La Columna de Matemática Computacional

Sección a cargo de

Tomás Recio

El objetivo de esta columna es presentar de manera sucinta, en cada uno de los números de La Gaceta*, alguna cuestión matemática en la que los cálculos, en un sentido muy amplio, tengan un papel destacado. Para cumplir este objetivo el editor de la columna (sin otros méritos que su interés y sin otros recursos que su mejor voluntad) quisiera contar con la colaboración de los lectores, a los que anima a remitirle (a la dirección que se indica al pie de página*[1]*) los trabajos y sugerencias que consideren oportunos.*

En este número...

Para este número de *La Gaceta* hemos solicitado la colaboración del profesor de la Universidad de La Rioja, Juan Luis Varona Malumbres. Juan Luis es alfarense (hasta el año 2001) y alfareño (en la actualidad, según el Diccionario de la RAE), doctor por la Universidad de Cantabria (con el inolvidable Chicho como director) y hombre de variados intereses, como el desarrollo del grupo de usuarios en castellano de TEX (www.cervantex.org).

Tras varios números de La Gaceta en los que esta columna ha estado dedicada a diversos aspectos computacionales de los números primos y a la Criptografía, la contribución del prof. Varona versa ahora sobre un tema muy diferente: la generación por ordenador de gráficos de fractales mediante el uso de algún programa de Cálculo Simbólico.

Como se verá en el artículo, esta tarea está lejos de ser automática (como suele ocurrir, por suerte para los matemáticos, cada vez que los ordenadores parecen capaces de automatizar ciertas cosas), sino que requiere determinadas astucias matemáticas preparatorias. Confiamos que el lector disfrute con la descripción de las mismas y con la belleza de los resultados.

[1]Tomás Recio. Departamento de Matemáticas. Facultad de Ciencias. Universidad de Cantabria. 39071 Santander. recio@matesco.unican.es

Representación gráfica de fractales mediante un programa de cálculo simbólico

por

Juan L. Varona

Introducción

No es el propósito de este artículo dar una visión general ni una teoría sobre fractales, ni mostrar magníficos dibujos elaborados quién sabe cómo. Hay muchos libros que se dedican a distintos aspectos del estudio de fractales. Por citar unos pocos, mencionemos [6, 11], en los que podemos encontrar bastantes ideas intuitivas, así como gráficos y explicaciones detalladas de cómo generarlos. Más avanzado es [7] (aunque también hace hincapié en los aspectos gráficos) y, aún más, [8, 3]; en particular, este último analiza en profundidad la iteración de funciones racionales, una de las mayores fuentes de fractales. En español, un interesante libro que muestra multitud de algoritmos y gráficos es [2]; mucho más contenido matemático tiene [9]. Queremos también citar [10], una traducción de un clásico de Mandelbrot, uno de los padres de la geometría fractal (sobre todo en el aspecto divulgador), y que, además, fue quien inventó el término *fractal*.

En este artículo, simplemente, nos dedicaremos a mostrar cómo lograr representaciones gráficas de fractales mediante uno de los potentes programas de cálculo simbólico disponibles actualmente. Veremos que es una tarea que implica cierta actividad matemática: no se podrá, en general, tratar de programar, directamente, alguno de los algoritmos que definen el fractal.

En pocas palabras, y sin profundidad, demos algunas ideas intuitivas de lo que se entiende normalmente por un fractal. Técnicamente, los fractales son conjuntos cuya dimensión de Hausdorff es *fraccionaria*. ¿Qué quiere esto decir? Centrándonos en los que se pueden representar en el plano, y teniendo en cuenta que una recta (o una línea curva) tiene dimensión 1, y el plano dimensión 2, un fractal sería un intrincado subconjunto del plano que no llega a tener superficie no nula, pero que tampoco la longitud es útil para medirlo, pues la longitud de cualquier trozo no trivial del fractal es infinita. Así, ni la longitud (medida 1-dimensional) ni la superficie (medida 2-dimensional) sirven como método útil para efectuar mediciones en el fractal; para ello, puede ser necesario, por ejemplo, utilizar una medida $(\log 4/\log 3)$-dimensional. (De todas formas, es bastante común que, aunque sepamos que algún objeto es un fractal, desconozcamos cuál es su dimensión.) En la práctica, los fractales suelen tener estructuras que se repiten por doquier y a cualquier escala (autosimilitudes), ya sea de manera exacta o con un aspecto similar: si nos aproximamos al fractal y lo miramos de mucho más cerca (como con un potente *zoom* de una cámara) volvemos a ver trozos del fractal que nos recuerdan lo ya observado antes. Y esto, tanto como queramos.

No me resisto a poner un ejemplo extraído de la naturaleza. Quizás alguno se haya preguntado por qué en los datos sobre países no suele aparecer la longitud de su costa. Simplemente, ¡porque no tiene sentido! Para medir la costa de un país tendríamos que establecer, en algún sentido, con qué detenimiento medimos. No es lo mismo medir sólo aproximadamente en un mapa, dedicarse a medir con cuidado la longitud de golfos y cabos, prestar atención a los contornos rocosos, los granos de arena, ...; dependiendo de nuestro afán, íbamos a obtener cantidades completamente diferentes. También es un fractal la superficie de la tierra, con sus diferentes elevaciones y depresiones (y, de hecho, se suelen utilizar estructuras fractales para lograr representaciones gráficas realistas de montañas). Así, tampoco tendría sentido decir cuál es la superficie de un país, pero esto tiene remedio: podemos medir la proyección sobre la esfera de radio el de la tierra.

Las representaciones gráficas de objetos fractales pueden tener gran belleza y colorido. Esto, unido a la cada vez mayor disponibilidad y potencia de los actuales ordenadores, ha motivado el enorme interés por los fractales en las últimas décadas del siglo XX y ha impulsado, indudablemente, su estudio teórico. Pero las bases de la teoría, debidas a Julia, Fatou y Hausdorff, son bastante anteriores, de comienzos del siglo XX (incluso a finales del siglo XIX ya había análisis abstractos de objetos que hoy llamamos fractales); lógicamente, ellos no pudieron ver ninguna representación gráfica de los objetos con los que estaban tratando, ni, posiblemente, se imaginaron su belleza plástica. En esa época, la falta de herramientas que permitieran una representación precisa hizo que este tema fuera considerado de interés puramente académico.

