ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

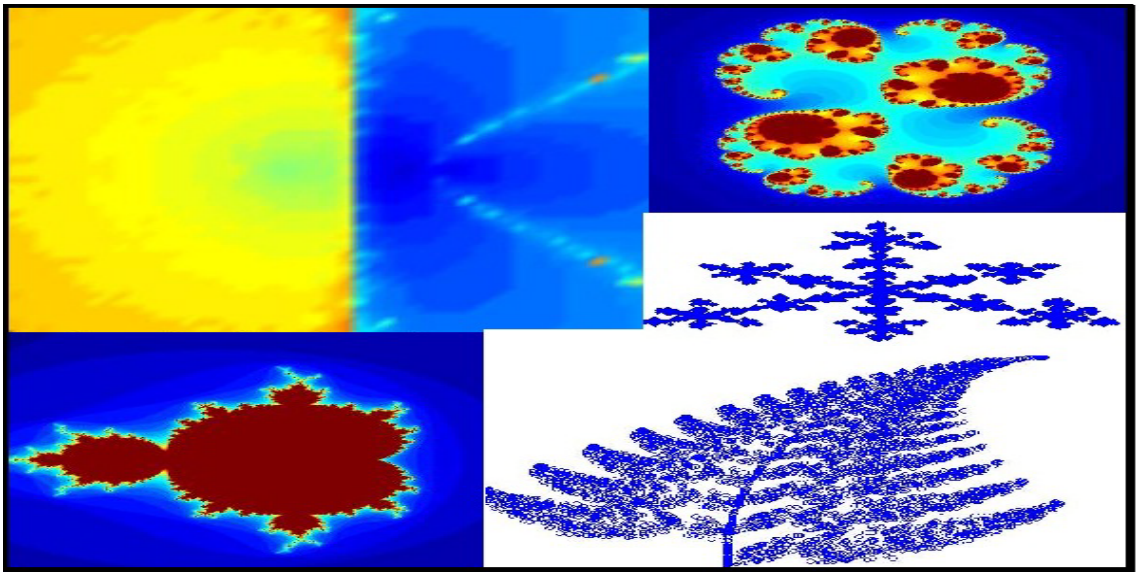
Proyecto final de carrera

CONJUNTOS DE JULIA Y MANDELBROT.

CUENCAS DE ATRACCIÓN.

SISTEMAS DE FUNCIONES ITERADAS.

APLICACIONES.



AUTOR: Juan Molero Jiménez

DIRECTORES: Juan Carlos Trillo Moya

Concepción Bermúdez Edo

Conjuntos de Julia y Mandelbrot. Cuencas de atracción. Sistemas de Funciones Iteradas. Aplicaciones

Autor: Juan Molero Jiménez
Directores: Juan Carlos Trillo Moya
Concepción Bermúdez Edo

Diciembre 2011

Autor	Juan Molero Jiménez
E-mail del Autor	molero.juan@gmail.com
Director	Juan Carlos Trillo Moya Concepción Bermúdez Edo
E-mail del Director	jctrillo@upct.es
Codirector	
Título del PFC	Conjuntos de Julia y Mandelbrot. Cuencas de atracción. Sistemas de Funciones Iteradas.Aplicaciones
Resúmen	Estudio teórico de los Conjuntos de Julia y Mandelbrot y obtención de las imágenes generadas. Estudio de las cuencas de atracción de funciones basándose en métodos numéricos en el plano complejo, así como la generación de las imágenes que representan dicho estudio. Análisis de los Sistemas de Funciones Iteradas (IFS) y de los algoritmos utilizados para para la obtención de un fractal asociado a un sistema de funciones iteradas. Generación de programas que producen un efecto de movimiento, tanto simple como compuesto, sobre un determinado fractal. Programación en Matlab de los algoritmos y métodos mencionados anteriormente. Creación de una interfaz gráfica desde la cual pueden ejecutarse programas Matlab implementados al efecto.
Titulación	Ingeniero Técnico de Telecomunicaciones, Especialidad Telemática
Departamento	Matemática Aplicada y Estadística
Fecha de Presentación	Diciembre 2011

A mi familia, amigos y a mi perra Llum.

Índice general

Índice de figuras

1 Introducción

Un fractal es un objeto semigeométrico cuya estructura básica, fragmentada o irregular, se repite a diferentes escalas.

Vamos a estudiar con mayor precisión algunos conceptos referentes a los fractales y su temática:

1.1. El padre de los fractales

En 1975, Benoît B. Mandelbrot acuñó la palabra fractal en su ensayo *"Les objets fractales: Forme, hasard et dimension"*, de la editorial Flammarion, París. En la introducción de dicho ensayo se puede leer:

«*El concepto que hace de hilo conductor será designado por uno de los dos neologismos sinónimos, "objeto fractal" y "fractal", términos que he inventado, ..., a partir del adjetivo latino "fractus"...*»

En 1982 publica un nuevo libro, con gráficos espectaculares creados con la tecnología informática que, por aquel entonces, estaba a su disposición: *"The Fractal Geometry of Nature"*, de la editorial W.H. Freeman & Co., Nueva York. En este libro Mandelbrot propone una definición de fractal, pero a la vez reconoce que la definición no incluye ciertos conjuntos que, por otras razones, deben incluirse en la categoría de fractales:

«*Un fractal es, por definición, un conjunto cuya dimensión fractal es estrictamente mayor que su dimensión topológica.*»

Después de más de veinte años, se han propuesto otras muchas definiciones que también se aproximan, pero hasta ahora no existe ninguna comúnmente aceptada.

1.2. Concepto de estructura fractal

En 1990, Kenneth Falconer, en su obra titulada *"Fractal Geometry: Mathematical Foundations and Applications"*, de la editorial John Wiley and Sons, explica que una estructura fractal debe satisfacer alguna o algunas de las propiedades siguientes:

1. Posee detalle a todas las escalas de observación.
2. No es posible describirlo con geometría Euclidiana, tanto local como globalmente.
3. Posee alguna clase de autosemejanza, posiblemente estadística.
4. Su dimensión fractal es mayor que su dimensión topológica.
5. El algoritmo que sirve para describirlo es muy simple, y posiblemente de carácter recursivo.

La primera propiedad establece que un fractal no tiene ninguna escala característica, sino que cualquier escala es buena para representarlo. De esta forma, independientemente de la escala a la que nos encontremos, el nivel de detalle del fractal seguirá siendo el mismo.

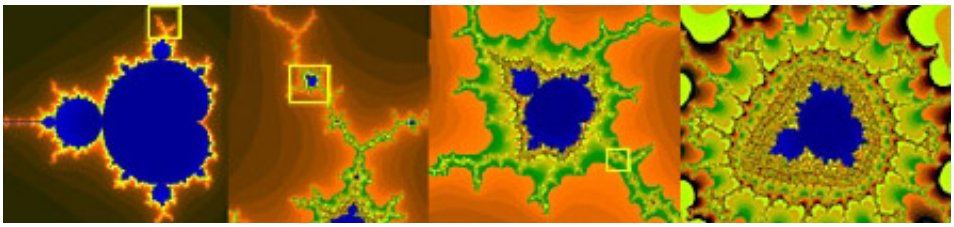


Figura 1.1: Un fractal no tiene una escala característica.

La segunda propiedad establece que al intentar representar un fractal en términos de geometría clásica, éste pierde todo el caos y autosimilitud que le caracteriza, siendo necesaria la creación de una nueva geometría, la geometría fractal.

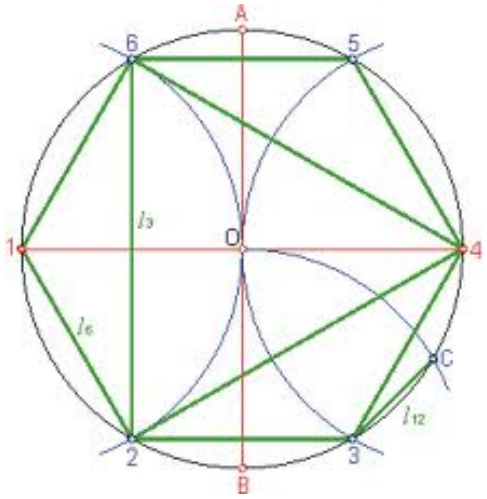


Figura 1.2: La geometría clásica es incapaz de representar fractales.

La tercera propiedad introduce el concepto de autosemejanza. En general, una estructura se dice autosemejante si puede ser construida como una reunión de estructuras, cada una de las cuales es una copia de la original pero a tamaño reducido.

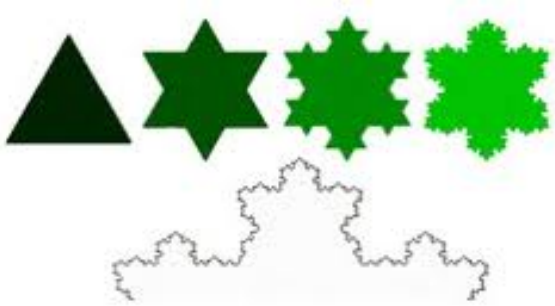


Figura 1.3: Formación del *Copo de nieve de Koch* a partir de tres curvas de Koch. Una de estas curvas se ve en la parte inferior de la imagen. La curva de Koch es autosemejante.

La cuarta propiedad introduce el concepto de dimensión fractal, concepto que se estudia con mayor profundidad más adelante.

Por último, la quinta propiedad establece que para generar un fractal basta con muy poca información. La clave se encuentra en la iteración, que consigue generar una gran cantidad de estructuras a partir de esa información inicial.

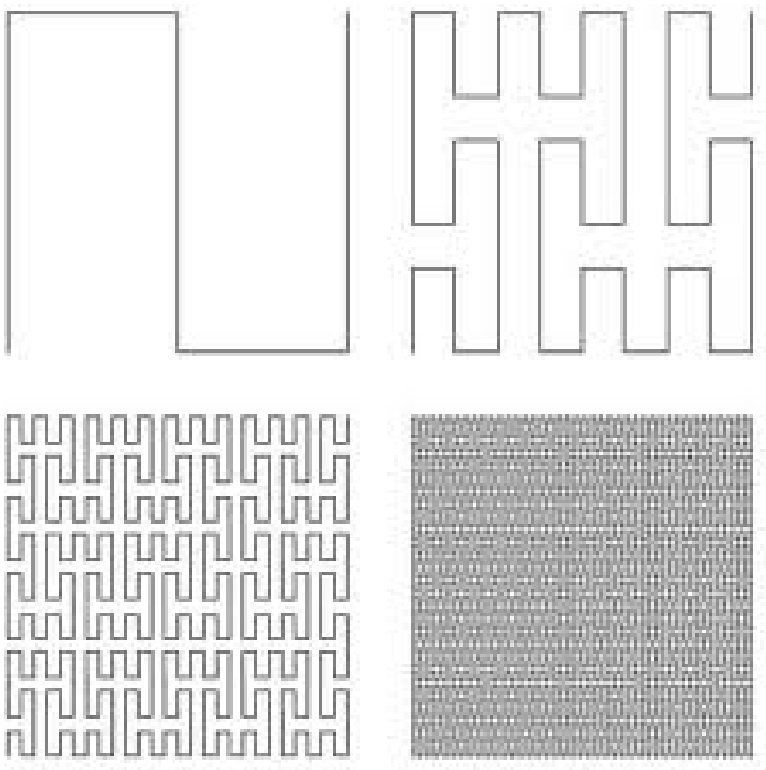


Figura 1.4: Varias iteraciones para la generación del fractal *Curva de Peano* que en el límite recubre el plano.

1.3. La dimensión fractal

La dimensión fractal es una característica fundamental de los objetos fractales. Es por ello que los matemáticos han definido varias formas de calcularla, y aunque no todas ellas son equivalentes, todas dan el mismo resultado cuando se aplican a conjuntos autosemejantes.

Un punto tiene dimensión 0, una línea dimensión 1, una superficie como un cuadrado dimensión 2 y un volumen como un cubo 3 dimensiones. Para determinar un lugar en una recta necesitamos un sólo dato (dimensión=1), si el conjunto es una superficie con 2 datos basta (dimensión=2) y si es un volumen necesitamos 3 datos (dimensión=3). Estas 3 dimensiones se corresponden con el número mínimo de coordenadas necesarias para fijar un punto en dicho conjunto.

Los objetos fractales y ciertos atractores a los que tiende el comportamiento de sistemas dinámicos caóticos, necesitan de otros tipos de dimensión. De hecho, no existen en las dimensiones euclídeas, sino en las dimensiones fraccionarias, que son descubrimiento del matemático alemán Félix Hausdorff que las describió ya en 1919 y que más tarde completó Besicovitch.

Para todo objeto de tamaño T, construido de unidades más pequeñas de tamaño t siendo N el número de veces que t cabe en T se cumple que su dimensión fractal o dimensión Hausdorff es:

$$N = \left(\frac{T}{t}\right)^D \Rightarrow \log(N) = D \cdot \log\left(\frac{T}{t}\right) \Rightarrow D = \frac{\log(N)}{\log\left(\frac{T}{t}\right)}$$

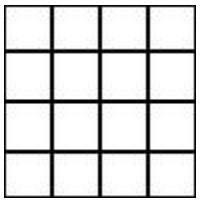
O si se prefiere, como T será por sistema la unidad y t un divisor de T, quedará que:

$$D = \frac{\log(N)}{\log\left(\frac{T}{t}\right)} = \frac{\log(N)}{\log\left(\frac{1}{\frac{1}{t}}\right)} = \frac{\log(N)}{\log(t)} \Rightarrow D = \frac{\log(N)}{\log(t)}$$

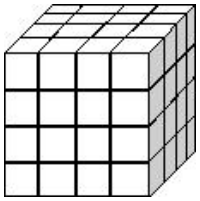
Así, para las 3 dimensiones euclídeas tenemos que coinciden con su homónimo Hausdorff-Besicovitch:

$$D = \frac{\log(N)}{\log\left(\frac{T}{t}\right)} = \frac{\log(4)}{\log\left(\frac{1}{\frac{1}{4}}\right)} = \frac{\log(4)}{\log(4)} = 1$$

$$D = \frac{\log(N)}{\log\left(\frac{T}{t}\right)} = \frac{\log(16)}{\log\left(\frac{1}{\frac{1}{4}}\right)} = \frac{2\log(4)}{\log(4)} = 2$$



$$D = \frac{\log(N)}{\log\left(\frac{T}{t}\right)} = \frac{\log(64)}{\log\left(\frac{1}{\frac{1}{4}}\right)} = \frac{3\log(4)}{\log(4)} = 3$$



Para determinar la dimensión fractal de una curva, la encerramos en una cuadrícula, de forma que la curva quede completamente dentro de ella. A continuación se cuentan todos aquellos cuadros de la cuadrícula que son cortados por la curva.

Este proceso se va repitiendo una y otra vez, pero en cada paso la longitud de los cuadros de la cuadrícula se va dividiendo entre dos.

Se representa por $N(k)$ el número de cuadros de la cuadrícula que son cortados por la curva en la iteración k. De esta forma, para un segmento rectilíneo, en cada iteración se duplicaría el número de cuadros cortados, ya que el lado de cada uno de ellos se ha reducido a la mitad. $N(k)$ se aproximaría entonces a:

$$N(k) \sim 2^k$$

Pero si la curva es una curva fractal, como la de la figura 1.5, entonces el número de cuadros que la cortan crecerá con k a un ritmo mayor. Se puede aproximar entonces $N(k)$ a:

$$N(k) \sim 2^D k, D > 1$$

El exponente D que multiplica a k es la dimensión fractal. La ecuación para obtener esta dimensión fractal es la siguiente:

$$D = \lim_{k \to \infty} \frac{\log N(k)}{\log k}$$

En la práctica, ya que es imposible examinar un número infinito de cuadrículas, se determina un rango adecuado para k según el objeto que se esté estudiando

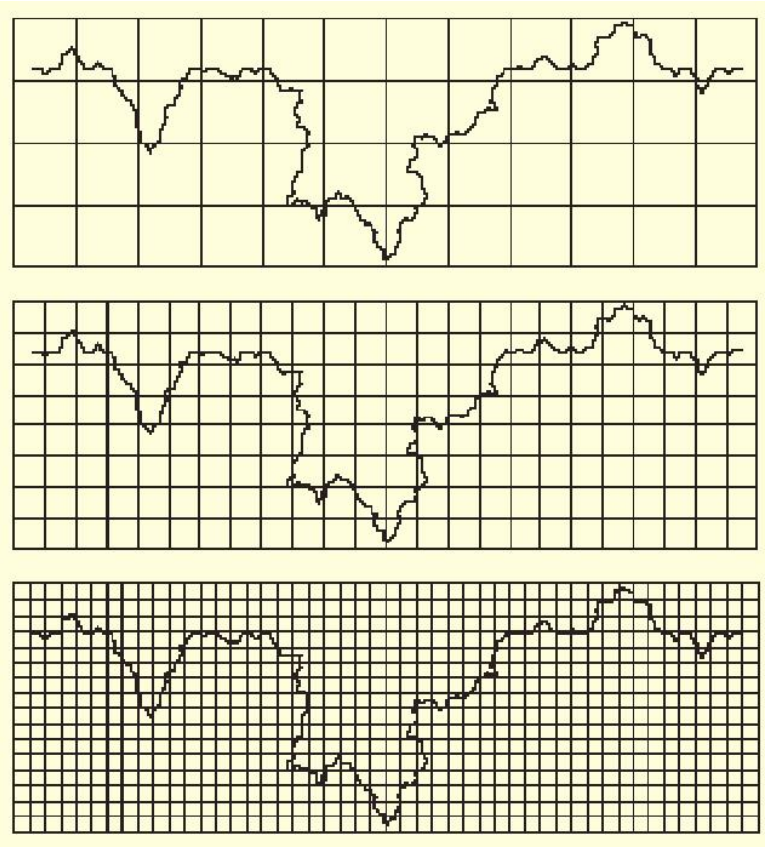


Figura 1.5: Cuadrícula para el cálculo de una curva fractal.

1.4. Aplicación de los fractales en las telecomunicaciones : Antenas fractales

El auge y crecimiento de las telecomunicaciones abren cada vez más las puertas de la exploración de nuevas alternativas en diseño que cubran las exigencias en ancho de banda, eficiencia, rapidez y economía. En la última década, una nueva y revolucionaria teoría: los fractales, se ha abierto paso, proponiendo modelos para el diseño de antenas permitiendo la implementación de nuevos y mejores servicios en los sistemas móviles, circuitos RFID (circuitos de identificación por radiofrecuencia), dispositivos de microonda y otros.

Las propiedades de los fractales, antes expuestas, se aprovechan en la construcción de antenas que pueden obtener anchos de banda de 10 a 40 % de la frecuencia central superiores a las antenas clásicas, patrones de radiación estables y gran número de bandas determinado por el número de iteraciones del fractal.

Podemos decir que una antena fractal posee estas 3 principales características especiales:

1. Un gran ancho de banda y comportamiento multibanda. El rango de frecuencia es especificada por el tamaño más pequeño y más grande presente en la antena.

2. En la mayoría de los casos tienen una ganancia considerable, por encima de una antena dipolo normal, y esta ganancia depende muy poco de la frecuencia en un rango de frecuencias grande.

3. Poseen un patrón de radiación estable para un rango amplio de frecuencias.

Los típicos inconvenientes de baja resistencia de radiación en el diseño de antenas cortas *(small antennas)* que operan a una longitud de onda mayor que su tamaño, pueden ser resueltos con curvas fractales que aumentan el perímetro de la antena conservando o minimizando su área.

Sin embargo, esta nueva concepción en el diseño de antenas, nos enfrenta ante problemas teóricos difíciles de resolver, ya que no podríamos tomar las expresiones de la teoría electromagnética en este tipo de curvas. Estos y otros inconvenientes, hacen indispensable utilizar complejos métodos numéricos para encontrar los campos. Pero por otro lado, estos métodos iterativos simplifican el cálculo de los campos de radiación de dichas antenas.

Los diseños y aplicaciones de las antenas fractales son muchos, dado que el avance de los sistemas de comunicaciones y el importante incremento de otras aplicaciones de los sistemas inalámbricos, las antenas de banda ancha y de bajo contorno, tienen gran demanda tanto para aplicaciones comerciales como militares.

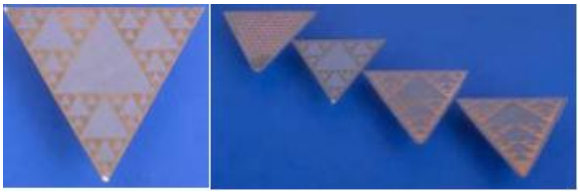
La teoría de los fractales esta abriendo un universo de posibilidades en la exploración de nuevas alternativas científicas para resolver y optimizar

sistemas.

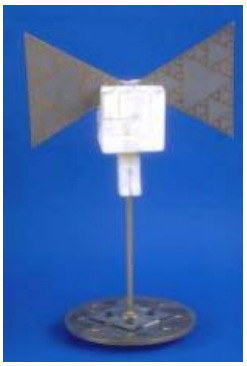
Sus propiedades especiales como la autosimilitud, rugosidad, dimensión fraccionaria, etc., se están empleando en el diseño de nuevas y mejores antenas que abrirán las posibilidades de las nuevas generaciones de sistemas de comunicaciones, permitiendo una integración eficiente de los nuevos servicios.

Actualmente los esfuerzos se centran principalmente en el diseño de antenas para los sistemas móviles, dando una solución barata, fácil y rápida.

Sin duda, estos nuevos diseños se constituirán en una pieza clave en el avance de los sistemas de telecomunicaciones.

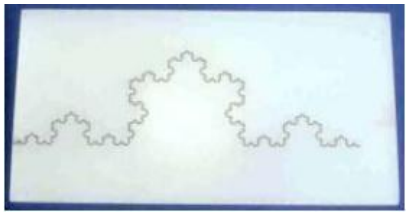


(a) Modelos tipo Sierpinski



(b) Antena monopolo en plano de tierra que usa tarjetas Sierpinski

(c) Antena en forma de árbol fractal



(d) Antena según modelo fractal de Koch

Figura 1.6: Ejemplos de antenas fractales

1.5. Organización del proyecto : Aplicaciones de los fractales

En este proyecto se estudian diferentes temáticas acerca de los fractales.

En el **Capítulo 2**, se explican los *Conjuntos de Julia y Mandelbrot* y se presentan los programas creados para su visualización.
Estas familias de conjuntos son las más famosas y estudiadas entre todos los conjuntos de fractales.
Los fractales se caracterizan por la belleza de sus formas, pero su uso no solo responde a manifestaciones artísticas, también son herramientas de gran potencia para el estudio de diferentes temas como pueden ser la compresión de imágenes, de audio y de vídeo, generación de efectos especiales, modelado del tráfico en redes, estudios de sistemas dinámicos (como se verá en el tercer capítulo), análisis de patrones sísmicos e incluso análisis bursátil y de mercado.

En el **Capítulo 3**, se lleva a cabo con mayor profundidad el estudio de *sistemas dinámicos en el cálculo de raíces de polinomios*. Se presentarán diferentes programas de métodos matemáticos para dicho cálculo y se analizará su dinámica mediante tres representaciones gráficas distintas.

En el **Capítulo 4** se ofrece una explicación teórica acerca de los *Sistemas de Funciones Iteradas*, una teoría unificada para la obtención de una amplia gama de objetos fractales.
El método consiste en establecer una semilla inicial y ejercer sobre ella una serie de transformaciones. Este resultado de sucesivas iteraciones recibe el nombre de atractor del sistema. El conjunto de transformaciones que se aplican para llegar hasta el atractor del sistema es lo que se denomina *Sistema de Funciones Iteradas (SFI)*.
Además se detallan los programas utilizados para representar diferentes imágenes basándose en estos sistemas.

En el **Capítulo 5** se desarrolla tanto la teoría como las aplicaciones para el desarrollo del *movimiento de fractales* utilizando los Sistemas de Funciones Iteradas como la base de esta temática.

En el **Capítulo 6** se documenta la *interfaz gráfica* creada donde se encuentran diferentes pantallas para la creación y visualización de todas las aplicaciones detalladas en los anteriores capítulos.

En el **Capítulo 7** se extraen algunas *conclusiones* acerca del trabajo realizado.

2 Conjuntos de Julia y de Mandelbrot

2.1. Conceptos Básicos

Sea $\overline{\mathbb{C}} = \mathbb{C} \cup \{\infty\}$ la esfera de Riemann. Más precisamente, pensamos la esfera de Riemann como la esfera 2-dimensional en $\mathbb{R}^3$,

$$S^2 = \{(x_1, x_2, x_3) \in \mathbb{R}^3 : x_1^2 + x_2^2 + x_3^2 = 1\}$$

dotada con el atlas analítico que contiene las cartas $\varphi_1 : \mathbb{C} \to S^2 - \{(0,0,1)\}$ y $\varphi_2 : \mathbb{C} \to S^2 - \{(0,0,-1)\}$, dadas por

$$\varphi_1(z) = \varphi_1(x_1, ix_2) = \left(\frac{2x_1}{x_1^2 + x_2^2 + 1}, \frac{2x_2}{x_1^2 + x_2^2 + 1}, \frac{x_1^2 + x_2^2 - 1}{x_1^2 + x_2^2 + 1} \right),$$

$$\varphi_2(z) = \varphi_2(x_1, ix_2) = \left(\frac{2x_1}{x_1^2 + x_2^2 + 1}, \frac{2x_2}{x_1^2 + x_2^2 + 1}, \frac{1 - x_1^2 - x_2^2}{x_1^2 + x_2^2 + 1} \right).$$

Estas cartas son llamadas *proyecciones estereográficas*, y determinan la estructura analítica de la esfera de Riemann. Recordemos que una función racional

$$R(z) = \frac{P(z)}{Q(z)},$$

donde P y Q son polinomios sin factores comunes, es una función analítica de $\overline{\mathbb{C}}$ en $\overline{\mathbb{C}}$. El grado es definido como

$$\text{grado} R = \max\{\text{grado}(P), \text{grado}(Q)\}.$$

Definición: *Dado un punto $z_0 \in \overline{\mathbb{C}}$, la órbita positiva de z_0 es el conjunto $O^+(z_0) = \{z_n : n = 0, 1, 2, \ldots\}$, donde $z_n = R(z_{n-1}) = R^n(z_0), n = 1, 2, \ldots$*

Una manera elemental de distinguir órbitas es contar el número de puntos en ellas.

Definición: *Sea $z_0 \in \overline{\mathbb{C}}$ y $O^+(z_0)$ su órbita por R. Decimos que $O^+(z_0)$ es periódica de período n si, $z_0 = R^n(z_0)$ y $R^j(z_0) \neq z_0$ para $1 \leq j \leq n-1$. Una órbita periódica de período uno, es decir, $R(z_0) = z_0$, la llamaremos punto fijo.*

Definición: *Sea $\alpha = \{z_0, R(z_0), \ldots, R^{n-1}(z_0)\}$ una órbita periódica de R. Su multiplicador $\lambda = \lambda(\alpha)$ es $(R^n)'(z_0)$*

Definición: *Decimos que una órbita periódica*

$$\alpha = \{z_0, R(z_0), \ldots, R^{n-1}(z_0)\}$$

es,

$$\left.\begin{array}{r} superatractora \\ atractora \\ indiferente \\ repulsora \end{array}\right\} \quad si\ y\ sólo\ si \quad \begin{cases} \lambda = 0 \\ 0 < |\lambda| < 1 \\ |\lambda| = 1 \\ |\lambda| > 1 \end{cases}$$

En lo que sigue nos centraremos sólo en la dinámica de funciones racionales R con $grado(R) \geq 2$.

Definición: *Un punto $z_0 \in \overline{\mathbb{C}}$ es un punto crítico de R si $R'(z_0) = 0$. Su valor $\omega_0 = R(z_0)$ es llamado valor crítico.*

Proposición: *Sea $R : \overline{\mathbb{C}} \to \overline{\mathbb{C}}$ una función racional de grado d. Entonces R tiene a los más $2d - 2$ puntos críticos.*

Definición: *Sea $z_0 \in \overline{\mathbb{C}}$ un punto fijo atractor. La cuenca de atracción de z_0 es el conjunto*

$$\mathcal{W}^S(z_0) = \{z_0 \in \overline{\mathbb{C}} : R^n(z) \to z_0 \quad cuando \quad n \to \infty\},$$

y su cuenca de atracción inmediata, $\mathcal{A}(z_0)$, es la componente conexa de $\mathcal{W}^S(z_0)$ que contiene a z_0.

Si $\alpha = \{z_0, R(z_0), \ldots, R^{n-1}(z_0)\}$ es una órbita periódica atractora, entonces su cuenca de atracción es

$$\mathcal{W}^S(\alpha) = \bigcup_{j=0}^{n-1} R^j(\mathcal{W}^S(z_0)).$$

Sean $D_j(\alpha)$ las componentes conexas de $\mathcal{W}^S(\alpha)$ que contienen los puntos $z_0, R(z_0), \ldots, R^{n-1}(z_0)$, respectivamente. Entonces $D(\alpha) = \bigcup_{j=0}^{n-1} D_j(\alpha)$ es llamada la cuenca de atracción inmediata de α.

Teorema: *(Fatou, Julia) La cuenca de atracción inmediata de una órbita periódica atractora contiene al menos un punto crítico.*

Corolario: *Una función racional $R : \overline{\mathbb{C}} \to \overline{\mathbb{C}}$ de grado d tiene a lo más $2d - 2$ órbitas periódicas atractoras*

Definición: *Una familia Γ de funciones meromorfas definidas en un dominio $U \subset \overline{\mathbb{C}}$*

$$\Gamma = \{f_i : U \to \overline{\mathbb{C}} : f_i \quad meromorfa\},$$

es normal si cada sucesión $(f_n)_{n\in\mathbb{N}}$ de elementos de Γ tiene una subsucesión $(f_{n_k})_{k\in\mathbb{N}}$ que converge uniformemente sobre cada subconjunto compacto de U.

Ahora nos centraremos en la familia de iterados $\{R^n : n = 0, 1, 2, 3, \ldots\}$ de una función racional $R : \overline{\mathbb{C}} \to \overline{\mathbb{C}}$.

Definición: *Un punto $z \in \overline{\mathbb{C}}$ pertenece al conjunto de Fatou $\mathcal{F}(R)$ (también llamado dominio de normalidad) si existe una vecindad U de z tal que la familia de iterados*

$$\Gamma = \{R^n : U \to \overline{\mathbb{C}} : n = 0, 1, 2, 3, \ldots\},$$

es normal en U.

El *Conjunto de Julia* de R, denotado por $\mathcal{J}(R)$ o simplemente por $\mathcal{J}$ cuando no exista peligro de confusión, es $\mathcal{J}(R) = \overline{\mathbb{C}} - \mathcal{F}(R)$. Es claro, a partir de su definición, que $\mathcal{F}(R)$ es abierto y en consecuencia $\mathcal{J}(R)$ es cerrado.

2.2. Conjuntos de Julia

2.2.1. Introducción

Gaston M. Julia y Pierre Fatou, trabajaron a principios del siglo XX (1918) en funciones de variable compleja. Iterándolas y observando su comportamiento, dieron con muchas de las propiedades básicas de la iteración en el plano complejo.

Comenzaron a trabajar con el polinomio $z^3 - 1 = 0$. Estudiaron las cuencas de atracción de sus soluciones y obtuvieron que la raíz actuaba como atractor para todo un conjunto de puntos z_0. Estas cuencas estaban muy enredadas y sus fronteras presentaban una complejidad inusitada.

Julia decidió que trabajaría sobre los polinomios de 2^o grado, que a buen seguro le darían más trabajo del que pudiera necesitar.
Con su trabajo se convirtió en un precursor de los sistemas dinámicos, sístemas muy sensibles a las condiciones iniciales, en las que una pequeña variación provoca un comportamiento radicalmente distinto del previsto; para un valor $z(0) = p$ la órbita generada era atraída al origen del plano, para otro valor $z(0) = p + 0{,}001i$ escapaba hacia el ∞.

De modo general entenderemos por sistema dinámico a un par (X, f) formado por un conjunto no vacío X y una aplicación $f : X \to X$. Dado un punto $x \in X$, se llama órbita de x a la sucesión $(f^n(x))$ donde f^n es la composición de f consigo misma n veces.

En un sistema dinámico (X, f) un punto $a \in X$ se dice que es un punto fijo si $f(a) = a$ y se dice punto periódico de periodo $n > 1$ si $f^n(a) = a$ y $f^i(a) \neq a$ para $n > i \geqslant 1$.

Los sistemas dinámicos complejos estudiados por Julia y Fatou eran (C, f_c) donde C es el campo complejo y $f_c : C \to C$. Para un número

complejo c, f_c viene dado por la expresión

$$f_c(z) = z^2 + c.$$

El trabajo de estos matemáticos se centró en determinar qué sucedía con un punto $z \in C$ en el sistema dinámico (C, f_c), llegando a la conclusión de que para ciertos valores de c, las órbitas de los puntos en un entorno del origen convergían a un punto fijo de la aplicación f_c, mientras que las órbitas de los puntos más alejados del orígen se iban al ∞. Cada uno de estos tipos de puntos constituyen una región y en medio queda una frontera «infinítamente delgada» que se conoce con el nombre de **Conjunto de Julia**.

Julia intuía la importancia de su trabajo pero no tuvo oportunidad durante sus años de fructífera producción-investigación matemática de gozar de un ordenador y proyectar una imagen de su conjunto en el monitor.

No todos los valores $z(0)$ que escapan al ∞ lo hacen a la misma velocidad, algunos lo hacen a la tercera iteración, otros a la décima y otros debemos iterarlos cientos de veces para preveer su comportamiento. Igualmente, tampoco todos los valores cuya órbita queda delimitada en el plano lo hacen a la misma velocidad.

Realmente no sabemos si realmente escapan o no al ∞, intuimos su comportamiento comprobando unas pocas iteraciones, sino estaríamos siempre calculando el primer valor dado a $z(0)$.

Lo riguroso sería iterarlo infinitas veces, pero como ello no es posible, nos limitamos a iterar las suficientes veces cada valor de $z(0)$ y estudiando la sucesión obtenida determinamos su tendencia. Según el valor escogido de $z(0)$ el crecimiento de la sucesión obtenida hacia el ∞ será más o menos rápido, o su tendencia a ser atraido por una cuenca de atracción más o menos rápida.

En la figura 2.1 vemos algunos ejemplos de Conjuntos de Julia.

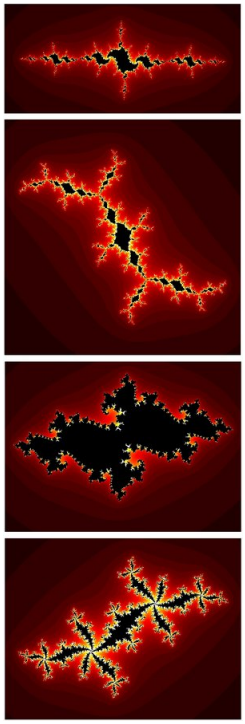


Figura 2.1: Ejemplos de varios Conjuntos de Julia.

2.2.2. Propiedades Básicas de los Conjuntos de Julia

En lo que sigue $R : \overline{\mathbb{C}} \to \overline{\mathbb{C}}$ será una función racional de grado mayor o igual que 2.

Proposición: *Sea $z_0 \in \overline{\mathbb{C}}$. Supongamos que $O^+(z_0)$ es periódica. Entonces si es (super)atractora, está contenida en el conjunto de Fatou, y si es repulsora está contenida en el conjunto de Julia de R.*

Proposición: *El conjunto de Julia $\mathcal{J}(R)$ es no vacío.*

Proposición: *Si z_r es un punto periódico repulsor, entonces*

$$\mathcal{J}(R) = \overline{\{z \in \overline{\mathbb{C}} : R^n(z) = z_r, n \in \mathbb{N}\}}.$$

Esta proposición nos da un algoritmo para graficar el conjunto de Julia. Para ello basta encontrar un punto fijo o periódico repulsor y considerar sus preimágenes. Computacionalmente este algoritmo es lento, pero para polinomios de grado pequeño es efectivo.

Proposición: *Fundamental (Julia, Fatou) Los puntos periódicos repulsores son densos en $\mathcal{J}(R)$, es decir,*

$$\mathcal{J}(R) = \overline{\{z \in \overline{\mathbb{C}} : z \quad \textit{punto periódico repulsor de } R\}}.$$

En particular, cada punto $z \in \mathcal{J}(R)$ es obtenido como límite de puntos periódicos repulsores.

Observación: De hecho, G. Julia comienza su memoria focalizando su atención sobre la clausura del conjunto de puntos periódicos repulsores, y muestra que su complemento es la unión de dominios sobre el cual la familia de iterados $\{R^n : n \geq 0\}$ es normal.

Teorema: *El conjunto de Julia de una función racional es un conjunto perfecto (conjunto cerrado en el que todos sus puntos son de acumulación).*

Definición: *Sea U un conjunto abierto no vacío de $\overline{\mathbb{C}}$. Definimos el conjunto de puntos omitidos de la aplicación racional $R|_U$ como el conjunto*

$$E_U = \overline{\mathbb{C}} - \bigcup_{n \geq 0} R^n(U).$$

Corolario: *Sea $z \in \mathcal{J}(R)$ y sea U una vecindad abierta de z. Entonces E_U contiene a lo más 2 puntos.*

Definición: *Sean R_1 y R_2 dos funciones racionales. Decimos que ellas son conjugadas si existe una transformada de Möbius $M : \overline{\mathbb{C}} \to \overline{\mathbb{C}}$ tal que*

$$R_1 = M^{-1} \circ R_2 \circ M.$$

Observación: *Si R_1 y R_2 son conjugadas por M, entonces se tiene que $M(\mathcal{J}(R_1)) = \mathcal{J}(R_2)$ y $M(\mathcal{F}(R_1)) = \mathcal{F}(R_2)$.*

Proposición: *El conjunto de puntos omitidos de una función racional R es independiente del punto $z \in \mathcal{J}$ usado para definirlo. Por lo tanto lo podemos denotar simplemente por E_R.*

Proposición: *Si $\overline{z} \in \mathcal{J}(R)$ entonces*

$$\mathcal{J}(R) = \overline{\{z \in \overline{\mathbb{C}} : R^n(z) = \overline{z}, \quad \textit{para algún } n \in \mathbb{N}\}}.$$

Proposición: *$R(\mathcal{J}(R)) = \mathcal{J}(R) = R^{-1}(\mathcal{J}(R))$, es decir, $\mathcal{J}(R)$ es completamente invariante.*

Proposición: *Los conjuntos de Julia de R y de $R^m, m \in \mathbb{N}$ son el mismo, esto es, $\mathcal{J}(R) = \mathcal{J}(R^m)$.*

Proposición: *Sea z_a un punto atractor para una función racional R, entonces $\mathcal{J}(R) = \delta\mathcal{W}^S(z_a)$ (δA denota la frontera del conjunto A).*

Esta proposición nos da una algoritmo bastante eficiente para graficar el conjunto de Julia. Para ello basta encontrar un punto fijo atractor z_a de R y fijando un error $\epsilon > 0$ pintamos de un color determinado los puntos z en una región acotada tales que para algún $n \geq 1$, se tiene $|R^n(z) - z_a| < \epsilon$. Este algoritmo se conoce como *algoritmo tiempo de escape*. Por otra parte, podemos definir los conjuntos de nivel, $L_k(z_a), k = 1, 2, 3, \ldots$ como sigue: sea $0 < \epsilon << 1$ y sea

$$L_0(z_a) = \{z \in \mathbb{C} : |z - z_a| < \epsilon\},$$

y para $k = 0, 1, \ldots$

$$L_{k+1}(z_a) = \{z \in \mathbb{C} - L_k(z_a) : R^k(z) \in L_k(z_a)\}.$$

se tiene que

$$\delta L_k(z_a) \to \mathcal{J}(R),$$

donde el límite es tomado respecto a la métrica de Hausdorff en $\mathcal{K}(\overline{\mathbb{C}}) = \{K \subset \overline{\mathbb{C}} : \text{K es compacto}\}$.

Una forma de obtener gráficas vistosas del conjunto de Julia con este algoritmo es colorear cada conjunto de nivel L_k con el color k correspondiente.

Teorema: *(Montel) Si $z \in \mathcal{J}(R)$ y U es una vecindad de z, entonces $\{R^m(U)\}_{m\in\mathbb{N}}$ cubre todo $\overline{\mathbb{C}}$, excepto a lo más dos puntos.*

Corolario: *Si $\mathcal{J}(R)$ tiene un punto interior, entonces $\mathcal{J}(R) = \overline{\mathbb{C}}$. En consecuencia, $\mathcal{F}(R)$ es vacío.*

Demostración: *Sea U un dominio contenido en $\mathcal{J}(R)$. Como $\mathcal{J}(R)$ es invariante,*

$$\mathcal{J}(R) \supset \cup_{n\geq 0} R^n(U) = \overline{\mathbb{C}} - E_R.$$

Además, como $\mathcal{J}(R)$ es cerrado y E_R contiene a lo más dos puntos, se tiene que $\mathcal{J}(R) = \overline{\mathbb{C}}$.

Corolario: *Si D es un dominio con $D \cap \mathcal{J}(R) = \mathcal{J}^* \neq \emptyset$, entonces existe $m \in \mathbb{N}$ tal que $R^m(\mathcal{J}^*) = \mathcal{J}(R)$.*

Definición: *Sea D una componente del conjunto de Fatou. El dominio D es periódico si existe $n \geq 1$ tal que $R^n(D) = D$. El dominio D es eventualmente periódico si existe $k \geq 1$ tal que $R^k(D)$ es periódico.*

Teorema: *(Sullivan) Todas las componentes del conjunto de Fatou son eventualmente periódicas. Además sólo existe una cantidad finita de componentes periódicas.*

2.2.3. Resultados generales sobre iteración de polinomios

En esta sección veremos algunos resultados correspondientes a la dinámica de las iteraciones de esta clase particular de funciones racionales.

Recordemos que estamos trabajando en el plano complejo extendido $\overline{\mathbb{C}} = \mathbb{C} \cup \{\infty\}$, el cual es representado geométricamente por la esfera unitaria S^2 en $\mathbb{R}^3$.

Para las iteraciones de polinomios, el punto $z = \infty$ tiene un carácter especial. Sea

$$p(z) = a_d z^d + a_{d-1} z^{d-1} + \ldots + a_1 z + a_0, \quad a_d \neq 0,$$

un polinomio de grado $d \geq 2$.

Denotemos por $C_1 = C_1(p)$ el conjunto de puntos críticos del polinomio p. Como veremos más adelante, este conjunto determina la topología de $\mathcal{J}(p)$.

