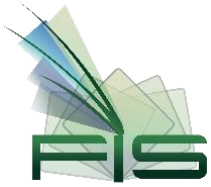

ESCUELA POLITÉCNICA NACIONAL
FACULTAD DE INGENIERÍA DE SISTEMAS

MÉTODOS NUMÉRICOS

PROYECTO II BIMESTRE

Andrés Fernández

Docente: Ing. Jonathan Zea

Integrantes: Andrés Fernandez, David Guachamin, David Puga, María Belén Raura

Carrera: Ciencias de la Computación
Curso: GR1CC
Grupo: 4
Fecha: 12/02/2025

Periodo Académico
2024 B

OBJETIVOS

Desarrollar un programa en Python para calcular y visualizar un fractal del Conjunto de Julia, implementando técnicas numéricas para asegurar precisión en los cálculos y utilizando una escala de colores que represente el número de iteraciones hasta la divergencia, permitiendo así un análisis visual e interactivo del comportamiento del fractal.

OBJETIVOS ESPECÍFICOS

- Determinar el número de iteraciones necesarias hasta que cada punto diverja, estableciendo un criterio de escape.
- Aplicar una escala de color para representar visualmente el número de iteraciones antes de la divergencia.
- Utilizar técnicas de métodos numéricos para garantizar precisión en los cálculos del conjunto de Julia.

INTRODUCCIÓN

Los fractales son estructuras matemáticas que cuentan con la característica de tener una complejidad infinita y la repetición de patrones a diferentes escalas. Estos diagramas no solo poseen un atractivo visual, sino que también cuentan con aplicaciones en campos de las ciencias como la física, la biología, la economía y en este caso ciencias de la computación.

En este proyecto el enfoque reside en la generación y visualización de los fractales a partir del conjunto de puntos F_8 en un plano complejo, definidos mediante la ecuación iterativa:

$$z_{n+1} = z_n^2 + c$$

Donde la condición inicial y el criterio de divergencia dado por $|z| \leq 2$. La ecuación en este contexto tiene una representación clásica de los sistemas dinámicos no lineales y permite explorar la frontera entre los puntos que divergen y aquellos que estarán acotados permitiendo de esta manera la creación de patrones visuales complejos y fascinantes. La riqueza de los patrones y la diversidad de formas que emergen de la ecuación iterativa manifiestan la magnificencia de las matemáticas y su capacidad para describir caos de manera ordenada.

Como se mencionaba anteriormente en los objetivos este trabajo es para obtener el cálculo del número de iteraciones necesarias para que los puntos en $c \in \mathbb{C}$ diverjan y representen de manera visual por medio de Python los resultados para cada punto en el plano complejo. Se vera el coloreado de acuerdo con el número de iteraciones requeridas para alcanzar el límite de divergencia, el fractal generado mostrara los detalles y variaciones cromáticas siendo el enfoque no solo visualizar la dinámica del sistema, sino también identificar las zonas estables y las zonas de caos en el plano.

MARCO TEÓRICO

El conjunto de Mandelbrot es un fractal definido mediante una sucesión iterativa en el plano complejo. Cada punto ccc en el plano complejo es evaluado aplicando la función

$$z_{n+1} = z_n^2 + c$$

generando una secuencia de valores que pueden divergir a infinito o permanecer acotados.

Si la sucesión permanece acotada para un número infinito de iteraciones, el punto ccc pertenece al conjunto de Mandelbrot; en caso contrario, no pertenece. Matemáticamente, esto se expresa como:

$$\limsup_{n\to\infty} |z_n| < 2 \Rightarrow c \in \text{Mandelbrot}$$

Por otro lado, si $|z_n|$ supera el valor 2 en alguna iteración, se garantiza que la sucesión diverge.

Este conjunto ha sido ampliamente estudiado en matemáticas y física debido a su estructura fractal infinita y su complejidad emergente a partir de una regla iterativa simple. Es utilizado en gráficos computacionales, caos y sistemas dinámicos para estudiar comportamientos no lineales.