Como hemos señalado, en este artículo, simplemente, nos dedicaremos a mostrar cómo lograr representaciones gráficas de fractales mediante uno de los potentes programas de cálculo simbólico disponibles actualmente. Para ello, deberemos escribir pequeños módulos de código basados, lógicamente, en la definición matemática (y las propiedades) del fractal que queramos representar. Sólo veremos unos pocos ejemplos, pero los cambios requeridos para lograr dibujar otros tipos de fractales son, muchas veces, sencillos: básicamente, modificar el algoritmo que define el fractal.

No podemos pasar por alto que un lenguaje de programación ordinario es, típicamente, cientos de veces más rápido; y la velocidad de cómputo siempre es importante cuando hay que hacer muchos cálculos. A este respecto, citemos que [6, 9] usan BASIC, [7] Logo, y [2] C. Pero los dibujos son mucho más sencillos de obtener si empleamos un paquete informático con amplias capacidades gráficas, como Mathematica, Maple o Matlab; afortunadamente, la potencia de los ordenadores actuales nos permite hacerlo. Hemos elegido Mathematica [13], pero serviría cualquier otro.

Sólo dibujaremos fractales en el plano. Además, nos centraremos en un sólo tipo, los relacionados con iteración de funciones racionales o meromorfas. Pero existen muchos otros que no analizaremos aquí, algunos muy conocidos y de nombre llamativo: conjunto de Cantor (a veces designado como polvo de Cantor), copo de nieve de von Koch, triángulo y tapiz de Sierpiński, curvas de Hilbert y de Peano, esponja de Sierpiński-Menger; su descripción está ba-

sada en *sencillas* manipulaciones geométricas que se repiten sin cesar, lo cual permite conocer su dimensión de Hausdorff. Otras estructuras fractales están relacionadas con el caos y los sistemas dinámicos, como la aplicación logística y los atractores de Hénon, Pickover y Lorenz (conocidos, en general, como atractores extraños), por citar algunos. También existen curvas dragón, biomorfos, árboles fractales, montañas, fractales aleatorios, movimiento browniano, ..., y podríamos seguir así durante mucho rato. Pueden encontrarse descripciones, algoritmos y representaciones gráficas en [2, 6, 7, 9].

Finalmente, es obligatorio que advirtamos al lector de que, si su único interés es lograr bonitos dibujos, existen multitud de programas especializados en ello, algunos de ellos joyas gratuitas. Estos programas pueden ser rápidos, eficaces y altamente configurables; además, a veces incorporan técnicas de suavizado que dan un mejor aspecto a los gráficos. Pero, a juicio del que escribe, no nos proporcionan todo el control, flexibilidad y capacidad de experimentación que podemos conseguir gracias a las capacidades gráficas y de programación incorporadas en los potentes programas de cálculo. No citaremos ninguno de ellos; quien desee alguno, tiene todo Internet a su disposición para encontrarlos. Por supuesto, también podemos hallar preciosas imágenes con fractales de todo tipo, previamente representados por alguna otra persona (aunque, en general, sin que nosotros podamos saber cómo, ni qué significado matemático tienen sus magníficos colores, ni siquiera a qué estructura fractal corresponden).

El conjunto de Mandelbrot

El conjunto de Mandelbrot, que denotaremos como $\mathcal{M}$, es, sin duda, el conjunto fractal más conocido. Los primeros dibujos suyos son debidos a Brooks y Matelski en 1978, en conexión con grupos discretos; poco después, fue Mandelbrot quien comenzó a ocuparse de él. Para describirlo matemáticamente, basta definir su complementario: para cada punto c del plano complejo $\mathbb{C}$, decimos que $c \notin \mathcal{M}$ si el proceso iterativo

$$z_0 = c, \qquad z_n = z_{n-1}^2 + c$$

produce una sucesión $\{z_n\}_{n=0}^{\infty}$ tal que $|z_n| \to \infty$. Como curiosidad, comentemos algunas propiedades topológicas: $\mathcal{M}$ es compacto, simplemente conexo y conexo; este último es un notable resultado probado por Douady y Hubbard en 1982. Pero no haremos más hincapié en ello, sino que nuestro objetivo es poder dibujarlo.

En primer lugar, observemos que su definición directa no nos sirve para representar gráficamente $\mathcal{M}$: podríamos necesitar infinitas iteraciones. Pero llega en nuestra ayuda una importante propiedad (cuya demostración es bastante sencilla): si, a partir de $z_0 = c$, aparece un z_n con $|z_n| \geq 2$, entonces $c \notin \mathcal{M}$. Así, en la práctica, no representamos $\mathcal{M}$, sino algo parecido: fijamos un número máximo de iteraciones y, para cada punto $c \in \mathbb{C}$, suponemos que

c está en $\mathcal{M}$ si, al iterar partiendo de c, aún no ha aparecido ningún z_n de módulo 2 o mayor.

Por otra parte, según la definición, el conjunto de Mandelbrot es, sólo, un subconjunto del plano. Es decir, un punto del plano está en $\mathcal{M}$ o no lo está. Así, podríamos pintar en negro los puntos de $\mathcal{M}$ y en blanco el resto. Pero esto, obviamente, no introduce ningún colorido. ¿Cómo lograrlo? De nuevo acudimos para ello a la propiedad antes descrita: a un punto c le asignamos el color negro cuando hemos decidido que está en $\mathcal{M}$; en caso contrario, le asignamos un color o un nivel de gris distinto dependiendo del número de iteraciones que han sido efectuadas hasta que ha aparecido un $|z_n| \geq 2$.