Teorema: *Sea p un polinomio de grado $d \geq 2$. Entonces*

1. *$z = \infty$ es un punto atractor;*
2. *$z = \infty$ es un punto crítico de orden $d - 1$;*
3. *$\mathcal{W}^S(\infty)^* = \mathcal{W}^S(\infty)$, donde $\mathcal{W}^S(\infty)^*$ es la componente conexa de $\mathcal{W}^S(\infty)$ que contiene a ∞, es decir, su cuenca de atracción inmediata.*

El hecho que $z = \infty$ es un punto fijo atractor de un polinomio, implica el siguiente

Corolario: *$\mathcal{J}(p)$ es acotado y no igual a todo el plano complejo. En particular, $\mathcal{J}(p)$ no contiene puntos interiores.*

Teorema: *El conjunto de Julia $\mathcal{J}(p)$ es conexo si, y sólo si $\mathcal{W}^S(\infty) \cap C_1 = \emptyset$. Equivalentemente, todos los puntos críticos tienen sus órbitas acotadas.*

Corolario: *Si $C_1 \subset \mathcal{J}(p)$, entonces $\mathcal{J}(p)$ es conexo, y $\mathcal{W}^S(\infty)$ es simplemente conexo.*

Teorema: *Si un polinomio p tiene un punto fijo atractor finito $\alpha \in \mathbb{C}$, tal que $C_1 \subset \mathcal{W}^*(\alpha)$, entonces $\mathcal{J}(p)$ es una curva de Jordan (curva cerrada y simple).*

El siguiente teorema nos dice que bajo ciertas condiciones el conjunto de Julia es computacionalmente despreciable.

Teorema: *Si $p(z)$ es un polinomio tal que $C_1 \subset \mathcal{W}^S(\infty)$, entonces*

a) $\mathcal{J}(p)$ es totalmente disconexo,

b) $\mathcal{J}(p)$ tiene medida de Lebesgue nula en $\mathbb{C}$, y

c) si $\mathcal{J}(p) \subset L$, donde L es una línea recta, entonces $\mathcal{J}(p)$ tiene medida de Lebesgue nula en $\mathbb{R}$.

La condición del teorema nos dice que los iterados de los puntos críticos de p deben converger a ∞. Esto es fácil de detectar computacionalmente. Basta calcular los módulos de las sucesivas iteraciones de los puntos críticos y ver que ellos van creciendo de forma indefinida.

2.2.4. Gráficas del conjunto de Julia para polinomios

Un algoritmo de fácil implementación computacional para obtener gráficas de conjuntos de Julia de polinomios es el siguiente, conocido como *algoritmo de tiempo de escape al infinito*
Sea p un polinomio de grado $d \geq 2$. Sea $K > 0$ una constante suficientemente grande. Definimos los *conjuntos de nivel* L_k, como sigue:

$$L_0 = \{z \in \mathbb{C} : |z| > K\}.$$

para $k = 0, 1, \ldots$, sean

$$L_{k+1} = \{z \in \mathbb{C} - L_k : p(z) \in L_k\}.$$

Se tiene así que,

$$\delta L_k(\alpha_i) \to \mathcal{J}(N)$$

donde el límite es tomado respecto a la métrica de Hausdorff en $\mathcal{K}(\overline{\mathbb{C}})$.
Además, podemos pintar cada conjunto de nivel L_k con su correspondiente color.

2.2.5. Código de Matlab para la obtención de gráficas del conjunto de Julia $z^2 + c$

```
function Julia_plot(n,c,k,Xr,Yr)


% Función que dibuja el conjunto de Julia dado un parámetro c
% Julia_plot(n,c,k,Xr,Yr)
% Variables de entrada:
% n - número de puntos de la matriz donde se pinta el conjunto de
% Julia
% c - parámetro en la ecuación z = z^2 + c
% k - número de iteraciones
% Xr - rango de valores del eje x, Xr(1,1) - valor mín, Xr(1,2)
% - valor máx
% Yr - rango de valores del eje y, Yr(1,1) - valor mín, Yr(1,2)
% - valor máx
% ejemplo:
% Julia_plot(500,25i,100,[-2 2],[-2 2])


% Se crea la matriz con el tamaño definido

x = linspace(Xr(1,1),Xr(1,2),n);
y = linspace(Yr(1,1),Yr(1,2),n);
[X,Y] = meshgrid(x,y);
W = zeros(length(X),length(Y));


% Se rellena la matriz utilizando el método de Julia

for m = 1:size(X,2)
    for j = 1:size(Y,2)
        [w,iter] = Julia(X(m,j)+Y(m,j)*1i,c,k);
        W(m,j) = W(m,j) + iter;
    end
end
```

```
% Se pinta el conujnto de Julia

hold on;
colormap(jet);
pcolor(X,Y,W);
shading interp;
hold off;


% Función para calcular el conjunto de Julia

function [pri,it] = Julia(z,c,k)

R = max(abs(c),2);
i = 0;
while i < k
if abs(z) > R
        pri = 1;
        it = i;
        return;
    end

    z = z^2 + c;
    i = i + 1;
end
pri = 0;
it = i;
```

A continuación veremos diferentes ejemplos de conjuntos de Julia $z^2 + c$:

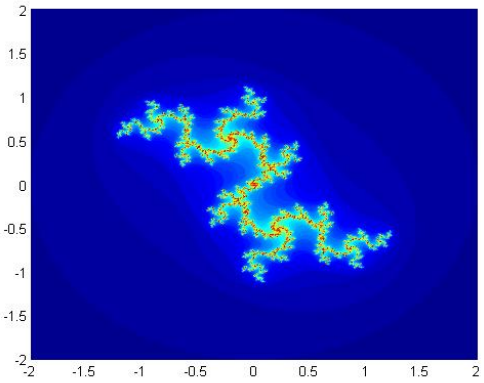


Figura 2.2: Conjunto de Julia con $c = 0{,}012 + 0{,}74i$ y $k = 50$ iteraciones.

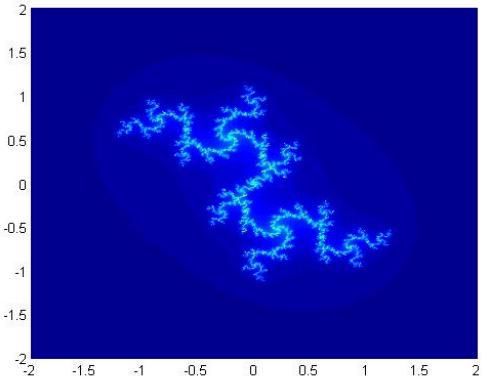


Figura 2.3: Conjunto de Julia con $c = 0{,}012 + 0{,}74i$ y $k = 5000$ iteraciones.

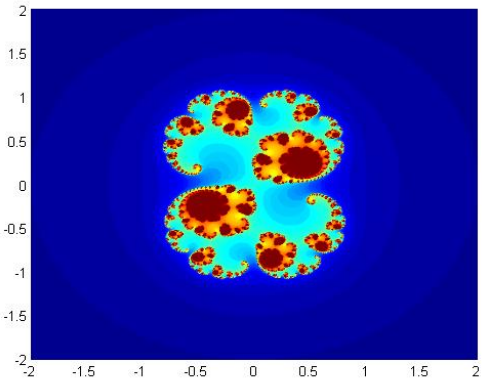


Figura 2.4: Conjunto de Julia con $c = 0{,}285 + 0{,}01i$ y $k = 50$ iteraciones.

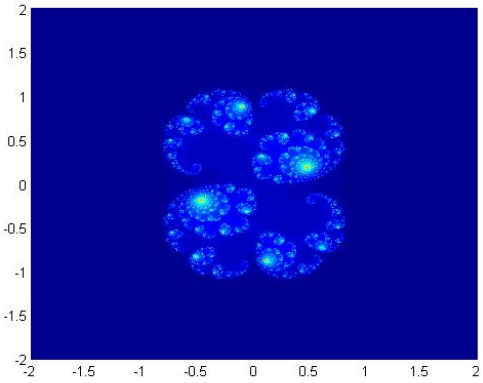


Figura 2.5: Conjunto de Julia con $c = 0{,}285 + 0{,}01i$ y $k = 5000$ iteraciones.

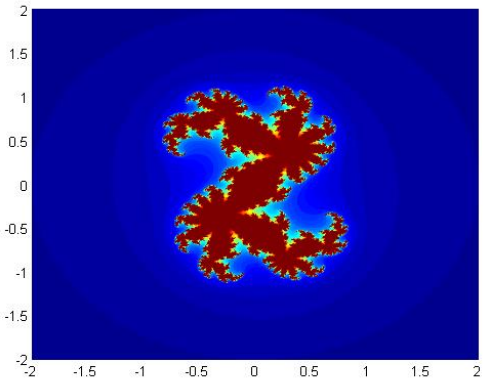


Figura 2.6: Conjunto de Julia con $c = 0{,}360 + 0{,}1003i$ y $k = 50$ iteraciones.

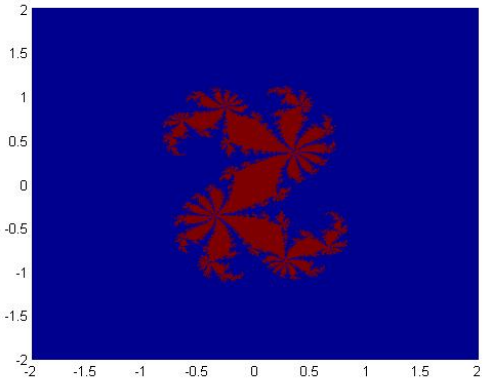


Figura 2.7: Conjunto de Julia con $c = 0{,}360 + 0{,}1003i$ y $k = 5000$ iteraciones.

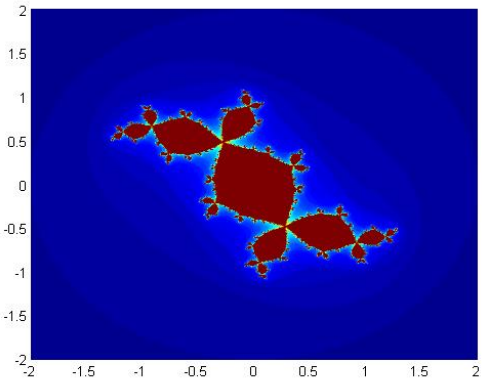


Figura 2.8: Conjunto de Julia con $c = -0{,}123 + 0{,}745i$ y $k = 50$ iteraciones.

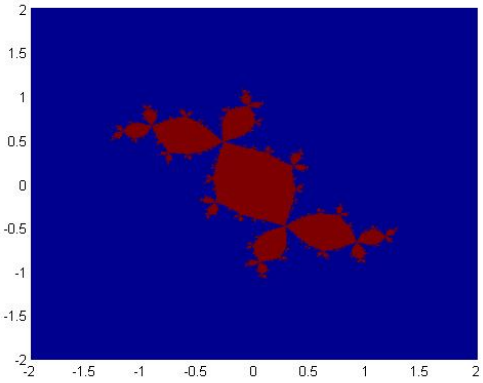


Figura 2.9: Conjunto de Julia con $c = -0{,}123+0{,}745i$ y $k = 5000$ iteraciones.

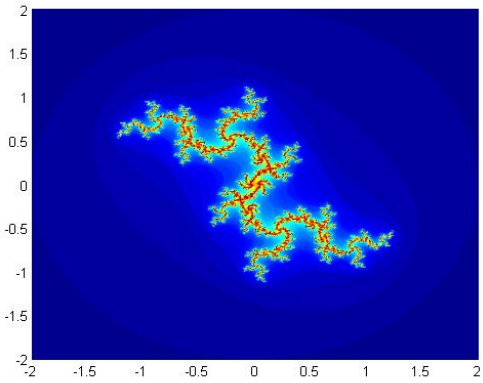


Figura 2.10: Conjunto de Julia con $c = 0{,}75i$ y $k = 50$ iteraciones.

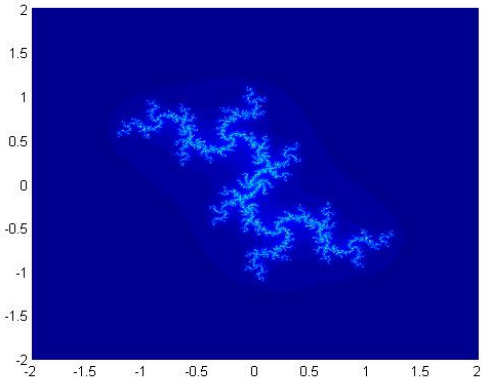


Figura 2.11: Conjunto de Julia con $c = 0{,}75i$ y $k = 5000$ iteraciones.

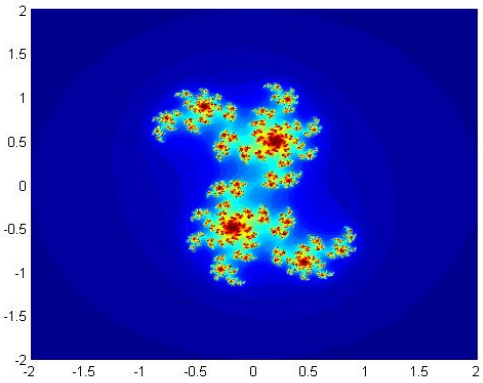


Figura 2.12: Conjunto de Julia con $c = 0{,}4 + 0{,}3i$ y $k = 50$ iteraciones.

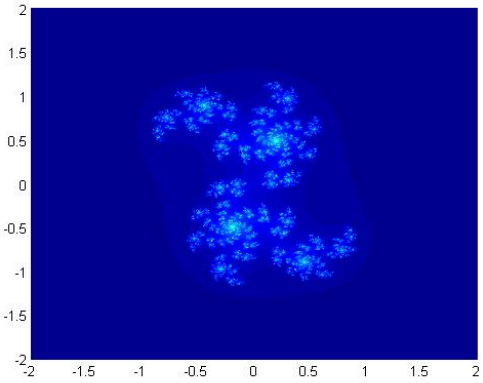


Figura 2.13: Conjunto de Julia con $c = 0{,}4 + 0{,}3i$ y $k = 5000$ iteraciones.

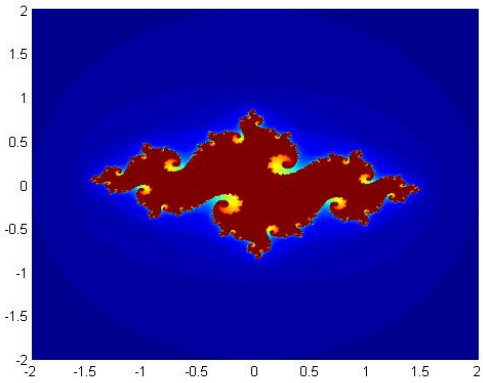


Figura 2.14: Conjunto de Julia con $c = -0{,}742 + 0{,}1i$ y $k = 50$ iteraciones.

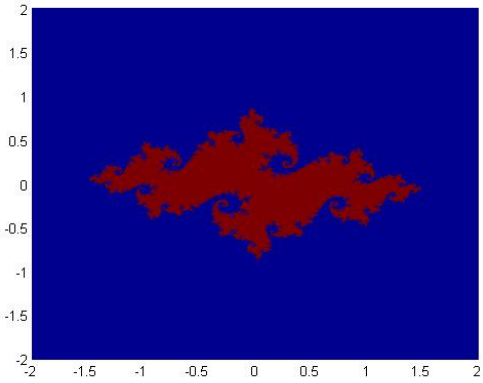


Figura 2.15: Conjunto de Julia con $c = -0{,}742 + 0{,}1i$ y $k = 5000$ iteraciones.

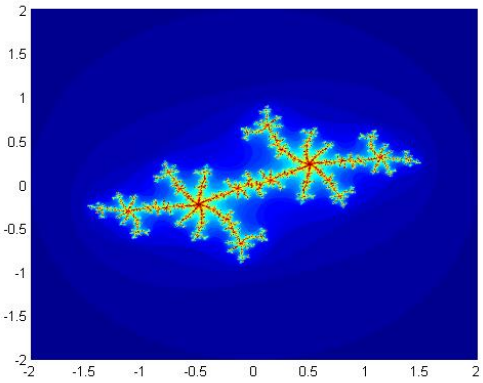


Figura 2.16: Conjunto de Julia con $c = -0{,}689 - 0{,}4626i$ y $k = 50$ iteraciones.

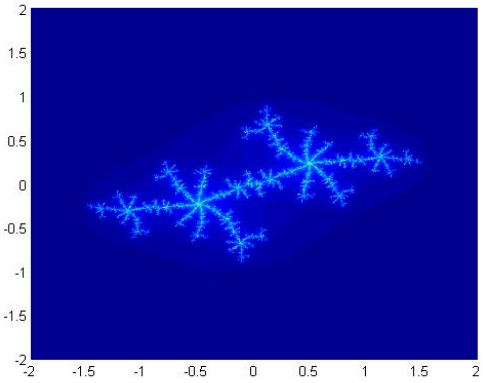


Figura 2.17: Conjunto de Julia con $c = -0{,}689 - 0{,}4626i$ y $k = 5000$ iteraciones.

2.2.6. Código de Matlab para la obtención de gráficas del conjunto de Julia generalizado

```
function Julia_general(n,f,k,R,Xr,Yr)


% Función que dibuja el conjunto de Julia
% Julia_general(n,f,k,R,Xr,Yr)
% Variables de entrada:
% n - número de puntos de la matriz donde se pinta el conjunto de
% Julia
% f - función a calcular
% k - número de iteraciones
% R - límite a partir del cual determinamos si la sucesión diverge
% Xr - rango de valores del eje x, Xr(1,1) - valor mín, Xr(1,2)
% - valor máx
% Yr - rango de valores del eje y, Yr(1,1) - valor mín, Yr(1,2)
% - valor máx
% ejemplo:
% Julia_general(500,'z^3 - i',100,100,[-2 2],[-2 2])


% Se crea la matriz con el tamaño definido

x = linspace(Xr(1,1),Xr(1,2),n);
y = linspace(Yr(1,1),Yr(1,2),n);
[X,Y] = meshgrid(x,y);
W = zeros(length(X),length(Y));


% Se rellena la matriz utilizando el método de Julia

for m = 1:size(X,2)
    for j = 1:size(Y,2)
        [w,iter] = Julia_gen(X(m,j)+Y(m,j)*1i,f,R,k);
        W(m,j) = W(m,j) + iter;
    end
end
```

```
% Se pinta el conujnto de Julia

hold on;
colormap(jet);
pcolor(X,Y,W);
shading interp;
hold off;


% Función para calcular el conjunto de Julia

function [pri,it] = Julia_gen(zn,f,R,k)

syms z;
i = 0;
while i < k
if abs(zn) > R
        pri = 1;
        it = i;
        return;
    end

    zn = subs(f,zn);
    i = i + 1;
end
pri = 0;
it = i;
```

A continuación veremos diferentes ejemplos de conjuntos de Julia generalizados:

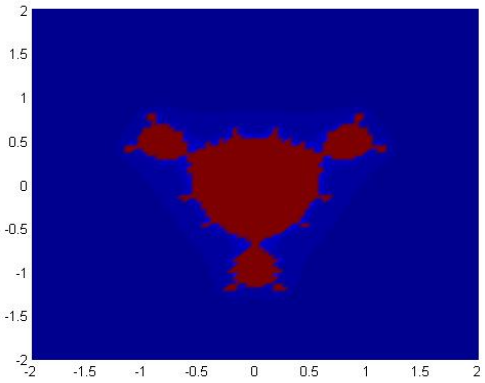


Figura 2.18: Conjunto de Julia $f(z) = z^3 - i$ con 100 puntos en la matriz, $k = 100$ iteraciones, $R = 25$.

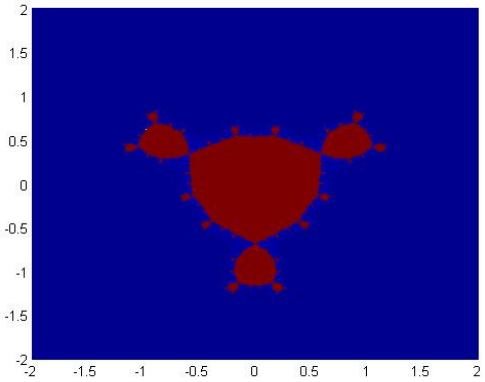


Figura 2.19: Conjunto de Julia $f(z) = z^3 - i$ con 500 puntos en la matriz, $k = 200$ iteraciones, $R = 25$.

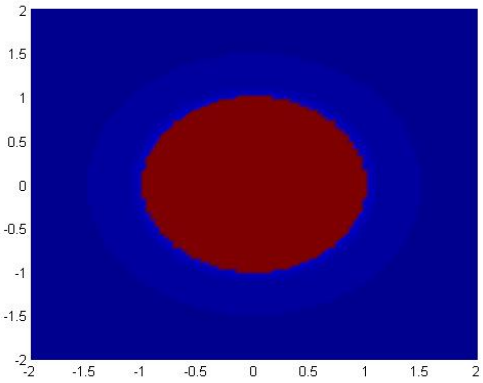


Figura 2.20: Conjunto de Julia $f(z) = z^2$ con 100 puntos en la matriz, $k = 100$ iteraciones, $R = 25$.

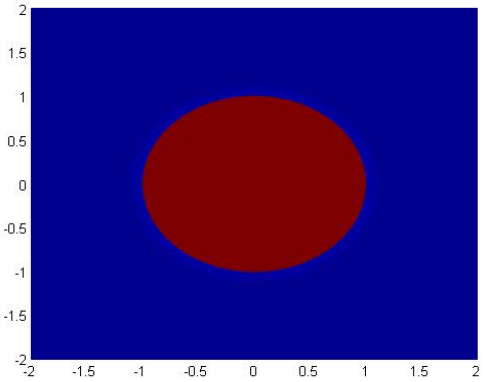


Figura 2.21: Conjunto de Julia $f(z) = z^2$ con 500 puntos en la matriz, $k = 200$ iteraciones, $R = 25$.

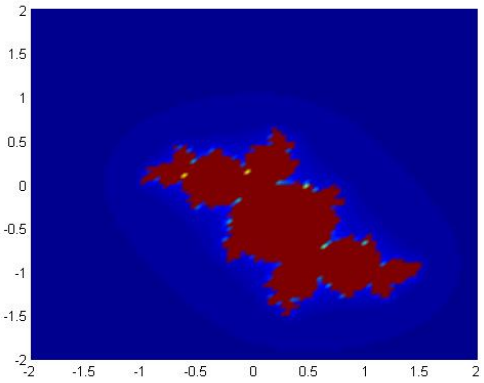


Figura 2.22: Conjunto de Julia $f(z) = e^{\frac{2\pi i}{3}} z + z^2$ con 100 puntos en la matriz, $k = 100$ iteraciones, $R = 25$.

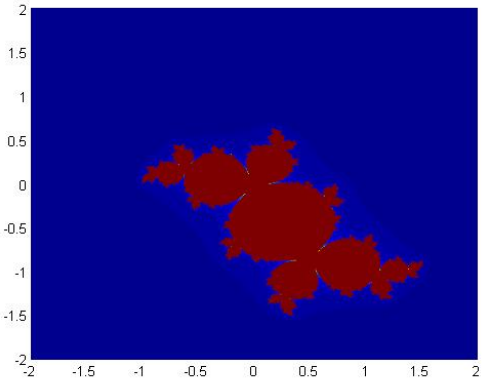


Figura 2.23: Conjunto de Julia $f(z) = e^{\frac{2\pi i}{3}} z + z^2$ con 500 puntos en la matriz, $k = 200$ iteraciones, $R = 25$.

2.3. Conjuntos de Mandelbrot

2.3.1. El Conjunto de Mandelbrot

El caso de polinomios cuadráticos es particularmente interesante por la propiedad de dicotomía que posee el conjunto de Julia.
Sea p un polinomio de grado 2, entonces p sólo tiene un punto crítico, c_p. Por lo tanto, $\mathcal{J}(p)$ es totalmente disconexo o es conexo si y sólo si $\lim_{n\to\infty} p^n(c_p)$ pertenece o no a $\mathcal{W}^S(\infty)$, respectivamente.
Consideremos ahora la familia a 1-parámetro $p_c(z) = z^2 - c$, $c \in \mathbb{C}$.

Proposición: *Cada polinomio de grado 2 es topológicamente conjugado a p_c para algún c.*

Demostración: *Sea $p(z) = az^2 + bz + c$ un polinomio cuadrático. Sea $\tau(z) = \alpha z + \beta, \alpha \neq 0$. Entonces,*

$$\tau \circ p \circ \tau^{-1}(z) = \tau\left(p\left(\frac{z}{\alpha} - \frac{\beta}{\alpha}\right)\right)$$

$$= \alpha\left(a\left(\frac{z}{\alpha} - \frac{\beta}{\alpha}\right)^2 + b\left(\frac{z}{\alpha} - \frac{\beta}{\alpha}\right) + c\right) + \beta$$

$$= \frac{a}{\alpha}(z^2 - 2\beta z + \beta^2) + bz - b\beta + \alpha c + \beta$$

$$= \frac{a}{\alpha}z^2 + \left(b - \frac{2a\beta}{\alpha}\right)z + \frac{a\beta^2}{\alpha} - b\beta + \alpha c + \beta.$$

Ahora hacemos $\left(\frac{a}{\alpha}\right) = 1$ y $2\beta = b$, y tenemos $\tau \circ p \circ \tau^{-1}(z) = z^2 - \frac{b^2-2b-4ac}{4}$. Usando la dicotomía anterior, y el hecho que cada polinomio cuadrático tiene su dinámica representada por algún elemento de la familia cuadrática a 1-parámetro $p_c(z) = z^2 - c$, se define el conjunto de Mandelbrot de p_c como

$$\mathcal{M} = \{c \in \mathbb{C} : J(p_c) \textit{es conexo}\}.$$

Como los puntos críticos de p_c son dados por la ecuación $p_c'(z) = 0$, ellos son $z = 0$ y $z = \infty$. Tenemos $c_0 = p_c(0) = -c, c_1 = p_c^2(0) = p_c(p_c(0)) = c^2 - c, c_3 = p_c^3(0) = p_c(p_c^2(0)) = (c^2 - c)^2 - c, \ldots$. Luego,

$$\mathcal{M} = \{c \in \mathbb{C} : \quad \text{la sucesión} \{c_n\}_{n\in\mathbb{N}} = \{p_c^n(0)\}_{n\in\mathbb{N}} \quad \text{permanece acotada}\}$$

$$= \{c \in \mathbb{C} : c_n = p_c^n(0) \nrightarrow \infty\}$$

La siguiente figura muestra al conjunto de Mandelbrot y los gráficos de algunos conjuntos de Julia correspondientes a los valores de $c \in \mathcal{M}$

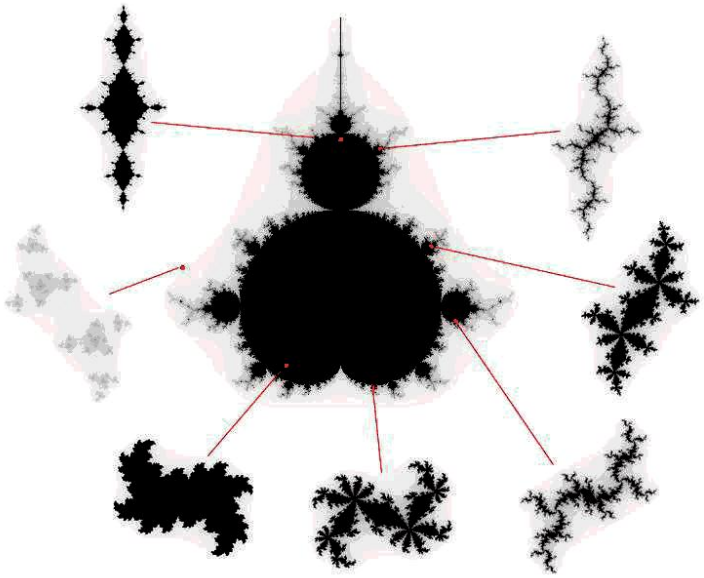


Figura 2.24: Componente principal del Conjunto de Mandelbrot.

2.3.2. Código de Matlab para la obtención del conjunto de Mandelbrot

```
function Mandelbrot_plot(n,k,Xr,Yr)


% Función que dibuja el conjunto de Mandelbrot
% Mandelbrot_plot(n,k,Xr,Yr)
% Variables de entrada:
% n - número de puntos de la matriz
% k - número de iteraciones
% Xr - rango de valores del eje x, Xr(1,1) - valor mín, Xr(1,2)
% - valor máx
% Yr - rango de valores del eje y, Yr(1,1) - valor mín, Yr(1,2)
% - valor máx
% ejemplo:
% Mandelbrot_plot(500,100,[0 2],[0 2])


% Se crea la matriz con el tamaño definido

x = linspace(Xr(1,1),Xr(1,2),n);
y = linspace(Yr(1,1),Yr(1,2),n);
[X,Y] = meshgrid(x,y);
W = zeros(length(X),length(Y));


% Se rellena la matriz utilizando el método de Mandelbrot

for m = 1:size(X,2)
    for j = 1:size(Y,2)
        [w,iter] = Mandelbrot(X(m,j)+Y(m,j)*1i,k);
        W(m,j) = W(m,j) + iter;
    end
end


% Se pinta el conujnto de Mandelbrot

hold on;
colormap(jet);
pcolor(X,Y,W);
shading interp;
hold off;
```

```
% Función para calcular el conjunto de Mandelbrot

function [pri,it] = Mandelbrot(c,m)
k = 0;
z = c;
while k < m
    if abs(z) > 2
        pri = 1;
        it = k;
        return;
    end

    z = z^2 + c;
    k = k + 1;
end
pri = 0;
it = k;
```

A continuación se observan diferentes conjuntos de Mandelbrot generados con el anterior programa cambiando la variable k correspondiente al número de iteraciones que se realizan en el cálculo:

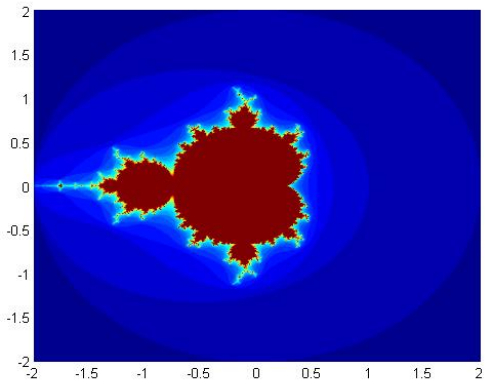


Figura 2.25: Conjunto de Mandelbrot con $k = 30$ iteraciones.

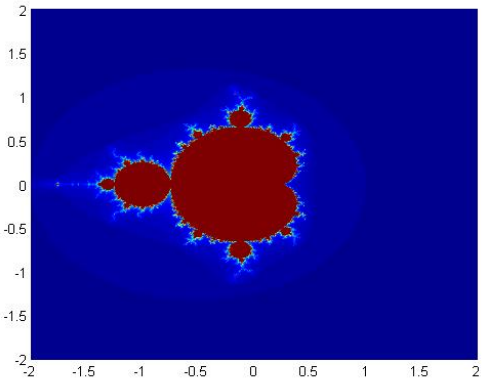


Figura 2.26: Conjunto de Mandelbrot con $k = 100$ iteraciones.

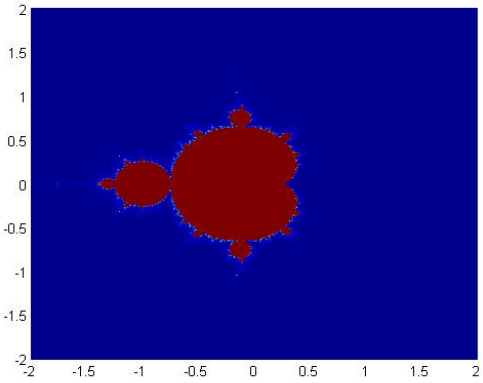


Figura 2.27: Conjunto de Mandelbrot con $k = 500$ iteraciones.

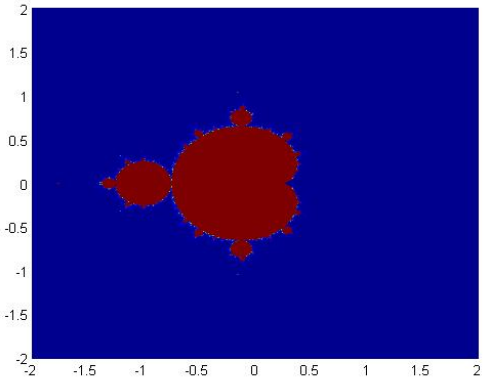


Figura 2.28: Conjunto de Mandelbrot con $k = 1000$ iteraciones.

3 Métodos numéricos en el plano complejo

Gran parte de la teoría en la que se basa este capítulo ha sido extraída de [5].

3.1. Nota histórica

Sea $f : \mathbb{C} \to \mathbb{C}$ una función analítica. Se define la transformada de Newton $N(z) := N_f(z)$ de f como

$$N(z) = z - \frac{f(z)}{f'(z)} = \frac{zf'(z) - f(z)}{f'(z)}$$

en todos los puntos $z \in \mathbb{C}$ donde $f'(z) \neq 0$.

En 1870/71 E. Schröder y en 1879 A.Cayley proponen extender el método de Newton conocido en el caso de funciones de variable real a valores reales, para polinomios en el plano complejo.

Sea $p(z) = a_0 + a_1 z + \ldots + a_k z^k$ un polinomio en el plano complejo. Ambos autores estudian la cuenca de atracción de una raíz α, es decir, el conjunto $\mathcal{W}^S(\alpha) = \{z \in \mathbb{C} : N^n(z) \to \alpha, n \to \infty\}$. Desde el punto de vista local la situación es simple, pues α es una raíz simple de p si y sólo si es un punto fijo superatractor de N, esto es $N(\alpha) = \alpha$ y $N'(\alpha) = 0$. Por lo tanto tomando una condición inicial z_0 próxima a α, las sucesivas iteraciones de N convergen rápidamente a α, de hecho la convergencia es cuadrática.

Por otra parte, si α es una raíz múltiple, digamos de multiplicidad k de p entonces $N'(\alpha) = (k-1)/k$, y las sucesivas iteraciones de N convergen a la raíz α de p , sólo que esta vez lo hacen en forma más lenta (linealmente).

Cayley estudia el problema anterior para el polinomio cuadrático $p(z) = z^2 - 1$ y obtiene una caracterización completa de las cuencas de atracción de las raíces 1 y -1 de $p(z)$. Ellas son dadas por $\mathcal{W}^S(1) = \{z \in \mathbb{C} : \Re e(z) > 0\}$ y $\mathcal{W}^S(-1) = \{z \in \mathbb{C} : \Re e(z) < 0\}$, donde $\Re e(z)$ es la parte real del número complejo z.

3.2. Problema de Cayley

El problema de Cayley de estudiar las cuencas de atracción de las raíces del polinomio cúbico $p(z) = z^3 - 1$, está relacionado con el siguiente problema:
«*Usando tres colores pintar un cuadrado, de modo que si en un punto dos colores se encuentran, entonces los tres colores se encuentran en ese punto*»

La teoría de Julia-Fatou está relacionada con el problema de Cayley de describir la dinámica de las iteraciones de una función racional, pues si $p(z)$ es un polinomio complejo, entonces

$$N_p(z) = z - \frac{p(z)}{p'(z)} = \frac{zp'(z) - p(z)}{p(z)}$$

es una función racional. El conjunto de Julia de N_p está contenido en el conjunto de puntos donde el método de Newton no converge a una raíz de la ecuación $p(z) = 0$, pero este último conjunto puede ser bastante más grande. Por ejemplo, cuando N_p posee órbitas periódicas atractoras de período mayor o igual que dos.

Consideremos la ecuación polinomial $p(z) = z^3 - 1 = 0$. Sus raíces en el plano complejo $\mathbb{C}$, son $\alpha_1 = 1, \alpha_2 = e^{\frac{2\pi i}{3}} = \cos(\frac{2\pi}{3}) + i\sin(\frac{2\pi}{3}) = -\frac{1}{2} + i\frac{\sqrt{3}}{2}$ y $\alpha_3 = e^{\frac{4\pi i}{3}} = \cos(\frac{4\pi}{3}) + i\sin(\frac{4\pi}{3}) = -\frac{1}{2} - i\frac{\sqrt{3}}{2}$.

El problema ahora es describir las cuencas de atracción $\mathcal{W}^S(\alpha_i), i = 1, 2, 3$. La transformada de Newton de $p(z) = z^3 - 1$ es

$$N(z) = z - \frac{p(z)}{p'(z)} = \frac{2z^3 + 1}{3z^2},$$

y por lo visto anteriormente, se tiene

$$\mathcal{J}(N) = \delta\mathcal{W}^S(\alpha_1) = \delta\mathcal{W}^S(\alpha_2) = \delta\mathcal{W}^S(\alpha_3).$$

Las gráficas de los conjuntos de Julia de $p(z) = z^3 - 1$, muestran la solución del problema de los tres colores de Cayley.

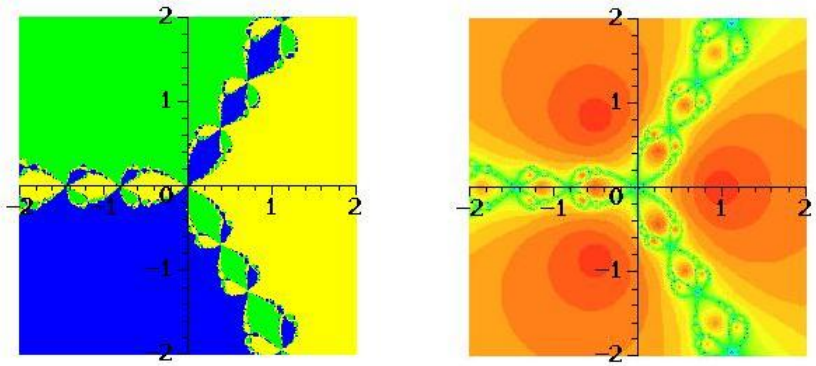


Figura 3.1: Solución del problema de Cayley.

3.3. Método de Newton en el plano complejo

3.3.1. Propiedades

Sea $p(z) = a_d z^d + \ldots + a_1 z + a_0 \neq 0$, un polinomio de grado d en $\mathbb{C}$, y sea

$$N_p(z) = z - \frac{p(z)}{p'(z)},$$

su transformada de Newton.
Veamos algunas propiedades elementales de la dinámica de N_p:

1. *$N_p(z_0) = z_0$ si y sólo si $p(z_0) = 0$, es decir, los puntos fijos de N_p son las raíces de p.*

2. *$z = \infty$ es siempre un punto fijo de N_p, y como $N_p'(\infty) = \frac{d-1}{d}$ este punto fijo es atractor.*

3. *Puesto que $N_p'(z) = \frac{p(z)p''(z)}{(p'(z))^2}$, si z_0 es una raíz simple de p, se tiene que $N_p'(z) = 0$, esto es, z_0 es un punto superatractor de N_p, lo cual implica que N_p es conjugada a la aplicación $z \to z^k$, para algún $k > 1$ en una vecindad de z_0.*

4. *Las raíces múltiples de p son puntos fijos atractores, pero no superatractores, de N_p, pues si z_0 tiene multiplicidad $m > 1$, entonces $N_p'(z_0) = \frac{m-1}{m} < 1$.*

5. *Para polinomios genéricos de grado d, la transformada de Newton es una función racional de grado d. Cuando el polinomio tiene raíces múltiples, N_p tiene grado menor que d.*

6. *Los puntos críticos de N_p son las raíces simples y los puntos de inflexión de p. Los puntos críticos de N_p que no son raíces de p los llamaremos puntos críticos libres. Las propiedades del conjunto de Julia de una función analítica en $\mathbb{C}$ son frecuentemente determinadas por las órbitas de sus puntos críticos.*

7. *Los puntos críticos de p, es decir, las raíces de $p'(z) = 0$, son los polos de N_p. Por lo tanto órbitas que evitan los puntos críticos de p tienen posibilidad de converger.*

Teorema: *(Lucas, 1874) Los puntos críticos de p están contenidos en la envoltura convexa de sus raíces.*

Teorema: *El método de Newton es, en general, no convergente.*

Demostración: *Sea $p(z) = \sum_{j=0}^{d} a_j z^j$ un polinomio de grado ≥ 3. Fijemos $a_0 = 1, a_1 = -1$ y $a_2 = 0$. Tenemos entonces que $p(z) = 1 - z + a_3 z^3 + \ldots + a_d z^d$.*
Ahora, $p'(z) = -1 + 3a_3 z^2 + 4a_4 z^3 + \ldots + d a_d z^{d-1}$ y $p''(z) = 6a3z + 12a_4 z^2 + \ldots + d(d-1)a_d z^{d-2}$. Así

$$N_p(z) = z - \frac{1 - z + a_3 z^3 + \ldots + a_d z^d}{-1 + 3a_3 z^2 + 4a_4 z^3 + \ldots + d a_d z^{d-1}}$$

Como $N_p'(z) = \frac{p(z)p''(z)}{(p'(z))^2}$, tenemos $N_p(0) = 1$ y $N_p'(0) = 0$, y si $N_p'(1) \neq \infty$ entonces $(N_p^2)'(0) = N_p'(N_p(0))N_p'(0) = N_p'(1)N_p'(0) = 0$.

Luego, si $N_p(1) = 0$ y $N_p'(1) \neq \infty$, se tiene que 0 es un punto periódico, de período 2, superatractor para N_p.
Ahora bien,

$$N_p(1) = 0 \quad \textit{si, y sólo si} \quad -1 + 2a_3 + 4a_4 + \ldots + (d-1)a_d = 0 \quad \textit{(I)}$$

y la condición $N_p'(1) \neq \infty$ se satisface si $p'(1) \neq 0$, es decir, si

$$-1 + 3a_3 + 4a_4 + \ldots + da_d \neq 0 \quad \textit{(II)}.$$

Estas condiciones son satisfechas en un conjunto abierto y denso de un hiperplano de $\{(a_3, a_4, \ldots, a_d) : a_i \in \mathbb{C}\} = \mathbb{C}^{d-2}$. Lo que termina la prueba.

