ' + '\n'
    texto_help_html += ' ' + '\n'
```

```
    texto_help_html += """<b>Para FGO: al ingresar el comando /FGO te saldra el mensaje "Ingresa la FGO", aqui debes escribir un mensaje con tu F(z)teniendo en cuenta que:</b>""" + '\n'
    texto_help_html += ' ' + '\n'
    texto_help_html += '        Si es fraccion: ()/() por ejemplo: (1)/(1-z)**2' + '\n'
    texto_help_html += ' ' + '\n'
    texto_help_html += '        Si es polinomio, por ejemplo: 1+z+z**2' + '\n'
    texto_help_html += ' ' + '\n'
    texto_help_html += ' Luego te saldra el mensaje "Ingresa el orden", este es el numero de la longitud de la secuencia, ejemplo: 20' + '\n'
    texto_help_html += ' ' + '\n'
    texto_help_html += ' Lo que recibiras sera un mensaje con la secuencia generada, la funcion de recurrencia y la funcion no recurrente ' + '\n'
    texto_help_html += ' ' + '\n'
    texto_help_html += ' ' + '\n'
    texto_help_html += """<b> Para La Funcion de Recurrencia: al ingresar el comando /f_n te saldra el mensaje "Ingresa la funcion de recurrencia:", aqui debes escribir un mensaje con tu f(n) teniendo en cuenta que:</b>""" + '\n'
    texto_help_html += ' ' + '\n'
    texto_help_html += '        Puede ser homogenea o no homogenea ' + '\n'
    texto_help_html += ' ' + '\n'
    texto_help_html += '        Solo se resuelven f(n) lineales ' + '\n'
    texto_help_html += ' ' + '\n'
    texto_help_html += '        El formato es: 2*f(n-2)-f(n-1)' + '\n'
    texto_help_html += ' ' + '\n'
    texto_help_html += ' recibiras un mensaje "Ingresa los valores iniciales separados por comas", por Ejemplo: 1,2' + '\n'
    texto_help_html += ' ' + '\n'
    texto_help_html += 'Lo que recibiras sera un mensaje con la secuencia generada y la funcion no recurrente ' + '\n'
    texto_help_html += ' ' + '\n'
    texto_help_html += ' ' + '\n'
    texto_help_html += """<b>Para la secuencia: al ingresar el comando /sec te saldra el mensaje "Ingresa la secuencia", aqui debes escribir un mensaje con la secuencia de la siguiente manera: 1,2,3,4,5,6,7, 8,9,10 Recomendaciones:</b>""" + '\n'
    texto_help_html += ' ' + '\n'
    texto_help_html += '        Ingresa al menos 10 valores ' + '\n'
    texto_help_html += ' ' + '\n'
    texto_help_html += '        Asegurate de poner las comas para separar los valores' + '\n'
    texto_help_html += ' ' + '\n'
    texto_help_html += ' Lo que recibiras sera un mensaje con la funcion recurrente y la funcion recurrente ' + '\n'
    texto_help_html += ' ' + '\n'
    texto_help_html += ' ' + '\n'
    texto_help_html += '<b>INSTRUCCIONES BOT-ESTELAR :</b> ' + '\n'
    texto_help_html += ' ' + '\n'
```

```python
    texto_help_html += ' ' + '\n'
    texto_help_html += ' /buscar : Te mostrara una lista de la constelaciones que tiene solo tienes que ingresar el nombre tal cual como esta en la lista ' + '\n'
    texto_help_html += ' ' + '\n'
    texto_help_html += ' /Estrellas : Aqui no tienes que hacer nada, yo te mando las entrellas y tu feliz ' + '\n'
    texto_help_html += ' ' + '\n'
    texto_help_html += '/TODAS : Aqui tampoco te tienes que preocupar, te dare TOOOOOODAS las constelaciones que tengo. ' + '\n'
    bot.reply_to(message, texto_help_html, parse_mode="html")