Veamos como programar Mathematica para llevar a cabo esta tarea. El método iterativo lo definimos mediante el siguiente bloque de código:

```
iterMandel[x_, y_, maxIter_] := Block[{c, z, contador},
  c = x + I*y; z = c; contador = 0;
  While[(Abs[z] < 2.0) && (contador < maxIter),
    z = z*z + c; contador++
  ];
  Return[contador]
]
```

Obsérvese que esta rutina devuelve como resultado el número de iteraciones efectuadas; si sólo nos interesase saber si un punto está o no en el conjunto de Mandelbrot, cambiaríamos `Return[contador]` por `If[contador >= maxIter, Return[1.0], Return[0.0]]`.

Entonces, tras fijar `maxIter`, para dibujar el conjunto de Mandelbrot bastará que utilicemos la orden `DensityPlot` aplicada a la función `iterMandel` (en realidad, como queremos que $\mathcal{M}$ aparezca en negro, representaremos `-iterMandel`). Elegiremos un cuadrado del plano, que aquí caracterizamos mediante su centro y longitud del lado. Además, permitiremos elegir la resolución (número de puntos) empleada para dibujar $\mathcal{M}$ (lógicamente, una mayor resolución requiere más tiempo de cálculo y más consumo de memoria; la orden `// Timing` tiene como objetivo, únicamente, mostrarnos el tiempo empleado). Así, definimos

```
mandelbrot[centro_, lado_, maxIter_, resolucion_] := Block[
  {xxMin = centro[[1]]-lado/2, xxMax = centro[[1]]+lado/2,
    yyMin = centro[[2]]-lado/2, yyMax = centro[[2]]+lado/2},
  DensityPlot[-iterMandel[x, y, maxIter],
    {x, xxMin, xxMax}, {y, yyMin, yyMax},
    PlotPoints -> resolucion, Mesh -> False,
    PlotRange -> {-maxIter, 0}, ColorFunction -> GrayLevel
  ]
] // Timing
```

De este modo, por ejemplo, `mandelbrot[{-0.5, 0.0}, 3, 30, 400]` genera el dibujo mostrado en la figura 1.

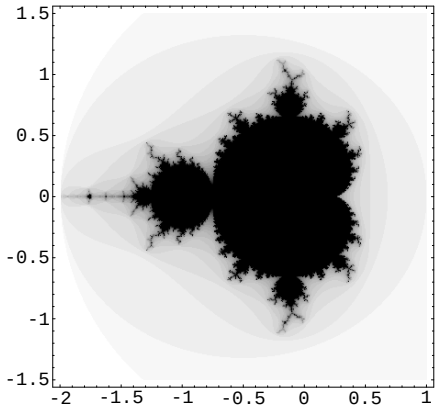

Figura 1: Conjunto de Mandelbrot.

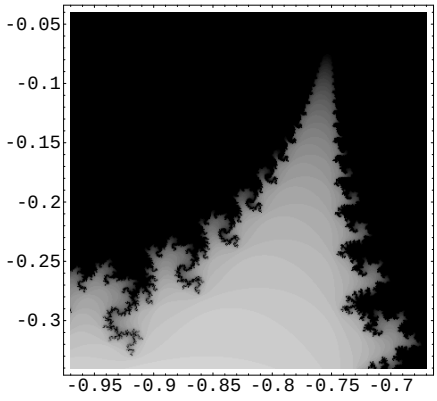

Figura 2: Detalle.

En la práctica, para lograr mucha mayor velocidad, es conveniente que usemos una versión previamente compilada de la función `iterMandel`. Esto permite a Mathematica evaluarla mucho más rápidamente (en nuestro ejemplo, el tiempo se divide, aproximadamente, por un factor 10). El código alternativo para ello es el siguiente:

```
iterMandel = Compile[
  {{x, _Real}, {y, _Real}, {maxIter, _Integer}},
  Block[{c = x + I*y, z = x + I*y, contador = 0},
    While[(Abs[z] < 2.0) && (contador < maxIter),
      z = z*z + c; contador++
    ];
  Return[contador]
  ]
]
```

En el conjunto de Mandelbrot, el fractal es su frontera; se desconoce su dimensión de Hausdorff. Como es característico de todos los fractales, la frontera tiene multitud de recovecos. Si nos acercamos a ella, siguen apareciendo extrañas formas, muchas de ellas semejantes al conjunto original. Para hacerlo con nuestro programa, basta tomar las coordenadas de un punto[2] cercano a la frontera e indicar el tamaño deseado para el lado del nuevo cuadrado que se debe representar. Por supuesto, esto podemos hacerlo

[2]En Mathematica, para conocer las coordenadas de un punto de un gráfico se procede del siguiente modo: Tras seleccionar el gráfico, con la tecla control (en un PC) o comando (en un Macintosh) pulsada, hacemos clic con el ratón en el punto deseado. Así, sus coordenadas aparecen escritas en la parte inferior izquierda de la ventana; además, se pueden *copiar* utilizando el correspondiente comando del menú de Mathematica.

tantas veces como deseemos. Como ejemplo, en la figura 2 hemos representado `mandelbrot[{-0.82, -0.19}, 0.3, 40, 400]`. Si fijamos `maxIter` y `resolucion`, y elegimos un `centro` y un `lado` iniciales, podríamos definir una función `zoom[centro_, factor_]` que se encargara de automatizar el proceso, calculando el nuevo `lado` y llamando ella misma a la función `mandelbrot` que realiza el dibujo. Así, bastaría con que, cada vez, le suministráramos las coordenadas del punto cuyo entorno queremos observar con más detalle.