Metodología

1. Análisis del Problema

En esta fase inicial, se identifican los principales objetivos del proyecto:

- Cálculo del conjunto de Julia: Se calcula el número de iteraciones necesarias para que cada punto c∈ C diverja, utilizando la función de iteración

$$z_{n+1} = z_n^2 + c$$

- Coloreo de puntos: Cada punto se colorea de acuerdo al número de iteraciones antes de que diverja, lo que genera una representación visual del conjunto.
- Optimización con métodos numéricos: Se emplean técnicas numéricas para asegurar que el cálculo de las iteraciones sea eficiente y preciso.
- Ajuste con mínimos cuadrados: Se utilizará el método de mínimos cuadrados para encontrar una función polinómica que ajuste el número de iteraciones como una aproximación a los puntos del conjunto.

2. Diseño del Algoritmo

El algoritmo se divide en dos etapas fundamentales:

1. **Cálculo del conjunto de Julia**:

- Se genera una grilla de puntos en el plano complejo dentro de los límites definidos por xmin,xmax,ymin,ymax.
- Para cada punto, se aplica la fórmula iterativa

  $$z_{n+1} = z_n^2 + c$$

  hasta que el valor de |Z| supera 2 o se alcanza el número máximo de iteraciones.
- Se guarda el número de iteraciones antes de que el punto diverja, lo que proporciona el mapa de colores.

2. **Ajuste con mínimos cuadrados**:

   - Se representa cada punto en el plano con sus coordenadas (x,y)(x, y)y el número de iteraciones z.
   - Se ajusta una función polinómica de segundo grado en x y y, es decir,

     $$f(x, y) = ax^2 + by^2 + cxy + dx + ey + f$$

     a través del método de **mínimos cuadrados**.
   - Para el ajuste, se utiliza la librería scipy.optimize.curve_fit, que optimiza los coeficientes del polinomio para minimizar el error cuadrático.

3. Implementación en Python

La implementación se realiza en Python utilizando las siguientes librerías:

- **NumPy**: Para realizar cálculos con matrices y operaciones complejas.
- **Matplotlib**: Para generar la visualización del conjunto de Julia y los puntos de iteración.
- **SciPy**: Para realizar la regresión por mínimos cuadrados y ajustar la función.

Desarrollo:

Las herramientas utilizadas por lenguajes de programación **Python** y en las librerías estaría **numpy** para operaciones matemáticas y manipulación de arreglos, **matplotlib** para la visualización de los fractales, **pillow** para la manipulación de imágenes y **ttkbootstrap** en la creación de interfaz gráfica todas con un enfoque especifico en la resolución del problema principal.

En cuanto al algoritmo y lógica de la programación se desarrolló un conjunto de scripts que generan estos fractales de Julia y Mandelbrot con la siguiente lógica:

- Creación de la cuadrícula de puntos complejos: Se definieron rangos de valores para los ejes X e Y, por ejemplo, $-1.5 < x < 1.5$ y $–1.2 < x < 1.2$ y, con una resolución de 800x800 píxeles.
- Iteración: Para cada punto en la cuadrícula, se aplicó la ecuación iterativa hasta que se cumplió la condición de divergencia o se alcanzó un límite máximo de iteraciones.
- Asignación de colores: Se usaron funciones logarítmicas para mapear el número de iteraciones a una escala de colores, mejorando la visualización de los detalles.

El proyecto incluye una interfaz gráfica desarrollada con **Tkinter** y **ttkbootstrap**, que permite a los usuarios seleccionar diferentes scripts para visualizar variaciones del conjunto de Julia. Además, se implementó funcionalidad de zoom interactivo utilizando eventos del ratón:

- Zoom In: Al hacer clic izquierdo en una región de la imagen, se reduce el rango visible, permitiendo acercar los detalles del fractal.
- Zoom Out: Al hacer clic derecho, se amplía el rango visible, alejándose de la imagen.

Resultados:

¿Cómo afecta la resolución de la cuadrícula al resultado visual?

La resolución de la cuadrícula tiene un impacto directo en la calidad visual del fractal. Una mayor resolución (mayor cantidad de píxeles) permite visualizar más detalles, haciendo que las transiciones entre las áreas de divergencia y no divergencia sean más suaves y precisas. Por el contrario, una resolución baja puede generar una imagen pixelada y menos detallada, perdiendo la riqueza de las estructuras fractales.