Teorema: *Sea R una función racional de grado $d \geq 2$. Entonces las siguientes afirmaciones son equivalentes:*

1. *R tiene exactamente d puntos fijos finitos $\xi_1, \xi_2, \ldots, \xi_d$ con $R'(\xi_j) = 1 - h$ para $j = 1, 2, \ldots, d$ y $R(\infty) = \infty$.*
2. *R es el método de Newton relajado, N_h, para $p(z) = (z-\xi_1)\cdots(z-\xi_d)$.*

Demostración: *La parte, (2) $\Longrightarrow$ (1) es inmediata, es decir, un simple cálculo.*
(1) $\Longrightarrow$ (2). Sea R una función racional de grado $d \geq 2$ que satisface (1). Entonces

$$R(z) = z - \frac{P(z)}{Q(z)},$$

donde P y Q son polinomios sin factores comunes. Las condiciones sobre R implican las siguientes desigualdades para los grados de $P(z), Q(z)$ y $zQ(z) - P(z)$, respectivamente,

$$grado(P) \geq d \geq grado(Q), \quad d \geq grado(zQ - P) \geq grado(Q).$$

De estas desigualdades obtenemos que $grado(P) \in \{d, d+1\}$. Si $grado(P) = d + 1$, entonces la segunda desigualdad implica que $grado(Q) = d$ lo cual nos da $grado(zQ - P) = grado(Q)$ lo cual es una contradicción puesto que $R(\infty) = \infty$. Por lo tanto $grado(P) = d$.
La derivada de R es

$$R'(z) = \frac{(Q(z))^2 - P'(z)Q(z) + Q'(z)P(z)}{(Q(z))^2}.$$

Como $R'(\xi_v) = 1 - h$ *y* $Q(\xi_v) \neq 0$ *se sigue que* P *es un factor del polinomio* $(Q(z))^2 - P'(z)Q(z) + Q'(z)P(z) - (1-h)(Q(z))^2 = Q(z)(hQ(z) - P'(z)) + Q'(z)P(z)$*, por lo tanto,* P *es un factor de* $hQ(z) - P'(z)$*. Ahora* $grado(hQ - P') \leq d - 1$*, por lo tanto* $hQ(z) = P'(z)$*.*

3.3.2. Código Matlab : Método de Newton

```
function [x,i,y]=Mnewton(f,x0,n,tol)


% Método de Newton para la resolución de una ecuación no lineal
% N(x) = x - (f(x)/df(x))
% [x,i,y]=Mnewton(f,x0,n,tol)
% Variables de entrada:
% f - función que determina la ecuación a resolver f(x)=0
% x0 - aproximación inicial a un cero de f
% n - número máximo de iteraciones
% tol - tolerancia, la usamos para hacer un criterio de parada,
% en caso que f/df sea más pequeño que tol en valor absoluto, el
% método parará
% Variables de salida:
% x - aproximación a la raíz
% i - número de iteraciones realizadas
% y - valor de la función evaluada en la aproximación x, es decir
% f(x)
% ejemplo:
% [x,i,y]=Mnewton('x^3 - 3*x^2',5,500,10E-8)


% Calculamos la derivada de manera simbólica

df=diff(f);


% Inicialización de la aproximación

x=x0;


for i=1:n
```

```
% Calculamos el cociente entre la función y su derivada

    co=subs(f/df,x);
    x=x-co;

% Criterio de parada

    if abs(co)<tol

% Dos iterados sucesivos están suficientemente cerca

        break;
    end
end

% Evaluación de la función en la aproximación

y=subs(f,x);
```

3.4. Método de Halley

Sea $f : \mathbb{C} \to \mathbb{C}$ una función analítica. La función de iteración de Halley es definida como

$$H_f(z) = z - \frac{2f(z)f'(z)}{2(f'(z))^2 - f(z)f''(z)}.$$

Con el propósito que esta función de iteración resulte ser una función racional de $\overline{\mathbb{C}}$ en si misma tomamos $f(z)$ polinomial.
Para obtener esta fórmula, una manera es aplicar el método de Newton a la función

$$g(z) = \frac{f(z)}{\sqrt{f'(z)}},$$

o bien, considerar la serie de Taylor de $f(z)$ para z_n próximo a una raíz de $f(z) = 0$, es decir,

$$w = f(z_n) + f'(z_n)(z - z_n) + \frac{f''(z_n)}{2}(z - z_n)^2.$$

En consecuencia para obtener z_{n+1} suponemos que este es una raíz de la ecuación anterior, luego evaluando obtenemos,

$$0 = f(z_n) + (z_{n+1} - z_n)\left(f'(z_n) + \frac{f''(z_n)}{2}(z_{n+1} - z_n)\right)$$

y de aquí se sigue que

$$z_{n+1} = z_n - \frac{f(z_n)}{f'(z_n) - \frac{f''(z_n)}{2}(z_{n+1} - z_n)}.$$

Ahora aproximamos la diferencia $z_{n+1} - z_n$ por el método de Newton, es decir, por $-\frac{f(z_n)}{f'(z_n)}$, y obtenemos

$$z_{n+1} = z_n - \frac{2f(z_n)f'(z_n)}{2(f'(z_n))^2 - f(z_n)f''(z_n)},$$

es decir, la función de iteración de Halley.

Proposición:

(a) Si α es un cero simple de f, entonces $H_f(\alpha) = \alpha$, es decir, es un punto fijo de H_f. Además, $H'_f(\alpha) = H''_f(\alpha) = 0$ y $H'''_f(\alpha) \neq 0$, por lo tanto H_f es una función de iteración de orden tres.

(b) $H'''_f(\alpha) = -\left(\frac{f'''(\alpha)}{f'(\alpha)} - \frac{3}{2}\left(\frac{f''(\alpha)}{f'(\alpha)}\right)^2\right)$

3.4.1. Código Matlab : Método de Halley

```
function [x,i,y]=Mhalley(f,x0,n,tol)


% Método de Halley para la resolución de una ecuación no lineal
% H(x) = x - ((2f(x)df(x))/(2(df(x)^2 - f(x)df2(x))
% [x,i,y]=Mhalley(f,x0,n,tol)
% Variables de entrada:
% f - función que determina la ecuación a resolver f(x)=0
% x0 - aproximación inicial a un cero de f
% n - número máximo de iteraciones
```

```
% tol - tolerancia, la usamos para hacer un criterio de parada,
% en caso que el cociente entre el numerador y el denominador sea
% más pequeño que tol en valor absoluto, el método parará
% Variables de salida:
% x - aproximación a la raíz
% i - número de iteraciones realizadas
% y - valor de la función evaluada en la aproximación x, es decir
% f(x)
% ejemplo:
% [x,i,y]=Mhalley('x^3-3*x^2',5,500,10E-8)


% Calculamos la derivada de manera simbólica

df=diff(f);


% Calculamos la derivada segunda de manera simbólica

df2=diff(f,2);


% Inicialización de la aproximación

x=x0;


for i=1:n


% Calculamos el numerador de la ecuación

    num=2*(f*df);


% Calculamos el denominador de la ecuación

    den=2*(df^2)-(f*df2);



% Calculamos el cociente entre el numerador y denominador
```

```
    co=subs(num/den,x);
    x=x-co;


% Criterio de parada

    if abs(co)<tol


% Dos iterados sucesivos están suficientemente cerca

        break;
    end
end


% Evaluación de la función en la aproximación

y=subs(f,x);
```

3.5. Método de Schröder

Sea $F : \mathbb{C} \to \mathbb{C}$ una función analítica y $p \in \mathbb{C}$ un punto fijo atractor de F. Dado $z_0 \in \mathbb{C}$ definimos la sucesión de iterados $(z_n)_{n\in\mathbb{N}}$, $z_{n+1} = F(z_n)$, $n = 0, 1, \ldots$. Supongamos que $z_n \to p$ cuando $n \to \infty$, y sea $e_n = z_n - p$ el error asociado al iterado n-ésimo.
Entonces

$$e_{n+1} = z_n - p = F(e_n + p) - F(p) = \frac{1}{m!}F^{(m)}(p)e_n^m + O(e_n^{m+1}),$$

donde m es el menor entero tal que $F^{(m)}(p) \neq 0$. Decimos entonces que F es una función de iteración de orden m. Por ejemplo, en general, la función de iteración de Newton es de orden 2.

Ahora, si $|h|$ es pequeño, entonces

$$F(z + h) = F(z) + \sum_{n=1}^{\infty} b_n(z)h^n = F(z) + B(z),$$

donde $b_n(z) = F^{(n)}(z)/n!$, $n = 1, 2, \ldots$. $B(z)$ puede ser considerada como una serie de potencias formal en h, cuyos coeficientes dependen de z. Si $b_1(z) = F'(z) \neq 0$, entonces $B(z)$ puede ser invertida, y

$$B^{-1}(z) = \sum_{n=1}^{\infty} c_n(z) h^n,$$

donde $c_n(z) = \frac{1}{n} res(B^{-n}(z))$, $n = 1, 2 \ldots$ (fórmula de Lagrange-Bürmann).

Como los coeficientes de B^{-n} son expresiones racionales en las funciones $b_1(z), b_2(z), \ldots$ y en cada una de ellas el denominador es una potencia de $b_1(z) = F'(z) \neq 0$, lo mismo vale para las funciones $c_n(z)$. Estas son funciones analíticas en todas partes donde $F'(z) \neq 0$.

Ahora sea $f : D \subset \mathbb{C} \to \mathbb{C}$ una función analítica. En general, para aplicar la teoría de Julia-Fatou, suponemos que f es polinomial. Para cada $m = 2, 3, \ldots$, definimos

$$S_m(z) = z + \sum_{k=1}^{m-1} c_k(z)(-f(z))^k$$

(funciones de iteración de Schröder).

Observación: *S_m es el truncamiento de la serie infinita en f cuyos tres primeros términos son*

$$S(z) = z - \frac{1}{f'(z)} f(z) - \frac{f''(z)}{2(f'(z))^3}(f(z))^2 - \frac{\frac{1}{2}(f''(z))^2 - \frac{1}{6}f'(z)f'''(z)}{(f'(z))^5}(f(z))^3 - \cdots$$

Es claro que S_2 coincide con la función de iteración de Newton, y

$$S_3(z) = z - \frac{f(z)}{f'(z)} - \frac{f''(z)(f(z))^2}{2(f'(z))^3}$$

$$S_4(z) = z - \frac{f(z)}{f'(z)} - \frac{f''(z)(f(z))^2}{2(f'(z))^3} - \frac{(\frac{1}{2}(f''(z))^2 - \frac{1}{6}f'(z)f'''(z))(f(z))^3}{(f'(z))^5}$$

Tenemos el siguiente

Teorema: *Sea f una función analítica en una región $D \subset \mathbb{C}$ y $f'(z) \neq 0$ para todo $z \in D$. Entonces, $S_2(z), S_3(z), \ldots$ son analíticas en D. Además, para cada $w \in D$ tal que $f(w) = 0$ se tiene*

$$S_m(w) = w, y \quad S'_m(w) = S''_m(w) = \cdots = S_m^{m-1}(w) = 0,$$

esto es, cada cero de f es un punto fijo superatractor de orden, al menos m, para las funciones S_m.

Ahora, si $m > 2$, de la ecuación de puntos fijos, $S_m(z) = z$, tenemos

(i) $f(z) = 0$, ó

(ii) $T_m(z) = \sum_{n=0}^{m-2} c_{n+1}(z)(-f(z))^n = 0$.

Por lo tanto, pueden aparecer puntos fijos de S_m, que no son raíces de $f(z) = 0$. Estos son las soluciones de la ecuación

$$T_m(z) = \sum_{n=0}^{m-2} c_{n+1}(-f(z))^n = 0$$

que no son soluciones de $f(z) = 0$. A estos nuevos puntos fijos los llamaremos puntos fijos extraños de S_m, los cuales aún cuando sean repulsores o indiferentes alteran las cuencas de atracción de las raíces. Cuando los puntos fijos extraños son atractores los llamaremos atractores extraños de S_m.

Los atractores extraños de S_m pueden atrapar una sucesión de iterados, dando resultados erróneos para una raíz z_0 de f.

3.5.1. Código Matlab : Método de Schröder con orden 3

```
function [x,i,y]=Mschroder3(f,x0,n,tol)


% Método de Schröder, con m = 3, para la resolución de una ecuación
% no lineal
% Hs3(x) = x - (f(x)/df(x)) - (df2(x)(f(x)^2)/2(df(x))^3)
% [x,i,y]=Mschroder3(f,x0,n,tol)
% Variables de entrada:
% f - función que determina la ecuación a resolver f(x)=0
% x0 - aproximación inicial a un cero de f
% n - número máximo de iteraciones
% tol - tolerancia, la usamos para hacer un criterio de parada,
% en caso que la resta de los tres términos sea más pequeña que
```

```
tol
% en valor absoluto, el método parará
% Variables de salida:
% x - aproximación a la raíz
% i - número de iteraciones realizadas
% y - valor de la función evaluada en la aproximación x, es decir
% f(x)
% ejemplo:
% [x,i,y]=MMschroder3('x^3-3*x^2',5,500,10E-8)


% Primer término de la ecuación (Método de Newton)
% Calculamos la derivada de manera simbólica

df=diff(f);


% Segundo término de la ecuación
% Calculamos la derivada segunda de manera simbólica

df2=diff(f,2);


% Inicialización de la aproximación

x=x0;


for i=1:n


% Calculamos el numerador de la primera ecuación

    num1=f;


% Calculamos el denominador de la primera ecuación

    den1=df;


% Calculamos el cociente del primer término

    co1=subs(num1/den1,x);
```

```
% Calculamos el numerador de la segunda ecuación

    num2=(df2*f)^2;

% Calculamos el denominador de la segunda ecuación

    den2=(2*df)^3;

% Calculamos el cociente del segundo término

    co2=subs(num2/den2,x);

% Calculamos el cociente del método

    co=co1-co2;
    x=x-co;

% Criterio de parada

    if abs(co)<tol


% Dos iterados sucesivos están suficientemente cerca

        break;
    end
end

% Evaluación de la función en la aproximación

y=subs(f,x);
```

3.5.2. Código Matlab : Método de Schröder con orden 4

```
function [x,i,y]=Mschroder4(f,x0,n,tol)


% Método de Schröder, con m = 4, para la resolución de una ecuación
% no lineal
% Hs4(x) = x − (f(x)/df(x)) − (df2(x)(f(x)^2)/2(df(x))^3)
% −((1/2(df2^2) − 1/6df(x)df3(x))(f(x))^3)/(df(x)^5)
% [x,i,y]=Mschroder4(f,x0,n,tol)
% Variables de entrada:
% f - función que determina la ecuación a resolver f(x)=0
% x0 - aproximación inicial a un cero de f
% n - número máximo de iteraciones
% tol - tolerancia, la usamos para hacer un criterio de parada,
% en caso que el resultado sea más pequeño que tol en valor absoluto,
% el método parará
% Variables de salida:
% x - aproximación a la raíz
% i - número de iteraciones realizadas
% y - valor de la función evaluada en la aproximación x, es decir
% f(x)
% ejemplo:
% [x,i,y]=MMschroder4('x^3 − 3 ∗ x^2',5,500,10E-8)


% Primer término de la ecuación (Método de Newton)
% Calculamos la derivada de manera simbólica

df=diff(f);


% Segundo término de la ecuación
% Calculamos la derivada segunda de manera simbólica

df2=diff(f,2);


% Tercer término de la ecuación
% Calculamos la derivada tercera de manera simbólica

df3=diff(f,3);
```

```
% Inicialización de la aproximación
x=x0;

for i=1:n

% Calculamos el numerador de la primera ecuación
    num1=f;

% Calculamos el denominador de la primera ecuación
    den1=df;

% Calculamos el cociente del primer término
    co1=subs(num1/den1,x);

% Calculamos el numerador de la segunda ecuación
    num2=(df2*f)^2;

% Calculamos el denominador de la segunda ecuación
    den2=(2*df)^3;

% Calculamos el cociente del segundo término
    co2=subs(num2/den2,x);

% Calculamos el numerador de la tercera ecuación
    num3= (((1/2)*df2)^2 - ((1/6)*df*df3))*f*f*f;
```

```
    % Calculamos el denominador de la tercera ecuación
     den3=df^5;

% Calculamos el cociente del tercer término
     co3=subs(num3/den3,x);

% Calculamos el cociente del método
     co=co1-co2-co3;
     x=x-co;

% Criterio de parada
     if abs(co)<tol


% Dos iterados sucesivos están suficientemente cerca
         break;
     end
end

% Evaluación de la función en la aproximación
y=subs(f,x);
```

3.6. Método de König

Otra función de iteración de orden m para encontrar soluciones de la ecuación $f(z) = 0$, con $f : \mathbb{C} \to \mathbb{C}$ analítica, es dada por

$$K_m(z) = z + (m-1)\frac{(1/f(z))^{(m-2)}}{(1/f(z))^{(m-1)}}.$$

Es claro que $K_2 = N_f$ (método de Newton).
Por ejemplo,

$$K_3(z) = z - \frac{2f(z)f'(z)}{2(f'(z))^2 - f(z)f''(z)} = H_f(z) \quad \text{(método de Halley)}$$

$$K_4(z) = z - \frac{2(f'(z))^2 - 3f(z)(f(z)f''(z))}{6(f'(z))^3 - 6f(z)f'(z)f''(z) + (f(z))^2 f'''(z)}.$$

Como en el caso enterior, tenemos que si $f(w) = 0$ entonces $K_m^{(i)}(w) = 0, i = 1, 2, \ldots, m-1$, luego las raíces de $f(z) = 0$ son puntos fijos superatractores de orden al menos m de $K_m, m = 2, 3, \ldots$. La afirmación recíproca no es verdadera en general. Por ejemplo, si todas las raíces de f son simples, entonces podemos escribir

$$h(z) = (f(z))^{-1} = \sum_{i=1}^{n} A_i (z - z_i^*)^{-1},$$

donde $z_i^*, i = 1, \ldots, n$ son las raíces de $f(z) = 0$.
Luego, $h^{(m)}(z) = (-1)^m m! \sum_{i=1}^{n} A_i (z - z_i^*)^{-(m+1)}$. Reemplazando en la fórmula para K_m, tenemos

$$K_m(z) = z - f(z)\frac{L_{m-1}(z)}{L_m(z)}$$

donde

$$L_m(z) = \sum_{i=1}^{n} A_i \prod_{j \neq i}^{n} (z - z_j^*)^m.$$

Para $m > 2$, la condición de punto fijo $K_m(p) = p$ nos da

(i) $f(p) = 0$, *esto es, las raíces de* $f(z) = 0$ *o*

(ii) $L_{m-1}(p) = 0$,

lo cual da origen a puntos fijos extraños, es decir, soluciones de $L_{m-1}(z) = 0$ que no son soluciones de $f(z) = 0$. Estos puntos fijos pueden ser repulsores, indiferentes o atractores extraños, y en cualquier caso, ellos alteran las cuencas de atracción de las raíces de $f(z) = 0$.
Para K_3 se tiene

Teorema: *Todos los puntos fijos de* K_3 *que no son raíces de* f*, son repulsores.*

Demostración: *Un cálculo directo muestra que*

$$K_3'(z) = \frac{(f(z))^2(3(f''(z))^2 - 2f'(z)f''(z))}{(f(z)f''(z) - 2(f'(z))^2)^2},$$

luego, $K_3'(\alpha) = 3$ *para todo* α *tal que* $K_3(\alpha) = \alpha$ *y* $f(\alpha) \neq 0$.

Teorema: *Para* $f(z) = z^2 - 1, \mathcal{J}(K_m) = \{iy : y \in \mathbb{R}\}, m = 2, 3, \ldots$.

Teorema: *Para* $n \geq 2$, *sea* $g_n(z) = z^n - 1$. *Sea* $Z_n = \{z_j^* = e^{2\pi j/n}, j = 1, 2, \ldots, n-1\}$ *el conjunto de las raíces de* g_n. *Entonces para* $m \geq 3$, *todos los puntos fijos extraños de* $K_m(z)$ *son repulsores y están ubicados sobre los rayos* $arg(z) = (2j+1)\pi/n$.

3.6.1. Código Matlab : Método de König

```
% Método de König para la resolución de una ecuación no lineal
% Km(x) = x + (m-1)(df(1/f(x))^(m-2))/(df(1/f(x))^(m-1))
% [x,i,y]=Mkonig(f,x0,n,tol,m)
% Variables de entrada:
% f - función que determina la ecuación a resolver f(x)=0
% x0 - aproximación inicial a un cero de f
% n - número máximo de iteraciones
% tol - tolerancia, la usamos para hacer un criterio de parada,
% en caso que el cociente entre el numerador y el denominador sea
% más pequeño que tol en valor absoluto, el método parará
% m - orden de la derivación
% Variables de salida:
% x - aproximación a la raíz
% i - número de iteraciones realizadas
% y - valor de la función evaluada en la aproximación x, es decir
% f(x)
% ejemplo:
% [x,i,y]=Mkonig('x^3-3*x^2',5,500,10E-8,4)


% Calculamos la derivada de orden (m-2) de manera simbólica
h = strcat (strcat('1/(',f),')');
df=diff(h,m-2);
```

```
% Calculamos la derivada de orden (m-1) de manera simbólica
df2=diff(df,1);

% Inicialización de la aproximación
x=x0;


for i=1:n

% Calculamos el numerador de la ecuación
    num=(m-1)*df;

% Calculamos el denominador de la ecuación
    den=2*(df^2)-(f*df2);


% Calculamos el cociente entre el numerador y denominador
    co=subs(num/den,x);
    x=x+co;


% Criterio de parada
    if abs(co)<tol


% Dos iterados sucesivos están suficientemente cerca
        break;
    end
end

% Evaluación de la función en la aproximación
y=subs(f,x);
```

3.7. Método de Newton Relajado

Sea $p : \mathbb{C} \to \mathbb{C}$ un polinomio. Supongamos que p tiene una raíz α de multiplicidad $m \geq 2$. Como vimos anteriormente, el método de Newton converge linealmente a dicha raíz (punto fijo de N_p) en una vecindad de α. Aplicando el método de Newton a $h(z) = \sqrt[n]{p(z)}$, tenemos

$$N_m(z) = z - \frac{mp(z)}{p'(z)}$$

y N_m converge cuadráticamente a la raíz múltiple de orden m, para cada punto en su cuenca de atracción, esto es, $N'_m(z) = 0$ cuando z es una raíz múltiple de orden m de p.

Teorema: *Sea $p(z) = (z-a)^2(z-b)$ un polinomio cúbico con una raíz doble. Entonces el método de Newton relajado N_2 aplicado a p es conjugado, por la transformación de Möbius de $M : \mathbb{C} \to \mathbb{C}, M(z) = \frac{3z+a-4b}{2(z-a)}$, a $f(z) = z^2 - \frac{3}{4}$.*

Demostración: *Aplicando el método de Newton para raíces dobles al polinomio cúbico $p(z) = (z-a)^2(z-b)$, tenemos*

$$N_2(z) = z - \frac{2p(z)}{p'(z)} = \frac{z^2 + az - 2ab}{3z - a - 2b}$$

y

$$N'_2(z) = \frac{(z-a)(3z+a-4b)}{(3z-a-2b)^2}.$$

Los puntos fijos de N_2 son $z = a$ y $z = b$. Como $N'_2(a) = 0$, a es un punto fijo superatractor, y como $N'_2(b) = -1$, b es un punto fijo indiferente. Los puntos críticos de N_2 son $z = a$ y $z = \frac{4b-a}{3}$. Los puntos fijos de la aplicación cuadrática $p_c(z) = z^2 + c$ en la esfera de Riemann son $z = 0$ y $z = \infty$. Notemos que el punto $z = \infty$ también es un punto fijo superatractor. Ahora, la transformación de Möbius

$$M(z) = \frac{3z + a - 4b}{2(z-a)}$$

lleva a en ∞, y $\frac{4b-a}{3}$ en 0. Luego, aplica los puntos críticos de N_2 en los de la aplicación cuadrática p_c. Conjugando N_2, por la transformación h, obtenemos la aplicación cuadrática

$$h \circ N_2 \circ h^{-1}(z) = z^2 - \frac{3}{4} = p_{\frac{3}{4}}(z).$$

Para el polinomio $p(z) = q(z)(z-b)^k$, *donde* $q(b) \neq 0$, *el punto* $z = b$ *es una raíz de orden* k *de* p, *y*

$$N_m(b) = 1 - \frac{m}{k},$$

luego b *es un punto fijo superatractor sólo si* $k = m$. *Por otra parte, si* $k < \frac{m}{2}, |N'_m(b)| > 1$ *y por lo tanto* b *es entonces un punto fijo repulsor. Si* $k = \frac{m}{2}, |N'_m(b)| = 1$ *y* b *es un punto fijo indiferente. Finalmente, si* $k > \frac{m}{2}, |N'_m(b)| < 1$ *y* b *es un punto fijo atractor para* N_m, *pero la convergencia es sólo lineal, no cuadrática.*

Teorema: *El método de Newton relajado,* N_m, *aplicado a cualquier polinomio de grado* $m+1$, *con una raíz de orden* m, *es conjugado por una transformación de Möbius de la esfera de Riemann, a la aplicación cuadrática*

$$p(z) = z^2 + \frac{1-m^2}{4}.$$

En particular, cuando $m = 3$, *el conjunto de Julia de* $z^2 - 2$ *es el segmento sobre el eje real entre* -2 *y* 2. *Por lo tanto, la cuenca de atracción de la raíz triple de* $(z-a)^3(z-b)$ *por iteraciones de* N_3 *es todo el plano complejo, excepto por una linea recta desde* $\frac{a+3b}{4}$ *a través de* b *a* ∞.

3.7.1. Código Matlab : Método de Newton Relajado

```
% Método de Newton relajado para la resolución de una ecuación no
% lineal
% N_m(x) = x - ((m * f(x))/df(x))
% [x,i,y]=MnewtonRelajado(f,x0,n,tol,m)
% Variables de entrada:
% f - función que determina la ecuación a resolver f(x)=0
% x0 - aproximación inicial a un cero de f
% n - número máximo de iteraciones
% tol - tolerancia, la usamos para hacer un criterio de parada,
```

```
% en caso que m*f/df sea más pequeño que tol en valor absoluto,
% el método parará
% m - orden de multiplicidad de la raíz
% Variables de salida:
% x - aproximación a la raíz
% i - número de iteraciones realizadas
% y - valor de la función evaluada en la aproximación x, es decir
% f(x)
% ejemplo:
% [x,i,y]=MnewtonRelajado('x^4-3*x^3-3*x^2+7*x+6',5,500,10E-8,2)



% Calculamos la derivada de manera simbólica

df=diff(f);


% Inicialización de la aproximación

x=x0;


for i=1:n


% Calculamos el cociente entre la función y su derivada

    co=subs(f/df,x);
    co=m*co;
    x=x-co;


% Criterio de parada

    if abs(co)<tol



% Dos iterados sucesivos están suficientemente cerca

        break;
    end
end
```

```
% Evaluación de la función en la aproximación

y=subs(f,x);
```

3.8. Método de Newton para Raíces Múltiples

Si $g : \mathbb{C} \to \mathbb{C}$ tiene una raíz múltiple, w_0, de multiplicidad $k \geq 2$,vimos que $N'_g(w_0) = \frac{k-1}{k}$ y por lo tanto la convergencia en la cuenca de atracción de w_0 es lineal, por lo tanto lenta.
Es fácil ver que si g tiene una raíz múltiple de orden $k \geq 2$ en w_0, entonces la función $f(z) = \frac{g(z)}{g'(z)}$ tiene una raíz simple en w_0. Aplicando el método de Newton a f, obtenemos

$$M_g(z) = z - \frac{g(z)g'(z)}{(g'(z))^2 - g(z)g''(z)}.$$

Esta función es más conocida como *Método de Newton modificado para raíces múltiples*. Ahora es fácil verificar que M_g tiene convergencia cuadrática en las raíces múltiples de g. Este método también puede ser aplicado cuando existen dos raíces, que aunque distintas, estén muy próximas.

3.8.1. Código Matlab : Método de Newton para Raíces Múltiples

```
function [x,i,y]=MnewtonRaicesMultiples(f,x0,n,tol)


% Método de Newton para raíces múltiples para la resolución de una
% ecuación no lineal
% Mg(x) = x - (f(x)*df(x))/(df(x))^2 - (f(x)*df2(x))
% [x,i,y]=MnewtonRaicesMultiples(f,x0,n,tol)
% Variables de entrada:
% f - función que determina la ecuación a resolver f(x)=0
% x0 - aproximación inicial a un cero de f
% n - número máximo de iteraciones
% tol - tolerancia, la usamos para hacer un criterio de parada,
% en caso que el resultado sea más pequeño que tol en valor absoluto,
% el método parará
```

```
% Variables de salida:
% x - aproximación a la raíz
% i - número de iteraciones realizadas
% y - valor de la función evaluada en la aproximación x, es decir
% f(x)
% ejemplo:
% [x,i,y]=MnewtonRaicesMultiples('x^3 - 3 * x^2',5,500,10E-8)


% Calculamos la derivada de manera simbólica

df=diff(f);


% Calculamos la derivada segunda de manera simbólica

df2=diff(f,2);


% Inicialización de la aproximación

x=x0;


for i=1:n


% Calculamos el numerador de la función

    num=(f*df);



% Calculamos el denominador de la función

    den=(df)^2 - (f*df2);



% Calculamos el cociente entre la función y su derivada

    co=subs(num/den,x);
    x=x-co;
```

```
% Criterio de parada

    if abs(co)<tol


% Dos iterados sucesivos están suficientemente cerca

        break;
    end
end


% Evaluación de la función en la aproximación

y=subs(f,x);
```

3.9. Cálculo computacional de los métodos para la resolución de una ecuación no lineal

Calcular el tiempo que es necesario para obtener el resultado en cada uno de los métodos y poder así realizar una comparativa sobre el cálculo computacional es imprescindible para presentar un estudio riguroso y completo.

Para ello se ha calculado en un mismo ordenador una ecuación para todos los métodos utilizando los mismos datos en los parámetros del cálculo y utilizando el comando de Matlab *cputime* para obtener el tiempo necesario.

Hemos tomado como referencia la ecuación test $x^5-11x^4+27x^3+71x^2-328x+240$ y se han fijado los siguientes parámetros para su cálculo : $n=50$ iteraciones, tolerancia $=10^{-5}$. Se ha ejecutado el algoritmo *M_dinamica3* y los resultados obtenidos bajo estas condiciones se muestran en la siguiente gráfica.

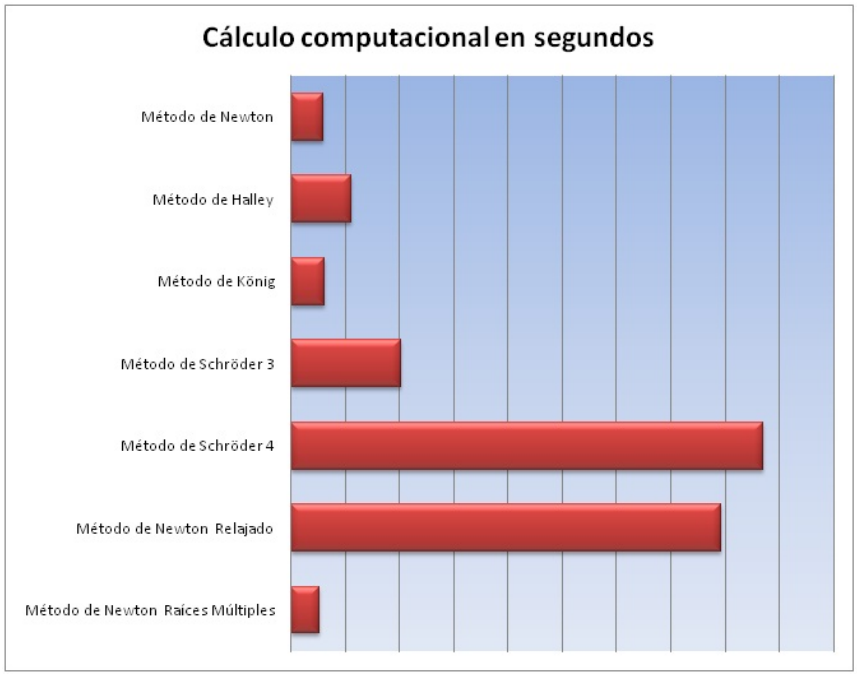


Figura 3.2: Cálculo computacional de los diferentes métodos.

Como se puede observar en el gráfico, los métodos *Método de Schröder 4* y *Método de Newton Relajado* son los de mayor coste computacional, seguidos del *Método de Schröder 3*.

Evaluando funcionalmente cada método se pueden explicar los resultados obtenidos. Entre los métodos computacionalmente mas atractivos se encuentran aquellos cuya fórmula es menos compleja.

En el caso del *Método de Newton*, la fórmula para calcularlo es $N(x) = x - \frac{f(x)}{f'(x)}$. Esto significa una operación sencilla sin apenas gasto computacional al tener que repetirla en cada iteración.

Un caso similar es el *Método de Halley*, cuya fórmula $H(x) = x - \frac{2f(x)f'(x)}{2(f'(x))^2 - f(x)f''(x)}$ es algo más compleja debido que se ha de calcular hasta la derivada segunda de la función introducida en cada iteración, pero sigue teniendo un gasto computacional bajo.

El caso del *Método de König* es diferente puesto que depende del or-

den de derivación con el que se desea calcular (m). Para el cálculo del gasto computacional se ha optado por $m = 2$ (que da lugar a que el resultado sea idéntico al del *Método de Newton*), lo que optimiza el tiempo de las operaciones del método puesto que la fórmula usada para éste es $K_m(x) = x + (m-1)\frac{(1/f(x))^{(m-2)}}{(1/f(x))^{(m-1)}}$.

El *Método de Newton Raíces Múltiples* funcionalmente genera un gasto computacional bajo porque en este caso su fórmula $M_g(x) = x - \frac{g(x)g'(x)}{(g'(x))^2 - g(x)g''(x)}$ tanto para raíces múltiples como para raíces simples converge rapidamente.

Por el contrario, nos encontramos con las dos versiones del *Método de Schröder*, denominadas *Método de Schröder 3* y *Método de Schröder 4*, cuya formulación más compleja tiene como consecuencia unos tiempos de generación mucho mayores que el resto. La función de iteración de Schröder 3 es la siguiente: $S_3(x) = x - \frac{f(x)}{f'(x)} - \frac{f''(x)(f(x))^2}{2(f'(x))^3}$. Esta función ya presenta elementos de cálculo más complejos que relentizan el cálculo en cada iteración. Pero sin duda, la función de iteración con más elementos y más lenta es sin duda la de Schröder 4, $S_4(z) = z - \frac{f(x)}{f'(x)} - \frac{f''(x)(f(x))^2}{2(f'(x))^3} - \frac{(\frac{1}{2}(f''(x))^2 - \frac{1}{6}f'(x)f'''(x))(f(x))^3}{(f'(x))^5}$, donde en cada una de las iteraciones se calcula hasta la tercera derivada y se realizan bastantes multiplicaciones y divisiones que generan un gasto computacional alto, lo que significa mayor tiempo de espera para la visualización de los resultados.

Para explicar el elevado tiempo de cálculo del *Método de Newton Relajado* hay que buscar la explicación en la naturaleza de la función introducida como test. La función elegida tiene como raíces $x = -3$, $x = 5$, $x = 1$ y $x = 4$, pero solo esta última con multiplicidad 2. Uno de los parámetros de entrada del método es la multiplicidad de la raíz (m), por lo que se ha introducido el valor $m = 2$, por lo que para calcular el resto de las raíces cuya multiplicidad es 1 la función diverge en lugar de converger con un número bajo de iteraciones como ocurre en el resto de los métodos mencionados.

3.10. Primera representación para el estudio de las cuencas de atracción de polinomios

La primera representación, cuyo código en Matlab se ha denominado *M_Dinamica*, calcula la velocidad tanto de la divergencia como de la con-

vergencia del polinomio. La gama de colores describen el porcentaje de iteraciones sobre el máximo que hayamos elegido, para así poder estudiar en que puntos el método necesita más iteraciones y en cuales necesita menos y poder así estudiar mejor su comportamiento con cada uno de los métodos.

Se considera que diverge en un punto cuando el método alcanza el número máximo de iteraciones sin alcanzar la convergencia.

3.10.1. Código Matlab de la primera versión para la visualización de las cuencas de atracción

Este programa representa las cuencas de atracción del polinomio y las dibuja en diferentes colores según la velocidad en la que la función converga o diverga a una solución sin separar las raíces por colores.

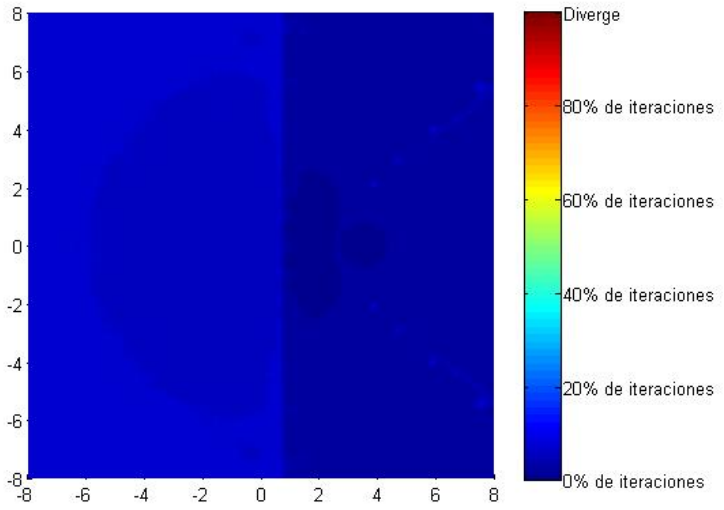
```
function M_Dinamica(f,met,param,mat_x,mat_y)


% Esta función dibuja en diferentes colores las cuencas de atracción
% según la velocidad de convergencia o divergencia
% M_Dinamica(f,met,param,mat_x,mat_y)
% Variables de entrada:
% f - función a calcular. Debe estar escrita con los monomios del
% polinomio en orden descendiente según el grado, en caso de tener
% un factor multiplicando debe ser escrito antes de la x y con un
% signo *.
% met - método que nos da la función de iteración
% param - array con los parámetros necesarios de cada método
% mat_x - longitud del eje de abscisas
% mat_y - longitud del eje de ordenadas
% ejemplo:
% M_Dinamica('x^5+x^3-5*x^2+1',@Mnewton,[500,10E-3],100,100)


% Calculamos el tamaño del array de parámetros

tam = length(param);
```

```
% Primero convertimos la función a un vector con los coeficientes
% numéricos

p=str2pol(f);


% Obtenemos el número de operandos de la ecuación

n=length(p);


% Calculamos lambda

for i=1:n-1
    res(i)=abs((p(i+1)/p(1)));
end
lambda=max(res);
M=lambda+1;


% Calculamos el mallado para dibujar la función

x = linspace(-M,M,mat_x);
y = linspace(-M,M,mat_y);
[X,Y] = meshgrid(y,x);
W = zeros(mat_x,mat_y);


% Utilizamos el método seleccionado para calcular el resultado


for m = 1:size(X,2)
    for j = 1:size(X,1)


% Si el método necesita solo 2 parámetros de entrada

        if tam ==2
            [xc,it] = met(f,X(j,m)+Y(j,m)*1i,param(1),param(2));


% Si el método necesita 3 parámetros de entrada
```

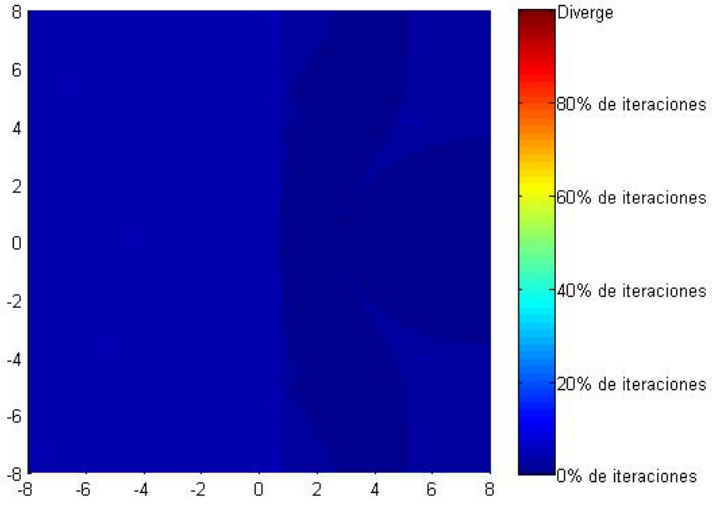
```
        else
            [xc,it] = met(f,X(j,m)+Y(j,m)*1i,param(1),param(2),param(3));
        end
        W(j,m) = W(j,m) + it;
    end
end
W(1,1)=0;
W(mat_x, mat_y)=param(1);


% Representamos el resultado final

hold on;
pcolor(X,Y,W);
colormap(jet);
ch = colorbar;
set(ch, 'YLim', [0 param(1)]);
labels = get(ch,'YLim');
dif = labels(2) - labels(1);
tramo = dif/5;
set(ch,'YTick',[labels(1) (labels(1)+tramo) (labels(1)+(2*tramo))
      (labels(1)+(3*tramo)) (labels(1)+(4*tramo)) labels(2)]...
      ,'YTickLabel',{''0% de iteraciones','20% de iteraciones','40% de iteraciones',...
      '60% de iteraciones','80% de iteraciones','Diverge'});
axis image;
shading interp;
hold off;
```

3.10.2. Resultados gráficos de la primera versión para el método de Newton

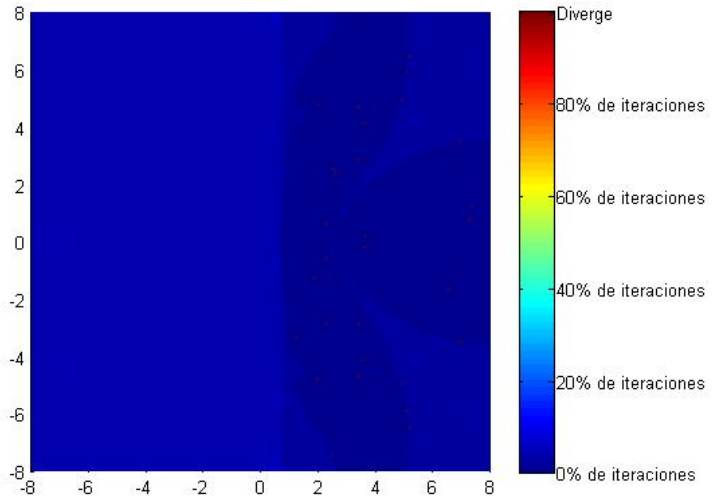
Para observar los resultados de la primera versión de las representaciones gráficas de las cuencas de atracción, se ha elegido el polinomio $x^4 - 3x^3 - 3x^2 + 7x + 6$, cuyas raíces son $x = 3$,$x = 2$ y $x = -1$ (esta última con multiplicidad 2).