En este artículo sólo mostramos figuras en escala de grises pero, en un ordenador, no es difícil generar dibujos en color que son, indudablemente, mucho más llamativos. Para conseguir gráficos coloreados, hay que indicarle a la función `mandelbrot` que lo haga. Para ello, en lugar de asignar `ColorFunction -> GrayLevel`, que proporciona niveles de gris (y que, en realidad, ni siquiera es necesario, pues es la opción que `DensityPlot` toma por defecto), hay que utilizar otra función de color. Así, a cada punto le asignaríamos un color dependiente del número de iteraciones efectuadas, es decir, del valor `contador` devuelto por `iterMandel`. Aquí, dos técnicas usuales son las siguientes: (a) utilizar un número fijo k de colores y asignar el color según sea el valor de `contador` módulo k; (b) introducir un degradado continuo (variando con `contador`) entre dos colores extremos. Con Mathematica, podemos definir la funciones de color a medida (más adelante, en este mismo artículo, hay un ejemplo; y también en [12]) pero, en todo caso, un método sencillo de obtener colores (sin preocuparnos cuáles) es usar `ColorFunction -> Hue`.

Es sencillo representar proyecciones en tres dimensiones del conjunto de Mandelbrot: asignamos la altura en función del número de iteraciones realizadas. Un programa que lleva a cabo esta tarea es el siguiente:

```
mandelbrot3D[centro_, lado_, maxIter_, resolucion_] := Block[
  {xxMin = centro[[1]]-lado/2, xxMax = centro[[1]]+lado/2,
    yyMin = centro[[2]]-lado/2, yyMax = centro[[2]]+lado/2},
  Plot3D[iterMandel[x, y, maxIter],
    {x, xxMin, xxMax}, {y, yyMin, yyMax},
    PlotPoints -> resolucion, Mesh -> False, Boxed -> False,
    Axes -> False, PlotRange -> {0, maxIter}
  ]
] // Timing
```

En la figura 3 podemos ver `mandelbrot3D[{0.1, -0.75}, 0.6, 25, 200]`. Aunque no lo mostraremos aquí, también son espectaculares las proyecciones del conjunto de Mandelbrot sobre la superficie de una esfera (la esfera de Riemann). La imaginación de los matemáticos ha dado lugar a numerosas ideas que han proporcionado magníficos gráficos.

Existen muchos fractales basados en ligeras modificaciones de la definición del conjunto de Mandelbrot, y que dan lugar a gráficos distintos; por ejemplo, cambiando z^2 por otra potencia de z (véase [1]). Recientemente, en [4] se ha descrito y estudiado una nueva modificación. Con nuestros programas, podemos explorarlos sin dificultad.

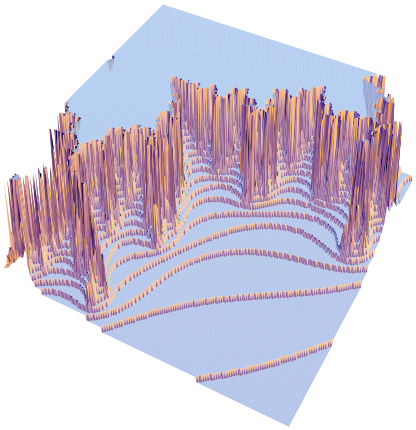

Figura 3: Conjunto de Mandelbrot en 3 dimensiones.

Conjuntos de Julia

Denotemos $\mathbb{C}_\infty$ al conjunto $\mathbb{C}$ con el punto del infinito, esto es, la esfera de Riemann, con su métrica habitual. Dada una función $f\colon \mathbb{C}_\infty \to \mathbb{C}_\infty$, para cada $z_0 \in \mathbb{C}_\infty$ podemos definir el método iterativo $z_n = f(z_{n-1}) = f^n(z_0)$, donde f^n denota f compuesta consigo mismo n veces.

Intuitivamente, describamos qué es el conjunto de Julia de f, que denotaremos $J(f)$. Al tomar dos puntos iniciales próximos, $z_0 \approx z'_0$, nos podemos preguntar si, tras efectuar iteraciones partiendo de ellos, los sucesivos valores z_n y z'_n también permanecen próximos (es decir, ¿$z_n \approx z'_n$?). Si la respuesta es negativa decimos que z_0 está en el conjunto de Julia de f; y si es afirmativa, que z_0 está en el conjunto de Fatou de f. Así, en $J(f)$ el método iterativo es sensible a las condiciones iniciales: aparece caos. Aunque no nos preocuparemos de ello en lo que sigue, con rigor matemático, el conjunto de Fatou de f es el subconjunto abierto maximal de $\mathbb{C}_\infty$ en el que la familia de funciones $\{f^n\}_{n=0}^\infty$ es equicontinua; y el conjunto de Julia de f es su complementario en $\mathbb{C}_\infty$.

Salvo para funciones f muy sencillas (por ejemplo, transformadas de Möbius $f(z) = (az+b)/(cz+d)$, $ad-bc \neq 0$), el conjunto de Julia suele ser no vacío. Habitualmente, además, es un fractal (esto no siempre es así: por ejemplo, si $f(z) = z^2$, $J(f)$ es la circunferencia unidad; y lo mismo para $f(z) = \alpha z^2$, con $|\alpha| = 1$).

Esto es cierto, en particular, para cada función cuadrática $f_c(z) = z^2 + c$ con $c \in \mathbb{C} \setminus \{0\}$. Veamos, en primer lugar, cómo representar gráficamente los correspondientes conjuntos de Julia $J_c = J(f_c)$. En realidad, es imposible que conozcamos J_c exactamente, luego en la práctica dibujaremos *algo* relacionado con ellos.

No es difícil comprobar que, para z_0 suficientemente grande, la sucesión $\{z_n\}_{n=0}^\infty$ converge a infinito (recordemos que estamos usando la métrica de

$\mathbb{C}_\infty$). Todos estos puntos, que pertenecen al conjunto de Fatou (el complementario al de Julia), decimos que están en la componente del infinito, F_∞. Así, lo que intentaremos representar es el complementario de F_∞. Se puede demostrar que la frontera de este conjunto es, precisamente, el conjunto de Julia. En otras palabras,

$$J_c = J(f_c) = \mathrm{Fr}\{z : f_c^n(z) \not\to \infty\}.$$

(Debemos recalcar que esta propiedad no es cierta para funciones y conjuntos de Julia $J(f)$ cualesquiera, pero sí cuando la función f es un polinomio.)