En este proyecto, se utilizó una resolución estándar de 800x800 píxeles, que ofrece un buen balance entre calidad visual y rendimiento computacional. Sin embargo, al aumentar la resolución, también incrementa el tiempo de cálculo, ya que el algoritmo debe procesar más puntos en el plano complejo.

¿Cuál es la resolución máxima de su programa?

La resolución máxima está limitada por la capacidad de procesamiento del hardware utilizado y la optimización del código. En pruebas realizadas, el programa soportó resoluciones de hasta 2000x2000 píxeles, aunque con un tiempo de cálculo considerablemente mayor. Superar esta resolución puede requerir técnicas de optimización adicionales, como la paralelización del procesamiento o el uso de hardware gráfico especializado (GPU).

Análisis y Discusión:

Primero debemos abarcar los problemas encontrados y como lo solucionamos

- Lentitud en la generación de imágenes de alta resolución: Se optimizó el código utilizando operaciones vectorizadas de NumPy para acelerar los cálculos.
- Gestión de eventos de zoom: Se implementó un sistema eficiente para actualizar los límites de la visualización sin necesidad de recalcular toda la imagen desde cero.

Con pruebas comparativas con fractales estándar para asegurar la precisión del algoritmo. Además, se verificó la estabilidad del programa al manejar resoluciones altas y la capacidad de la interfaz para gestionar eventos de zoom sin errores para obtener una validación.

Obtuvimos una serie de resultados intermedios que generaron diferentes conjuntos de Julia variando el valor de C y la resolución de la cuadrícula. Las pruebas demostraron que, al aumentar la resolución, los detalles finos del fractal emergen con mayor claridad, revelando la complejidad infinita de estas estructuras matemáticas.

Y ya para concluir, el proyecto nos ayudó a comprender la importancia de la resolución y la optimización en la generación de fractales. La interactividad, a través de la funcionalidad de zoom, ha enriquecido la experiencia de visualización, permitiendo explorar la complejidad de los fractales a diferentes escalas. Sin embargo, también se

identificaron áreas para mejoras futuras, como la implementación de paralelización más avanzada y la adaptación del código para aprovechar el procesamiento por GPU.

CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES

1. La implementación del conjunto de Julia mediante métodos numéricos permitió visualizar de manera efectiva la complejidad de los fractales, validando la importancia de la precisión en los cálculos iterativos.
2. El uso de Python y librerías especializadas como NumPy y Matplotlib facilitó el desarrollo del algoritmo, optimizando la representación gráfica y mejorando la eficiencia computacional.
3. La aplicación del ajuste por mínimos cuadrados mostró resultados prometedores para modelar la relación entre el número de iteraciones y la ubicación de los puntos en el fractal, aunque con ciertas limitaciones en la aproximación.
4. La funcionalidad de zoom interactivo permitió una exploración detallada de las estructuras fractales, mejorando la experiencia de análisis visual.

RECOMENDACIONES

1. Se recomienda mejorar el uso de operaciones vectorizadas con NumPy para reducir el tiempo de ejecución en la generación del fractal. Además, se podría explorar el uso de multiprocessing o joblib para paralelizar cálculos y aprovechar mejor los recursos del procesador.
2. Se sugiere incluir otros conjuntos fractales, como el conjunto de Mandelbrot, con el fin de comparar sus propiedades y su visualización en distintos contextos.

3. Resulta conveniente considerar el uso de librerías como Pillow o OpenCV para mejorar la manipulación y visualización de imágenes fractales dentro del entorno de Python.
4. Se recomienda realizar pruebas con distintas configuraciones de parámetros para analizar su impacto en la calidad de los resultados y su estabilidad numérica.

BIBLIOGRAFÍA

1. Barnsley, M. F. (2012). *Fractals Everywhere*. Academic Press.
2. Falconer, K. (2014). *Fractal Geometry: Mathematical Foundations and Applications* (3rd ed.). John Wiley & Sons.
3. Peitgen, H.-O., Jürgens, H., & Saupe, D. (2004). *Chaos and Fractals: New Frontiers of Science* (2nd ed.). Springer.
4. Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (2007). *Numerical Recipes: The Art of Scientific Computing* (3rd ed.). Cambridge University Press.
5. Trefethen, L. N., & Bau, D. (1997). *Numerical Linear Algebra*. SIAM.