@bot.message_handler(commands=['f_n'])
def enviar_respuesta_f_n(message):
    chat_id = message.chat.id

    def obtener_recurrencia(m):
        recurrencia = m.text.strip()
        bot.send_message(chat_id, f"Recurrencia ingresada: {recurrencia}")
        bot.send_message(chat_id, "Ingresa los valores iniciales separados por comas:")
        bot.register_next_step_handler(m, obtener_valores_iniciales, recurrencia)

    def obtener_valores_iniciales(m, recurrencia):
        try:
            valores_iniciales = [int(val.strip()) for val in m.text.split(",")]
        except Exception as e:
            bot.send_message(chat_id,
                             f"Ha ocurrido un error recuerda: Ingresa los valores iniciales separados por comas")
            return
        bot.send_message(chat_id, f"Valores iniciales ingresados: {valores_iniciales}")
        print(recurrencia)
        print(valores_iniciales)

        RRLHCCC_RRLNHCCC(recurrencia, valores_iniciales, chat_id)

    bot.send_message(chat_id, "Ingresa la recurrencia:")
    bot.register_next_step_handler(message, obtener_recurrencia)


@bot.message_handler(commands=['sec'])
def enviar_respuesta_sec(message):
    chat_id = message.chat.id

    def obtener_sec(m):
        sec = m.text.strip()
        bot.send_message(chat_id, f"Recurrencia ingresada: {sec}")
```

```
        obtener_valores_iniciales(m, sec)

    def obtener_valores_iniciales(m, sec):
        sucesion(sec, chat_id)

    bot.send_message(chat_id, "Ingresa la secuencia:")
    bot.register_next_step_handler(message, obtener_sec)


@bot.message_handler(commands=['FGO'])
def enviar_respuesta_fgo(message):
    chat_id = message.chat.id

    def obtener_orden(m, Fgo):
        orden = m.text.strip()
        bot.send_message(chat_id, f"Orden ingresado: {orden}")
        FGO2(int(orden), Fgo, chat_id)

    def obtener_funcion(m):
        Fgo = m.text.strip()
        bot.send_message(chat_id, f"Funcion ingresada: {Fgo}")
        bot.send_message(chat_id, "Ingresa el orden n (Ejemplo: 15):")
        bot.register_next_step_handler(m, obtener_orden, Fgo)

    bot.send_message(chat_id, "Ingresa la funcion generadora:")
    bot.register_next_step_handler(message, obtener_funcion)


@bot.message_handler(commands=['buscar'])
def obtener_nombre(message):
    chat_id = message.chat.id

    def llamar_funcion(m):
        nombre = m.text.strip()
        bot.send_message(chat_id, f"Nombre ingresado: {nombre}")
        buscar(nombre, chat_id)

    texto_html = '<b>Lista de constelaciones: </b>\n\n'
    texto_html += '<b>Boyero</b>\n'
    texto_html += '<b>Casiopea</b>\n'
    texto_html += '<b>Cazo</b>\n'
    texto_html += '<b>Cygnet</b>\n'
    texto_html += '<b>Geminis</b>\n'
    texto_html += '<b>Hydra</b>\n'
    texto_html += '<b>OsaMayor</b>\n'
    texto_html += '<b>OsaMenor</b>\n'

    bot.send_message(chat_id, texto_html, parse_mode="html")
    bot.send_message(chat_id, "Por favor, ingresa un nombre:")
    bot.register_next_step_handler(message, llamar_funcion)

@bot.message_handler(commands=['Estrellas'])
```

```python
def mostrar_todas_estrellas(message):
    chat_id = message.chat.id
    bot.send_message(chat_id, f"Aqui estan todas las estrellas")
    mostrarEstrellas(chat_id)

@bot.message_handler(commands=['TODAS'])
def mostrar_todas_estrellas(message):
    chat_id = message.chat.id
    bot.send_message(chat_id, f"Aqui estan TODAS las constelaciones")
    mostrar_constelaciones(chat_id)