Los conjuntos de Julia asociados a las funciones $f_c(z) = z^2 + c$ están muy relacionados con el conjunto de Mandelbrot: J_c es conexo si y sólo si $c \in \mathcal{M}$ (de hecho, esto se toma, a veces, como definición alternativa del conjunto de Mandelbrot). Cuando $c \notin \mathcal{M}$, J_c es totalmente disconexo y sin puntos aislados.

De nuevo, nuestro objetivo es representar los conjuntos de Julia J_c con Mathematica. En realidad, dibujaremos el *conjunto relleno* $J_c^* = \{z : f_c^n(z) \not\to \infty\}$, no su frontera. Puede comprobarse que $J_c^* \subset \{z : |z| \leq m_c\}$ con $m_c = 1/2 + \sqrt{1/4 + |c|}$. Así, para cada punto z, si $|f_c^n(z)|$ es bastante grande (mayor que m_c), entonces $z \notin J_c^*$. Pero, por otra parte, y como nos ocurría con $\mathcal{M}$, podríamos necesitar infinitas iteraciones para poder asegurar que $z \in J_c^*$. Así que, en la práctica, nos tendremos que contentar con un número máximo de iteraciones y ver si en ese número de iteraciones ha aparecido ya un valor suficientemente grande. Realmente, no es necesario que usemos m_c como *valor de escape*; podríamos haber cogido otro número mayor.

Como es habitual, para cada punto z procedemos del siguiente modo: le asignamos color negro si, en el número de iteraciones prefijado, no se ha alcanzado ningún $|f_c^n(z)| > m_c$; en otro caso, distintos colores o niveles de gris, dependiendo del primer n (lo que se suele denominar *tiempo de escape*) para el que $|f_c^n(z)| > m_c$. Todo esto, en un cuadrado centrado en el origen y de lado algo mayor que $2m_c$.

Veamos cómo hacerlo con Mathematica; en principio, emplearemos niveles de gris. El método iterativo se implementa del siguiente modo (mostramos ya únicamente la versión compilada):

```
iterJuliaCuadratico = Compile[
  {{c, _Complex}, {x, _Real}, {y, _Real}, {maxIter, _Integer}},
  Block[{z = x + y*I,
      mc = 0.5 + Sqrt[0.25 + Abs[c]], contador = 0},
    While[(Abs[z] <= mc) && (contador < maxIter),
      z = z*z + c; contador++
    ];
    Return[contador]
  ]
]
```

Y la rutina que permitirá generar los dibujos es

```
juliaCuadratico[c_, maxIter_, resolucion_] := Block[
  {semilado = Ceiling[0.5 + Sqrt[0.25 + Abs[c]]],
    xxMin = -semilado, xxMax = semilado,
    yyMin = -semilado, yyMax = semilado},
  DensityPlot[-iterJuliaCuadratico[c, x, y, maxIter],
    {x, xxMin, xxMax}, {y, yyMin, yyMax},
    PlotPoints -> resolucion, Mesh -> False
  ]
] // Timing
```

En las figuras 4 y 5 podemos ver `juliaCuadratico[-.3-.5I, 30, 400]` y `juliaCuadratico[-.75+.8I, 30, 400]`. Los valores de c más interesantes son los cercanos a la frontera del conjunto de Mandelbrot, pues es allí cuando el conjunto J_c pasa, bruscamente, de ser conexo a ser completamente disconexo.

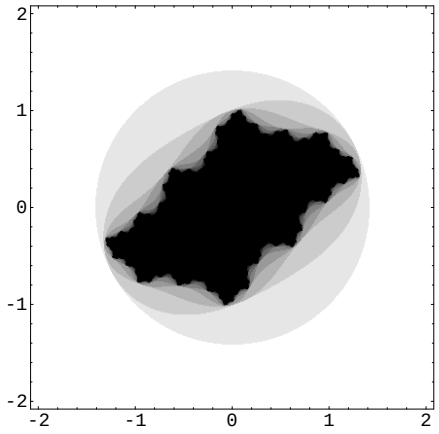

Figura 4: Conjunto de Julia conexo.

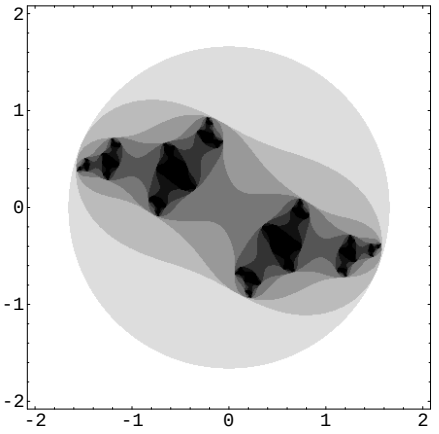

Figura 5: Conjunto de Julia totalmente disconexo.

Obviamente, podríamos introducir en el programa las modificaciones necesarias para dibujar otras zonas; en particular, para poder observar con más detalle el trozo que deseemos.

Hasta ahora, aunque la definición de conjuntos de Julia que hemos dado es bastante general, sólo hemos dibujado los correspondientes a funciones cuadráticas $f_c(z) = z^2 + c$.

Pero, por ejemplo, tiene perfecto sentido hablar del de Julia $J(f)$ de una función $f\colon \mathbb{C}_\infty \to \mathbb{C}_\infty$ racional o meromorfa. La dificultad radica en que, para dibujar $J(f_c)$, nos hemos servido de propiedades matemáticas suyas, y que no son válidas para conjuntos de Julia $J(f)$ generales (parece claro que no es posible utilizar directamente la definición). Así, para representar gráficamente conjuntos de Julia asociados a otro tipo de funciones, habrá que emplear otros métodos gráficos, basados en diferentes propiedades de los correspondientes conjuntos de Julia.