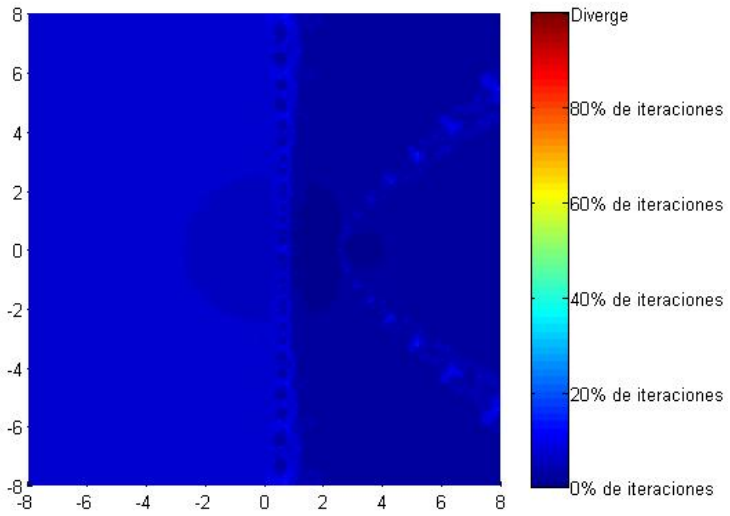
(a) Método de Newton con $n = 500$ iteraciones



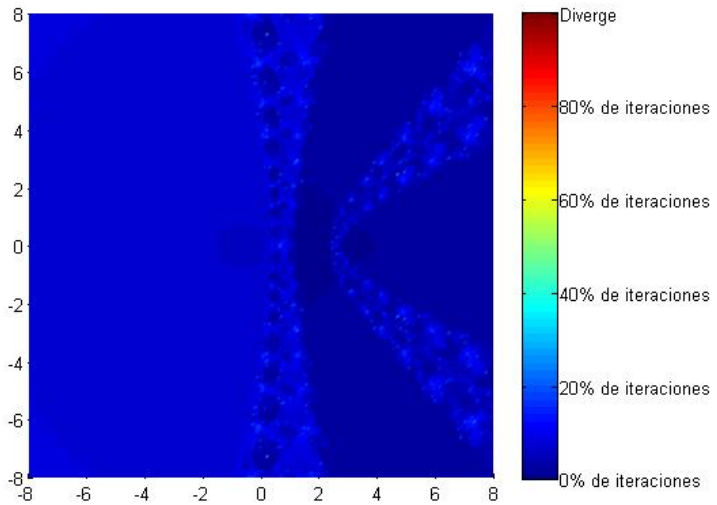
(b) Método de Halley con $n = 500$ iteraciones



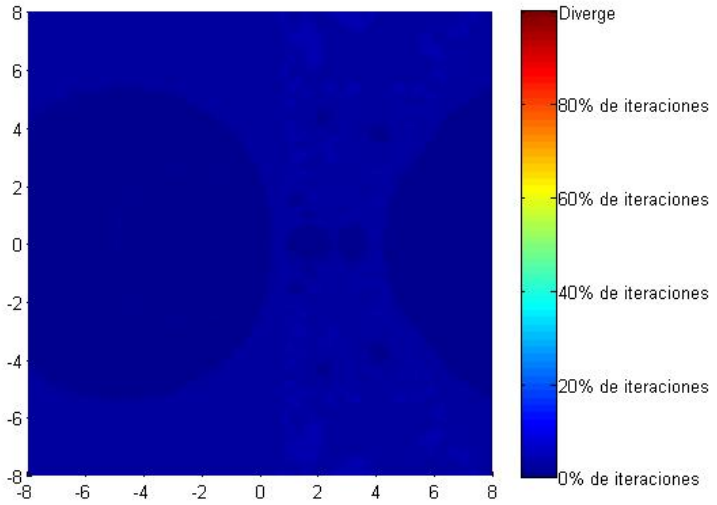
(c) Método de König con $n = 500$ iteraciones y $m = 3$



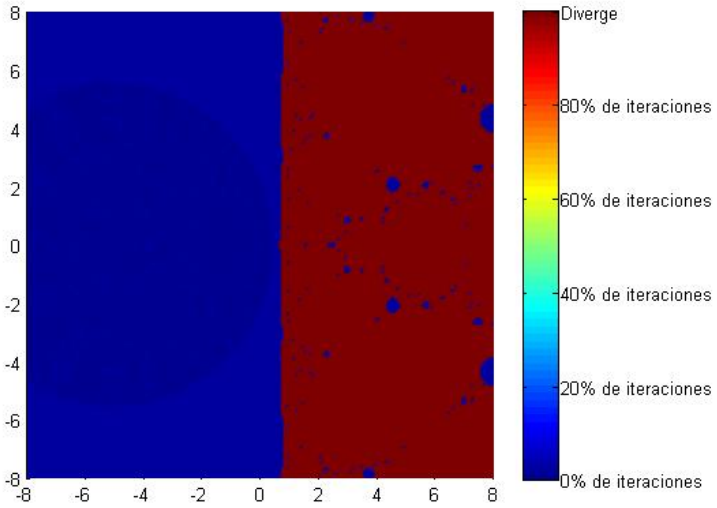
(d) Método de Schröder 3 con $n = 500$ iteraciones



(e) Método de Schröder 4 con $n = 500$ iteraciones



(f) Método de Newton Raíces Múltiples con $n = 500$ iteraciones



(g) Método de Newton Relajado con $n = 500$ iteraciones y $m = 2$

Figura 3.3: Representación de las cuencas de atracción para la primera versión sobre la función $x^4 - 3x^3 - 3x^2 + 7x + 6$, con 200 particiones del eje abscisas y 200 particiones del de coordenadas.

Para crear una imagen minimamente útil con la cual se pueda trabajar, es necesario utilizar unos valores altos tanto en el número de iteraciones como en los puntos de los ejes, al igual que introducir una tolerancia suficientemente baja.

3.11. Segunda representación para el estudio de las cuencas de atracción de polinomios

Para la segunda y tercera versión se utilizan colores para determinar cada raíz de la función. El significado de cada color es el siguiente:

Amarillo : **Diverge**

Rojo : **Primera raíz**

Verde : **Segunda raíz**

Azul : **Tercera raíz**

Blanco : **Cuarta raíz**

Negro : **Quinta raíz**

Gris : **Sexta raíz**

Cyan : **Séptima raíz**

Fucsia : **Octava raíz**

En este caso, el programa muestra en un color diferente cada raíz de la función introducida.

3.11.1. Código Matlab de la segunda versión para la visualización de las cuencas de atracción

```
function M_Dinamica2(f,met,param,mat_x,mat_y,ventana)


% Esta función dibuja en un color diferente cada raíz de la función
% introducida
% M_Dinamica2(f,met,param,mat_x,mat_y,ventana)
% Variables de entrada:
% f - función a calcular. Debe estar escrita con los monomios del
% polinomio en orden descendiente según el grado, en caso de tener
% un factor multiplicando debe ser escrito antes de la x y con un
% signo *.
% met - método que nos da la función de iteración
% param - array con los parámetros necesarios de cada método
% mat_x - longitud del eje de abscisas
% mat_y - longitud del eje de ordenadas
% ventana - interfaz donde se dibuja el resultado



% Calculamos el tamaño del array de parámetros

tam = length(param);


% Primero convertimos la función a un vector con los coeficientes
% numéricos

p=str2pol(f);


% Obtenemos el número de operandos de la ecuación

n=length(p);


% Lista con los colores (rojo, verde, azul, blanco, negro, gris,
% cyan, fucsia)
```

```
color_raiz_nuevo={[1,0,0],[0,1,0],[0,0,1],[1,1,1],[0,0,0],[0.5,0.5,0.5],
[0,1,1],[1,0,1]};


% Contador para asignar color

c_color=1;


% Array con las raíces del método y los colores

raices=[];


% Calculamos lambda

for i=1:n-1
    res(i)=abs((p(i+1)/p(1)));
end
lambda=max(res);
M=lambda+1;


% Calculamos el mallado para dibujar la función

W = zeros(mat_x,mat_y);
x = linspace(-M,M,mat_x);
y = linspace(-M,M,mat_y);
[X,Y] = meshgrid(y,x);


% Utilizamos el método seleccionado para calcular el resultado


for m = 1:size(X,2)
    for j = 1:size(X,1)


% Si el método necesita solo 2 parámetros de entrada

        if tam ==2
            [xc,it] = met(f,X(j,m)+Y(j,m)*1i,param(1),param(2));
```

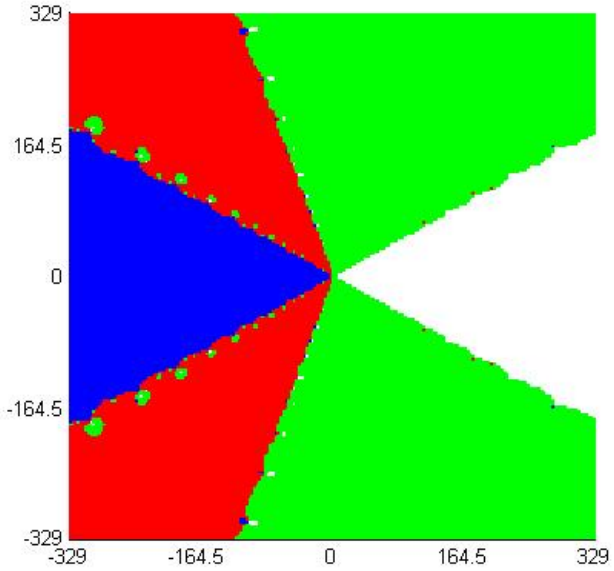
```
% Si el método necesita 3 parámetros de entrada

        else
            [xc,it] = met(f,X(j,m)+Y(j,m)*1i,param(1),param(2),param(3));
        end


% Si diverge se pinta de amarillo

        if it==param(1)
            W(j,m,1)=1;
            W(j,m,2)=1;
            W(j,m,3)=0;


% Si converge, se pinta del color predeterminado

        else
            [dist_raiz,ind]=min(abs(xc-raices));
            if dist_raiz<2*param(2)
                W(j,m,:)=color_raiz_nuevo{ind};
            else
                W(j,m,:)=color_raiz_nuevo{c_color};
                raices=[raices,xc];
                c_color=c_color+1;
            end
        end
    end
end


% Guardamos las raíces del polinomio

save raices.mat raices;


% Representamos el resultado final
```

```
hold on;
axes(ventana.axes1);


% Configuramos los parámetros de los ejes

xlab=[-M 0 M];
ylab=[-M 0 M];
xlabmax=[-M (-M/2) 0 M/2 M];
ylabmax=[-M (-M/2) 0 M/2 M];
dimx=[1 length(x)/2 length(x)];
dimy=[1 length(y)/2 length(y)];
dimxmax=[1 length(x)/4 length(x)/2 3*length(x)/4 length(x)];
dimymax=[1 length(y)/4 length(y)/2 3*length(y)/4 length(y)];

heatmap(W,x,y);


% Eje x

if length(x) > 20
    set(gca,'XTick',dimxmax,'XTickLabel',xlabmax);
elseif length(x) > 8
    set(gca,'XTick',dimx,'XTickLabel',xlab);
else
    set(gca,'XTickLabel',x);
end


% Eje y

if length(y) > 20
    set(gca,'YTick',dimymax,'YTickLabel',ylabmax);
elseif length(y) > 8
    set(gca,'YTick',dimy,'YTickLabel',ylab);
else
    set(gca,'YTickLabel',y);
end

axis image;
shading interp;
hold off;
```

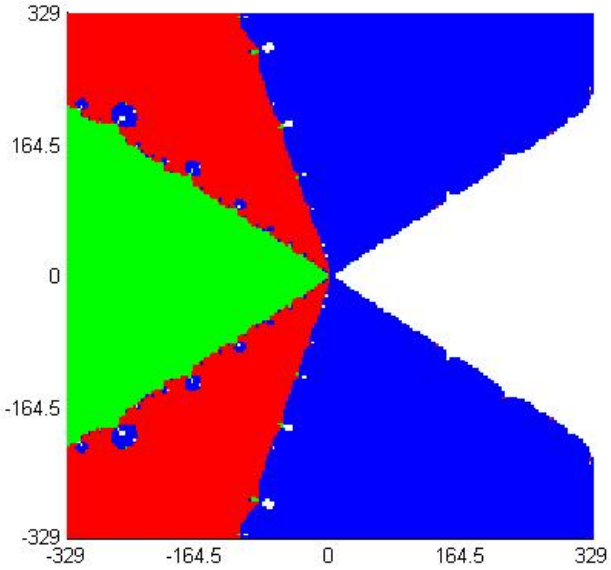
3.11.2. Resultados gráficos de la segunda versión

Para observar los resultados de la segunda versión de las representaciones gráficas de las cuencas de atracción, se ha elegido el polinomio $x^5 - 11x^4 + 27x^3 + 71x^2 - 328x + 240$, cuyas raíces son $x = -3$,$x = 5$,$x = 1$ y $x = 4$ (esta última con multiplicidad 2).

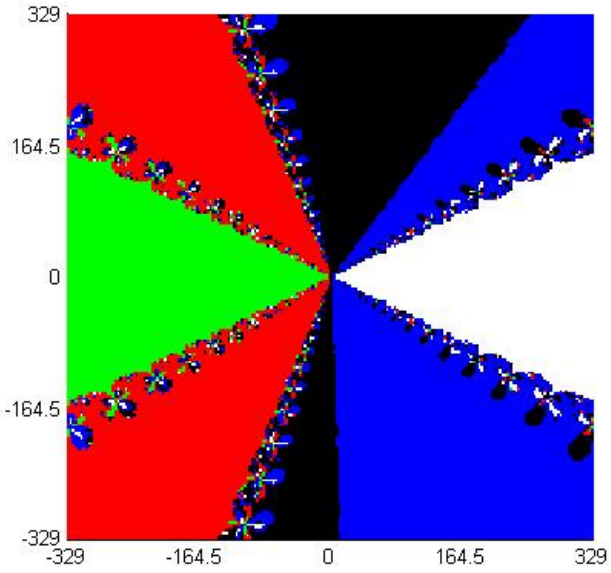
Como se observa en la figura 3.4, los resultados obtenidos al calcular el polinomio con cada uno de los métodos anteriormente explicados son los siguientes:



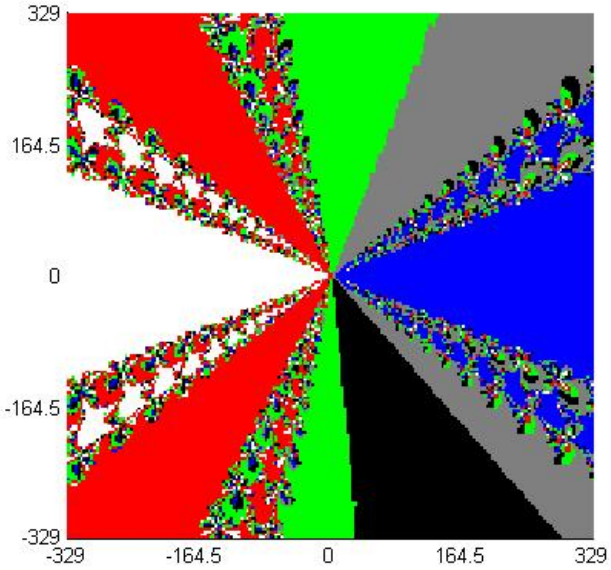
(a) Método de Newton



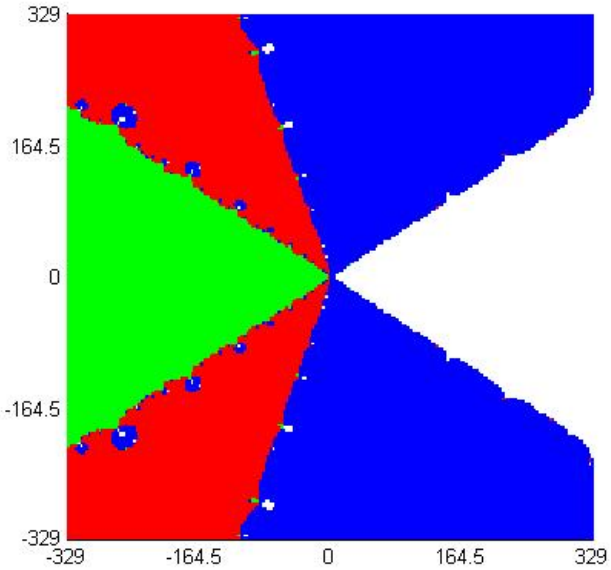
(b) Método de Halley



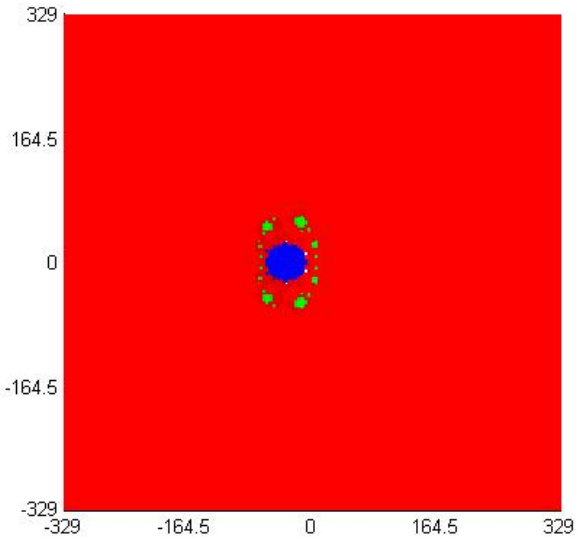
(c) Método de Schröder con orden 3



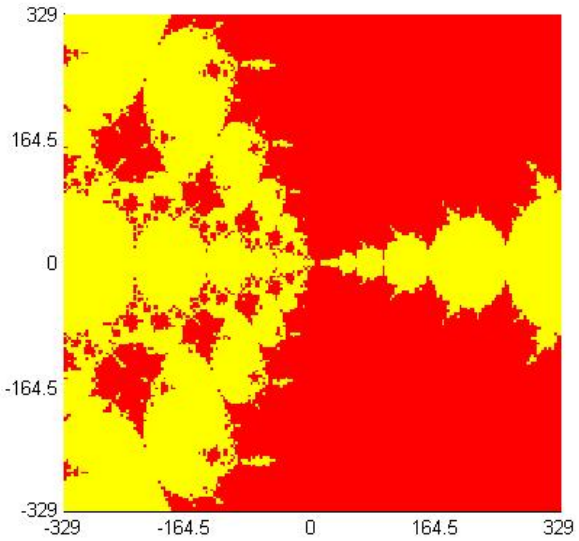
(d) Método de Schröder con orden 4



(e) Método de König cono orden de derivación = 2



(f) Método de Newton Raíces Múltiples



(g) Método de Newton Relajado

Figura 3.4: Representación de las cuencas de atracción de la segunda versión de la función $x^5 - 11x^4 + 27x^3 + 71x^2 - 328x + 240$, con $n = 800$ iteraciones, tolerancia $= 10^{-6}$, 200 particiones de los ejes de abscisas y coordenadas.

Como se puede observar en los resultados gráficos, las soluciones obtenidas representan de un color distinto cada una de las raíces de la función. Debido a la naturaleza de cada método utilizado, el orden de las raíces cambia, es por esta razón que las soluciones comparten patrón en la forma pero con diferentes colores para cada raíz.

En el caso de los métodos de Schröder 3 y Schröder 4, el resultado no es del todo limpio ni óptimo debido al fenómeno de *puntos extraos*, que como se explica anteriormente en los apartados de teória son *puntos fijos que alteran las cuencas de atracción de las raíces de* $f(z) = 0$

Para posteriores mejoras, se deberían mejorar los resultados de los métodos de Newton Relajado y de Newton Raíces Múltiples para mostrar resultados más realistas y de mayor precisión.

Al igual que en el resto de versiones, para crear una imágen minimamente útil con la cual se pueda trabajar, es necesario utilizar unos valores altos tanto en el número de iteraciones como en los puntos de los ejes, al igual que introducir ina tolerancia suficientemente baja.

3.12. Tercera representación para el estudio de las cuencas de atracción de polinomios

Esta versión es la más completa y es una composición de las dos anteriores. Como resultado del programa se observa en cada color cada raíz de la función pero esta vez también se puede observar la velocidad en la que cada raíz converge o diverge dependiendo de la intensidad del color.

3.12.1. Código Matlab de la tercera versión para la visualización de las cuencas de atracción

```
function M_Dinamica3(f,met,param,mat_x,mat_y,ventana)


% Esta función dibuja en un color diferente cada raíz de la función
% introducida y conforme se acerca a otra raíz va cambiando de color
```

```
% M_Dinamica3(f,met,param,mat_x,mat_y,ventana)
% Variables de entrada:
% f - función a calcular. Debe estar escrita con los monomios del
% polinomio en orden descendiente según el grado, en caso de tener
% un factor multiplicando debe ser escrito antes de la x y con un
% signo *.
% met - método que nos da la función de iteración
% param - array con los parámetros necesarios de cada método
% mat_x - longitud del eje de abscisas
% mat_y - longitud del eje de ordenadas
% ventana - interfaz donde se dibuja el resultado


% Calculamos el tamaño del array de parámetros

tam = length(param);


% Primero convertimos la función a un vector con los coeficientes
numéricos

p=str2pol(f);


% Obtenemos el número de operandos de la ecuación

n=length(p);


% Lista con los colores (rojo, verde, azul, blanco, negro, gris,
% cyan, fucsia)


color_raiz_nuevo={[1,0,0],[0,1,0],[0,0,1],[1,1,1],[0,0,0],[0.5,0.5,0.5],
[0,1,1],[1,0,1]};


% Lista con los colores anteriores o potenciados para las
% diferentes cuencas de atracción

color_raiz_nuevo2={[0.9,0,0],[0,0.9,0],[0,0,0.9],[0.9,0.9,0.9],
[0.1,0.1,0.1],[0.55,0.55,0.55],[0,0.9,0.9],[0.9,0,0.9]};
```

```
color_raiz_nuevo3={[0.8,0,0],[0,0.8,0],[0,0,0.8],[0.8,0.8,0.8],
[0.2,0.2,0.2],[0.6,0.6,0.6],[0,0.8,0.8],[0.8,0,0.8]};
color_raiz_nuevo4={[0.7,0,0],[0,0.7,0],[0,0,0.7],[0.7,0.7,0.7],
[0.3,0.3,0.3],[0.65,0.65,0.65],[0,0.7,0.7],[0.7,0,0.7]};
color_raiz_nuevo5={[0.6,0,0],[0,0.6,0],[0,0,0.6],[0.6,0.6,0.6],
[0.4,0.4,0.4],[0.7,0.7,0.7],[0,0.6,0.6],[0.6,0,0.6]};


% Contador para asignar color

c_color=1;


% Array con las raíces del método y los colores

raices=[];


% Calculamos lambda

for i=1:n-1
    res(i)=abs((p(i+1)/p(1)));
end
lambda=max(res);
M=lambda+1;


% Calculamos los parámetros necesarios para dibujar la función


W = zeros(mat_x,mat_y);
x = linspace(-M,M,mat_x);
y = linspace(-M,M,mat_y);
[X,Y] = meshgrid(y,x);


% Array donde diferenciamos los diferentes niveles de velocidad
% de iteración de la raíz

velocidad = linspace(1,param(1),6);


% Utilizamos el método seleccionado para calcular el resultado
```

```
for m = 1:size(X,2)
    for j = 1:size(X,1)


% Si el método necesita solo 2 parámetros de entrada

        if tam ==2
            [xc,it] = met(f,X(j,m)+Y(j,m)*1i,param(1),param(2));



% Si el método necesita 3 parámetros de entrada

        else
            [xc,it] = met(f,X(j,m)+Y(j,m)*1i,param(1),param(2),param(3));
        end



% Si diverge se pinta de amarillo

        if it==param(1)
            W(j,m,1)=1;
            W(j,m,2)=1;
            W(j,m,3)=0;



% Si converge, se pinta del color predeterminado

  else
      [dist_raiz,ind]=min(abs(xc-raices));
            if dist_raiz<2*param(2)
                if it>= velocidad(1) && it< velocidad(2)
                    W(j,m,:)=color_raiz_nuevo{ind};
                elseif it>= velocidad(2) && it< velocidad(3)
                    W(j,m,:)=color_raiz_nuevo2{ind};
                elseif it>= velocidad(3) && it< velocidad(4)
                    W(j,m,:)=color_raiz_nuevo3{ind};
                elseif it>= velocidad(4) && it< velocidad(5)
                    W(j,m,:)=color_raiz_nuevo4{ind};
                elseif it>= velocidad(5) && it<velocidad(6)
```

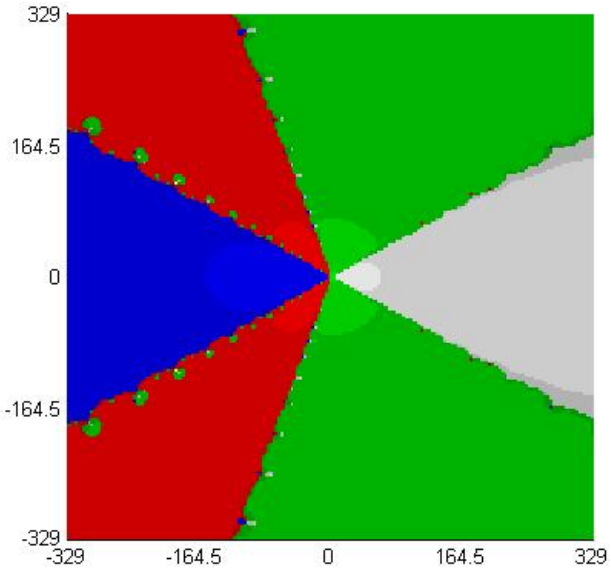
```
                    W(j,m,:)=color_raiz_nuevo5{ind};
                end
            else
                if it>= velocidad(1) && it< velocidad(2)
                    W(j,m,:)=color_raiz_nuevo{c_color};
                elseif it>= velocidad(2) && it< velocidad(3)
                    W(j,m,:)=color_raiz_nuevo2{c_color};
                elseif it>= velocidad(3) && it< velocidad(4)
                    W(j,m,:)=color_raiz_nuevo3{c_color};
                elseif it>= velocidad(4) && it< velocidad(5)
                    W(j,m,:)=color_raiz_nuevo4{c_color};
                elseif it>= velocidad(5) && it<velocidad(6)
                    W(j,m,:)=color_raiz_nuevo5{c_color};
                end
                raices=[raices,xc];
                c_color=c_color+1;
            end
        end
    end
end


% Guardamos las raíces del polinomio

save raices.mat raices;


% Representamos el resultado final

hold on;
axes(ventana.axes1);


% Configuramos los parámetros de los ejes

xlab=[-M 0 M];
ylab=[-M 0 M];
xlabmax=[-M (-M/2) 0 M/2 M];
ylabmax=[-M (-M/2) 0 M/2 M];
dimx=[1 length(x)/2 length(x)];
dimy=[1 length(y)/2 length(y)];
```

```
dimxmax=[1 length(x)/4 length(x)/2 3*length(x)/4 length(x)];
dimymax=[1 length(y)/4 length(y)/2 3*length(y)/4 length(y)];
heatmap(W,x,y);


% Eje x

if length(x) > 20
    set(gca,'XTick',dimxmax,'XTickLabel',xlabmax);
elseif length(x) > 8
    set(gca,'XTick',dimx,'XTickLabel',xlab);
else
    set(gca,'XTickLabel',x);
end


% Eje y

if length(y) > 20
    set(gca,'YTick',dimymax,'YTickLabel',ylabmax);
elseif length(y) > 8
    set(gca,'YTick',dimy,'YTickLabel',ylab);
else
    set(gca,'YTickLabel',y);
end
axis image;
shading interp;
hold off;
```

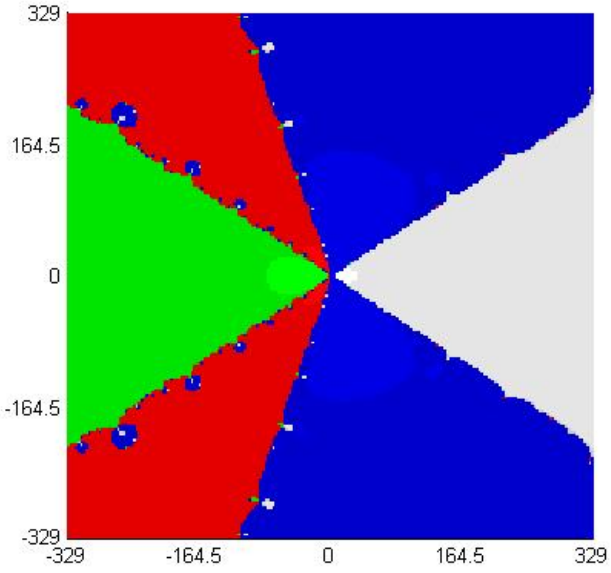
3.12.2. Resultados gráficos de la tercera versión

Para observar los resultados de la tercera versión de las representaciones gráficas de las cuencas de atracción, se ha elegido el mismo polinomio que en la anterior versión para que de esta manera se puedan observar con mayor facilidad las diferentes tonalidades del color que indican la velocidad.

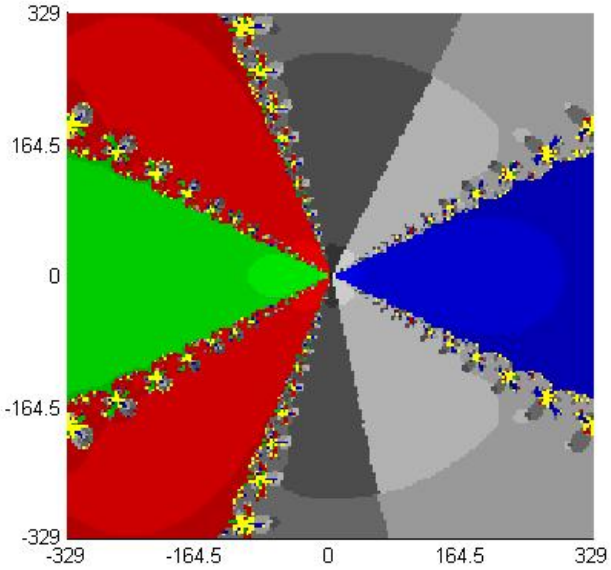
Como se observa en la figura 3.5, los resultados obtenidos al calcular el polinomio con cada uno de los métodos anteriormente explicados son los siguientes:



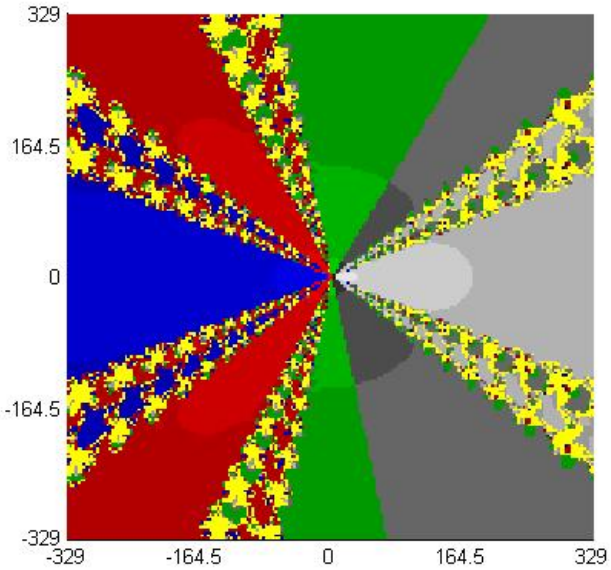
(a) Método de Newton



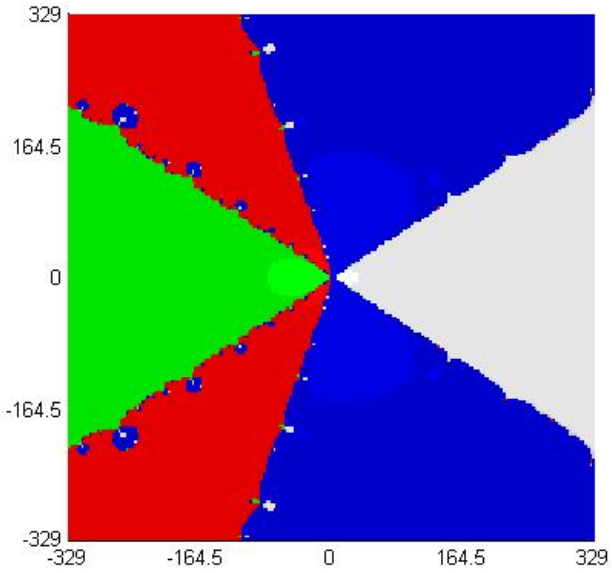
(b) Método de Halley



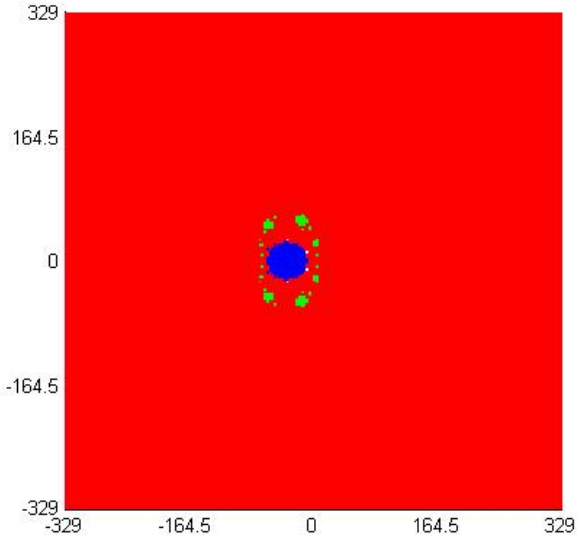
(c) Método de Schröder con orden 3



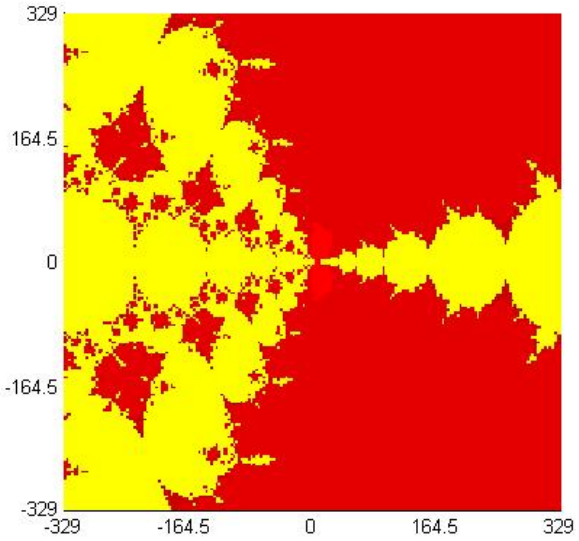
(d) Método de Schröder con orden 4



(e) Método de König cono orden de derivación = 2



(f) Método de Newton Raíces Múltiples



(g) Método de Newton Relajado

Figura 3.5: Representación de las cuencas de atracción según la tercera versión de la función $x^5-11x^4+27x^3+71x^2-328x+240$, con $n = 50$ iteraciones, tolerancia $= 10^{-5}$, 200 particiones de los ejes de abscisas y coordenadas.

Se observa claramente que al aplicar el programa sobre la misma función pero cambiando los parámetros para que sea mas exacta, en este caso aumentando el número de puntos sobre la matriz resultante, los cambios de colores respectivos a la velocidad de convergencia a cada raíz se ven con mejor claridad y facilita así su estudio sobre la atracción de las diferentes cuencas de atracción de la función dada.

3.13. Otros programas utilizados para la realización de las versiones

3.13.1. str2pol.m

La función *str2pol.m* se utiliza para convertir una cadena de carácteres que contiene un polinomio en un polinomio numérico que pueda ser empleado por *Matlab* para realizar las operaciones pertinentes.

```
function p=str2pol(f)


% Esta función convierte una cadena de carácteres que contiene un
% polinomio a un polinomio numérico
% p=str2pol(f)
% Variables de entrada:
% f - cadena de carácteres que contiene la expresión del polinomio
% Variables de salida:
% p - vector que contiene los coeficientes numéricos del polinomio
% ejemplo:
% str2pol('x^5 + 5 * x^3')

% Longitud de f

nf=length(f);


% Encontramos la posición en la cadena del caracter '+'y del caracter
% '-'

[f1,ind1]=find(f=='+' | f=='-');
ni=length(ind1);
```

```
% Componemos en p la cadena de caracteres conteniendo solo los coeficientes

if ind1(1)==1

% En este caso hay ni terminos

    n_ter=ni;

% Llamamos al método arreglar_monomio para rellenar el array p

    [p,g]=arreglar_monomio(f(ind1(1):ind1(2)-1));
    for i=2:ni-1
        [p_aux,g_aux]=arreglar_monomio(f(ind1(i):ind1(i+1)-1));
        aux=num2str(zeros(1,g-g_aux-1));
        p=[p,blanks(1),aux,blanks(1),p_aux];
        g=g_aux;
    end
    [p_aux,g_aux]=arreglar_monomio(f(ind1(ni):nf));
    aux1=num2str(zeros(1,g-g_aux-1));
    aux2=num2str(zeros(1,g_aux));
    p=[p,blanks(1),aux1,blanks(1),p_aux,blanks(1),aux2];

else
    [p,g]=arreglar_monomio(f(1:ind1(1)-1));
    for i=1:ni-1
        [p_aux,g_aux]=arreglar_monomio(f(ind1(i):ind1(i+1)-1));
        aux=num2str(zeros(1,g-g_aux-1));
        p=[p,blanks(1),aux,blanks(1),p_aux];
        g=g_aux;
    end
    [p_aux,g_aux]=arreglar_monomio(f(ind1(ni):nf));
    aux1=num2str(zeros(1,g-g_aux-1));
    aux2=num2str(zeros(1,g_aux));
    p=[p,blanks(1),aux1,blanks(1),p_aux,blanks(1),aux2];
end
```

```
% Convertimos los términos del vector p que provienen del método
% arreglar_monomio en datos válidos para usar en Matlab

p=str2num(p);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [h1,g]=arreglar_monomio(h)

% Función que ordena los monomios para que el programa los reconozca
% correctamente
% [h1,g]=arreglar_monomio(h)
% Variables de entrada:
% h - monomio que se desea arreglar
% Variables de salida:
% h1 - coeficiente del monomio
% g - grado del monomio

% Buscamos la posición de los comandos 'x'y '*'para proceder a ordenar
% el monomio

pos_prod=strfind(h,'*');
pos_x=strfind(h,'x');

% Calculamos el grado y el coeficiente del monomio

pos_pot=strfind(h,'^');
nh=length(h);
if isempty(pos_pot)
    if isempty(pos_x)
        g=0;
        h1=h;
    else
```

```
        g=1;
        if isempty(pos_prod)
           h1='1';
        else
           h1=h(1:pos_prod-1);
        end
    end
else
    g=str2num(h(pos_pot+1:nh))
    if isempty(pos_prod)
       h1='1';
    else
       h1=h(1:pos_prod-1);
    end
end
```

4 Sistemas de Funciones Iteradas (IFS)

Un método para generar imagenes fractales es el *Sistemas de Funciones Iteradas*, IFS, desarrollado en los años 80 fundamentalmente por J. E. Hutchinson y M. Barnsley basándose en el principio de autosemejanza. Se empleó estos sistemas para modelizar objetos que a diferentes escalas presentan una relación de semejanza con la figura completa.

4.1. Elementos de Topología

Para comprender este método, son necesarios algunos conceptos básicos de topología. Principalmente trabajaremos en el espacio euclidiano $\mathbb{R}^n$.
Todo lo que haremos es válido en espacios métricos.

Primero recordemos una serie de conceptos en espacios métricos.
Una métrica en un conjunto X es una función $\rho : X \times X \to \mathbb{R}$, que satisface:

1. $\rho(x, y) \geq 0, \quad x, y \in X,$
2. $\rho(x, y) = 0$ si y sólo si $x = y,$
3. $\rho(x, y) = \rho(y, x), \quad x, y \in X$ (simetría), y
4. $\rho(x, y) \leq \rho(x, z) + \rho(z, y), \quad x, y, z \in X$ (desigualdad triangular)

Consideremos un espacio métrico (X, ρ). Las siguientes propiedades y definiciones son usuales en topología:

1. El *diámetro* de un conjunto $A \subset X$ es $|A| = sup\{\rho(x, y) : x, y \in A\}$.

2. Decimos que una sucesión $(x_n)_{n\in\mathbb{N}}, x_n \in X$, converge a un punto x si, para cada $\epsilon > 0$ dado, existe $n_0 \in \mathbb{N}$ tal que

$$\rho(x_n, x) < \epsilon, \quad \text{cuando} \quad n \geq n_0.$$

   Usamos la notación $x = \lim_{n\to\infty} x_n$.

3. Un subconjunto $C \subset X$ es *cerrado* si para cada sucesión convergente $(x_n)_{n\in\mathbb{N}}$, con $x_n \in C$, se tiene $\lim_{n\to\infty} x_n \in C$.

4. Decimos que un subconjunto $K \subset X$ es *compacto* si cada sucesión $(x_n)_{n\in\mathbb{N}}$, con $x_n \in K$ posee una subsucesión convergente.

5. Dado $x \in X$, la bola abierta de centro en x y radio $r > 0$ es el conjunto $B_r(x) = \{y \in X : \rho(x, y) < r\}$.

6. Decimos que un subconjunto $A \subset \mathbb{R}^n$ es *acotado* si existe $r > 0$ tal que $||x|| < r$ para todo $x \in A$, es decir, A está contenido en una bola abierta $B_r(0)$ de radio $r > 0$ suficientemente grande.

   Los conjuntos compactos en $\mathbb{R}^n$ son caracterizados por la propiedad siguiente, la cual no es válida en espacios métricos arbitrarios.

   Un subconjunto $K \subset \mathbb{R}^n$ es compacto si, y sólo si, es cerrado y acotado.

7. Una sucesión $(x_n)_{n\in\mathbb{N}}$ en un espacio métrico (X, ρ) es una *sucesión de Cauchy* si, para cada $\epsilon > 0$ dado, existe $n_0 \in \mathbb{N}$ tal que $\rho(x_n, x_m) < \epsilon$, cuando $n, m \geq n_0$.

8. Un espacio métrico (X, ρ) es *completo* si, cada sucesión de Cauchy en X es convergente.

9. Sea $A \subset X$. Un punto $x \in X$ es *punto de acumulación* de A si, para todo $\epsilon > 0$, $(B_\epsilon(x) - \{x\}) \cap A \neq \emptyset$. El conjunto de los puntos de acumulación de A se denota por A' y es llamado el *conjunto derivado* de A. Por otro lado, $x \in A$ es un *punto aislado* de A si existe $\epsilon > 0$, tal que $B_\epsilon(x) \cap A = \{x\}$.

10. Un conjunto $A \subset X$ es un *conjunto perfecto* si todos sus puntos son de acumulación.

11. Un punto $x \in X$ es un *punto adherente* de A si, para cada $\epsilon > 0$ dado, existe $a \in A$ tal que $\rho(x, a) < \epsilon$.

    La *clausura*, $\overline{A}$, de $A \subset X$ es el conjunto de puntos adherentes a A. Decimos que $A \subset X$ es *denso* en X si $\overline{A} = X$.

12. Sea $A \subset X$. Decimos que $x \in X$ es un punto *frontera* de A, si para cada $\epsilon > 0$, $B_\epsilon(x) \cap A \neq \emptyset$ y $B_\epsilon(x) \cap (X - A) \neq \emptyset$. El conjunto de puntos frontera de A es denotado por ∂A.

4.2. Aplicaciones Contractivas y Teorema del Punto Fijo

El concepto básico en esta sección es el de *aplicación contractiva* o simplemente *contracción*, y el resultado fundamental es el teorema que garantiza que cada contracción en un espacio métrico completo tiene un (único) punto fijo.

Definición: *Una transformación $f : X \to X$ es una contración si existe un número, $0 \leq s < 1$, tal que para cada $x, y \in X$ se tiene $d(f(x), f(y)) \leq s\ d(x, y)$ (s es llamado factor de contractividad de f). Además,*

$$s = \sup_{\substack{x,y \in X \\ x \neq y}} \frac{d(f(x), f(y))}{d(x, y)}.$$

Notemos que cada contracción es una aplicación continua. Aplicando reiteradas veces la desigualdad $d(f(x), f(y)) \leq s\ d(x, y)$, obtenemos

$$\begin{array}{ccccc} d(f^2(x), f^2(y)) & \leq & sd(f(x), f(y)) & \leq & s^2 d(x, y), \\ d(f^3(x), f^3(y)) & \leq & sd(f^2(x), f^2(y)) & \leq & s^3 d(x, y), \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ d(f^n(x), f^n(y)) & \leq & sd(f^{n-1}(x), f^{n-1}(y)) & \leq & s^n d(x, y), \end{array}$$

como $s^n \to 0$ cuando $n \to \infty$ tenemos el siguiente

Teorema: *Si X es un espacio métrico completo y $f : X \to X$ es una contracción. Entonces*

$$x_f = \lim_{n \to \infty} f^n(x)$$

existe y es independiente de la elección de $x \in X$. Además, x_f es el único punto fijo de f.