@bot.message_handler(content_types=['text'])
def non_command_handler(message):
    command = message.text
    if command != "/help" or command != "/FGO" or command != "/Help" or
   command != "/f_n" or command != "/sec":
        bot.send_message(message.chat.id, "?")
        bot.send_message(message.chat.id,
                              "Lo siento, el comando: ('{}')  no existe,
   utilice /ayuda para saber que comando utilizar".format(
                                   command))
if __name__ == '__main__':
    print('Inicio el bot')
    bot.infinity_polling()
```

11.3.2. Tensores

- Filas, columnas, diagonales y ramas en un tensor

```
def fliplr(j):
  return nc-1-j
def flipud(i):
  return nf-1-i

nf = int(input())
nc = int(input())

#Se llena el tensor con unos
tensor = np.ones((nf, nc, 3))

# Dupla aleatoria
I, J = np.random.randint(0,nf-1),np.random.randint(0,nc-1)
print(I,J)

# Modificacion RGB
tensor[I,:,0] = 0  # Fila
tensor[:,J,1] = 0  # Columna

for i in range(min(I, J) + 1):
    tensor[I - i, J - i, 2] = 0  # dp arriba

for i in range(min(nf - I, nc - J)):
    tensor[I+i, J+i, 2] = 0  # dp abajo

for i in range(min(nf - I, J + 1)):
    tensor[I+i, J-i, 0] = 0 # ds arriba

for i in range(min(I+1, nc-J)):
    tensor[I-i, J+i, 0] = 0  # ds abajo

for i in range(nf): #r14
  j = J + abs(i-I)
  if j < nc:
    tensor[i, j, 1] = 0

for i in range(nf): #r23
  j = J - abs(i-I)
  if j >=0:
    tensor[i, j, 1] = 0

for j in range(nc): #r12
  i = I - abs(j-J)
  if i >=0:
    tensor[i, j, 0] = 0

for j in range(nc): #r34
  i = I + abs(j-J)
  if i < nf:
```

```
    tensor[i, j, 0] = 0
# Plot the tensor
plt.imshow(tensor, interpolation='nearest')
```

- Tensores y gráficas

```
def rectangulo(i, j, subtensor, tensor, int1=0, int2=0):
  if i == f - 1 and j == c - 1:
      subtensor[i, j] = [255, 0, 0]
      actualizar()
  elif i == f - 1:
      subtensor[i, j] = [0, 255, int2]
      rectangulo(0, j + 1, subtensor, tensor, int1, int2+10)
  else:
    if i == 0 and j == 0:
      subtensor[i, j] = [255, 40, 255]
    else:
      subtensor[i, j] = [0, int1, 255]
    rectangulo(i + 1, j, subtensor, tensor, int1 + 10, int2)
```

```
def triangulo_isoceles(i,j,subtensor, tensor,  int1=0, int2=0):

  if (i==f-1 and j==c-1):
    subtensor[i, j] = [255, 0, 0]
    actualizar()
  elif (i==f-1):
    if i == f-1 and j == 0:
      subtensor[i, j] = [255, 40, 255]
    else:
      subtensor[i, j] = [0, 255, int2]
    triangulo_isoceles(abs((j+1)-(f-1)),j+1,subtensor,  int1, int2
   +10)
  else:
    subtensor[i, j] = [0, int1, 255]
    triangulo_isoceles(i+1,j,subtensor, tensor, int1 + 10, int2)
```

```
def triangulo_rectangulo(i,j,subtensor, tensor,int1=0, int2=0):
  if (i==n-1 and j==n-1):
    subtensor[i, j] = [255, 0, 0]
    actualizar()
  elif i==n-1:
    subtensor[i, j] = [0, 255, int2]
    triangulo_rectangulo(j+1,j+1,subtensor, tensor,  int1, int2+10)
  else:
    if i == 0 and j == 0:
      subtensor[i, j] = [255, 40, 255]
    else:
      subtensor[i, j] = [0, int1, 255]
    triangulo_rectangulo(i+1, j, subtensor, tensor,  int1 + 10, int2)
```