Cuando f es una función racional, puede demostrarse que, para cualquier $z \in J(f)$,

$$J(f) = \overline{\cup_{n\geq 0} f^{-n}(\{z\})}$$

donde f^{-1} denota la antiimagen de un conjunto; y f^{-n}, efectuar n veces la misma *iteración inversa*. En la práctica, si no conocemos ningún $z \in J(f)$, se suele tomar $z \in \mathbb{C}$ cualquiera, se itera *hacia atrás* bastantes veces y se considera que el nuevo z que así se obtiene está ya en $J(f)$. Ahora, bastará tomar un N fijo (grande), encontrar suficientes puntos de $\cup_{0\leq n\leq N} f^{-n}(\{z\})$ y representarlos gráficamente. En cada uso de f^{-1}, no es imprescindible encontrar la antiimagen completa: podemos incluso contentarnos con tomar un solo punto cada vez.

Aunque no lo haremos, esto puede implementarse con Mathematica del siguiente modo: definimos una lista, a la que le vamos añadiendo los puntos con `AppendTo`, y finalmente los dibujamos usando `ListPlot`.

En particular, el algoritmo descrito es un método alternativo de dibujar los conjuntos de Julia cuadráticos J_c. Además, el gráfico que se obtiene es el de J_c, no el de J_c^*. Nótese aquí que el método que habíamos usado para generar J_c^* no es útil si queremos dibujarlo según su *definición matemática pura* (sin niveles de gris, sólo blanco y negro) cuando $c \notin \mathcal{M}$: como estos conjuntos de Julia son totalmente disconexos, es común que, cuando Mathematica (internamente) da valores a puntos para usar `DensityPlot`, no se tope con puntos del fractal; y no dibuja nada. En cambio, el método de las iteraciones inversas sí que sirve para representar J_c.

También se han estudiado propiedades que permiten representar los conjuntos de Julia asociados a algunas funciones trascendentes. En concreto, para funciones del tipo $\lambda\cos(z)$, $\lambda\,\mathrm{sen}(z)$, $\lambda\exp(z)$, siendo $\lambda \in \mathbb{C}$ una constante. Así, se tiene

$$J(f) = \overline{\{z : |\operatorname{Im}(f^n(z))| \to \infty\}}, \quad f(z) = \lambda\cos(z) \text{ o } f(z) = \lambda\,\mathrm{sen}(z)$$

y

$$J(f) = \overline{\{z : \operatorname{Re}(f^n(z)) \to \infty\}}, \quad f(z) = \lambda\exp(z).$$

Con estas ideas, vamos a usar Mathematica para dibujar el conjunto de Julia de la función $f(z) = (1 + 0.4i)\,\mathrm{sen}(z)$. En primer lugar, definimos la función, la condición de escape, y el método iterativo, como funciones compiladas:

```
f = Compile[{{z, _Complex}}, (1. + 0.4I)Sin[z]];
testNoEscape = Compile[{{z, _Complex}}, (Abs[Im[z]] < 50)];
iterJulia = Compile[
  {{x, _Real}, {y, _Real}, {maxIter, _Integer}},
  Block[{z = x + I*y, contador = 0},
    While[testNoEscape[z] && (contador < maxIter),
      z = f[z]; contador++
    ];
    Return[contador]
```

```
  ],
  {{f[_], _Complex}, {testNoEscape[_], True | False}}
]
```

Aunque los gráficos que aparecen en este artículo son en blanco y negro, aquí, con un poco más de esfuerzo, mostraremos una manera de dibujar los correspondientes conjuntos de Julia con colores. Para poder llamar a los colores por su nombre, cargamos el paquete

```
<< Graphics`Colors`
```

Así, podemos definir

```
coloresArcoIris = {Red, Orange, Yellow, Green,
  Blue, Indigo, Violet}
```

Entonces, la función que se encargará de efectuar los gráficos es

```
juliaArcoIris[centro_, lado_, maxIter_, resolucion_] := Block[
  {$Messages = {}, (* evita mensajes de underflow, etc. *)
    xxMin = centro[[1]]-lado/2, xxMax = centro[[1]]+lado/2,
    yyMin = centro[[2]]-lado/2, yyMax = centro[[2]]+lado/2},
  coloridoArcoIris[p_] := If[(p == maxIter),
    RGBColor[0., 0., 0.], coloresArcoIris[[Mod[p, 7] + 1]]
  ];
  DensityPlot[iterJulia[x, y, maxIter],
    {x, xxMin, xxMax}, {y, yyMin, yyMax},
    PlotPoints -> resolucion, Mesh -> False,
    PlotRange -> {0, maxIter},
    ColorFunction -> coloridoArcoIris,
    ColorFunctionScaling -> False
  ]
] // Timing
```

En la figura 6 mostramos `julia[{0., 0.}, 10, 20, 400]`, siendo `julia` una rutina análoga a `juliaArcoIris`, pero en la que se ha eliminado la definición de la función de color `coloridoArcoIris`, así como las asignaciones a `ColorFunction` y `ColorFunctionScaling` (y añadido un cambio de signo para que el conjunto de Julia salga en negro). En la figura 7 vemos una ampliación del cuadrado de centro `{1.2, 0.9}` y lado `1.5`. En [5, 6, 2] se pueden encontrar llamativos dibujos en color de este tipo de conjuntos de Julia.

Al generar estos gráficos, es interesante tener en cuenta que, debido a la extrema sensibilidad sobre las condiciones iniciales, pequeños cambios en los parámetros del fractal pueden producir resultados sorprendentemente distintos. Por otra parte, nótese que, si ponemos una condición de escape que no es la que corresponde, lo que aparece puede no tener nada que ver con el conjunto de Julia. Pero, y sobre todo si estamos utilizando colores, es habitual que quede un bonito dibujo.