Demostración: *Dado $x \in X$ definamos la sucesión $(x_n)_{n\in\mathbb{N}}$, como sigue: sea $x_0 = x$, $x_1 = f(x)$, $x_2 = f(f(x)) = f^2(x)$, ..., $x_n = f \circ f \circ \cdots \circ f(x) = f^n(x)$. Sea s el factor de contracción de f. Entonces*

$$\begin{aligned} d(x_{k+1}, x_k) &= d(f(x_k), f(x_{k-1})) \\ &\leq sd(x_k, x_{k-1}) \\ &\leq s^2 d(x_{k-1}, x_{k-2}) \\ &\vdots \\ &\leq s^k d(x_1, x_0), \end{aligned}$$

luego,

$$\begin{aligned} d(x_{k+\ell}, x_k) &\leq \sum_{i=0}^{\ell-1} d(x_{k+i+1}, x_{k+i}) \\ &\leq \sum_{i=0}^{\ell-1} s^{k+i} d(x_1, x_0) \\ &\leq s^k d(x_0, x_1) \sum_{i=0}^{\infty} s^i \\ &\leq \tfrac{s^k}{1-s} d(x_1, x_0). \end{aligned}$$

Como $0 < s < 1$ se tiene que $\lim_{k\to\infty} s^k = 0$, de donde se sigue que $lim_{k\to\infty} d(x_{k+\ell}, x_k) = 0$, esto es, la sucesión $(x_k)_{k\in\mathbb{N}}$ es de Cauchy en X, por lo tanto convergente. Sea $x_f = \lim_{k\to\infty} x_k$. Tenemos entonces que $x_f \in X$. Como f es continua, $f(x_f) = f(\lim_{k\to\infty} x_k) = \lim_{k\to\infty} f(x_k) = \lim_{k\to\infty} x_{k+1}$ $= x_f$. Para mostrar la unicidad de x_f, supongamos que existe $a \in X$ con $f(a) = a$. Entonces $d(x_f, a) = d(f(x_f), f(a)) \leq sd(x_f, a)$ y como $0 < s < 1$ se concluye que $d(x_f, a) = 0$, de donde $x_f = a$, lo que completa la prueba del teorema.

4.3. Contracciones Lineales

Con el fin de obtener un generador de *imágenes fractales*, trabajaremos con aplicaciones afines del plano $\mathbb{R}^2$ en si mismo. Esto es por la facilidad de implementación computacional que este tipo de transformaciones presenta. Por lo tanto dedicamos esta sección al estudio del problema de saber cuándo

una aplicación afín (composición de una transformación lineal con una traslación) es una contracción.

Si $L : \mathbb{R}^2 \to \mathbb{R}^2$ es una transformación lineal, entonces $L(x, y) = (ax + by, cx + dy)$. El problema que tratamos ahora de resolver es saber cuándo L es una contracción. Enseguida usaremos transformaciones afines, $A(x, y) = (ax + by + e, cx + dy + f)$, (lineal más una tralación). En este caso, componiendo A con una isometría I, es decir, una transformación lineal de $\mathbb{R}^2$ que preserva distancias, es decir, $d(I(x), I(y)) = d(x, y)$). Se tiene que $d(I \circ A(u), I \circ A(v)) = d(A(u), A(v))$, y podemos elegir $I(x, y) = (x-e, y-f)$, con lo cual obtenemos $I \circ A(0, 0) = (0, 0)$.

El caso fácil de analizar es cuando $b = c = 0$, es decir, $L(x, y) = (ax, dy)$. Tenemos $L(1, 0) = a(1, 0)$ y $L(0, 1) = d(0, 1)$, por lo tanto L tiene factores de escalamiento separados a lo largo del eje x y del eje y. Luego

$$||L(x_1, y_1)-L(x_2, y_2)|| = \sqrt{a^2(x_1 - x_2)^2 + d^2(y_1 - y_2)^2} \leq max\{|a|, |d|\}||(x_1-x_2, y_1-y_2)||.$$

Por lo tanto, si ambos a y d tienen valor absoluto menor que 1, el factor de contractividad de L es $max\{|a|, |d|\}$.

Para el caso general, sean $v_1 = (x_1, y_1)$, $v_2 = (x_2, y_2)$. Usando notación matricial tenemos

$$\begin{aligned} ||v_1 - v_2||^2 &= (x_1 - x_2)^2 + (y_1 - y_2)^2 \\ &= \begin{pmatrix} x_1 - x_2 & y_1 - y_2 \end{pmatrix} \begin{pmatrix} x_1 - x_2 \\ y_1 - y_2 \end{pmatrix} \\ &= \begin{pmatrix} x_1 - x_2 \\ y_1 - y_2 \end{pmatrix}^T \begin{pmatrix} x_1 - x_2 \\ y_1 - y_2 \end{pmatrix} \end{aligned}$$

luego,

$$||L(v_1)-L(v_2)||^2 = [L(v_1)-L(v_2)]^T[L(v_1)-L(v_2)] = [v_1-v_2]^T A^T A[v_1-v_2].$$

Queremos encontrar el máximo de

$$\frac{[v_1 - v_2]^T A^T A[v_1 - v_2]}{[v_1 - v_2]^T[v_1 - v_2]}.$$

Ahora, $A^T\ A$ es simétrica y tiene la propiedad $w^T\ A^T\ Aw = ||Aw||^2 \geq 0$ para todo vector w. Supongamos que $v_1 - v_2 = \begin{pmatrix} u \\ v \end{pmatrix}$ y que $u^2 + v^2 = 1$.

Entonces queremos el máximo de

$$(u \quad v)\begin{pmatrix} a & c \\ b & d \end{pmatrix}\begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} u \\ v \end{pmatrix} = F(u,v)$$

sujeto a la condición $u^2 + v^2 = 1$. Desarrollando obtenemos

$$F(u,v) = (au + bv)^2 + (cu + dv)^2.$$

Usando multiplicadores de Lagrange, cada punto extremo de F sujeto a la condición $u^2 + v^2 = 1$ debe satisfacer las siguientes ecuaciones

$$\frac{\partial}{\partial u}(F(u,v) + \lambda(1 - u^2 - v^2)) = 0$$

$$\frac{\partial}{\partial v}(F(u,v) + \lambda(1 - u^2 - v^2)) = 0$$

desarrollando estas nos queda

$$2(a^2 + c^2)u + 2(ab + cd)v - 2u\lambda = 0$$

$$2(ab + cd)u + 2(b^2 + d^2)v - 2v\lambda = 0$$

que matricialmente puede ser escrito en la forma

$$A^T A\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} a & c \\ b & d \end{pmatrix}\begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} u \\ v \end{pmatrix} = \lambda\begin{pmatrix} u \\ v \end{pmatrix}$$

es decir, $\begin{pmatrix} u \\ v \end{pmatrix}$ es un vector propio de $A^T A$ y el número λ es el valor propio asociado. Ahora, es fácil ver que el factor de escalamiento en la solución $\begin{pmatrix} u \\ v \end{pmatrix}$ es $\sqrt{|\lambda|}$. Luego, podemos determinar si L es o no una contracción calculando los valores propios de $A^T A$.

4.4. Sistemas de Funciones Iteradas

En lo que sigue trabajaremos en $\mathbb{R}^n$, pero todo lo que hagamos es válido en espacios métricos completos.

Consideremos el conjunto

$$\mathcal{K}(\mathbb{R}^n) = \{K \subset \mathbb{R}^n : K \quad \textit{compacto no vacío}\}.$$

Dotamos a este conjunto con la *métrica de Hausdorff*. Antes de definirla, introduzcamos la siguiente notación: dados $A \subset \mathbb{R}^n$ y $\varepsilon > 0$, sea A_ε el conjunto $A_\varepsilon = \{y \in \mathbb{R}^n : \quad \textit{existe} \quad x \in A \quad \textit{con} \quad d(x,y) < \varepsilon\}$, donde d es la métrica usual en $\mathbb{R}^n$. Este conjunto A_ε es llamado una ε-vecindad de A.

Ahora definimos la métrica de Hausdorff sobre $\mathcal{K}(\mathbb{R}^n)$ como

$$H(K,L) = inf\{\varepsilon > 0 : K \subset L_\varepsilon \quad y \quad L \subset K_\varepsilon\}.$$

Equivalentemente, la métrica de Hausdorff puede ser definida como

$$H(K,L) = max\{D(K,L), D(L,K)\}.$$

donde para $A, B \subset \mathbb{R}^n$, se define $D(A,B)$ como

$$D(A,B) = \sup_{x \in A} \inf_{y \in B} d(x,y).$$

Proposición : *H es una métrica en $\mathcal{K}(\mathbb{R}^n)$.*

Demostración : *Claramente, $H(K,L) \geq 0$ y $H(K,L) = H(L,K)$. Si $K = L$, para cada $\varepsilon > 0$, $K \subset L_\varepsilon$ y $L \subset K_\varepsilon$, por lo tanto $H(K,L) = 0$. Recíprocamente, si $H(K,L) = 0$, para cada $x \in K$ y cada $\varepsilon > 0$, tenemos $x \in L_\varepsilon$ y $dist(x,L) = \text{ínf}\{d(x,y) : y \in L\} = 0$. Como L es compacto, se tiene que $x \in L$ y por lo tanto $K \subset L$. En forma análoga se prueba que $L \subset K$. Por lo tanto $K = L$.*

Para la desigualdad triangular, sean $K, L, M \in \mathcal{K}(\mathbb{R}^n)$ y $\varepsilon > 0$. Si $x \in K$, existe $y \in L$ tal que $d(x,y) < H(K,L) + \varepsilon$. Análogamente, existe $z \in M$ con $d(y,z) < H(L,M) + \varepsilon$, es decir, K está contenido en la $(H(K,L) + H(L,M) + 2\varepsilon)$-vecindad de M. En forma análoga se prueba que M está contenido en la $(H(K,L)+H(L,M)+2\varepsilon)$-vecindad de K. Por lo tanto $H(K,M) \leq H(K,L) + H(L,M)$.

Proposición : *$(\mathcal{K}(\mathbb{R}^n), H)$ es un espacio métrico completo.*

Demostración : *Sea $(K_j)_{j\in\mathbb{N}}$ una sucesión de Cauchy en $\mathcal{K}(\mathbb{R}^n)$. Definamos el conjunto*

$$K = \{x \in \mathbb{R}^n : \quad \text{existe una sucesión} \quad (x_j)_{j\in\mathbb{N}} \quad \text{con} \quad x_j \in K_j \quad y \quad \lim_{j\to\infty} x_j = x\}.$$

Afirmación : $\lim_{j\to\infty} K_j = K$, *donde el límite es tomado respecto de la métrica de Hausdorff.*

Sea $\varepsilon > 0$ dado. Entonces existe $N \in \mathbb{N}$ tal que $j, \ell \geq N$ implica $H(K_j, K_\ell) < \varepsilon/2$. Sea $j \geq N$. Afirmamos que $H(K, K_j) < \varepsilon$.

En efecto, si $x \in K$, existe una sucesión $(x_k)_{k\in\mathbb{N}}$, con $x_k \in K_k$ y $\lim_{k\to\infty} x_k = x$. Luego, para k suficientemente grande, $d(x_k, x) < \varepsilon/2$. Por lo tanto si $k \geq N$, existe $y \in K_j$ con $d(x_k, y) < \varepsilon/2$, pues $H(K_j, K_k) < \varepsilon/2$, y tenemos $d(y,x) \leq d(y, x_k) + d(x_k, x) \leq \varepsilon$, esto es, $K \subset (K_n)_\varepsilon$.

Supongamos ahora que $y \in K_j$. Elijamos enteros $k_1 < k_2 < \cdots$ tales que $k_1 = n$ y $H(K_{k_i}, K_m) < 2^{-i}\varepsilon$ para todo $m \geq k_i$. Definamos una sucesión $(y_k)_{k\in\mathbb{N}}$ con $y_k \in K_k$ como sigue: para $k < j$, podemos elegir $y_k \in K_k$ arbitrariamente, tomamos por lo tanto $y_k = y$. Para $k_i < k < k_{i+1}$, elegimos $y_k \in K_k$ con $d(y_{k_i}, y_k) < 2^{-i}\varepsilon$. Es claro que con estas elecciones, $(y_k)_{k\in\mathbb{N}}$ es una sucesión de Cauchy en $\mathbb{R}^2$, por lo tanto converge. Sea $x = \lim_{k\to\infty} y_k$. Tenemos entonces que $x \in K$. Ahora, como $d(y,x) = \lim_{k\to\infty} d(y, y_k) < \varepsilon$, se sigue que $y \in K_\varepsilon$. Esto prueba que $K_j \subset K_\varepsilon$, por lo tanto, $H(K, K_j) \leq \varepsilon$, es decir, $(K_n)_{n\in\mathbb{N}}$ converge a K en la métrica de Hausdorff.

Como caso especial, tenemos la siguiente

Proposición : *Sea $(K_j)_{j\in\mathbb{N}}$ una sucesión en $\mathcal{K}(\mathbb{R}^n)$. Supongamos que $K_1 \supset K_2 \supset \cdots$. Entonces $(K_j)_{j\in\mathbb{N}}$ converge en la métrica de Hausdorff a $K = \cap_{j\in\mathbb{N}} K_j$.*

Demostración : *Inmediata*

Sean $w_1, \ldots, w_\ell : \mathbb{R}^n \to \mathbb{R}^n$, contracciones con factor de contracción $r_i (i = 1, \ldots, \ell)$, es decir, $d(w_i(x), w_i(y)) \leq r_i d(x, y)$ y definamos la aplicación

$$W = \mathcal{K}(\mathbb{R}^n) \to \mathcal{K}(\mathbb{R}^n), \quad W(K) = \cup_{i=1}^{\ell} w_i(K)$$

W es llamado *operador de Hutchison.*

Es claro que W está bién definida, pues por la Proposición anterior la imagen de un conjunto compacto por una aplicación continua es un conjunto compacto, y unión finita de conjuntos compactos es un conjunto compacto, como se verifica facilmente.

La colección de pares $\mathcal{W} = \{(w_i, r_i) : i = 1, \ldots, \ell\}$, donde w_i es contracción con factor de contracción (razón) $0 \leq r_i < 1$ en $\mathbb{R}^n$ (o más general en un espacio métrico (X, ρ)) es llamado un *Sistemas de Funciones Iteradas* y usamos la abreviación, IFS (del inglés Iterated Functions System).

Teorema : *La aplicación W definida arriba es una contracción en el espacio métrico $(\mathcal{K}(\mathbb{R}^n), H)$.*

Demostración : *Sea $r = max\{r_i : i = 1, \ldots, \ell\}$; es claro entonces que $0 \leq r < 1$. Sean $K, L \in \mathcal{K}(\mathbb{R}^n)$. Vamos a probar que $H(W(K), W(L)) \leq rH(K, L)$.*

Sea $q > H(K, L)$. Si $x \in W(K)$, entonces $x = w_i(x')$ para algún $i = 1, \ldots, \ell$ y algún $x' \in K$. Como $q > H(K, L)$, existe $y' \in L$ con $d(x', y') < q$. Luego $y = w_i(y') \in W(L)$ satisface $d(x, y) \leq r_i d(x', y') \leq rq$. Esto es verdadero para todo $x \in W(K)$, por lo tanto $W(K)$ está contenido en L_{rq}. Análogamente vemos que $W(L)$ está contenido en K_{rq}, esto es, $H(W(K), W(L)) \leq rq$, y como esto vale para $q > H(K, L)$, obtenemos que $H(W(K), W(L)) \leq rH(K, L)$.

Definición : *Sea $f : X \to X$. Decimos que un conjunto $A \subset X$ es invariante por f si $f(A) = A$.*

Note que si $A \subset X$ es invariante por f, entonces $f^n(A) = A$, para todo $n \in \mathbb{N}$.

Nota : Por lo probado arriba, el factor de contracción de W en la métrica de Hausdorff de $\mathcal{K}(\mathbb{R}^n)$ es $r = max\{r_i : i = 1, \ldots, \ell\} < 1$. Tenemos entonces el siguiente

Corolario : *Sea $K_0 \in \mathcal{K}(\mathbb{R}^n)$. Defina la sucesión $(K_m)_{m\in\mathbb{N}}$ en $\mathcal{K}(\mathbb{R}^n)$ como sigue*

$$K_{m+1} = W(K_m) = \cup_{i=1}^{\ell} w_i(K_m) = W^{m+1}(K_0).$$

Entonces la sucesión $(K_m)_{m\in\mathbb{N}}$ converge en la métrica de Hausdorff al único conjunto invariante $K \in \mathcal{K}(\mathbb{R}^n)$ del IFS. Además, K satisface

$$K = W(K) = \cup_{i=1}^{\ell} w_i(K)$$

(ecuación de autosimiliridad), esto es, K es un punto fijo de W y está formado por la unión de sus imágenes por las aplicaciones $w_i, i = 1, 2, \ldots, \ell$. Además K es caracterizado por

$$K = \lim_{m\to\infty} K_m = \lim_{m\to\infty} W^{m+1}(K_0).$$

Equivalentemente, $\lim_{m\to\infty} H(K_m, K) = 0$. *El conjunto K es llamado atractor del IFS, $\mathcal{W} = \{(w_i, r_i) : i = 1, \ldots, \ell\}$.*

Demostración : *Sigue inmediatamente de los Teoremas*

La ecuación de autosimilaridad proporciona el carácter de autosimilar a K y la ecuación que le sigue nos dice que para generar computacionalmente imagenes a partir de un IFS, basta tomar cualquier conjunto compacto K_0 en $\mathbb{R}^n$ y calcular las sucesivas iteraciones de K_0 por la aplicación W. En general, se toma $K_0 = \{punto\}$, pues corresponde al más simple de los compactos no vacíos de $\mathbb{R}^n$ y formamos los conjuntos $K_j \subset \mathbb{R}^n$, dados por

$$K_j = W^j(K_0) = W(W^{j-1}(K_0)), \quad j \geq 1,$$

donde $W^0 = Id$ es la aplicación identidad. Esto define una sucesión creciente de conjuntos compactos que convergen, en la métrica de Hausdorff, al atractor K del IFS. La ecuación anterior corresponde al más simple de los algorítmos para generar imagenes a partir de un IFS.

En general se usan contracciones afines, esto es, transformaciones de la forma $L : \mathbb{R}^n \to \mathbb{R}^n, L(x) = Ax + b$, donde A es una matriz de orden $n \times n$ y $b \in \mathbb{R}^n$ es un vector de traslación. Para que L sea contracción es suficiente que la norma, $||A||$, de A

$$\begin{aligned} ||A|| &= \sup\{||Ax|| : x \in \mathbb{R}^n \quad \text{con} \quad ||x|| = 1\} \\ &= \sup\{||Ax|| : x \in \mathbb{R}^n \quad \text{con} \quad ||x|| \leq 1\} \end{aligned}$$

sea menor que 1.

La razón para tomar transformaciones afines es la facilidad de cálculo y de programación que estas ofrecen, pero se pueden usar contracciones cualesquiera.

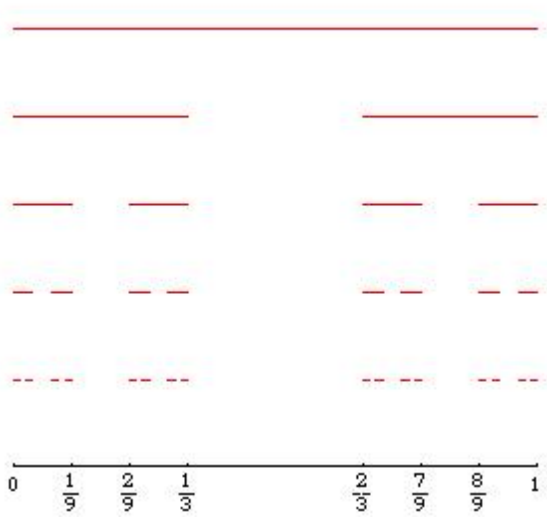
Ejemplos :

Conjunto de Cantor

Sea $X = \mathbb{R}$ con la distancia usual. Consideremos las transformaciones afines, $w_1(x) = \frac{x}{3}$ y $w_2(x) = \frac{x}{3} + \frac{2}{3}$. Sea $K \subseteq \mathbb{R}$ un compacto no vacío. Entonces

$$\begin{aligned} W(K) &= \tfrac{1}{3}K \cup \left(\tfrac{1}{3}K + \tfrac{2}{3}\right) \\ W^2(K) &= \tfrac{1}{9}K \cup \left(\tfrac{1}{9}K + \tfrac{2}{9}\right) \cup \left(\tfrac{1}{9}K + \tfrac{2}{3}\right) \cup \left(\tfrac{1}{9}K + \tfrac{8}{9}\right), \\ &\vdots \\ W^{n+1}(K) &= \left(\tfrac{1}{3}W^n(K)\right) \cup \left(\tfrac{1}{3}W^n(K) + \tfrac{2}{3}\right). \end{aligned}$$

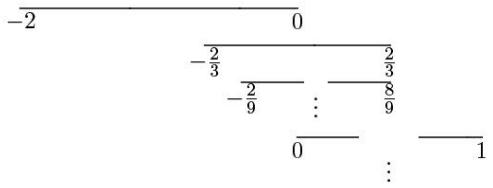
Por ejemplo, tomando $K = [0, 1]$, nos queda



Ahora, si tomamos $K = [-2, 0]$, nos queda

$$\begin{aligned} W(K) &= \left[-\tfrac{2}{3}, 0\right] \cup \left[0, \tfrac{2}{3}\right] \\ W^2(K) &= \left[-\tfrac{2}{9}, 0\right] \cup \left[0, \tfrac{2}{9},\right] \cup \left[-\tfrac{4}{9}, \tfrac{2}{3}\right] \cup \left[-\tfrac{2}{3}, \tfrac{8}{9}\right] \\ &\vdots \end{aligned}$$

y se tiene la convergencia de $W^n(K)$ al conjunto de Cantor clásico.



Para $0 < \alpha < 1/2$, sea $K \subset \mathbb{R}^2$ el atractor del IFS definido por las cuatro contracciones siguientes

$$\psi_1(x, y) = \begin{pmatrix} \alpha & 0 \\ 0 & \alpha \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

$$\psi_2(x, y) = \begin{pmatrix} \alpha & 0 \\ 0 & \alpha \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 1-\alpha \\ 0 \end{pmatrix}$$

$$\psi_3(x, y) = \begin{pmatrix} \alpha & 0 \\ 0 & \alpha \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0 \\ 1-\alpha \end{pmatrix}$$

$$\psi_4(x, y) = \begin{pmatrix} \alpha & 0 \\ 0 & \alpha \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 1-\alpha \\ 1-\alpha \end{pmatrix}.$$

El radio de contracción de cada una de estas transformaciones afines es α, con $0 < \alpha \leq 1/2$. Para $\alpha = 1/3$, K es el producto del conjunto de Cantor clásico consigo mismo.

Triángulo de Sierpinski

Comencemos con un triángulo T, por ejemplo, el triángulo equilátero con vértices en $v_1 = (0, 0)$, $v_2 = (2, 0)$ y $v_3 = 1 + \sqrt{3}i = (1, \sqrt{3})$. Los puntos medios de los tres lados son $u_3 = 1$, $u_2 = \frac{1+\sqrt{3}i}{2}$ y $u_1 = \frac{3+\sqrt{3}i}{2}$. Existen tres

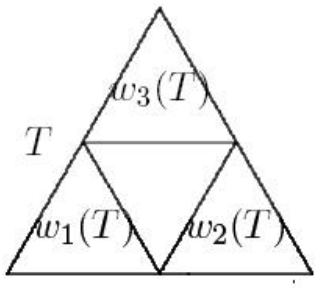
triángulos equiláteros en cada vértice v_j, T_1 con vértices $\{v_1, u_2, u_3\}$, T_2 con vértices $\{v_2, u_1, u_3\}$ y T_3 con vértices $\{v_3, u_1, u_2\}$. Definimos tres transformaciones afines, $\{w_1, w_2, w_3\}$ con las condiciones $w_1(T) = T_1$, $w_2(T) = t_2$ y $w_3(T) = T_3$, que fijan los vértices v_j y preservan la orientación. Por ejemplo, $w_1(v_1) = v_1$, $w_1(v_2) = u_3$ y $w_1(v_3) = u_2$. Cada w_j tiene factor de escalamiento $\frac{1}{2}$, y las fórmulas para cada una de ellas son: $w_1(z) = \frac{1}{2}z$, $w_2(z) = \frac{1}{2}z + 1$ y $w_3(z) = \frac{1}{2}z + \frac{1+\sqrt{3}i}{2}$, donde $z \in \mathbb{C}$. Matricialmente,

$$w_1(x, y) = \begin{pmatrix} 0{,}5 & 0 \\ 0 & 0{,}5 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

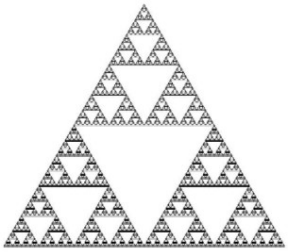
$$w_2(x, y) = \begin{pmatrix} 0{,}5 & 0 \\ 0 & 0{,}5 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$w_3(x, y) = \begin{pmatrix} 0{,}5 & 0 \\ 0 & 0{,}5 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0{,}5 \\ 0{,}866 \end{pmatrix}$$

La siguiente figura muestra la primera etapa de la construcción del triángulo de Sierpinski, T, y sus imágenes por las transformaciones w_1, w_2 y w_3

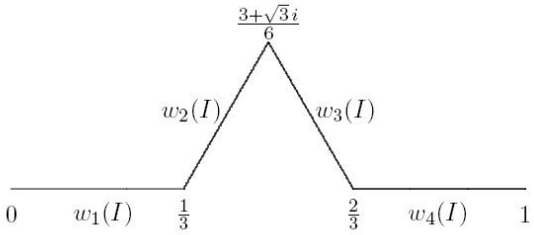


El triángulo de Sierpinski es el atractor del IFS, $\{w_1, w_2, w_3\}$ y una imagen de este se muestra en la figura



Curva de Koch

Consideremos las transformaciones afines $w_j(z) = a_j z + b_j$, donde $z \in \mathbb{C}$ y $a_j, b_j \in \mathbb{C}$, con $j \in \{1, 2, 3, 4\}$ son constantes. Queremos transformar un intervalo unitario $I = [0, 1]$ en la siguiente figura mediante las transformaciones w_j



Conociendo la localización de las imágenes de dos puntos podemos determinar a_j y b_j. A partir de la figura tenemos

$$\begin{aligned} w_1(0) &= b_1 = 0, & w_2(0) &= b_2 = \tfrac{1}{3} \\ w_1(1) &= a_1 + b_1 = \tfrac{1}{3}, & w_2(1) &= a_2 + b_2 = \tfrac{3+\sqrt{3}i}{6} \\ w_3(0) &= b_3 = \tfrac{3+\sqrt{3}i}{6}, & w_4(0) &= b_4 = \tfrac{2}{3} \\ w_3(1) &= a_3 + b_3 = \tfrac{2}{3}, & w_4(1) &= a_4 + b_4 = 1. \end{aligned}$$

Podemos resolver esas ecuaciones para encontrar a_j y b_j. Otra forma es observar que cada pedazo tiene longitud $\frac{1}{3}$ de la longitud del intervalo I. Luego, el factor de escalamiento es $\frac{1}{3}$; para encontrar a_j sólo necesitamos

encontrar la rotación producida. La constante b_j es siempre la imagen de 0. Un cálculo fácil muestra que $w_1(z) = \frac{1}{3}z$, $w_2(z) = \frac{1}{3}e^{i\frac{\pi}{3}}z + \frac{1}{3}$, $w_3(z) = \frac{1}{3}e^{-i\frac{\pi}{3}}z + \frac{3+\sqrt{3}i}{6}$, y $w_4(z) = \frac{1}{3}z + \frac{2}{3}$, donde $e^{\alpha+i\beta} = e^{\alpha}(\cos(\beta) + i\,\mathrm{sen}(\beta))$ es la exponencial compleja clásica.
Matricialmente, las transformaciones $w_i, i = 1, 2, 3, 4$ vienen dadas por

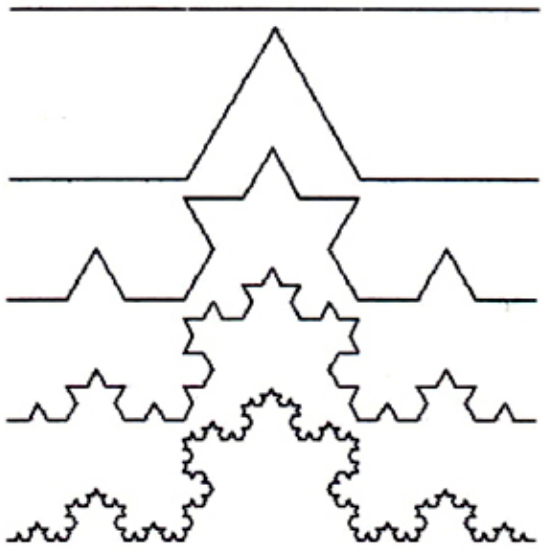
$$w_1(x,y) = \begin{pmatrix} 1/3 & 0 \\ 0 & 1/3 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

$$w_2(x,y) = \begin{pmatrix} 1/6 & -\sqrt{3}/6 \\ \sqrt{3}/6 & 1/6 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 1/3 \\ 0 \end{pmatrix}$$

$$w_3(x,y) = \begin{pmatrix} 1/6 & \sqrt{3}/6 \\ -\sqrt{3}/6 & 1/6 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 1/2 \\ \sqrt{3}/6 \end{pmatrix}$$

$$w_4(x,y) = \begin{pmatrix} 1/3 & 0 \\ 0 & 1/3 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 2/3 \\ 0 \end{pmatrix}$$

El atractor K de este sistema de funciones iteradas es conocido como *curva de Koch.*



Pequeños pedazos de la curva de Koch son obtenidos combinando varias de las transformaciones w_j. Cada $w_i(K)$ es un tercio del tamaño de K, luego cada conjunto $w_i(w_j(K))$ tiene tamaño igual a un noveno del tamaño de K. También K esta formado por todos los pedazos $w_i(w_j(K))$ para todos los índices $i, j \in \{1, 2, 3, 4\}$.

Pedazos cada vez más pequeños son obtenidos aplicando más y más w_j's, por ejemplo,

$$w_{a_1} \circ w_{a_2} \circ \cdots w_{a_n}(K),$$

donde cada subíndice $a_i \in \{1, 2, 3, 4\}$. Este pedazo tiene diámetro igual a $\frac{1}{3^n}\times$ diámetro de K. Cuando $n \to \infty$ esos pedazos cada vez más y más pequeños convergen a un sólo punto de K. Luego, si llamamos a $\{1, 2, 3, 4\}$ el *espacio de códigos*, a cada sucesión $\alpha = \{a_1, a_2, a_3, \ldots\}$ en el espacio de códigos le corresponde un punto $z(\alpha)$ en la curva de Koch $K(w_1, w_2, w_3, w_4)$. Algunos puntos de la curva de Koch corresponden a dos elementos distintos del espacio de códigos, esto es, existen sucesiones $\alpha \neq \beta$ con $z(\alpha) = z(\beta)$. En general, la mayoría de los puntos corresponde exactamente a una única sucesión del espacio de códigos, mientras que una cantidad numerable corresponde a dos sucesiones distintas.

Haciendo un análisis similar al anterior, usando un espacio de código de dos elementos $\{1, 2\}$ para el conjunto de Cantor, es fácil ver que a cada sucesión $\alpha = \{a_1, a_2, \ldots\}, a_i \in \{1, 2\}$ le corresponde sólo un punto de $C = K(w_1, w_2)$, y recíprocamente, cada punto de C determina una única sucesión α en el espacio de códigos.

Curva de Keisswetter

Sean $w_1, w_2, w_3, w_4 : \mathbb{R}^n \to \mathbb{R}^n$, las contracciones afines dadas por

$$w_1(x, y) = \begin{pmatrix} 1/4 & 0 \\ 0 & -1/2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

$$w_2(x, y) = \begin{pmatrix} 1/4 & 0 \\ 0 & 1/2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 1/4 \\ -1/2 \end{pmatrix}$$

$$w_3(x, y) = \begin{pmatrix} 1/4 & 0 \\ 0 & 1/2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 1/2 \\ 0 \end{pmatrix}$$

$$w_4(x, y) = \begin{pmatrix} 1/4 & 0 \\ 0 & 1/2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 3/4 \\ 1/2 \end{pmatrix}.$$

El atractor K de este IFS es conocido como *curva de Kiesswetter*. Es claro que $\{w_i : i = 1, \ldots, 4\}$ es un IFS con factor de contractividad $s = \frac{1}{2}$. De hecho, cada transformación afín tiene factor de contractividad $s_i = \frac{1}{2}, i = 1, \ldots, 4$.

Helecho

$$w_1(x,y) = \begin{pmatrix} 0 & 0 \\ 0 & 0{,}16 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0{,}203 \\ -0{,}035 \end{pmatrix}$$

$$w_2(x,y) = \begin{pmatrix} 0{,}2 & 0{,}26 \\ -0{,}23 & 0{,}22 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0{,}173 \\ 1{,}628 \end{pmatrix}$$

$$w_3(x,y) = \begin{pmatrix} -0{,}15 & -0{,}28 \\ -0{,}26 & 0{,}24 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0{,}222 \\ 0{,}465 \end{pmatrix}$$

$$w_4(x,y) = \begin{pmatrix} 0{,}85 & -0{,}04 \\ 0{,}04 & 0{,}85 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0{,}029 \\ 1{,}597 \end{pmatrix}.$$

Familia de Dragones

Las imágenes generadas por la siguiente familia de contracciones son comunmente llamados dragones, por su similitud con aquellos de los grabados chinos,

$$S_{\pm}(x,y) = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \pm 1 \\ 0 \end{pmatrix}$$

donde $|a_i j| < 1$

Curva de Levi

Esta figura fractal es obtenida como el atractor del siguiente sistema de funciones iteradas:

$$T_1(x,y) = \begin{pmatrix} 0{,}5 & -0{,}5 \\ 0{,}5 & 0{,}5 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

$$T_2(x,y) = \begin{pmatrix} 0{,}5 & 0{,}5 \\ -0{,}5 & 0{,}5 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0{,}5 \\ 0{,}5 \end{pmatrix}$$

4.5. Calculando el Atractor por Iteración Aleatoria

Notenos que lo descrito anteriormente es de carácter determinista, y que en efecto nos provee de un algoritmo determinista para calcular al atractor de un IFS. Es fácil convencerse que el cálculo del atractor de un IFS mediante el algoritmo determinista es bastante lento. Por ejemplo, si tenemos tres transformaciones, w_1, w_2, w_3, y tomando K como el más simple de los compacto no vacíos, es decir, $K = \{p\}$ (un punto); se tiene que

$$\begin{aligned} W(p) &= w_1(p) \cup w_2(p) \cup w_3(p) \\ W^2(P) &= w_1(w_1(p)) \cup w_1(w_2(p)) \cup w_1(w_3(p)) \cup w_2(w_1(p)) \\ &\quad \cup w_2(w_2(p)) \cup w_2(w_3(p)) \cup w_3(w_1(p)) \cup w_3(w_2(p)) \\ &\quad \cup w_3(w_3(p)) \\ &\vdots \end{aligned}$$

y en general, se tiene

$$W^n(p) = \bigcup_{\substack{i_1,i_2,\ldots,i_n \\ i_j \in \{1,2,3\}, j=1,\ldots,n}} w_{i_1 i_2 \ldots i_n}(p).$$

Un análisis muestra que $W(\{p\})$ está formado por 3 puntos, $W^2(\{p\})$ está formado por 3^2 puntos, y en general $W^n(\{p\})$ está formado 3^n puntos, cada uno de los cuales debe ser evaluado por w_1, w_2 y w_3 para obtener $W^{n+1}(\{p\})$. Esto, como es fácil de convencerse, consume mucho tiempo y memoria computacional.

Dado que el algoritmo determinista es muy lento, podemos intentar un algoritmo aleatorio , el cual es conocido con el nombre de *Chaos Game*. Para esto, asociamos a cada contracción w_i una probabilidad p_i, $0 < p_i < 1$, de modo que $\sum_{i=1}^{n} p_i = 1$. Esta probabilidad p_i representa la oportunidad de w_i de ser seleccionada para aplicarla en el próximo paso de la iteración. Por ejemplo, si elegimos $i_j = 1$ en cada etapa de la iteración de un punto p, la

sucesión $p_m = w_i^m(p)$ converge rápidamente al único punto fijo de w_1. La sucesión de elección tiene probabilidad $\lim_{m\to\infty}(1-p_1)^m = 0$.

Ahora, cada punto de $K(w_1,\ldots,w_n)$ está asociado a una sucesión en el espacio de códigos, $\beta = \{b_1, b_2, \ldots,\}$. Dos sucesiones en el espacio de códigos corresponden a puntos próximos en $K(w_1,\ldots,w_n)$ si ellas coinciden en sus primeros N (grande) elementos. La sucesión $w_{b_1}\cdots w_{b_N}$ tiene probabilidad pequeña, pero positiva, de ocurrir, la cual es dada por $p_{b_1}\cdots p_{b_N}$, asumiendo que cada término es elegido independientemente.

Consideremos la órbita $p_m = w_{a_m}(p_{m-1})$. Como estamos eligiendo una cantidad infinita de sucesiones de N términos, tenemos probabilidad 1 de eventualmente elegir $a_m = b_1$, $a_{m-1} = b_2, \cdots, a_{m-N+1} = b_N$. Se tiene entonces que

$$p_m = w_{a_m}\circ\cdots\circ w_{a_1}(p) = w_{b_1}\circ\cdots\circ w_{b_N}\circ w_{a_m-N}\circ\cdots w_{a_1}(p)$$

está en la parte de $K(w_1,\ldots,w_n)$ correspondiente a la sucesión $\{b_1,\ldots,b_n\}$. Esto justifica porqué la sucesión $(p_m)_{m\in\mathbb{N}}$ se aproxima arbitrariamente a cada punto de $K(w_1,\ldots,w_n)$. De lo anterior se tiene la siguiente

Proposición : *La sucesión $(p_n)_{n\in\mathbb{N}}$ construida anteriormente es densa en $K(w_1,\ldots,w_n)$.*

Observación : Si las $w_i, i = 1,\ldots,n$ son transformaciones afines, $w_i(x,y) = A_i(x,y) + b_i$ donde A_i es una matriz de contracción 2×2 y b_i es vector de traslación, una manera natural de elegir las probabilidades p_i asociadas es

$$p_i = \frac{|det(A_i)|}{\sum_{i=1}^{n}|det(A_i)|}.$$

4.6. Teorema del Pegamiento (Collage)

El estudio de los IFS tiene una importancia práctica en el llamado *problema inverso* de la geometría fractal: «comenzar con un fractal y encontrar el IFS u otro sistema dinámico que converge a este fractal». Por otra parte, una imagen típica puede requerir para ser almacenada una estructura de datos pixel por pixel muy grande, y un IFS complicado consume sólo unos cuantos centenares de carácteres para ser escrito. Luego, si conocemos el IFS que genera una imagen necesitamos mucho menos memoria computacional que la

figura original para almacenarlo. Esto es llamado *Compresión de Imágenes*. Para descomprimir la imagen aplicamos el método de iteración aleatorio al IFS almacenado y rápidamente generamos la imagen.

Problema : ¿ Cómo encontrar el IFS asociado a una imagen?

El ingrediente clave es el *Teorema del Collage de Barnsley*. Supongamos que tenemos un conjunto fractal compacto L que queremos codificar como el atractor de un IFS. Examinamos el conjunto y tratamos de descubrir las partes de este que pueden ser asemejadas al conjunto entero (autosimilaridad). Esto es, encontramos contracciones $w_1, w_2, \ldots, w_n$ tales que cada $w_j(L)$ es aproximadamente una pequeña parte de L. La unión de todos esos pedazos es aproximadamente L, es decir

$$L \approx w_1(L) \cup \cdots \cup w_n(L).$$

La pregunta clave aquí es ¿qué significa «aproximadamente»?

Para responder a esto, usamos la noción de distancia de Hausdorff entre conjuntos compactos. Luego, nos estamos preguntando por

$$h(L, w_1(L) \cup \cdots \cup w_n(L)) < \varepsilon$$

$\varepsilon > 0$ pequeño. Sea $0 \leq s < 1$ el factor de contractividad del IFS $\{w_1, \ldots, w_n\}$. Vimos que existe un atractor $K = K(w_1, \ldots, w_n)$ asociado al IFS, y tenemos el siguiente

Teorema : *(del Collage). Sea $\{(w_i, r_i) : i = 1, \ldots, \ell\}$ un IFS con atractor $K \in \mathcal{K}(\mathbb{R}^n)$. Dado $\varepsilon > 0$, sea $L \in \mathcal{K}(\mathbb{R}^n)$ tal que*

$$H(L, \cup_{i=1}^{\ell} w_i(L)) \leq \varepsilon.$$

Entonces

$$H(L, K) \leq \frac{\varepsilon}{1 - r}$$

donde $r = max\{r_i : i = 1, \ldots, \ell\}$ es el radio de contracción de la aplicación $W : \mathcal{K}(\mathbb{R}^n) \to \mathcal{K}(\mathbb{R}^n)$ dada por $W(M) = \cup_{i=1}^{\ell} w_i(M)$. Equivalentemente

$$H(L, K) \leq \frac{1}{1 - r} H(L, \cup_{i=1}^{\ell} w_i(L))$$

.

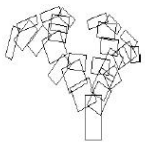
Si $s < 0{,}5$, por ejemplo, entonces $h(L, K(w_1, \ldots, w_n)) \leq 2\varepsilon$. Entonces el atractor $K(w_1, \ldots, w_n)$ es una buena aproximación del conjunto original L, tanto como lo es el «collage $w_1(L) \cup \cdots \cup w_n(L)$». La mejor situación

posible es cuando s es tan pequeño cuánto sea posible. Para s bastante pequeño usamos contracciones afines, w_j tales que $w_j(L)$ representa una parte pequeña de L. Luego existe una relación directa entre la precisión del atractor y el tamaño del IFS usado para representar L.

Ahora observemos que si w es una transformación afín del plano, entonces $w(x, y) = (ax+by+e, cx+dy+f)$. Luego w es determinada por seis parámetros a, b, c, d, e y f. Supongamos que tenemos seis puntos $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2)$, $P_3 = (x_3, y_3)$, $Q_1 = (u_1, v_1)$, $Q_2(u_2, v_2)$ y $Q_3(u_3, v_3)$. Entonces las ecuaciones,

$$w(P_1) = Q_1, \quad w(P_2) = Q_2 \quad \text{y} \quad w(P_3) = Q_3.$$

producen seis ecuaciones lineales en las seis incógnitas a, b, c, d, e y f. Si los puntos P_1, P_2, P_3 no pertenecen todos a la misma recta, esas ecuaciones tienen solución única. Luego, las transformaciones afines son determinadas por la forma en que ellas transforman un triángulo.