```
def rombo(i,j,subtensor, tensor, int1=0, int2=0):

```

```
  if (i==n-1 and j==v):
    subtensor[i, j] = [255, 0, 0]
    actualizar()
  elif (j==(n-1)-abs(v-i)):
    if i == 0 and j == v:
      subtensor[i, j] = [255, 40, 255]
    else:
      subtensor[i, j] = [0, 255, int2]
    rombo(i+1,abs(v-(i+1)),subtensor, tensor, int1, int2+10)
  else:
    subtensor[i, j] = [0, int1, 255]
    rombo(i,j+1,subtensor, tensor,int1 + 10, int2)
```

```
# Calcular los indices de inicio para el subtensor
def inicio_tensor(nf, nc, f, c):
  start_row = (nf - f) // 2
  start_col = (nc - c) // 2
  subtensor = tensor[start_row:start_row+f, start_col:start_col+c, :]
  return subtensor
```

```
# Actualizar el tesnor original con los datos modificados del
  subtensor
def actualizar():
  start_row = (nf - f) // 2
  start_col = (nc - c) // 2
  tensor[start_row:start_row + f, start_col:start_col + c, :] =
  subtensor
  plt.imshow(tensor)
  plt.axis('off')
```

```
def PNG():
  # URL de la imagen en linea
  url = 'https://raw.githubusercontent.com/Slrosales/
  PC_AGUACATE_202310/main/aguacate.png'

  # Obtener la imagen directamente desde la URL
  image = Image.open(requests.get(url, stream=True).raw)

  # Transformacion de la imagen a tensor
  tensor = np.array(image)
  tensor = tensor[:, :, :3]
  print(tensor.shape)

  # Obtencion del numero de filas y columnas del tensor obtenido
  nf, nc = tensor.shape[0], tensor.shape[1]
  return tensor, nf, nc
```

```
def SVG():
  # URL de la imagen en linea
  url = 'https://raw.githubusercontent.com/Slrosales/
  PC_AGUACATE_202310/126c303d636915ffcb3dbaa90425330df871ad1a/
  Aguacate.svg'
```

```

  # Obtener la imagen directamente desde la URL
  image = cairosvg.svg2png(url=url)
  image = io.BytesIO(image)
  image = Image.open(image)

  # Convertir la imagen en un tensor
  tensor = np.array(image)
  tensor = tensor[:, :, :3]
  print(tensor.shape)

  # Obtencion del numero de filas y columnas del tensor obtenido
  nf, nc = tensor.shape[0], tensor.shape[1]
  return tensor, nf, nc
```

```
def EPS():
  # URL de la imagen en linea
  eps_url = 'https://raw.githubusercontent.com/Slrosales/
   PC_AGUACATE_202310/main/avocado.eps'

  # Descargar el archivo EPS
  response = requests.get(eps_url)
  eps_bytes = response.content

  # Abrir el archivo EPS
  image = Image.open(io.BytesIO(eps_bytes))

  # Guardar la imagen en formato PNG
  png_path = 'avocado.png'
  image.save(png_path, 'PNG')

  # Convertir la imagen en un tensor
  tensor = np.array(image)
  tensor = tensor[:, :, :3]
  print(tensor.shape)

  # Obtencion del numero de filas y columnas del tensor obtenido
  nf, nc = tensor.shape[0], tensor.shape[1]
  return tensor, nf, nc
```

Menú para que el usuario elija que figura quiere dibujar

```
print("1. Rectangulo")
print("2. Triandulo Isoceles")
print("3. Triangulo Rectangulo")
print("4. Rombo")
print("5.salir")

res = int(input("Que le quieres aplicar a la imagen?: "))
while (res <1 or res > 5):
  res = int(input("Error, intenta nuevamente: "))

#PNG
```

```
# URL de la imagen en Google Drive
url = 'https://raw.githubusercontent.com/Slrosales/PC_AGUACATE_202310
    /main/aguacate.png'