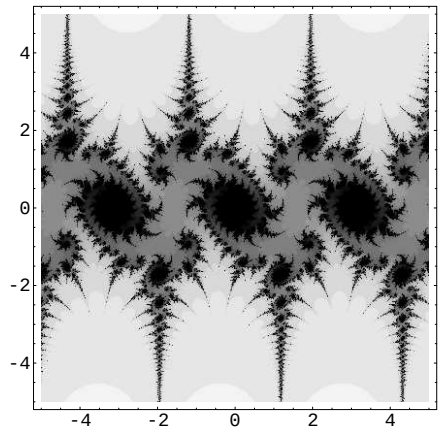

Figura 6: Conjunto de Julia de $(1 + 0.4i)\,\mathrm{sen}(z)$.

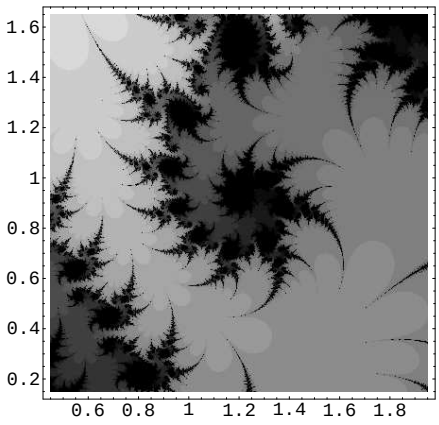

Figura 7: Detalle.

Fractales asociados a métodos iterativos de resolución de ecuaciones

Sea g una función $g\colon \mathbb{R} \to \mathbb{R}$ y ζ una raíz, es decir, $g(\zeta) = 0$. Es bien conocido que, si tomamos x_0 cerca de ζ, y bajo ciertas condiciones de las que no nos preocuparemos aquí, el método de Newton (o de las tangentes, por su interpretación geométrica)

$$x_{n+1} = x_n - \frac{g(x_n)}{g'(x_n)}, \quad n = 0, 1, 2, \ldots$$

genera una sucesión $\{x_n\}_{n=0}^{\infty}$ que converge a ζ.

En 1879, Cayley trató de usar el método para encontrar raíces de funciones complejas $g\colon \mathbb{C} \to \mathbb{C}$. Partiendo de $z_0 \in \mathbb{C}$ e iterando

$$z_{n+1} = z_n - \frac{g(z_n)}{g'(z_n)}, \quad n = 0, 1, 2, \ldots,$$

él buscaba bajo qué condiciones y hacia qué raíz de g convergía cada sucesión $\{z_n\}_{n=0}^{\infty}$. En otras palabras, intentaba identificar la *cuenca de atracción* de cada raíz. El problema es sencillo cuando g es un polinomio cuadrático, pero muy complicado en el caso general, incluso para polinomios cúbicos (lo cual hizo que Cayley renunciara a continuar tras varios años de intentos). Ahora sabemos que la frontera entre las cuencas de atracción tiene naturaleza fractal. Esta frontera es el conjunto de Julia de la función $f(z) = z - g(z)/g'(z)$.

A este respecto, el ejemplo más conocido es el de las cuencas de atracción de las tres raíces $e^{2k\pi i/3}$, $k = 0, 1, 2$, de la función $g(z) = z^3 - 1$. Existen

multitud de dibujos generados por ordenador que nos muestran la complejidad de estas intrincadas regiones.

Generalmente, existen tres estrategias para generar estos gráficos:

(a) A cada punto z se le asigna un color (o un nivel de gris) dependiendo a qué raíz de g converge el método de Newton que parte de z. Y marcamos el punto como negro (por ejemplo) si el método no converge. Por supuesto, deberemos fijar un número máximo de iteraciones permitido, y con qué precisión se desea alcanzar las raíces. De este modo, se distinguen las cuencas de atracción de las raíces.

(b) En lugar de asignar el color según la raíz alcanzada por el método, lo asignamos según el número de iteraciones efectuadas hasta alcanzarla con la precisión prefijada. Esto no genera un conjunto de Julia, pero también aparecen bonitos gráficos.

(c) Esta tercera estrategia es combinación de las dos anteriores. Asignamos un color para cada raíz, y hacemos que sea más claro o más oscuro según el número de iteraciones efectuadas hasta alcanzar la raíz con la precisión requerida. Como antes, usamos negro si el método no converge. Los dibujos generados de esta manera son, en opinión del que escribe, los más interesantes; y son fáciles de producir en una pantalla de ordenador.

Aquí sólo seguiremos la primera estrategia y, además, con niveles de gris. Para ver dibujos similares con colores, junto con el código en Mathematica para generarlos, consúltese [12].

Antes de continuar, no podemos pasar por alto que existen otros métodos iterativos para hallar raíces de funciones. El de Newton es el más conocido, pero hay muchos más: de nuevo en [12] se muestran bastantes de ellos, así como referencias de dónde estudiarlos con detalle.

Veamos cómo programar todo esto con Mathematica. Lo haremos de modo que sea suficientemente sencillo de usar para una función y un método de resolución de ecuaciones genéricos. E intentando, además, utilizar funciones compiladas, para aumentar la velocidad de ejecución. En primer lugar, definimos `ejecutarRutinasRaices`, que tiene como argumento una función genérica `g`. Cuando la usemos, se encargará de hallar las raíces de la función y definir la rutina que comprueba si un complejo `z` es una raíz (con aproximación `eps`) o no.

```
ejecutarRutinasRaices[g_] := Block[{},
  hallaRaices = NSolve[g[z] == 0, z];
  numRaices = Length[hallaRaices];
  Do[raiz[k] = hallaRaices[[k, 1, 2]], {k, 1, numRaices}];
  posicionRaiz = Compile[{{z, _Complex}},
    Block[{eps = 10.0^(-3), k, numLocal = numRaices},
      For[k = 1, k <= numLocal, k++,
        If[Abs[z - raiz[k]] < eps, Return[k]]
      ];
```

```
      Return[0]
    ],
    {{raiz[_], _Complex}}
  ]
]
```

Nótese que, para hallar las raíces, usamos `NSolve`. Esto permite hallar todas las raíces de un polinomio o una función racional; pero si la función es de otro tipo, las raíces no se pueden calcular así y hay que modificar el programa. (Para aplicar la estrategia (b) antes descrita, ni siquiera es necesario conocer las raíces. Si no se puede comparar directamente con la raíz exacta, una raíz aproximada de g se identifica con las condiciones de parada habituales: $|g(z_n)| < \varepsilon$ o $|z_{n+1} - z_n| < \varepsilon$.)