4.7. Algoritmos utilizados para para la obtención de un fractal asociado a un sistema de funciones iteradas

En este capítulo se exponen dos algoritmos, el determinista y el aleatorio, proporcionando ambos el mismo resultado, que permiten obtener el atractor asociado a un sistema de funciones iteradas. También se estudia cómo obtener un nuevo algoritmo para generar fractales en movimiento a partir del algoritmo aleatorio. Solo se considera el caso en el que los sistemas de funciones iteradas están definidos sobre $\mathbb{R}^2$, por ser de más sencilla elaboración, aunque el desarrollo en cualquier otro espacio métrico es posible sin dificultad adicional.

4.7.1. Algoritmo determinista

Las pautas para la obtención del atractor de un SFI se pueden resumir en el siguiente algoritmo:

1. En primer lugar elegir un conjunto arbitrario $B \subset X$ compacto y no vacío.
2. Hacer $Z = B$.
3. Representar Z
4. Hacer desde $i = 1$ hasta M.
   a) Borrar Z.
   b) $F(Z) = \cup_{i=1}^{N} f_i(Z)$.
   c) Hacer $Z = F(Z)$
   d) Representar Z.
5. Fin.

Cuando este algoritmo termine de ejecutarse habremos obtenido $F^M(B)$, que para $M = 10$ nos da, en general, una muy buena aproximacin del atractor A.

4.7.2. Código Matlab : Algoritmo determinista

```
function p = IFS(trans,shape,n)


% Función que devuelve los puntos generados por el SFI
% IFS(trans,shape,n)
% Variables de entrada:
% trans - lista de transformaciones afines
% shape - forma: punto, segmento, triángulo, cuadrado
% n - número de iteraciones
% ejemplo:
% n = 4;
% trans = [0.4194 0.3629 -0.0000; 0.0376 0.3306 0.0000; 0 0 1.0000],...
% [0.5645 -0.2903 0; 0.0699 0.1855 0.0000; 0.8500 0.8250 1.0000];
% shape = [0 0 1; 0 0.5 1; 1 0.5 1; 1 0 1];
% p = IFS(trans,shape,n);
```

```
temp = shape;


% Forma: punto, segmento, triángulo, cuadrado

[iw,ik] = size(temp);


% Número de transformaciones

tr = length(trans);


% Guardamos los puntos transformados después de una iteración

temp1 = zeros(iw,ik*tr^n);


% Se inicializa w=1 y con el bucle for controlamos el
% número de iteraciones.


w = 1;

for i = 1:n
    k = 1;


% Aplicamos cada una de las transformaciones

    for j = 1:tr
        for m = 1:w
            temp1(:,1+(k-1)*ik:ik+(k-1)*ik) = ...
            temp(:,1+(m-1)*ik:ik+(m-1)*ik) * trans{j};
            k = k + 1;
        end
    end



% Se actualiza el número de puntos al que aplicar las transfomaciones
% en la siguiente iteración
```

```
    w = w * tr;
    temp = temp1(:,1:ik*w);
end
% Se guarda en p los puntos generados por el SFI.

p = temp;
```

4.7.3. Algoritmo aleatorio

Sea $\{f_1, f_2, \ldots, f_N\}$ un sistema de funciones iteradas en el plano. Asignamos a cada $f_i, 1 \leq i \leq N$, una cierta probabilidad $p_i > 0$ tal que $\sum_{i=1}^{N} p_i = 1$, y realizamos el siguiente proceso iterativo:
Se elige $x_0 \in \mathbb{R}^2$ arbitrario. A continuación se elige aleatoriamente

$$x_1 \in \{f_1(x_0), \ldots, f_N(x_0)\},$$

donde $f_i(x_0), 1 \leq i \leq N$, tiene una probabilidad p_i de ser elegido. Análogamente e independientemente del paso anterior, se elige aleatoriamente

$$x_2 \in \{f_1(x_1), \ldots, f_N(x_1)\},$$

según la misma distribución de probabilidades. Cuando tenemos construidos $\{x_0, x_1, \ldots, x_p\}$, se determina x_{p+1} mediante el mismo proceso anterior, es decir, eligiendo de manera independiente (de los anteriores pasos) y aleatoria

$$x_{p+1} \in \{f_1(x_p), \ldots, f_N(x_p)\},$$

según la distribución de probabilidades. Y así sucesivamente. Entonces *con probabilidad uno*, el conjunto obtenido $\{x_n\}_{n=0}^{\infty} \subset X$ converge en la métrica de Hausdorff al atractor A del SFI, en el sentido de que dado $\varepsilon > 0$, existe $K = K(\varepsilon) \in N$ tal que

$$lim_{M \to \infty} d_H(A, \{x_n : K \leq n \leq M\}) < \varepsilon.$$

De lo anterior se deduce que los puntos del conjunto $\{x_n\}_{n=0}^{\infty}$ que pueden estar a mayor distancia del atractor son los primeros puntos de la sucesión. Por este motivo, cuando se intenta aproximar el atractor mediante este algoritmo se suelen despreciar los primeros términos (con despreciar los primeros 50 es suficiente para obtener una buena aproximación del atractor).

Sea $\{f_1, \ldots, f_N\}$ un SFI con probabilidades $\{p_i\}_{i=1}^{N} (p_i > 0, 1 \leq i \leq N$ y $\sum_{i=1}^{N} p_i = 1)$. Una redacción más precisa del algoritmo aleatorio sería el siguiente pseudocdigo :

1. Elegir un punto arbitrario $x \in \mathbb{R}^2$.
2. Hacer desde $i = 1$ hasta M:
   a) Elegir j aleatoriamente entre $\{1, 2, \ldots, N\}$ con probabilidades $\{p_1, p_2, \ldots, p_N\}$.
   b) Hallar $y = f_j(x)$.
   c) Hacer $x = y$.
   d) Si $i > 50$ representar x.
3. Fin.

Cuando este algoritmo termine de ejecutarse habremos representado $M - 50$ puntos, que para $M = 5000$ nos da, en general, una muy buena aproximación del atractor A.

4.7.4. Código Matlab : Algoritmo aleatorio

```
function IFS1(trans,pr,pto_inicial,m,iden,ejes,op_visual,vel)


% Función que dibuja el atractor del SFI utilizando el algoritmo
% aleatorio
% IFS1(trans,pr,pto_inicial,m,iden,ejes,op_visual,vel)
% Variables de entrada:
% trans - lista de transformaciones afines
% pr - probabilidad de cada transformación
% pto_inicial - punto donde comienza la iteración
% m - número de iteraciones
% iden - posición donde se dibuja el resultado
% ejes - indica si se desea ver los ejes de la figura
% op_visual - contiene el valor 1 si se quieren visualizar las iteraciones
% y 0 en caso contrario
% vel - velocidad para dibujar los puntos de la figura


% Se declara una variable global para ser usada por otros programas.


global p;
```

```
% Con op_visual controlamos si se quieren visualizar las iteraciones
% o no.
% Si queremos visualizar las iteraciones
if op_visual==1
    i=1;

    while i<m & ishandle(iden)


% Se elige aleatoriamente la transformación a elegir según sus probabilidades


        j_ale=uint8(randp(pr,1));



% Se transforma el punto inicial
        pto_inicial=pto_inicial*trans{j_ale};

        if i>50
            if ishandle(iden)
                figure(iden);
                p_con=get(p,'Value');
                if p_con==1
                    waitfor(p,'Value',0);
                end
                if ishandle(iden)
                    plot(pto_inicial(1),pto_inicial(2),'ob','MarkerSize',5);
                    if ejes==0
                        axis off;
                    else
                        axis on;
                    end
                    hold on
                    pause(vel);
                end
            end
        end
        i=i+1;
```

```
    end


% Para no visulaizar las iteraciones

elseif op_visual==0
    puntos=zeros(m,2);
    for i = 1:m


% Se elige aleatoriamente la transformación a elegir según sus probabilidades


        j_ale=uint8(randp(pr,1));



% Se transforma el punto inicial

        puntos(i,:)=pto_inicial*trans{j_ale}(:,1:2);

        pto_inicial=[puntos(i,:),1];
    end

    figure(iden);
    plot(puntos(50:m,1),puntos(50:m,2),'ob','MarkerSize',5);
    if ejes==0
        axis off;
    else
        axis on;
    end
end
```

5 Fractales en movimiento

En esta sección nos hemos basado en el PFC [15] que a su vez estuvo basado en los trabajos [6], [9] y [13]. Hemos trabajado sobre sus movimientos aumentando el número de movimientos, pasándo de ser movimientos simples a movimientos compuestos.

Puesto que los conjuntos fractales que hemos considerado hasta ahora dependen directamente de una familia de funciones contractivas, parece razonable pensar que pequeñas variaciones es estas funciones produzcan pequeñas variaciones en el fractal generado.
Si esto fuese así se podría generar fractales muy próximos entre sí, que montados adecuadamente, podrían producir un efecto de movimiento sobre un determinado fractal.
Se supone que las aplicaciones contractivas que definen un SFI $\{f_1, \ldots, f_N\}$ no vienen unívocamente determinadas, sino que vienen definidas en función de un parmetro $\rho \in [\alpha, \beta] \subset \mathbb{R}$ del que dependen continuamente.
En el siguiente teorema se estudia como influyen pequeñas variaciones del parmetro ρ.

Teorema : *Para cada $\rho \in [\alpha, \beta] \subset \mathbb{R}$ sea $\{f_1(\rho), \ldots, f_N(\rho)\}$ un SFI de razón $r(\rho), 0 \leq r(\rho) \leq r < 1$ con atractor $A(\rho) \in \mathcal{H}(\mathbb{R}^n)$. Supongamos que, para cada $x \in X$ y $1 \leq i \leq N$ fijos, la función $f_i(\rho)(x) = f_i(\rho, x)$ es contínua respecto de $\rho \in [\alpha, \beta]$. Entonces el atractor $A(p)$ depende continuamente de $\rho \in [\alpha, \beta]$.*

Pasamos a detallar detenidamente cada uno de los posibles movimientos estudiados:

5.1. Movimientos simples

5.1.1. Giro sobre el punto (0,0)

Para calcular el giro del atractor alrededor del punto $(0,0)$ hemos de coger el SFI del conjunto en cuestión y aplicarle las matrices de giro.
Dichas matrices son:

$$G_\alpha(x,y) = \begin{pmatrix} cos(\alpha) & -sin(\alpha) \\ sin(\alpha) & cos(\alpha) \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix},$$

$$G_\alpha^{-1} = \begin{pmatrix} cos(\alpha) & sin(\alpha) \\ -sin(\alpha) & cos(\alpha) \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}.$$

La matriz G_α nos proporciona un giro de α grados del atractor y G_α^{-1} nos proporciona el giro inverso.
Para obtener el giro alrededor del punto $(0,0)$ hemos de calcularlo de la siguiente manera:
Cualquier funcin afín del SFI tiene la siguiente forma:

$$f(x,y) = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} e \\ f \end{pmatrix},$$

tomando

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix},$$

y

$$E = \begin{pmatrix} e \\ f \end{pmatrix},$$

debemos realizar la siguiente operación

$$(G_\alpha * A * G_\alpha^{-1}) \begin{pmatrix} x \\ y \end{pmatrix} + (G_\alpha * E),$$

a cada una de las funciones que componen el SFI a estudiar.
Para hacer esto de forma más eficiente hemos realizado un programa para que realice dichos cálculos

Programa de Matlab para la obtención del giro del atractor sobre el punto (0,0)

```
function [trans_matriz,trans_vector]=girar_atractor(fichero)


% Esta función nos proporciona los sistemas de funciones iteradas
% para girar un atractor sobre el punto (0,0) cuyo sistema de funciones
% iteradas viene dado en fichero
% [trans_matriz,trans_vector]=girar_atractor(fichero)
% Variables de entrada:
% fichero - nombre del archivo que contiene el sistema de funciones
% iteradas que se quiere girar
% Variables de salida:
% [trans_matriz,trans_vector] - SFI para girar el atractor


syms p;


% Definimos las matrices de giro

g=[cos(p), -sin(p); sin(p), cos(p)];
g_inv=[cos(p), sin(p); -sin(p), cos(p)];


% Leemos el sistema de funciones iteradas

[trans] = read_IFS(fichero);


% Transformación de cada una de las funciones iteradas

for i=1:length(trans)
    a=trans{i}(1:2,1:2)';
    tras=trans{i}(3,1:2)';
    trans_matriz{i}=simplify(g*a*g_inv);
    trans_matriz{i}
    trans_vector{i}=simplify(g*tras);
    trans_vector{i}
    pause
end
```

5.1.2. Giro sobre un punto cualquiera

Para poder girar un atractor alrededor de un punto cualquiera del plano hemos de coger el SFI del conjunto en cuestión y aplicarle las matrices de giro que nos proporcionen el efecto deseado.
Las matrices de giro respecto del punto $(0,0)$ son:

$$G_\alpha(x,y) = \begin{pmatrix} cos(\alpha) & -sin(\alpha) \\ sin(\alpha) & cos(\alpha) \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix},$$

$$G_\alpha^{-1} = \begin{pmatrix} cos(\alpha) & sin(\alpha) \\ -sin(\alpha) & cos(\alpha) \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}.$$

La matriz G_α nos proporciona un giro de α grados del atractor y G_α^{-1} nos proporciona el giro inverso.
Para obtener el giro alrededor de un punto cualquiera debemos transformar cada una de las funciones iteradas de la siguiente forma: Si cada una de las funciones iteradas son como se muestra a continuación:

$$f(x,y) = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} e \\ f \end{pmatrix},$$

tomando

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix},$$

y

$$E = \begin{pmatrix} e \\ f \end{pmatrix},$$

debemos realizar la siguiente operación

$$(G_\alpha * A * G_\alpha^{-1}) \begin{pmatrix} x \\ y \end{pmatrix} +$$

$$G_\alpha * A * \left(punto - G_\alpha^{-1} * punto\right) + G_\alpha * E + punto - G_\alpha * punto,$$

a cada una de las funciones que componen el SFI a estudiar, siendo la variable

$$punto = \begin{pmatrix} p_1 \\ p_2 \end{pmatrix},$$

el punto desde el cual se desea girar el atractor.
Como en el caso anterior hemos realizado un programa para ahorrar tiempo en la realización de dichos cálculos

Programa de Matlab para la obtención del giro del atractor sobre un punto cualquiera

```
function [trans_matriz,trans_vector]=girar_atractor_punto(fichero,punto)


% Esta función nos proporciona los sistemas de funciones iteradas
% para girar un atractor sobre cualquier punto cuyo sistema de funciones
% iteradas viene dado en fichero
% [trans_matriz,trans_vector]=girar_atractor_punto(fichero,punto)
% Variables de entrada:
% fichero - nombre del archivo que contiene el sistema de funciones
% iteradas que se quiere girar
% punto [c1,c2] - puntos alrededor de donde se gira el atractor
% Variables de salida:
% [trans_matriz,trans_vector] - SFI para girar el atractor


syms p;
punto=punto';


% Definimos las matrices de giro

g=[cos(p), -sin(p); sin(p), cos(p)];
g_inv=[cos(p), sin(p); -sin(p), cos(p)];
aux1=simplify(punto-g_inv*punto);
aux2=simplify(punto-g*punto);


% Leemos el sistema de funciones iteradas

[trans] = read_IFS(fichero);


% Transformación de cada una de las funciones iteradas

for i=1:length(trans)
    a=trans{i}(1:2,1:2)';
    tras=trans{i}(3,1:2)';
    trans_matriz{i}=simplify(g*a*g_inv);
    trans_matriz{i}
```

```
    trans_vector{i}=simplify(g*a*aux1+g*tras+aux2);
    trans_vector{i}
    pause
end
```

5.1.3. Traslación

Para obtener el SFI que nos permita trasladar el atractor vamos a proceder como sigue. Escribamos cada una de las funciones iteradas como

$$f(x,y) = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} e \\ f \end{pmatrix},$$

y el vector de traslacin como,

$$punto = \begin{pmatrix} p_1 \\ p_2 \end{pmatrix}.$$

Teniendo en cuenta que la traslación viene dada por

$$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + punto,$$

y la traslación inversa por

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} - punto,$$

tomando

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix},$$

y

$$E = \begin{pmatrix} e \\ f \end{pmatrix},$$

tengo que la transformación de cada función iterada queda,

$$A \begin{pmatrix} x \\ y \end{pmatrix} - t * A * punto + E + t * punto,$$

donde t es el parámetro que produce el movimiento progresivo de traslación. Entonces, esto es lo que debemos aplicar a cada una de las funciones para obtener el conjunto de funciones iteradas que nos trasladan el atractor.
Igual que en los casos anteriores el programa que nos realiza estos cálculos se detalla a continuación

Programa de Matlab para trasladar el atractor

```
function [trans_matriz,trans_vector]=trasladar_atractor(fichero,punto)


% Esta función nos proporciona los sistemas de funciones iteradas
% para trasladar un atractor cuyo sistema de funciones
% iteradas viene dado en fichero
% [trans_matriz,trans_vector]=trasladar_atractor(fichero,punto)
% Variables de entrada:
% fichero - nombre del archivo que contiene el sistema de funciones
% iteradas que se quiere girar
% punto [c1,c2] - puntos alrededor de donde se gira el atractor
% Variables de salida:
% [trans_matriz,trans_vector] - SFI para girar el atractor



syms p;
punto=punto';
aux1=simplify(p*punto);


% Leemos el sistema de funciones iteradas

[trans] = read_IFS(fichero);


% Transformación de cada una de las funciones iteradas

for i=1:length(trans)
    a=trans{i}(1:2,1:2)';
    tras=trans{i}(3,1:2)';
    trans_matriz{i}=simplify(a);
    trans_matriz{i}
    trans_vector{i}=simplify(aux1+tras-p*a*punto);
    trans_vector{i}
    pause
end
```

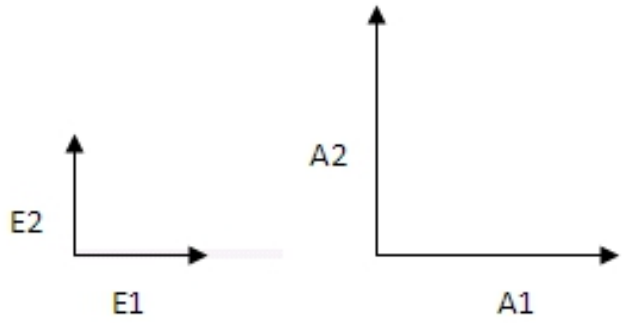


Figura 5.1: Transformación de los ejes de referencia en una dilatación

5.1.4. Dilatación-Contracción

Imaginemos que tenemos la imagen que se muestra en la Figura 5.1: Para dilatar la imagen tenemos que pasar de la imagen de la izquierda a la derecha y para ello necesitamos hacer un cambio de base:

De $(E1, E2)$ a $(A1, A2)$ con $|\beta| \leq 1$

$$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \tfrac{1}{\beta} \begin{pmatrix} x \\ y \end{pmatrix}.$$

Para contraer la imagen y pasar de $(A1, A2)$ a $(E1, E2)$

$$\begin{pmatrix} x \\ y \end{pmatrix} = \beta \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$$

Partiendo de que las funciones iteradas son:

$$f(x, y) = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} e \\ f \end{pmatrix},$$

tomando

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix},$$

y

$$E = \begin{pmatrix} e \\ f \end{pmatrix},$$

para contraer-dilatar el atractor tendremos que transformar las funciones contractivas según:

$$\beta\left(A\left(\tfrac{1}{\beta}\begin{pmatrix} x \\ y \end{pmatrix}\right)+E\right)=A\begin{pmatrix} x \\ y \end{pmatrix}+\beta\cdot E.$$

Debemos aplicarlo a cada una de las funciones iteradas.
El programa que nos facilita los cálculos lo podemos ver a continuación.

Programa de Matlab para contraer-dilatar el atractor

```
function [trans_matriz,trans_vector]=dilatar_contraer_atractor(fichero)


% Esta función nos proporciona los sistemas de funciones iteradas
% para dilatar-contraer un atractor cuyo sistema de funciones
% iteradas viene dado en fichero
% [trans_matriz,trans_vector]=dilatar_contraer_atractor(fichero)
% Variables de entrada:
% fichero - nombre del archivo que contiene el sistema de funciones
% iteradas que se quiere dilatar-contraer
% Variables de salida:
% [trans_matriz,trans_vector] - SFI para dilatar-contraer el atractor


% Definimos la variable simbólica p

syms p;


% Leemos el sistema de funciones iteradas

[trans] = read_IFS(fichero);


% Transformación de cada una de las funciones iteradas

for i=1:length(trans)
    a=trans{i}(1:2,1:2)';
    tras=trans{i}(3,1:2)';
    trans_matriz{i}=simplify(a);
    trans_matriz{i}
```

```
    trans_vector{i}=simplify(p*tras);
    trans_vector{i}
    pause
end
```

5.1.5. Simetría sobre el eje x

Para realizar una simetría sobre el eje x debemos obtener los sistemas de funciones iteradas que nos proporcionan dicha simetría. Esto se consigue como se muestra a continuación:
Imaginemos que tenemos la figura que se muestra:

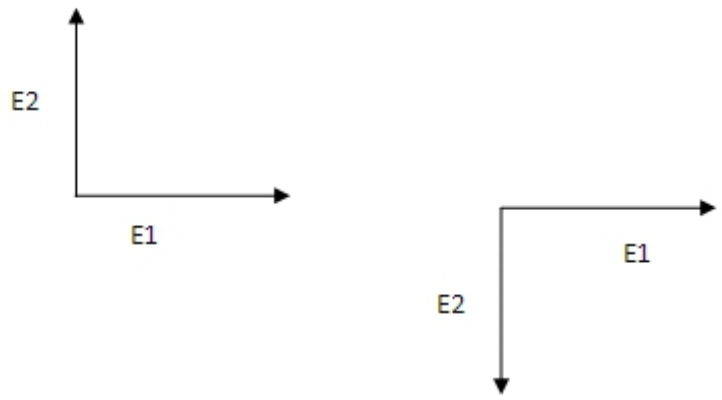


Figura 5.2: Transformación de los ejes de referencia en una simetría sobre el eje x

Lo primero es definir las matrices que nos proporcionan la simetría de manera contínua dependiendo de un parámetro p.
Estas son :

$$S_x \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & -p \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \end{pmatrix};$$
$$S_x^{-1} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & -\frac{1}{p} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}.$$

Teniendo en cuenta que

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix},$$

y

$$E = \begin{pmatrix} e \\ f \end{pmatrix},$$

la operación a realizar a cada una de las funciones para obtener los SFI que nos proporcionan dicha simetría es la siguiente.

$$S_x(A \cdot (S_x^{-1} \cdot \begin{pmatrix} x \\ y \end{pmatrix}) + E) = S_x \cdot A \cdot S_x^{-1} \begin{pmatrix} x \\ y \end{pmatrix} + S_x \cdot E.$$

5.1.6. Simetría sobre el eje y

Las matrices que nos proporcionan la simetría en este caso son:

$$S_y \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -p & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \end{pmatrix},$$
$$S_y^{-1} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} \frac{-1}{p} & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix},$$

y como en casos anteriores tomando :

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix},$$

y

$$E = \begin{pmatrix} e \\ f \end{pmatrix},$$

la operación a realizar es la siguiente

$$S_y(A \cdot (S_y^{-1} \begin{pmatrix} x \\ y \end{pmatrix}) + E) = S_y \cdot A \cdot S_y^{-1} \begin{pmatrix} x \\ y \end{pmatrix} + S_y \cdot E.$$

5.1.7. Simetría sobre el eje x=y

Para obtener simetría en el eje $x = y$ tenemos que:

$$S_{xy}\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1-p & p \\ p & 1-p \end{pmatrix}\begin{pmatrix} x_1 \\ y_1 \end{pmatrix},$$

$$S_{xy}^{-1}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \frac{1}{1-2\cdot p}\begin{pmatrix} 1-p & -p \\ -p & 1-p \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix},$$

La operación a realizar es la siguiente:

$$S_{xy}\cdot A\cdot S_{xy}^{-1}\begin{pmatrix} x \\ y \end{pmatrix} + S_{xy}\cdot E.$$

El programa que nos facilita los cálculos para realizar la simetría sobre el eje x, eje y y el eje $x = y$ lo podemos ver a continuación:

Programa de Matlab para calcular la simetría del atractor

```
function [trans_matriz,trans_vector]=simetria_axial_atractor(fichero,tipo)


% Esta función nos proporciona los sistemas de funciones iteradas
% para hacer una simetrá de un atractor cuyo sistema de funciones
% iteradas viene dado en fichero
% [trans_matriz,trans_vector]=simetria_axial_atractor(fichero,tipo)
% Variables de entrada:
% fichero - nombre del archivo que contiene el sistema de funciones
% iteradas hace una simetría del atractor
% tipo - selección de la simetría a usar
% Eje x 'x',
% Eje y 'y',
% Recta x=y 'xy'
% Variables de salida:
% [trans_matriz,trans_vector] - SFI para hacer una simetría del atractor


% Definimos la variable simbólica p

syms p;
```

```
% Definimos las matrices de simetría

if strcmp(tipo,'x')
    S=[1, 0; 0, -1/p];
    S_inv=[1, 0; 0, -p];
elseif strcmp(tipo,'y')
    S=[-1/p, 0; 0, 1];
    S_inv=[-p, 0; 0, 1];
elseif strcmp(tipo,'xy')
    S=1/(1-2*p)*[1-p, -p; -p, 1-p];
    S_inv=[1-p, p; p, 1-p];
else
    display('Lo sentimos, debe elegir correctamente el tipo de simetria');
    return;
end


% Leemos el sistema de funciones iteradas

[trans] = read_IFS(fichero);


% Transformación de cada una de las funciones iteradas

for i=1:length(trans)
    a=trans{i}(1:2,1:2)';
    tras=trans{i}(3,1:2)';
    trans_matriz{i}=simplify(S_inv*a*S);
    trans_matriz{i}
    trans_vector{i}=simplify(S_inv*tras);
    trans_vector{i}
    pause
end
```

5.2. Movimentos compuestos

Los movimientos compuestos que vamos a tratar en esta sección se basan en la combinación de dos de los movimientos simples anteriormente detallados. La combinación consiste en realizar la transformación de la función

mediante uno de los movimientos, y a continuación realizar la segunda transformación sobre la función anteriormente transformada.
Las combinaciones de movimientos simples estudiadas son las siguientes:

5.2.1. Traslación - Giro

Para obtener este movimiento, primero se transforma cada una de las funciones iteradas mediante traslación. Una vez obtenido el resultado, se transforma cada una de las funciones iteradas trasladadas mediante giro.
A continuación se muestra el programa Matlab para este movimiento:

Programa de Matlab para trasladar y girar el atractor

```
function [trans_matriz,trans_vector]=trasladar_girar_atractor1(fichero,
punto1,punto2)


% Esta función nos proporciona los sistemas de funciones iteradas
% para trasladar y girar un atractor cuyo sistema de funciones iteradas
% viene dado en fichero
% [trans_matriz,trans_vector]=trasladar_girar_atractor1(fichero,punto1,punto2)
% Variables de entrada:
% fichero - nombre del archivo que contiene el sistema de funciones
% iteradas que se quiere trasladar y girar
% punto1 - donde se inicia la traslación
% punto2 - donde se termina la traslación
% Variables de salida:
% [trans_matriz,trans_vector] SFI para traladar y girar el atractor


% Definimos las variables simbólicas p y q

syms p q;

punto=(punto2-punto1)';
aux1=simplify(p*punto);


% Leemos el sistema de funciones iteradas
```

```
[trans] = read_IFS(fichero);


% Transformación de cada una de las funciones iteradas mediante
% traslación

for i=1:length(trans)
    a=trans{i}(1:2,1:2)';
    tras=trans{i}(3,1:2)';
    trans_matriz{i}=simplify(a);
    trans_vector{i}=simplify(aux1+tras-p*a*punto);
end


% Transformación de cada una de las funciones iteradas mediante
% giro

fcentro=[punto1(1);punto1(2)]+p*punto;
g=[cos(q), -sin(q); sin(q), cos(q)];
g_inv=[cos(q), sin(q); -sin(q), cos(q)];
aux1=simplify(centro-g_inv*centro);
aux2=simplify(centro-g*centro);
for i=1:length(trans)
    a=trans_matriz{i};
    tras=trans_vector{i};
    trans_matriz{i}=simplify(g*a*g_inv);
    trans_matriz{i}
    trans_vector{i}=simplify(g*a*aux1+g*tras+aux2);
    trans_vector{i}
    pause
end
```

5.2.2. Traslación - Dilatación Contracción

Para la obtención del movimiento, primero se transforman las funciones iteradas mediante traslación y seguidamente, se transforma cada una de las funciones iteradas mediante dilatación-contracción.
El programa Matlab que realiza la combinación de movimientos de traslación con dilatación-contracción es el siguiente:

Programa de Matlab para trasladar y dilatar-contraer el atractor

```
function [trans_matriz,trans_vector]=trasladar_dilatar_contraer_atractor
(fichero,punto1,punto2)


% Esta función nos proporciona los sistemas de funciones iteradas
% para trasladar y dilatar-contraer un atractor cuyo sistema de
% funciones iteradas viene dado en fichero
% [trans_matriz,trans_vector]=trasladar_dilatar_contraer_atractor(fichero,
% punto1,punto2)
% Variables de entrada:
% fichero - nombre del archivo que contiene el sistema de funciones
% iteradas que se quiere trasladar-dilatar-contraer
% punto1 - donde se inicia la traslación
% punto2 - donde se termina la traslación
% Variables de salida:
% [trans_matriz,trans_vector] SFI para traladar y dilatar-contraer
% el atractor


% Definimos las variables simbólicas p y q

syms p q;

punto=(punto2-punto1)';
aux1=simplify(p*punto);


% Leemos el sistema de funciones iteradas

[trans] = read_IFS(fichero);


% Transformación de cada una de las funciones iteradas mediante
% traslación

for i=1:length(trans)
    a=trans{i}(1:2,1:2)';
    tras=trans{i}(3,1:2)';
    trans_matriz{i}=simplify(a);
    trans_vector{i}=simplify(aux1+tras-p*a*punto);
end
```

```
% Transformación de cada una de las funciones iteradas mediante
% dilatación-contracción

for i=1:length(trans)
    a=trans_matriz{i};
    tras=trans_vector{i};
    trans_matriz{i}=simplify(a);
    trans_matriz{i}
    trans_vector{i}=simplify(q*tras);
    trans_vector{i}
    pause
end
```

5.2.3. Giro - Dilatación Contracción

Para la obtención del movimiento, primero se transforman las funciones iteradas mediante dilatación-contracción y luego se transforma cada una de las funciones iteradas mediante giro.
El programa Matlab que realiza la combinación de movimientos de traslación con dilatación-contracción es el siguiente:

Programa de Matlab para girar y dilatar-contraer el atractor

```
function [trans_matriz,trans_vector]=girar_dilatar_contraer_atractor(fichero,
punto1,punto2)


% Esta función nos proporciona los sistemas de funciones iteradas
% para girar y dilatar-contraer un atractor cuyo sistema de
% funciones iteradas viene dado en fichero
% [trans_matriz,trans_vector]=girar_dilatar_contraer_atractor(fichero,punto1,punto2)
% Variables de entrada:
% fichero - nombre del archivo que contiene el sistema de funciones
% iteradas que se quiere girar-dilatar-contraer
% punto1 - donde se inicia el movimiento
% punto2 - donde se termina el movimiento
% Variables de salida:
% [trans_matriz,trans_vector] SFI para girar y dilatar-contraer
```

```
% el atractor


% Definimos las variables simbólicas p y q
syms p q;

punto=(punto2-punto1)';



% Leemos el sistema de funciones iteradas
[trans] = read_IFS(fichero);


% Transformación de cada una de las funciones iteradas mediante
% dilatación-contracción
for i=1:length(trans)
    a=trans{i}(1:2,1:2)';
    tras=trans{i}(3,1:2)';
    trans_matriz{i}=simplify(a);
    trans_vector{i}=simplify(p*tras);
end


% Transformación de cada una de las funciones iteradas mediante
% giro
centro=[punto1(1);punto1(2)]+p*punto;
g=[cos(q), -sin(q); sin(q), cos(q)];
g_inv=[cos(q), sin(q); -sin(q), cos(q)];
aux1=simplify(centro-g_inv*centro);
aux2=simplify(centro-g*centro);
for i=1:length(trans)
    a=trans_matriz{i};
    tras=trans_vector{i};
    trans_matriz{i}=simplify(g*a*g_inv);
    trans_matriz{i}
    trans_vector{i}=simplify(g*a*aux1+g*tras+aux2);
    trans_vector{i}
    pause
end
```

5.2.4. Traslación - Simetría

Este movimiento compuesto se realiza del siguiente modo: Primero se transforman las funciones iteradas mediante traslación y luego se transforma cada una de las funciones iteradas mediante el tipo de simetría escogido. El programa Matlab que realiza estos movimientos es el siguiente:

Programa de Matlab para trasladar y realizar la simetría al atractor

```
function [trans_matriz,trans_vector]=trasladar_simetria_atractor(fichero,
tipo,punto1,punto2)


% Esta función nos proporciona los sistemas de funciones iteradas
% para trasladar y hacer una simetría a un atractor cuyo sistema
% de funciones iteradas viene dado en fichero
% [trans_matriz,trans_vector]=trasladar_simetria_atractor(fichero,tipo,punto1,punto2)
% Variables de entrada:
% fichero - nombre del archivo que contiene el sistema de funciones
% iteradas que se quiere trasladar y realizar simetría
% tipo - selección de la simetría a usar
% Eje x 'x',
% Eje y 'y',
% Recta x=y 'xy'
% punto1 - donde se inicia la traslación
% punto2 - donde se termina la traslación
% Variables de salida:
% [trans_matriz,trans_vector] SFI para trasladar y realizar la simetría
% al atractor


% Definimos las variables simbólicas p y q
syms p q;

punto=(punto2-punto1)';
aux1=simplify(p*punto);


% Leemos el sistema de funciones iteradas
```

```
[trans] = read_IFS(fichero);


% Transformación de cada una de las funciones iteradas mediante
% traslación

for i=1:length(trans)
    a=trans{i}(1:2,1:2)';
    tras=trans{i}(3,1:2)';
    trans_matriz{i}=simplify(a);
    trans_vector{i}=simplify(aux1+tras-p*a*punto);
end


% Movimiento simétrico
% Definimos las matrices de simetría


if strcmp(tipo,'x')
    S=[1, 0; 0, -1/q];
    S_inv=[1, 0; 0, -q];
elseif strcmp(tipo,'y')
    S=[-1/q, 0; 0, 1];
    S_inv=[-q, 0; 0, 1];
elseif strcmp(tipo,'xy')
    S=1/(1-2*q)*[1-q, -q; -q, 1-q];
    S_inv=[1-q, q; q, 1-q];
else
    display('Lo sentimos, debe elegir correctamente el tipo de simetria');
    return;
end


% Transformación de cada una de las funciones iteradas mediante
% simetría

for i=1:length(trans)
    a=trans_matriz{i};
    tras=trans_vector{i};
    trans_matriz{i}=simplify(S_inv*a*S);
    trans_matriz{i}
    trans_vector{i}=simplify(S_inv*tras);
```

```
    trans_vector{i}
    pause
end
```

5.2.5. Giro - Simetría

Este movimiento compuesto se realiza del siguiente modo: Primero se transforman las funciones iteradas mediante giro y estas funciones transformadas se vuelven a transformar pero esta segunda vez mediante el tipo de simetría escogido.
El programa Matlab que realiza estos movimientos es el siguiente:

Programa de Matlab para girar y realizar la simetría al atractor

```
function [trans_matriz,trans_vector]=girar_simetria_atractor(fichero,
tipo,punto1,punto2)


% Esta función nos proporciona los sistemas de funciones iteradas
% para girar y hacer una simetría a un atractor cuyo sistema
% de funciones iteradas viene dado en fichero
% [trans_matriz,trans_vector]=girar_simetria_atractor(fichero,tipo,punto1,punto2)
% Variables de entrada:
% fichero - nombre del archivo que contiene el sistema de funciones
% iteradas que se quiere trasladar y realizar simetría
% tipo - selección de la simetría a usar
% Eje x 'x',
% Eje y 'y',
% Recta x=y 'xy'
% punto1 - donde se inicia el movimiento
% punto2 - donde se termina el movimiento
% Variables de salida:
% [trans_matriz,trans_vector] SFI para girar y realizar la simetría
% al atractor


% Definimos las variables simbólicas p y q

syms p q;
```

```
% Leemos el sistema de funciones iteradas

[trans] = read_IFS(fichero);


% Transformación de cada una de las funciones iteradas mediante
% giro

punto=(punto2-punto1)';
aux1=simplify(p*punto);
centro=[punto1(1);punto1(2)]+p*punto;
g=[cos(p), -sin(p); sin(p), cos(p)];
g_inv=[cos(p), sin(p); -sin(p), cos(p)];
aux1=simplify(centro-g_inv*centro);
aux2=simplify(centro-g*centro);
for i=1:length(trans)
    a=trans{i}(1:2,1:2)';
    tras=trans{i}(3,1:2)';
    trans_matriz{i}=simplify(g*a*g_inv);
    trans_vector{i}=simplify(g*a*aux1+g*tras+aux2);
end


% Movimiento simétrico
% Definimos las matrices de simetría


if strcmp(tipo,'x')
    S=[1, 0; 0, -1/q];
    S_inv=[1, 0; 0, -q];
elseif strcmp(tipo,'y')
    S=[-1/q, 0; 0, 1];
    S_inv=[-q, 0; 0, 1];
elseif strcmp(tipo,'xy')
    S=1/(1-2*q)*[1-q, -q; -q, 1-q];
    S_inv=[1-q, q; q, 1-q];
else
    display('Lo sentimos, debe elegir correctamente el tipo de simetria');
    return;
end
```

```
% Transformación de cada una de las funciones iteradas mediante
% simetría

for i=1:length(trans)
    a=trans_matriz{i};
    tras=trans_vector{i};
    trans_matriz{i}=simplify(S_inv*a*S);
    trans_matriz{i}
    trans_vector{i}=simplify(S_inv*tras);
    trans_vector{i}
    pause
end
```

5.2.6. Simetría - Dilatación Contracción

Para conseguir el atractor de este movimiento se ha desarrollado un programa Matlab que dado un SFI, transforma las funciones iteradas primero mediante simetra y después mediante dilatación-contracción. El programa Matlab que realiza estos movimientos es el siguiente:

Programa de Matlab para dilatar-contraer y realizar la simetría al atractor

```
function [trans_matriz,trans_vector]=simetria_dilatar_contraer_atractor(fichero,tipo)


% Esta función nos proporciona los sistemas de funciones iteradas
% para dilatar-contraer y hacer una simetría a un atractor cuyo
sistema
% de funciones iteradas viene dado en fichero
% [trans_matriz,trans_vector]=simetria_dilatar_contraer_atractor(fichero,tipo)
% Variables de entrada:
% fichero - nombre del archivo que contiene el sistema de funciones
% iteradas que se quiere dilatar-contraer y realizar simetría
% tipo - selección de la simetría a usar
% Eje x 'x',
% Eje y 'y',
% Recta x=y 'xy'
```

```
% Variables de salida:
% [trans_matriz,trans_vector] SFI para dilatar-contraer y realizar
la simetría
% al atractor


% Definimos las variables simbólicas p y q

syms p q;


% Leemos el sistema de funciones iteradas

[trans] = read_IFS(fichero);


% Movimiento simétrico
% Definimos las matrices de simetría


if strcmp(tipo,'x')
    S=[1, 0; 0, -1/q];
    S_inv=[1, 0; 0, -q];
elseif strcmp(tipo,'y')
    S=[-1/q, 0; 0, 1];
    S_inv=[-q, 0; 0, 1];
elseif strcmp(tipo,'xy')
    S=1/(1-2*q)*[1-q, -q; -q, 1-q];
    S_inv=[1-q, q; q, 1-q];
else
    display('Lo sentimos, debe elegir correctamente el tipo de simetria');
    return;
end


% Transformación de cada una de las funciones iteradas mediante
% simetría

for i=1:length(trans)
    a=trans{i}(1:2,1:2)';
    tras=trans{i}(3,1:2)';
    trans_matriz{i}=simplify(S_inv*a*S);
    trans_vector{i}=simplify(S_inv*tras);
end
```

```
% Transformación de cada una de las funciones iteradas mediante
% dilatación-contracción

for i=1:length(trans)
    a=trans_matriz{i};
    tras=trans_vector{i};
    trans_matriz{i}=simplify(a);
    trans_matriz{i}
    trans_vector{i}=simplify(q*tras);
    trans_vector{i}
    pause
end
```

5.3. Código Matlab para la generación de fractales en movimiento

```
function IFS2(ifs2,pr,pto_inicial,m,iden,ejes,val_ejes,mov)


% Función que dibuja el atractor del sistema de funciones iteradas
% utilizando el algoritmo para fractales en movimiento
% IFS2(ifs2,pr,pto_inicial,m,iden,ejes,val_ejes,mov)
% Variables de entrada:
% ifs2 - lista de transformaciones afines
% pr - probabilidad de cada transformación
% pto_inicial - punto donde comienza la iteración
% m - número de iteraciones
% iden - posición donde se dibuja el resultado
% ejes - indica si se desea ver los ejes de la figura
% val_ejes - vector que indica en qué región dibujar el atractor
% mov - rango de variación del parámetro del que depende continuamente
% el movimiento


% Se declaran las variables necesarias de modo global para poder
% ser usadas por otros programas.

global p;
global dete;
```

```
% En d se guarda el nmero de transformaciones afines.

d=max(size(ifs2));


% Control del cierre de la ventana.

dete=1;

ini=0;
inc=(mov(2)-mov(1))/100;
param_state='c';
param_alpha=mov(1);

while ishandle(iden) && dete==1
    p_con=get(p,'Value');
    if p_con==1
        waitfor(p,'Value',0);
    end


% Cálculo del atractor para cada paso de tiempo

    puntos=zeros(m,2);



% Para cada valor del parámetro mov se define el vector de probabilidades


    for i=1:d aux=pr{i};
        pr_v(i)=subs(aux,param_alpha);
    end



% Para cada valor del parámetro mov se define el conjunto de
% transformaciones afines

    for i=1:d
        aux=ifs2{i}{1}; trans(i,1,1)=subs(aux,param_alpha);
        aux=ifs2{i}{2}; trans(i,2,1)=subs(aux,param_alpha);
```

```
        aux=ifs2{i}{3}; trans(i,1,2)=subs(aux,param_alpha);
        aux=ifs2{i}{4}; trans(i,2,2)=subs(aux,param_alpha);
        aux=ifs2{i}{5}; trans(i,3,1)=subs(aux,param_alpha);
        aux=ifs2{i}{6}; trans(i,3,2)=subs(aux,param_alpha);
    end

    for i = 1:m


% Se elige aleatoriamente la transformación según sus
% probabilidades

        j_ale=uint8(randp(pr_v,1));

        aux=reshape(trans(j_ale,:,1:2),3,2);
        puntos(i,:)=pto_inicial*aux;


% Se transforma el punto inicial

        pto_inicial=[puntos(i,:),1];
    end

    if ishandle(iden)
        figure(iden);
        plot(puntos(50:m,1),puntos(50:m,2),'.b','MarkerSize',5);
        hold off
        axis(val_ejes);
        if ejes==0
            axis off;
        else
            axis on;
        end
        set(gcf,'CloseRequestFcn',{@my_close,iden});
    end

    if param_state=='c'
        param_alpha=param_alpha+inc;
        if param_alpha>mov(2)
```

```
            param_alpha=param_alpha-2*inc;
            param_state='d';
        end

    elseif param_state=='d'
        param_alpha=param_alpha-inc;
        if param_alpha<mov(1)
            param_alpha=param_alpha+2*inc;
            param_state='c';
        end
    end
end

set(gcf,'CloseRequestFcn','default');
close(iden);
```

6 Interfaz gráfica

Para poder visualizar de modo gráfico los algoritmos implementados se ha realizado una interfaz gráfica mediante *MATLAB*. Este programa nos proporciona una gran versatilidad para dicha labor mediante el entorno de desarrollo *GUIDE*.