# Obtener la imagen directamente desde la URL
image = Image.open(requests.get(url, stream=True).raw)

tensor = np.array(image)
tensor = tensor[:, :, :3]
print(tensor.shape)
nf, nc = tensor.shape[0], tensor.shape[1]

print("\n")
if res == 1:
  c = int(input('Ingrese el numero de columnas para la figura: '))
  f = int(input('Ingrese el numero de filas para la figura: '))
  while c > nc or f > nf:
    print('El numero de columnas o filas excede el maximo permitido')
    c = int(input('Ingrese el numero de columnas para la figura: '))
    f = int(input('Ingrese el numero de filas para la figura: '))
  subtensor = inicio_tensor(nf, nc, f, c)
  rectangulo(0, 0, subtensor, tensor)

elif(res == 2):
  # Solicitar al usuario el numero de columnas para la figura
  f = int(input('Ingrese el numero de filas para la figura: '))
  # Verificar si el numero de columnas excede el maximo
  while 2*f-1 > nc or f > nf:
      print(f'El calulo del numero de filas o columas excede el
    maximo permitido ({f}, {2*f-1})')
      f = int(input('Ingrese el numero de filas para la figura: '))
  c=2*f-1
  subtensor = inicio_tensor(nf, nc, f, c)
  triangulo_isoceles(f-1,0, subtensor, tensor)

elif(res == 3):
  # Solicitar al usuario el numero de columnas para la figura
  n = int(input('Ingrese el numero de la matriz nxn para la figura: '
    ))

  # Verificar si el numero de columnas excede el maximo
  while n > nc or n > nf:
      print('El numero de columnas o filas excede el maximo permitido
    .')
      n = int(input('Ingrese el numero de la matriz nxn para la
    figura: '))
  f, c = n,n
  subtensor = inicio_tensor(nf, nc, f, c)
  triangulo_rectangulo(0,0,subtensor, tensor)

elif(res == 4):
  # Solicitar al usuario el numero de columnas para la figura
```

```
  n = int(input('Ingrese el numero de impar columnas para la figura: '))

  while n > nc or n > nf:
    print('El numero de columnas excede el maximo permitido.')
    n = int(input('Ingrese el numero impar de la matriz nxn para la figura: '))

  if (n%2==0): n=n+1
  v = (n-1)//2
  f, c = n,n
  subtensor = inicio_tensor(nf, nc, f, c)
  rombo(0, v, subtensor, tensor)
else:
  exit()
```

Variables con valores predeterminados para mostrar la imagen con las 4 figuras

```
# Crear una figura con una cuadricula de 2x2 subfiguras
fig, axs = plt.subplots(2, 2, figsize=(10, 10))

# Configurar los titulos de las subfiguras
axs[0, 0].set_title('Rectangulo')
axs[0, 1].set_title('Triangulo Isoceles')
axs[1, 0].set_title('Triangulo Rectangulo')
axs[1, 1].set_title('Rombo')

tensor, nf, nc = EPS()
# Crear el rectangulo en la primera subfigura
c = 20
f = 25
subtensor = inicio_tensor(nf, nc, f, c)
rectangulo(0, 0, subtensor, tensor)
axs[0, 0].imshow(tensor)
axs[0, 0].axis('off')

tensor, nf, nc = EPS()
# Crear el triangulo isosceles en la segunda subfigura
f = 20
c = 2 * f - 1
subtensor = inicio_tensor(nf, nc, f, c)
triangulo_isoceles(f - 1, 0, subtensor, tensor)
axs[0, 1].imshow(tensor)
axs[0, 1].axis('off')

tensor, nf, nc = EPS()
# Crear el triangulo rectangulo en la tercera subfigura
n = 28
f, c = n, n
subtensor = inicio_tensor(nf, nc, f, c)
triangulo_rectangulo(0, 0, subtensor, tensor)
axs[1, 0].imshow(tensor)
axs[1, 0].axis('off')
```

```

tensor, nf, nc = EPS()
# Crear el rombo en la cuarta subfigura
n = 22
if n % 2 == 0:
    n += 1
v = (n - 1) // 2
f, c = n, n
subtensor = inicio_tensor(nf, nc, f, c)
rombo(0, v, subtensor, tensor)
axs[1, 1].imshow(tensor)
axs[1, 1].axis('off')

# Ajustar el espaciado entre las subfiguras
fig.tight_layout()
```