6.1. Manejo de la interfaz gráfica

Para ejecutar la interfaz gráfica, se debe introducir en el intérprete de comandos de *Matlab*

$$>> prueba\_interfaz$$

Antes de ejecutar dicha instrucción debemos situarnos en el directorio donde está contenida la interfaz gráfica de usuario.
Ejecutada la interfaz gráfica se nos abre una ventana como la que se ilustra en la figura 6.1
Esta pantalla nos muestra los diferentes menús con los algoritmos y opciones explicadas en capítulos anteriores. Estos son:

Conjuntos de Julia y Mandelbort

Cuencas de atracción

Sistemas de funciones iteradas

Fractales en movimiento

Figura 6.1: Ventana de la pantalla de inicio de la interfaz gráfica

6.1.1. Conjuntos de Julia y Mandelbrot

Si escogemos la primera opción *(Conjuntos de Julia y Mandelbrot)*, se muestra una pantalla como la siguiente:

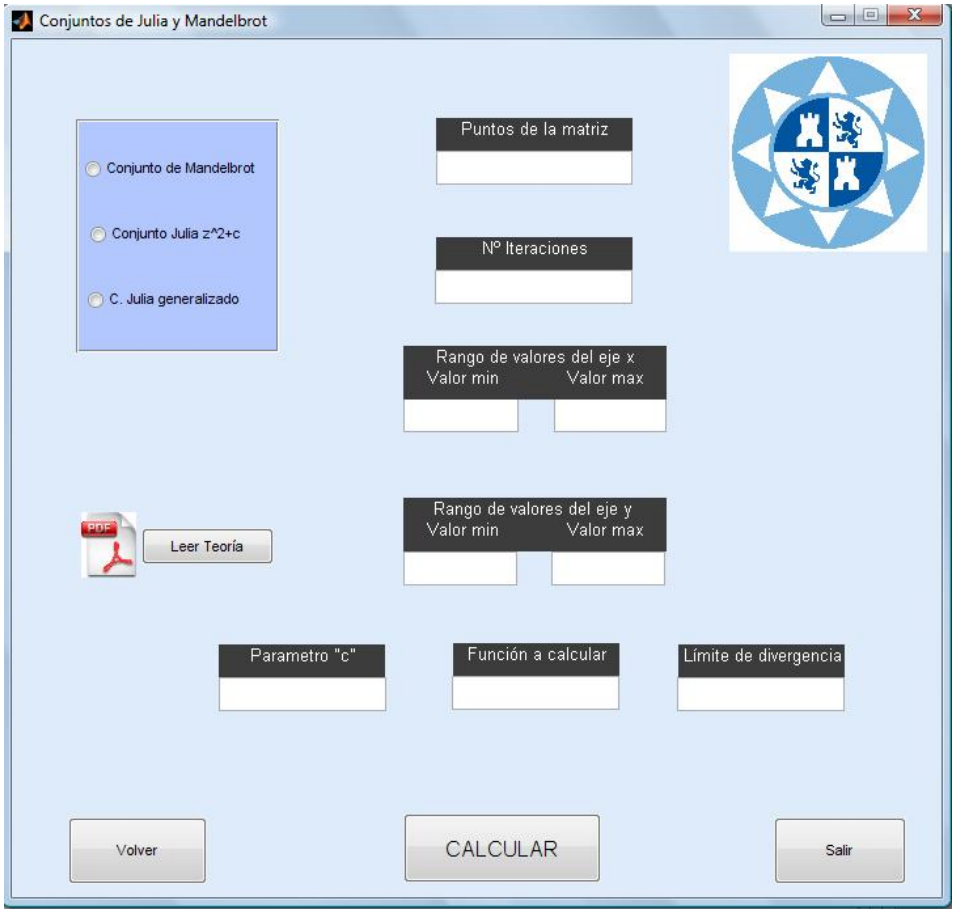


Figura 6.2: Ventana de la pantalla Conjuntos de Julia y Mandelbrot

Manejo de la pantalla Conjuntos de Julia y Mandelbrot

A continuación se detallan las diferentes partes y opciones de la pantalla *Conjuntos de Julia y Mandelbrot*

Conjuntos :

En este menú se muestran los tres conjuntos que se pueden calcular *(Conjunto de Mandelbrot, Conjunto de Julia $z^2 + c$ y Conjunto de Julia generalizado)*.

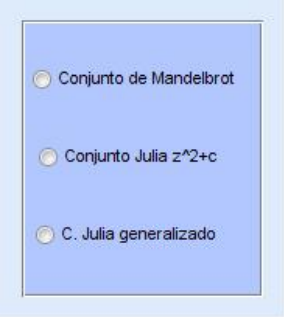


Figura 6.3: Conjuntos de Julia y Mandelbrot que se pueden calcular

Parámetros de entrada :

En estas casillas se introducen los datos necesarios para el cálculo de cada conjunto. Dependiendo del conjunto escogido, se deberán rellenar unos datos u otros.

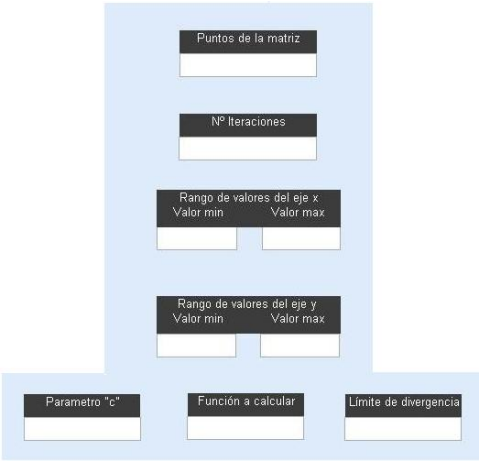


Figura 6.4: Parámteros de entrada de los Conjuntos de Julia y Mandelbrot

Leer teoría :

Pulsando este botón, se abre el archivo en formato *.pdf* correspondiente al capítulo 2 donde se encuentra el contenido teórico al que corresponde esta pantalla.

Figura 6.5: Acceso directo a la teoría de Conjuntos de Julia y Mandelbrot

Volver :

Con este botón se vuelve a la pantalla principal del programa.

Figura 6.6: Botón para retonran a la pantalla pricipal

Salir :

Botón para salir de la aplicación.

Figura 6.7: Botón para salir del programa

Calcular :

Una vez que se han introducido todos los datos para realizar el cálculo del conjunto se debe pulsar este botón para visualizar el resultado.

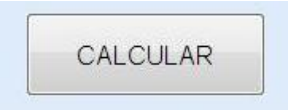


Figura 6.8: Botón para calcular el conjunto

Ejemplos de Conjuntos de Julia y Mandelbrot

A continuación se muestran algunos ejemplos de *Conjuntos de Julia y Mandelbrot* y su solución. En este caso vamos a calcular un *Conjunto de Julia* $z^2 + c$ y tambien mostraremos como se rellenan los otros casos y los posibles errores dados.

Como se observa en las figuras 6.9, 6.11 y 6.12 los campos que no es necesario rellenar están coloreados y anulados, para así facilitar el proceso y evitar posibles errores en el programa.

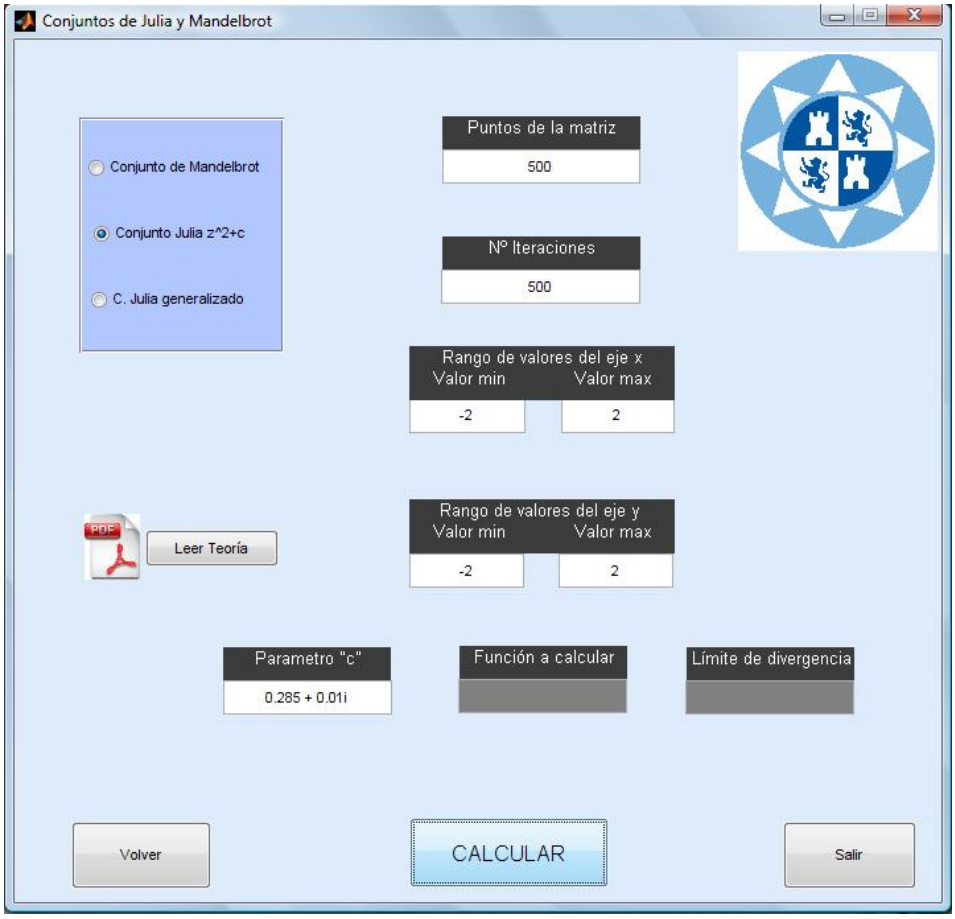


Figura 6.9: Ejemplo de cálculo de Conjunto de Julia $z^{`2} + c$

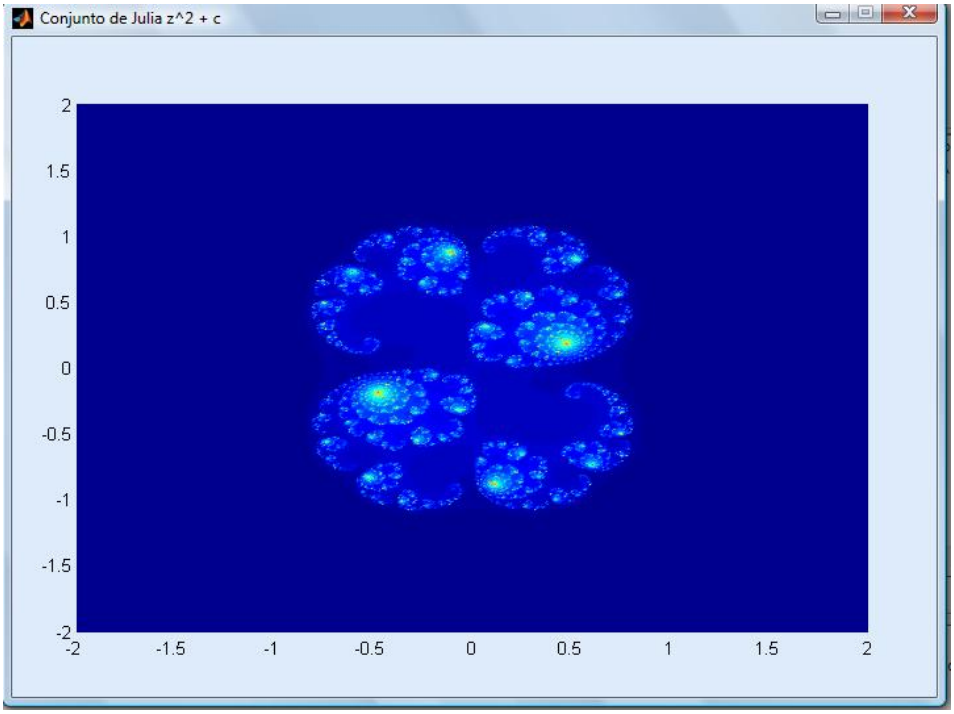


Figura 6.10: Solución de cálculo de Conjunto de Julia $z^{`2} + c$

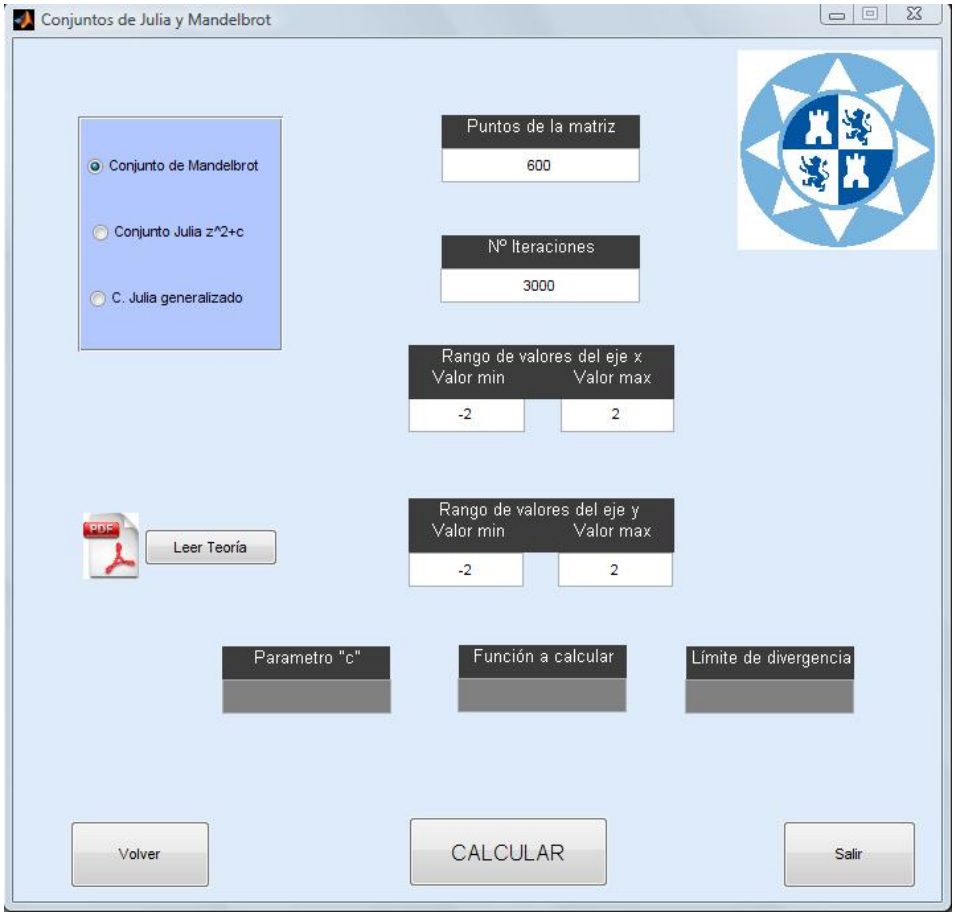


Figura 6.11: Ejemplo de cálculo de Conjunto de Mandelbrot

Figura 6.12: Ejemplo de cálculo de Conjunto de Julia generalizado

En caso de no rellenar completamente los parámetros de entrada, el programa avisará de que falta algún dato, al igual que si se han rellenado con datos erróneos.

Figura 6.13: Mensaje de error

6.1.2. Cuencas de atracción

Para calcular *cuencas de atracción*, tenemos la siguiente pantalla para rellenar los parámetros necesarios. La ventana se muestra a continuación:

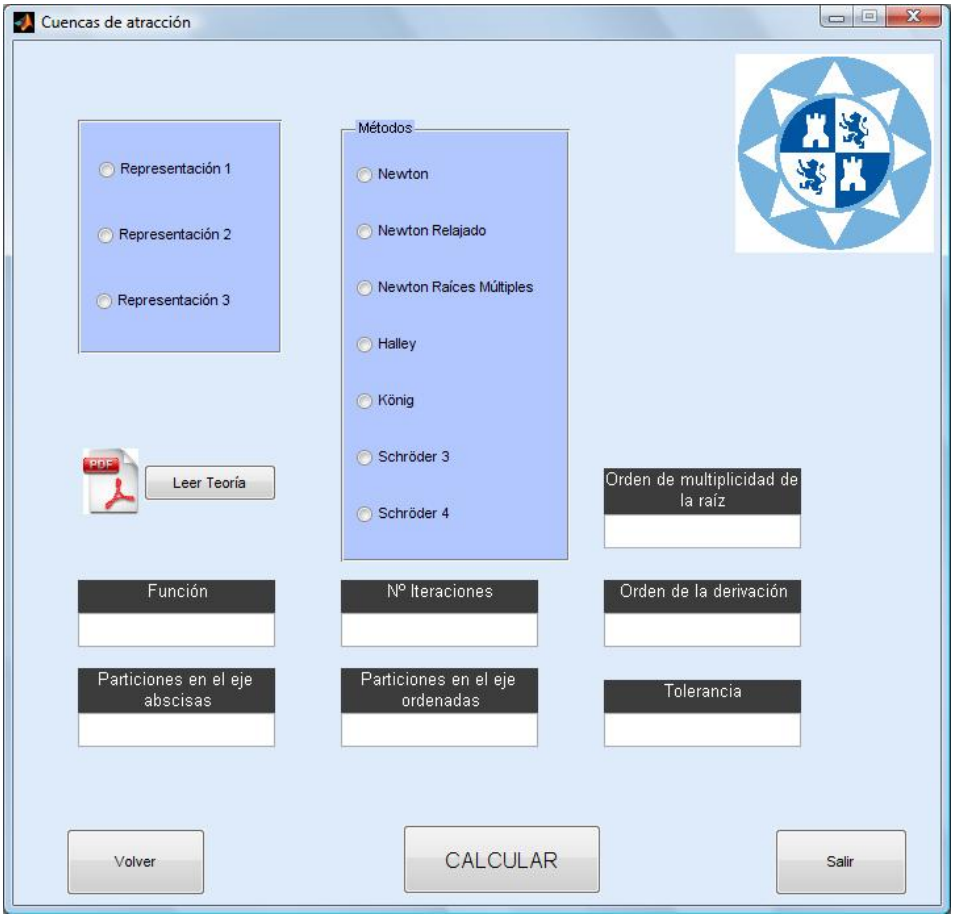


Figura 6.14: Ventana de la pantalla Cuencas de atracción

Manejo de la pantalla Cuencas de atracción

A continuación se detallan las diferentes partes y opciones de la pantalla *Cuencas de atracción*

Representaciones :

En este menú se muestran las tres versiones que se pueden calcular.

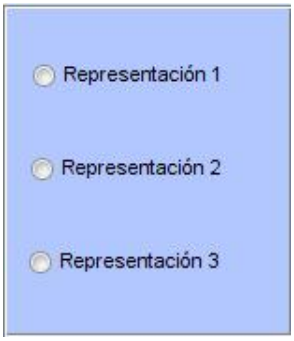


Figura 6.15: Las diferentes versiones de las cuencas de atracción que se pueden calcular

Leer teoría :

Al igual que en las otras ventanas, el botón de *Leer Teoría*, (figura 6.5), abre un archivo *.pdf* que corresponde al capítulo 3.

Volver :

Botón representado en la figura 6.6 para acceder a la pantalla principal de la interfaz.

Salir :

(Figura 6.7). Botón para salir de la aplicación.

Calcular :

Una vez estén todos los datos correctamente introducidos, pulsamos este botón (figura 6.8) para visualizar el resultado.

Métodos :

Listado de los métodos para realizar los cálculos de las cuencas de atracción:

Figura 6.16: Lista de métodos para el cálculo de las cuencas de atracción

Parámetros de entrada :

Datos de entrada de los métodos que se deben rellenar.

Figura 6.17: Parámteros de entrada de Cuencas de atracción

Ejemplos de Cuencas de atracción

Aquí veremos diferentes ejemplos de como rellenar satisfactoriamente los datos para realizar los cálculos de una manera correcta. A continuación se muestran las pantallas correspondientes, al igual que el resultado obtenido.

En primer lugar, calcularemos la primera versión (*Representación 1*). La función elegida es $6x^3 + 2x^2 - 5x$, el número de iteraciones es 500, el *Método de Newton*, una tolerancia de 10^{-9} y 100 particiones en ambos ejes.
Esta función tiene 3 raíces (0, $-1{,}0946$ y 0,7613), cuyas cuencas se pueden observar en la Figura 6.19 como los puntos donde la función converge con mayor velocidad, es decir, utilizan un menor número de iteraciones, que están representados por el color azul más oscuro.

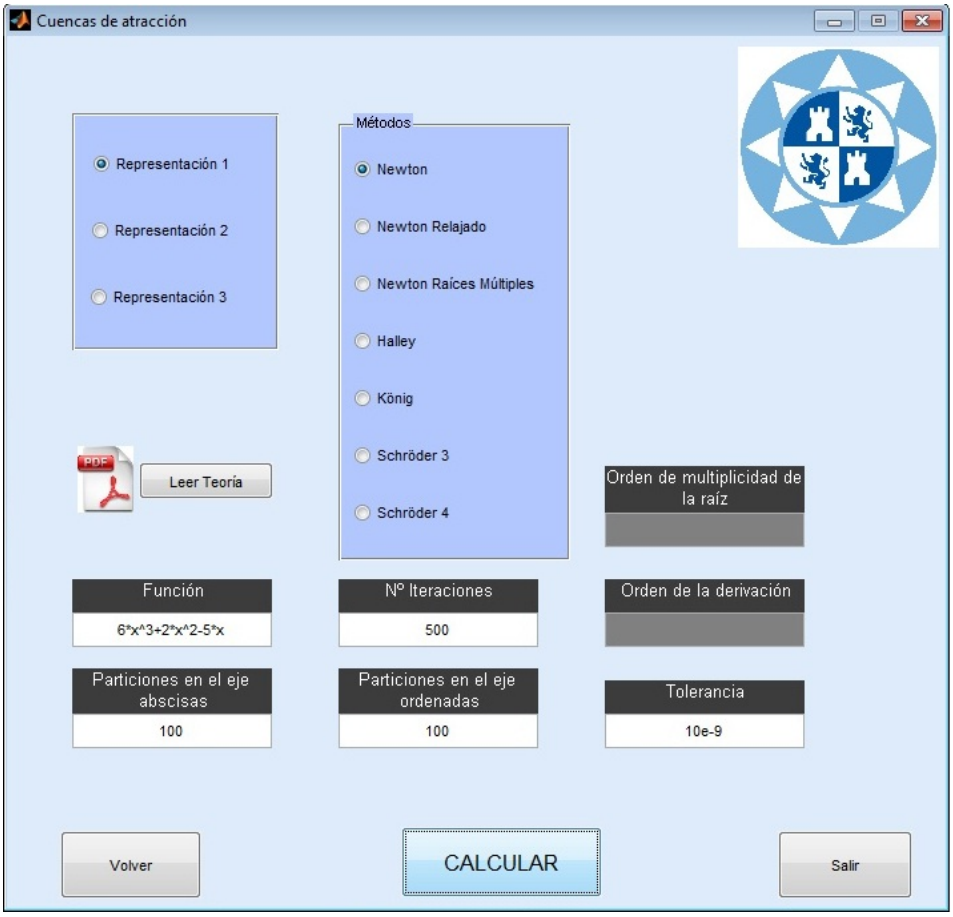


Figura 6.18: Ejemplo de la primera versión de Cuencas de atracción

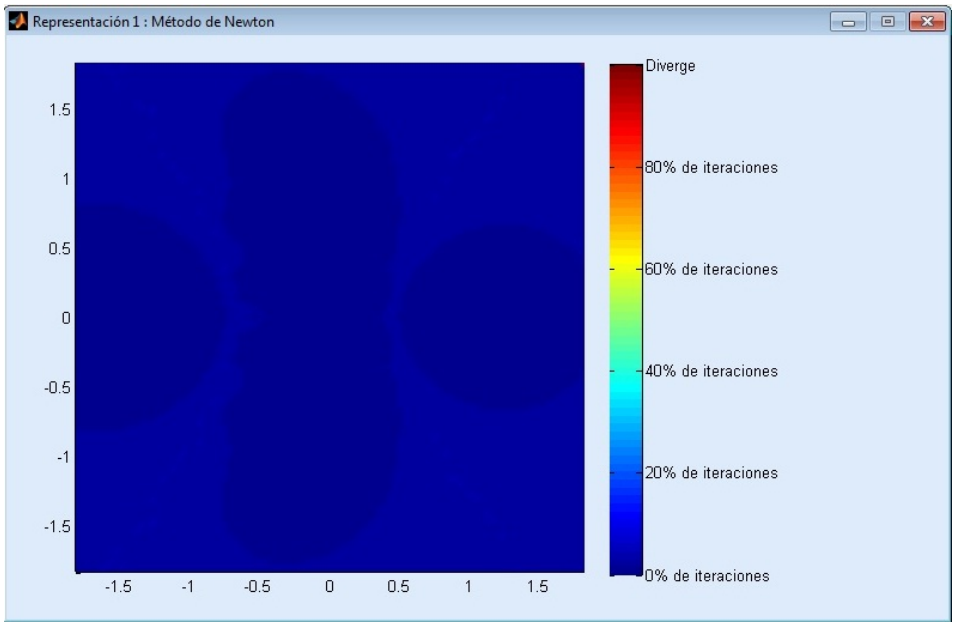


Figura 6.19: Solución de Representación 1, $f(x) = 6x^3 + 2x^2 - 5x$

El siguiente ejemplo es relativo a la segunda versión de las cuencas de atracción (*Representación 2*). Esta vez, usaremos el *Método de Halley*, la función es $f(x) = x^3 - 1$, $n = 1000$ iteraciones, una tolerancia de 10^{-9}, 150 particiones en el eje de abscisas y 150 en el de ordenadas.
El resultado de esta representación usando la función $x^3 - 1$ es similar al *Problema de Cayley*, explicado en capítulos anteriores y cuya solución se aprecia en la Figura 3.1. A la derecha de la imagen se pueden observar las raíces de la función y el color que las representa.

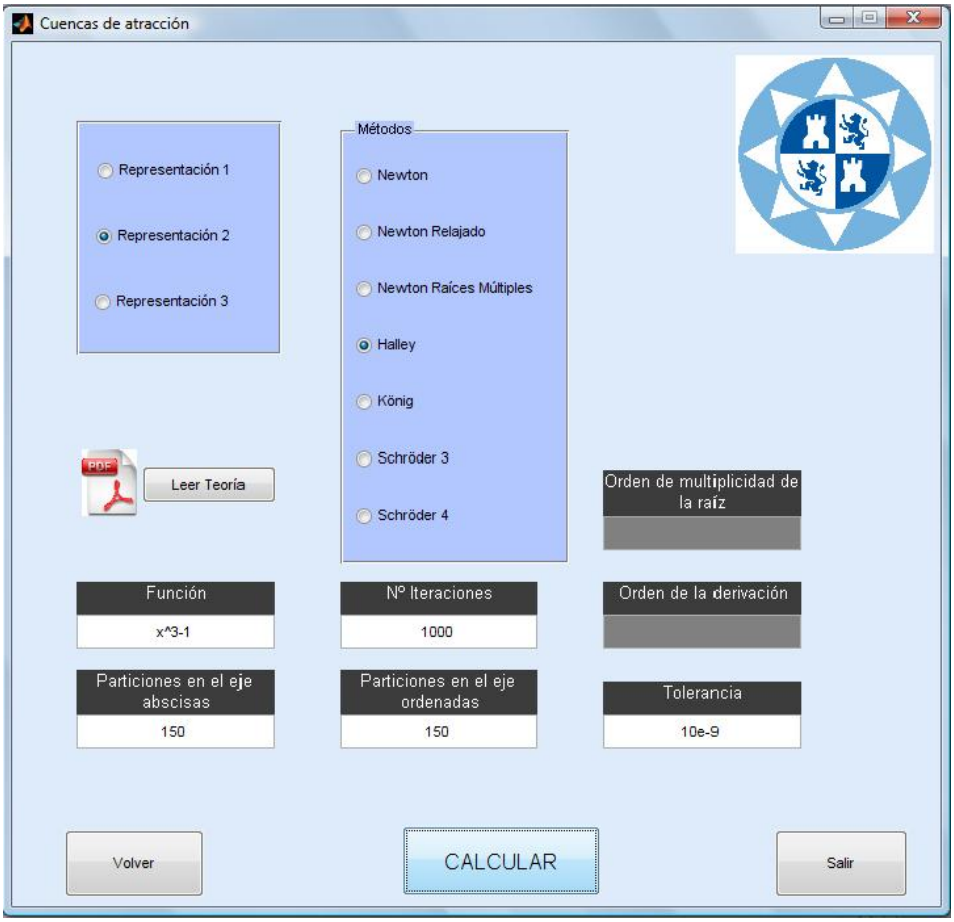


Figura 6.20: Ejemplo de la segunda versión de Cuencas de atracción

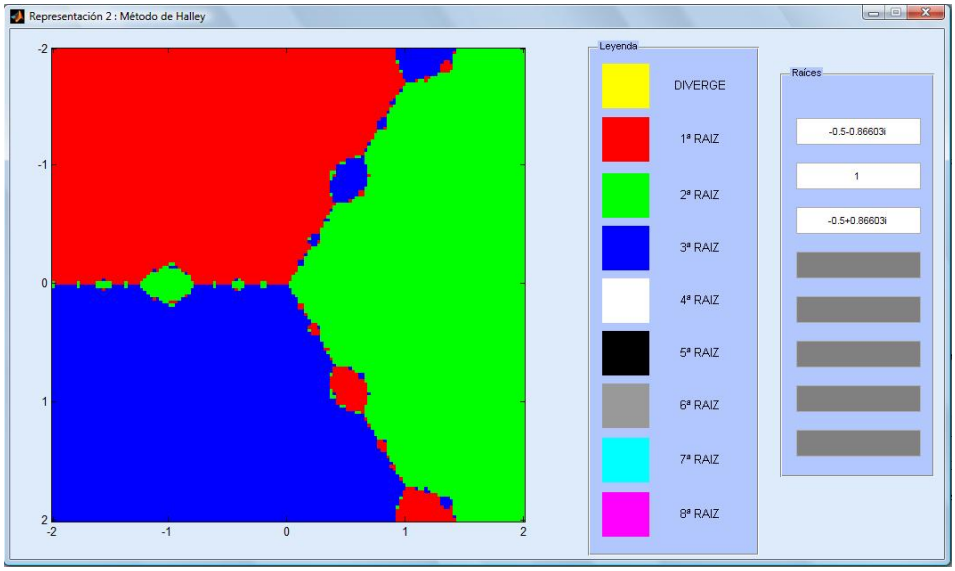


Figura 6.21: Solución de Representación 2, $f(x) = x^3 - 1$

Para la tercera versión (*Representación 3*) utilizaremos en modo de ejemplo la función $f(x) = x^3 - 1$ calculado gracias al *Método de Halley*, $n = 10$ iteraciones, 150 particiones en ambos ejes y una tolerancia igual a 10^{-9}.

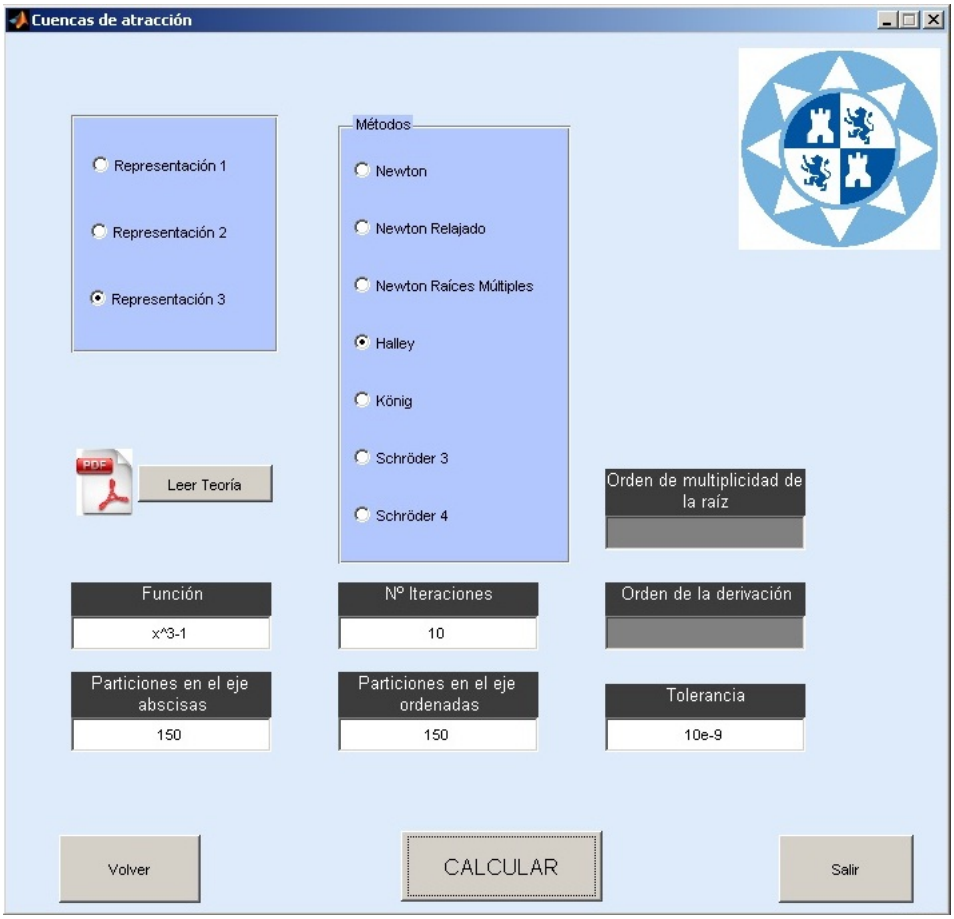


Figura 6.22: Ejemplo de la tercera versión de Cuencas de atracción

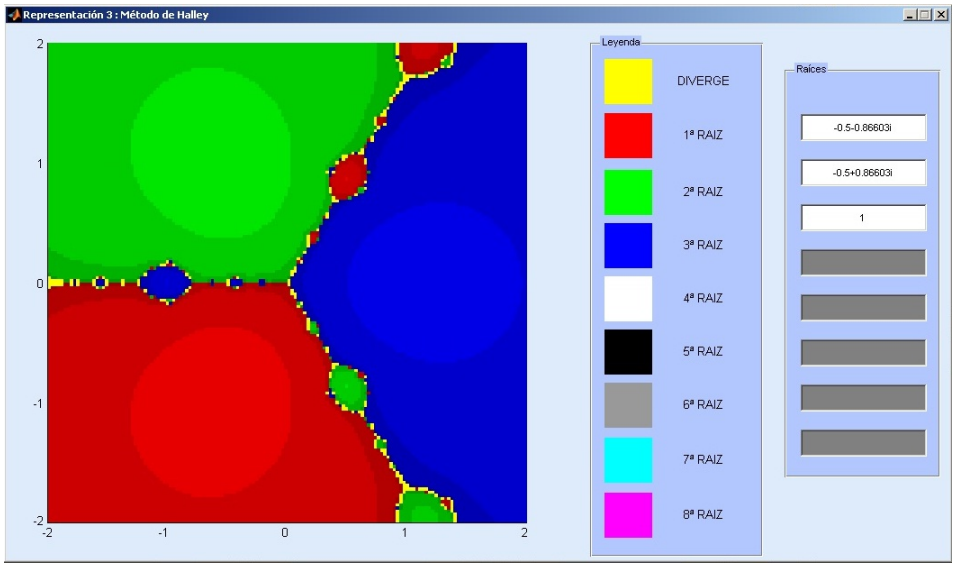


Figura 6.23: Solución de Representación 3, $f(x) = x^3 - 1$

Como en las otras pantallas, en caso de no rellenar o hacerlo con datos incorrectos, el programa mostrará mensajes de error informando acerca de aquello que impide realizar correctamente la operación pertinente.

6.1.3. Sistemas de funciones iteradas

Para calcular *Sistemas de funciones iteradas* tenemos la siguiente pantalla:

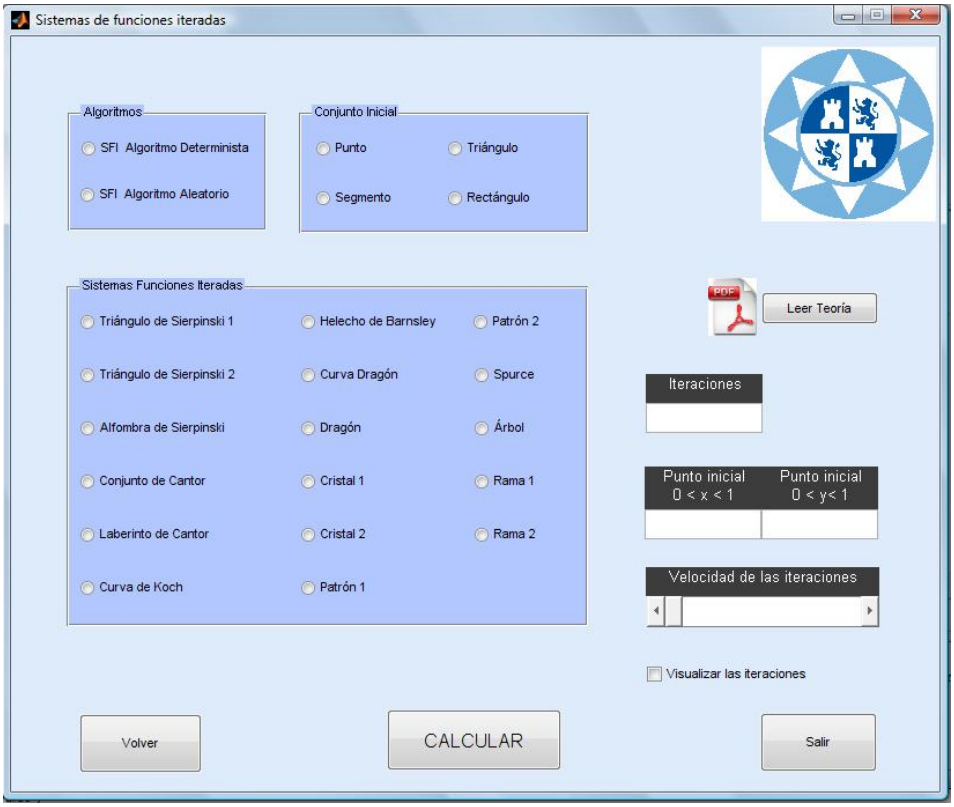


Figura 6.24: Ventana de la pantalla Sistemas de funciones iteradas

Manejo de la pantalla Sistemas de funciones iteradas

En este apartado se detallan las diferentes partes de la ventana para el cálculo de *Sistemas de funciones iteradas*.

Algoritmos :

Aquí se escoge el tipo de algoritmo que se va a utilizar *(Determinista o Aleatorio)*

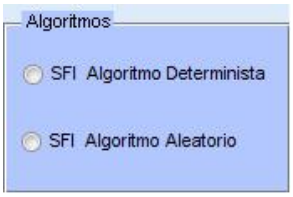


Figura 6.25: Los tipos de algoritmo para calcular SFI

Conjuntos iniciales :

Para elegir el conjunto inicial del *SFI* tenemos el siguiente panel de opciones:

Figura 6.26: Panel con los diferentes conjuntos iniciales

Sistemas de Funciones Iteradas :

En el siguiente panel de elección, se encuentran los distintos sistemas de funciones iteradas que se pueden representar:

Figura 6.27: Sistemas de funciones iteradas que se pueden visulaizar

Parámetros de entrada :

Para rellenar los datos de entrada para los *sistemas de funciones iteradas*:

Figura 6.28: Parámetros de entrada para visulaizar los Sistemas de funciones iteradas

Leer teoría :

Al igual que en las otras ventanas, el botón de *Leer Teoría*, (figura 6.5), abre un archivo *.pdf* que corresponde al capítulo 4.

Volver :

Botón representado en la figura 6.6 para acceder a la pantalla principal de la interfaz.

Salir :

(Figura 6.7). Botón para salir de la aplicación.

Calcular :

Una vez estén todos los datos correctamente introducidos, pulsamos este botón (figura 6.8) para visualizar el resultado.

Ejemplos de Sistemas de Funciones Iteradas

Veremos como insetar correctamente los datos para visualizar correctamente *Sistemas de Funciones Iteradas* con diversos ejemplos explicativos.

En primer lugar calculemos un *SFI Algoritmo Determinista*, con un *Conjunto inicial : Segmento*, 10 iteraciones y como *Sistemas de Funciones Iteradas* escogemos *Laberinto de Cantor* :

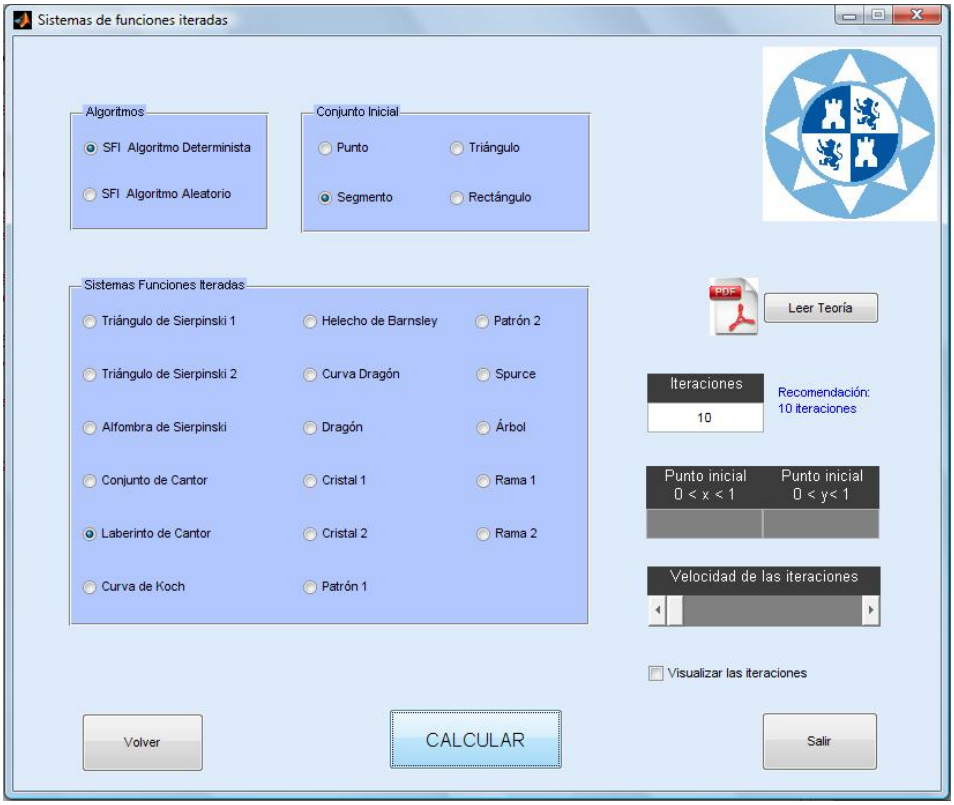


Figura 6.29: Ejemplo de SFI Algoritmo Determinista

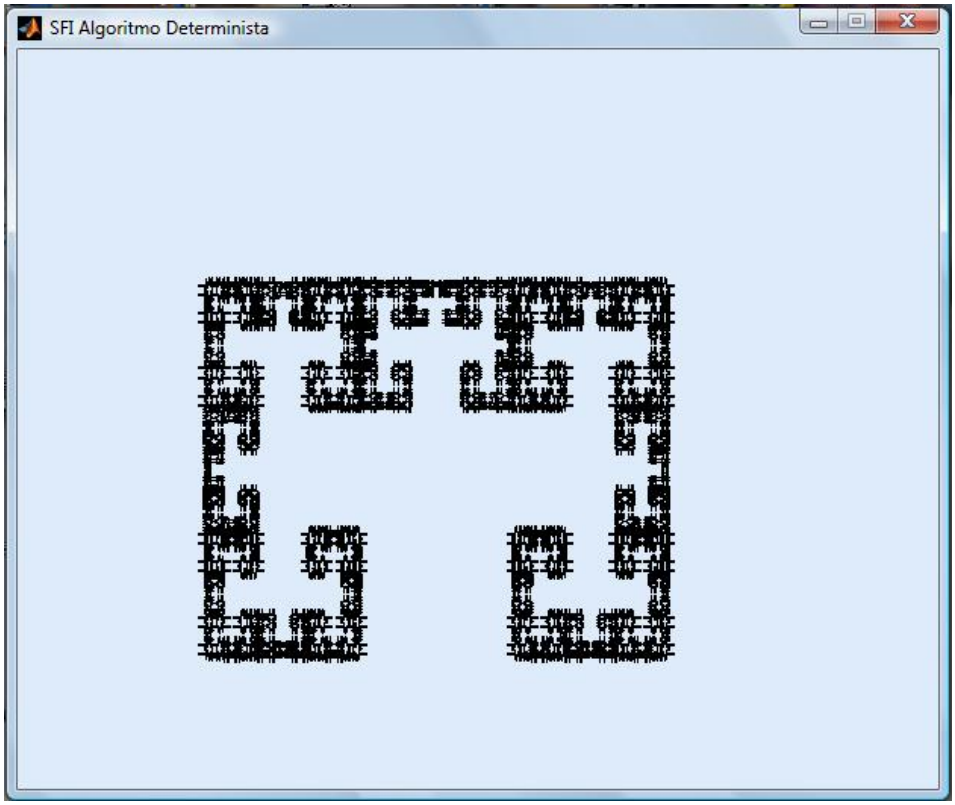


Figura 6.30: Representación del Laberinto de Cantor, SFI Algortimo Determinista

El siguiente ejemplo corresponde a un *SFI Algoritmo Aleatorio*, como *Conjunto inicial: Punto*, 5000 iteraciones y *Sistema de Funciones Iteradas: Helecho de Barnsley*, punto inicial x : 0,5 y punto inicial y : 0,8. En este ejemplo no se visualizan las iteraciones por su dificultad en plasmar dicha iteraciones en imágenes.

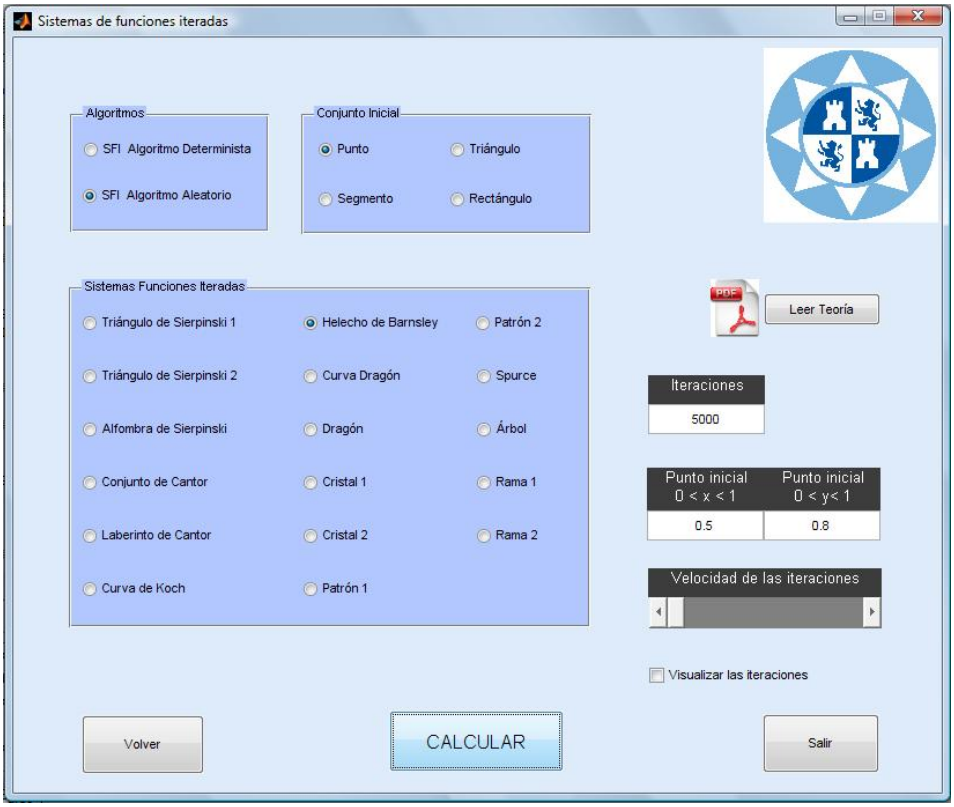


Figura 6.31: Ejemplo de SFI Algoritmo Aleatorio

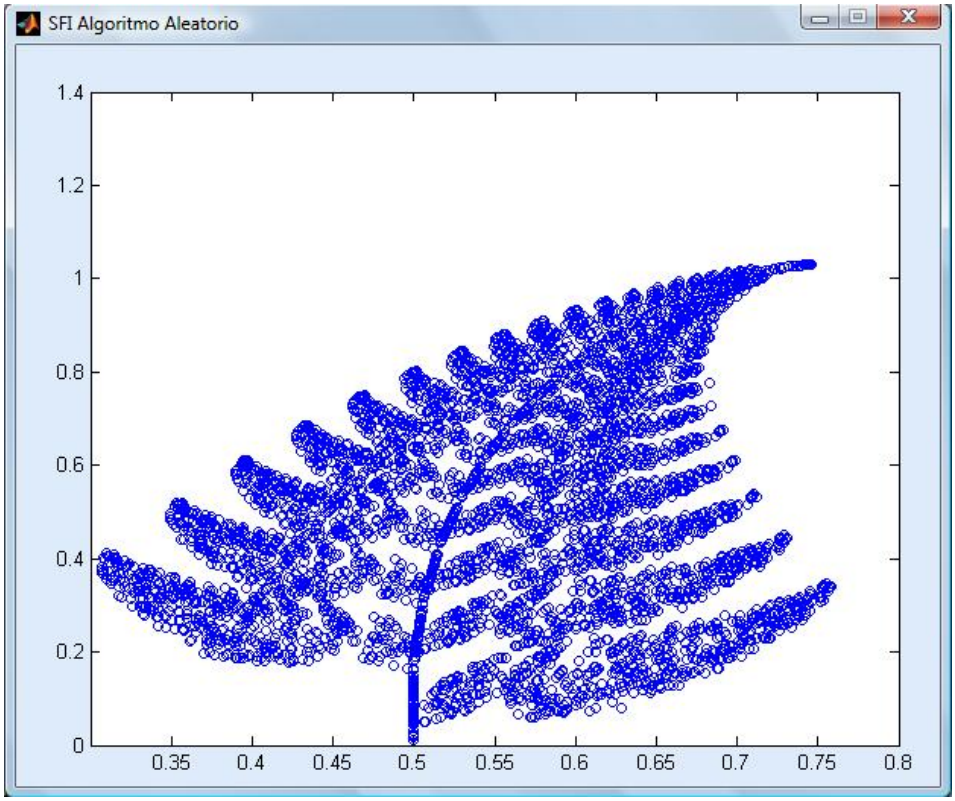


Figura 6.32: Representación del Helecho de Barnsley, SFI Algortimo Aleatorio

Como en las ventanas anteriores, en caso de intentar calcular sin haber introducido los datos correctamente aparecerán en pantalla un mensaje de error determinando la causa de este.

6.1.4. Fractales en movimiento

La pantalla principal para visulaizar los *Fractales en movimiento* es la mostrada a continuación:

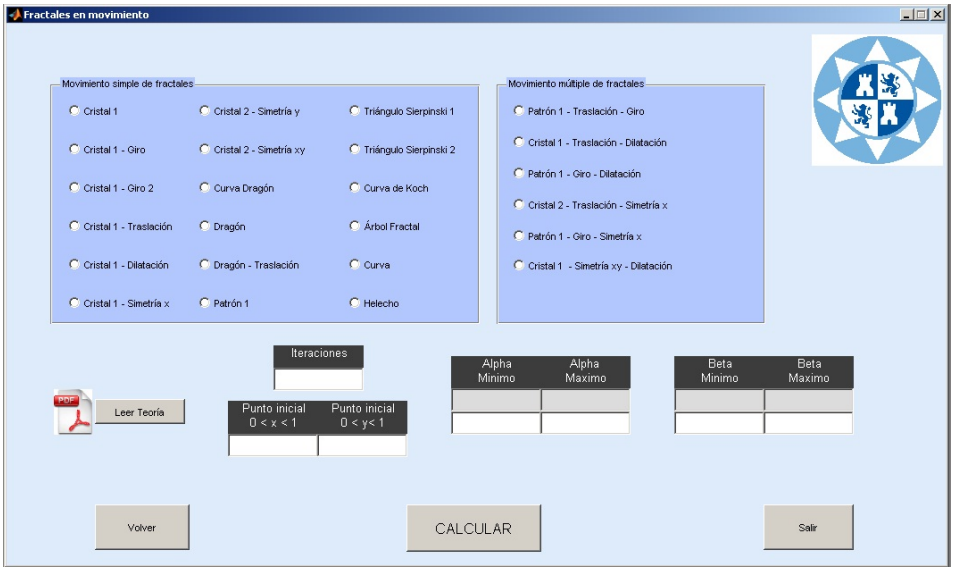


Figura 6.33: Ventana de la pantalla Fractales en movimiento

Manejo de la pantalla Fractales en Movimiento

La ventana de *Fractales en movimiento* se compone de las siguientes partes descritas a continuación:

Movimiento de fractales :

En estos recuadros se escogen el fractal y el movimiento al que viene asociado. Podemos elegir entre *Movimiento simple de fractales* y *Movimiento múltiple de fractales*

Figura 6.34: Movimientos simples de fractales que se pueden visualizar

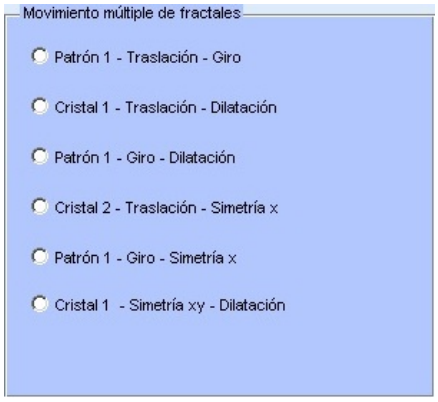


Figura 6.35: Movimientos múltiples de fractales que se pueden visualizar

Parámetros de entrada :

Datos de entrada para la visualización de los fractales en movimiento.

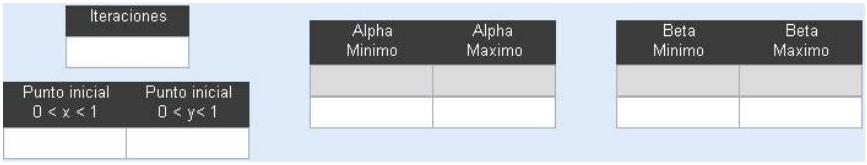


Figura 6.36: Parámetros de entrada para visualizar fractales en movimiento

Leer teoría :

Al igual que en las otras ventanas, el botón de *Leer Teoría*, (figura 6.5), abre un archivo *.pdf* que corresponde al capítulo 5.

Volver :

Botón representado en la figura 6.6 para acceder a la pantalla principal de la interfaz.

Salir :

(Figura 6.7). Botón para salir de la aplicación.

Calcular :

Una vez estén todos los datos correctamente introducidos, pulsamos este botón (figura 6.8) para visualizar el resultado.

Ejemplos de Fractales en Movimiento

En las siguientes pantallas se observa un ejemplo de ventana principal preparada para la visualización de un fractal en movimiento simple y en las siguientes diferentes momentos del movimiento de éste para observar su funcionamiento.

En este ejemplo, la opción escogida es: *Cristal 1 - Giro 2*, con 5000 iteraciones, 0,7 como punto inicial en ambos ejes, y los valores de *alpha* predeterminados:

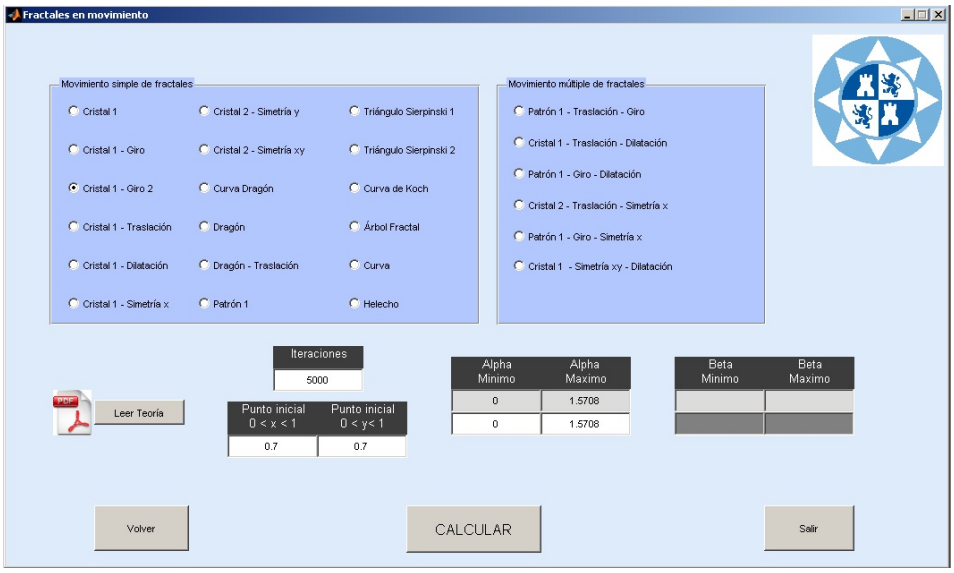
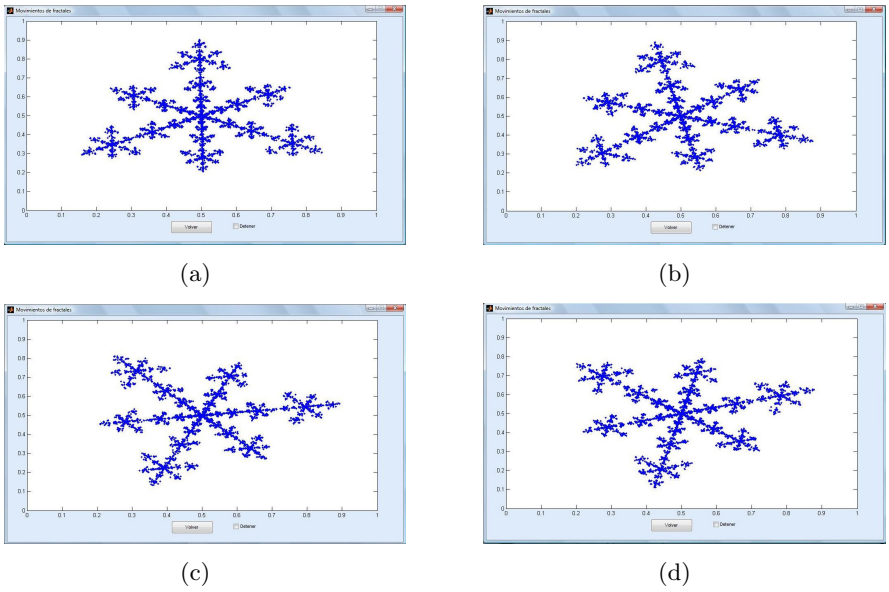


Figura 6.37: Ejemplo de cálculo de movimiento simple en Fractales en Movimiento

Las siguientes figuras muestran como la figura realiza el movimiento que hemos seleccionado.



(a) (b)

(c) (d)

Figura 6.38: Movimiento fractal simple

A continuación, se muestran los diversos movimientos múltiples de fractales y sus respectivas pantallas donde se han intriducido los parámetros:

En el primer movimiento, el fractal elegido *(Patrón 1)*, se desplaza mientras gira sobre sí mismo

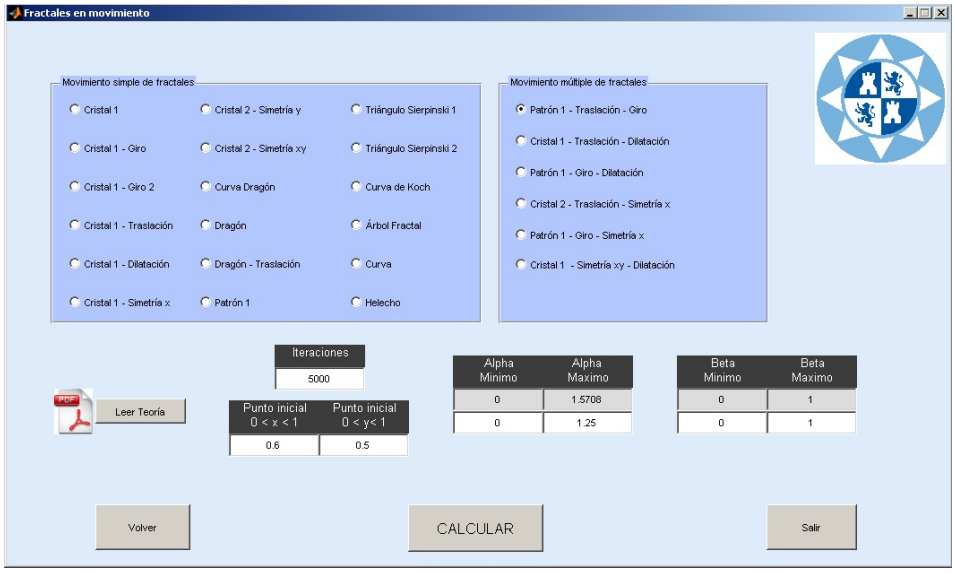
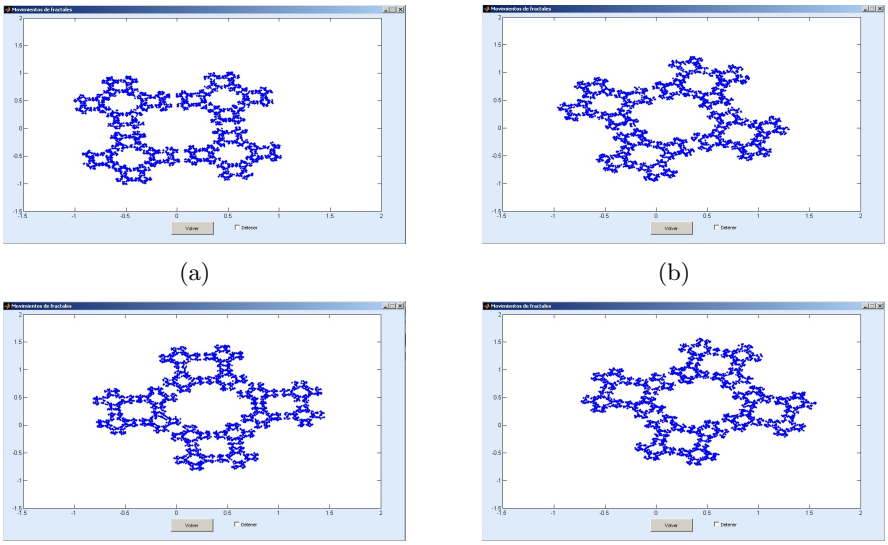


Figura 6.39: Ejemplo de cálculo de Patrón 1 Traslación-Giro



(a) (b)

(c) (d)

Figura 6.40: Movimiento Patrón 1 Traslación - Giro

El movimiento que se muestra a continuación reproduce al fractal *Cristal 1* trasaladándose mientras se dilata y se contrae.

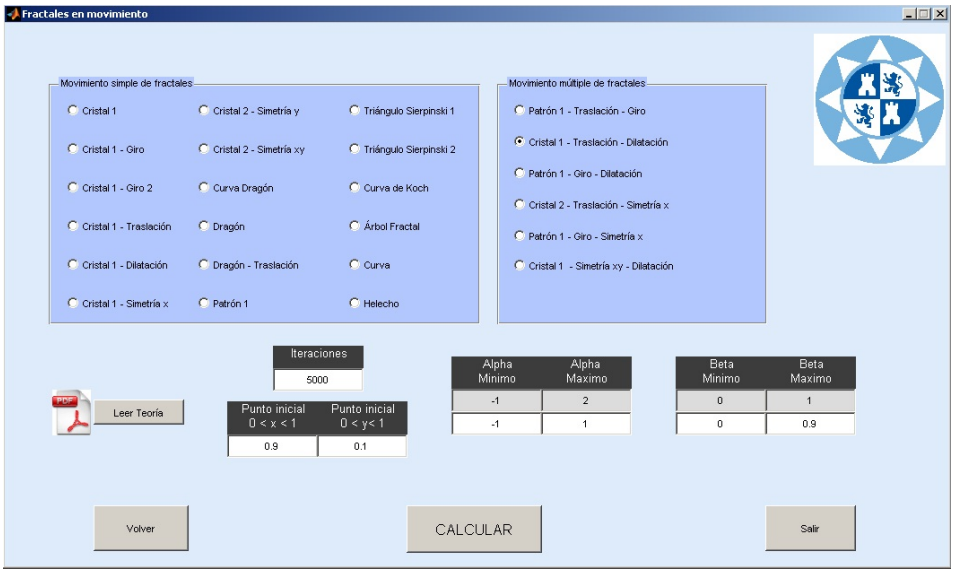
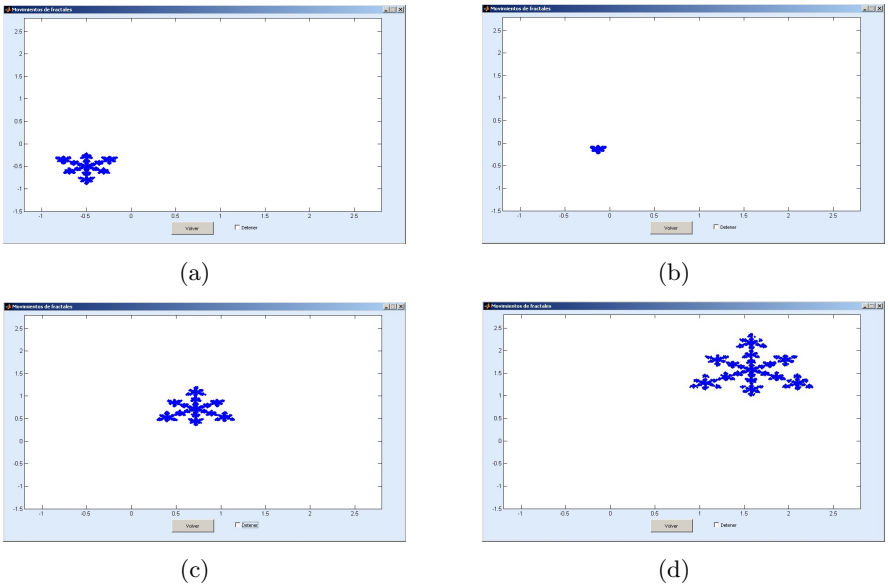


Figura 6.41: Ejemplo de cálculo de Cristal 1 Traslación-Dilatación



(a) (b)

(c) (d)

Figura 6.42: Movimiento Cristal 1 Traslación - Dilatación

Para este movimiento, el fractal elegido *(Patrón 1)*, gira sobre sí mismo a la vez que se dilata y se contrae.

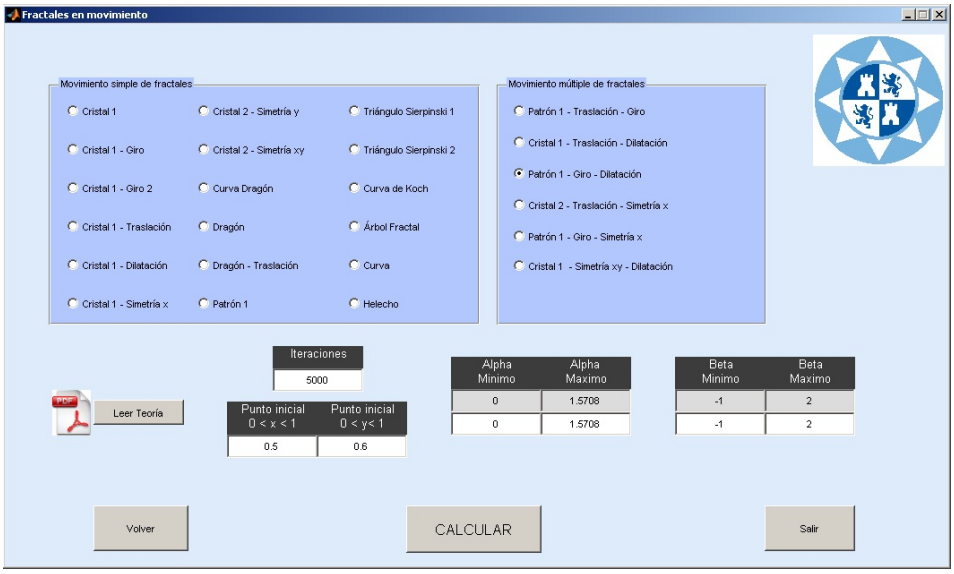
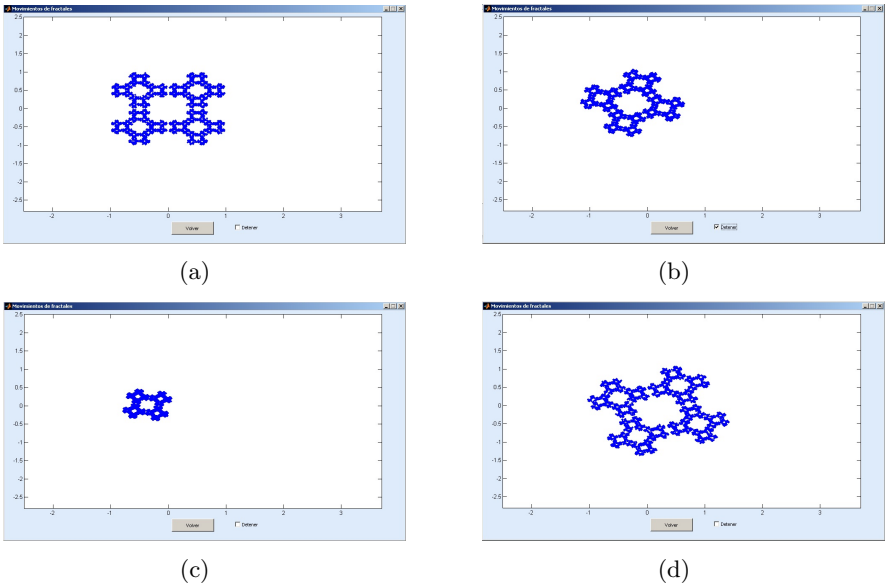


Figura 6.43: Ejemplo de cálculo de Patrón 1 Giro-Dilatación



(a) (b)

(c) (d)

Figura 6.44: Movimiento Patrón 1 Giro - Dilatación

Este movimiento presenta al fractal *Cristal 2* trasladándose y al misma vez ejerciendo simetría sobre el eje x.

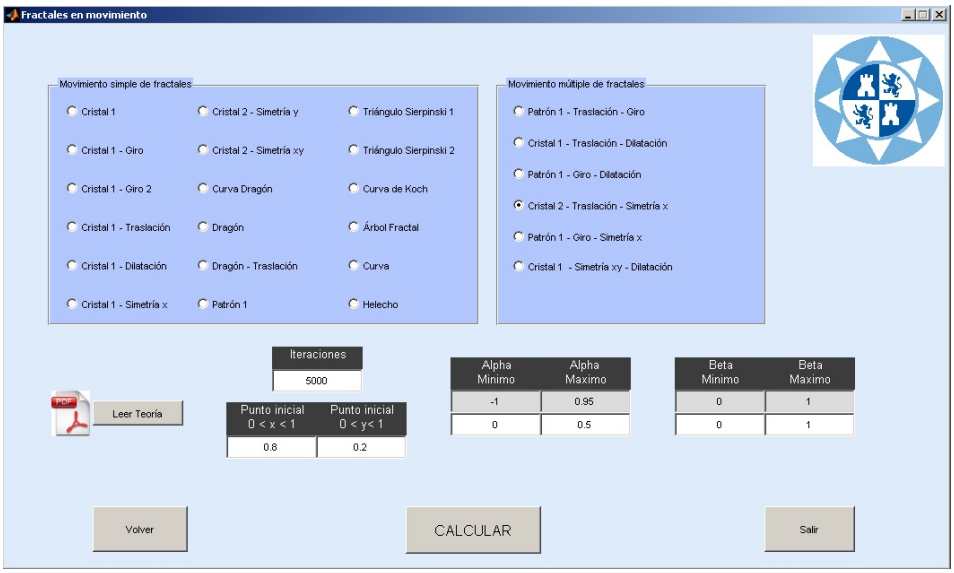
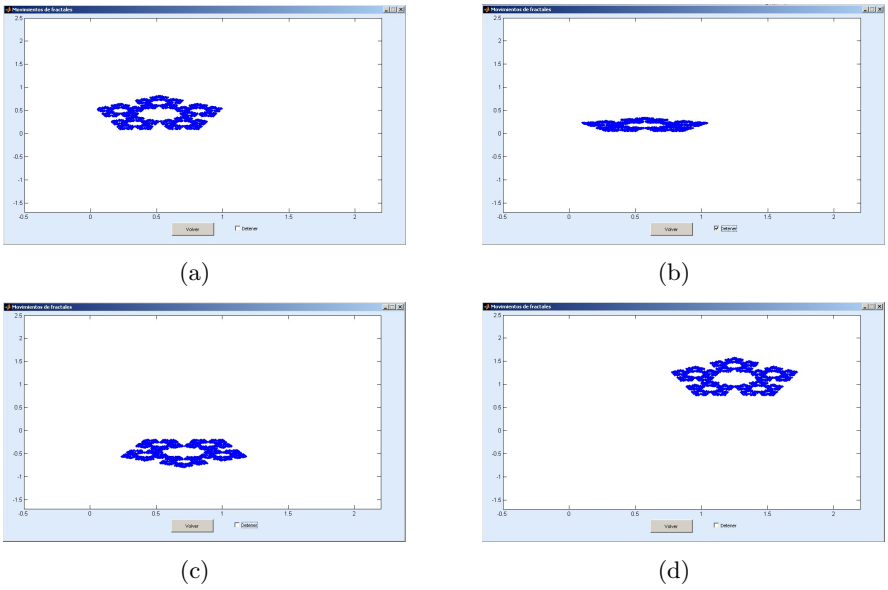


Figura 6.45: Ejemplo de cálculo de Cristal 2 Traslación-Simetría x



(a) (b)

(c) (d)

Figura 6.46: Movimiento Cristal 2 Traslación - Simetría x

Esta combinación de movimientos consiste en que el fractal *Patrón 1* gira sobre él mismo mientras que al mismo tiempo ejerce simetría sobre su eje x.

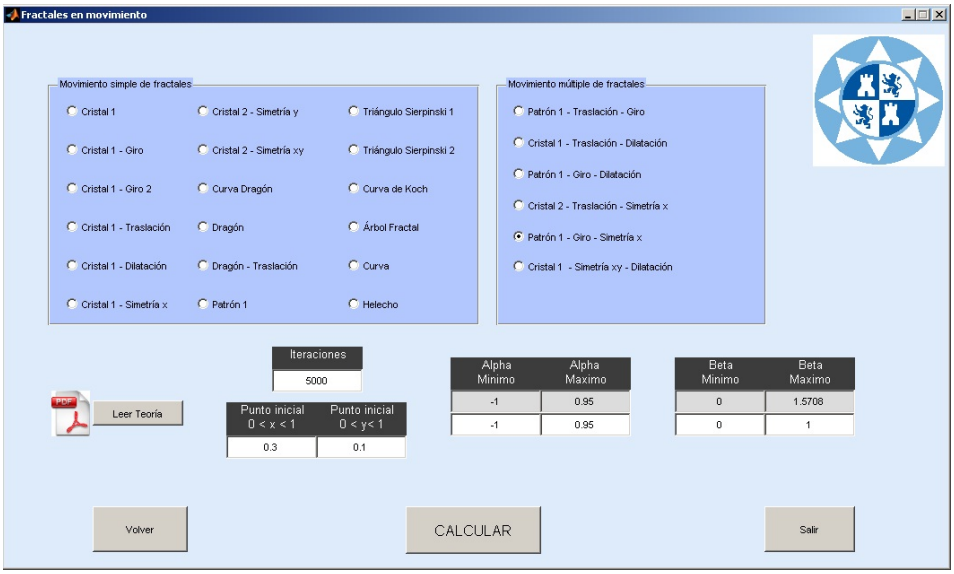
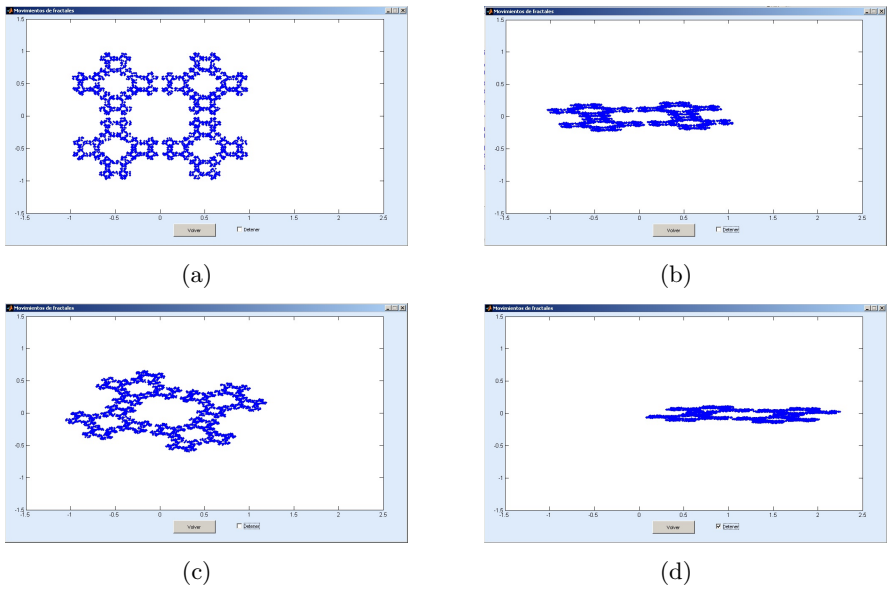


Figura 6.47: Ejemplo de cálculo de Patrón 1 Giro-Simetría x



(a) (b)

(c) (d)

Figura 6.48: Movimiento Patrón 1 Giro - Simetría x

La última combinación de movimientos consiste en que el fractal *Cristal 1* gira sobre él ejerce simetría sobre sus eje x e y a la misma vez que se dilata y contrae.

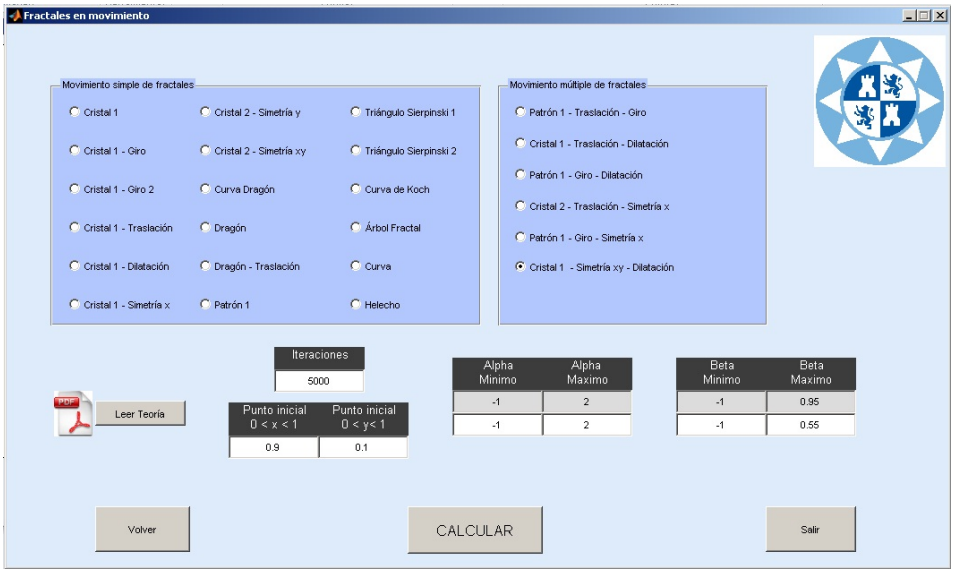
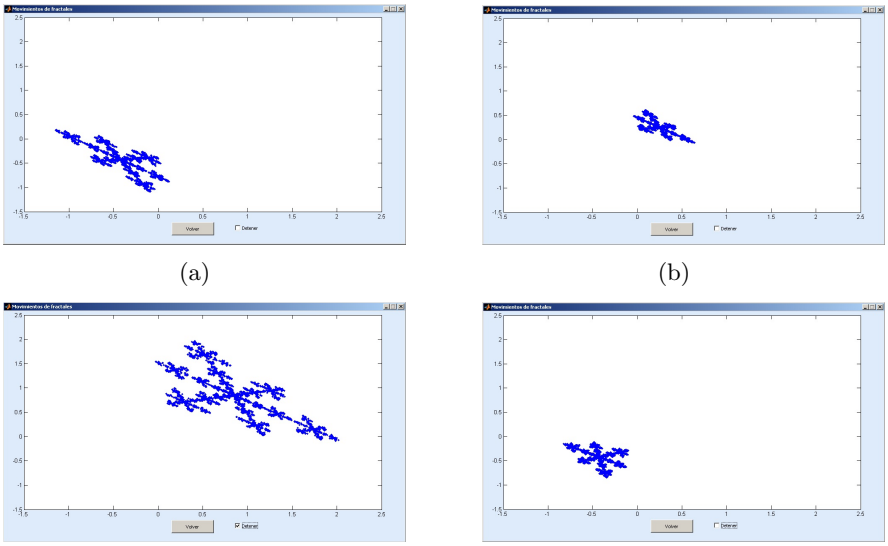


Figura 6.49: Ejemplo de cálculo de Cristal 1 Simetría xy - Dilatación



(a) (b)

(c) (d)

Figura 6.50: Movimiento Cristal 1 Simetría xy - Dilatación

En caso de que el programa detecte alguna irregularidad a la hora de introducir datos para la correcta visualización del resultado, éste mostrará mensajes de error indicando el motivo.

7 Conclusiones

En este proyecto hemos realizado el estudio de los conjuntos de Julia y Mandelbrot, de los sistemas de funciones iteradas, de los métodos numéricos en el plano complejo y de sus posibles aplicaciones. Además, basándonos en el algoritmo determinista y en el algoritmo aleatorio hemos generado fractales que combinan movimmientos y se han creado algunos ejemplos para su visualización y estudio.

Hemos desarrollado los programas en el lenguaje de programación Matlab para poner en práctica los algoritmos estudiados en la teoría, y para facilitar al usuario la ejecución de los programas también hemos creado una interfaz gráfica.

Este proyecto es útil para la generación de conjuntos de Julia y Mandelbrot y su posterior estudio. Al igual que para el caso de los métodos numéricos en el plano complejo y el estudio de las cuencas de atracción de las raíces con las tres representaciones para un mayor rango de carácteristicas a estudiar debido a los diversos parámetros disponibles. Una característica a tener en cuenta de los algoritmos fractales es que son muy lentos, con lo cual el factor tiempo resulta determinante para decantarse por uno u otro sistema.

Posibles trabajos a realizar en un futuro pueden ser implementar algoritmos con un menor tiempo de ejecución.También sería interesante crear nuevas representaciones para un mejor estudio de las cuencas de atracción de las raíces en los métodos numéricos en el plano complejo.

Otros posibles usos es la fabricación y mejora de las antenas fractales, para diseñar nuevos modelos que mejoren el ancho de banda de las antenas actualmente utilizadas.

Bibliografía

[1] Barnsley M., *Fractals Everywhere,* Londres, Academic Press, (1988).

[2] Guzman M. , Martin M.A., Moran M., Reyes M., *Estructuras Fractales y sus aplicaciones,* Editorial Labor, S.A..

[3] Barrallo Calonge J., *Geometra Fractal. Algortmica y presentación,* Edit.Anaya multimedia.

[4] Amat S., Bermúdez C., Busquier S., Leauthier P., Plaza S., *On the dynamics of some Newton's type iterative functions.*, Int. J. Comput. Msth., **86(4)**, 631-639, (2009).

[5] Plaza S., *Fractales y generación computacional de imagenes,* Dpto. de Matemática, Facultad de Ciencias, Universidad de Santiago de Chile.

[6] Arandiga F., Donat R., *Nonlinear Multi-scale Decomposition: The Approach of Harten A.*, Numerical Algorithms, 175-216, (2000).

[7] Barnsley M., *Fractal function and interpolation. Constructive aproximation*, 303-329, (1986).

[8] Barnsley M., *The desktop fractal design handlook,* San Diego Academic Press, 380, (1989).

[9] Guzman M., Martin M.A., Moran M., Reyes M., *Estructuras Fractales y sus aplicaciones*, Editorial Labor, (1993).

[10] Mandelbrot B., *How long is the coast of Britain? statical-self-similarity and fractional dimension*, Science, 636-638, (1967).

[11] Mandelbrot B., *Fractals: From, chance and dimension*, San Francisco Freeman W.H., (1982).

[12] Mandelbrot B., *The fractal geometry of nature*, Nueva York Freeman H., (1982).

[13] Harten A., *Discrete multiresolution analysis and generalized wavelets*, 153-192, (1993).

[14] Burgos J., *Algebra lineal*, Mc Graw-Hill., (1993).

[15] Lucas Martínez F.M., *Sistemas de funciones iteradas y aplicaciones*, P.F.C Universidad Politécnica de Cartagena., (2010).