A continuación, definimos la rutina que (cuando sea llamada) calcula la función iteradora `f` asociada a una función `g` y un método `metodo` genéricos (obsérvese que aprovechamos las posibilidades simbólicas de Mathematica para simplificar):

```
calcularFuncionIteradora[g_, metodo_] := Block[{},
  f = Compile[{{z, _Complex}},
    Evaluate[Simplify[metodo[g, z]]]
  ]
]
```

Y éste es el proceso iterativo que decide a qué raíz converge cada punto (o que no converge a ninguna) tras aplicar sucesivas veces `f`:

```
iterRaices = Compile[{{x, _Real}, {y, _Real},
    {maxIter, _Integer}},
  Block[{z, contador, r},
    z = x + y*I; contador = 0; r = posicionRaiz[z];
    While[(r == 0) && (contador < maxIter),
      contador++; z = f[z]; r = posicionRaiz[z]
    ];
    Return[r]
  ],
  {{f[_], _Complex}, {posicionRaiz[_], _Integer}}
]
```

Con todo esto, el fractal se dibujará invocando la siguiente rutina:

```
fractalRaices[centro_, lado_, maxIter_, resolucion_] :=
  Block[{$Messages = {},
      xxMin = centro[[1]]-lado/2, xxMax = centro[[1]]+lado/2,
      yyMin = centro[[2]]-lado/2, yyMax = centro[[2]]+lado/2},
    DensityPlot[iterRaices[x, y, maxIter],
      {x, xxMin, xxMax}, {y, yyMin, yyMax},
      PlotPoints -> resolucion, Mesh -> False,
```

```
      PlotRange -> {0, numRaices}
    ]
  ] // Timing
```

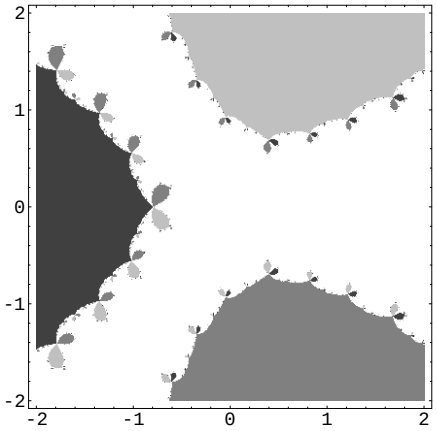

Figura 8: Método de Newton aplicado a $g(z) = z^4 + 2z - 1$.

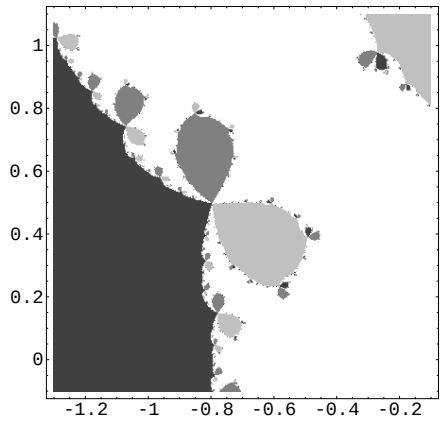

Figura 9: Detalle del método de Halley.

Concluimos mostrando un par de ejemplos de uso. Con

```
g[z_] := z^4 + 2z - 1;
metodo[g_, z_] := z - g[z]/g'[z]; (* Newton *)
ejecutarRutinasRaices[g];
calcularFuncionIteradora[g, metodo];
fractalRaices[{0, 0}, 4, 25, 400]
```

conseguimos el gráfico de la figura 8 (obsérvese que Mathematica se deberá encargar de efectuar, simbólicamente, las derivadas). Y con

```
metodo[g_, z_] :=
  z - 1/(g'[z]/g[z] - g''[z]/(2g'[z])); (* Halley *)
calcularFuncionIteradora[g, metodo];
fractalRaices[{-0.7, 0.5}, 1.2, 25, 400]
```

(como es la misma función `g`, ya no hace falta reevaluar las rutinas relacionadas con las raíces), la figura 9.

Bibliografía

[1] C. Alexander, I. Giblin y D. Newton, Symmetry groups on fractals, *The Mathematical Intelligencer* **14** (1992), n.º 2, 32–38.

[2] J. Barrallo Calonge, *Geometría fractal: Algorítmica y representación*, Anaya Multimedia, Madrid, 1993.

[3] A. F. Beardon, *Iteration of rational functions*, Springer-Verlag, Nueva York, 1991.

[4] M. Benito, J. M. Gutiérrez y V. Lanchares, El fractal de Chicho, en *Margarita Mathematica en memoria de José Javier (Chicho) Guadalupe Hernández* (L. Español y J. L. Varona, eds.), pp. 247–254, *Servicio de Publicaciones de la Universidad de La Rioja*, Logroño, 2001.

[5] R. L. Devaney, Film and video as a tool in mathematical research, *The Mathematical Intelligencer* **11** (1989), n.º 2, 33–38.

[6] R. L. Devaney, *Chaos, fractals, and dynamics*, Addison-Wesley, Nueva York, 1990.

[7] G. A. Edgar, *Measure, topology, and fractal geometry*, Springer-Verlag, Nueva York, 1990.

[8] K. Falconer, *Techniques in fractal geometry*, John Wiley & Sons, Chichester, 1997.

[9] M. de Guzmán, M. A. Martín, M. Morán y M. Reyes, *Estructuras fractales y sus aplicaciones*, Labor, Barcelona, 1993.

[10] B. Mandelbrot, *Los objetos fractales*, Tusquets, Barcelona, 1987.

[11] H. O. Peitgen y P. H. Richter, *The beauty of fractals*, Springer-Verlag, Nueva York, 1986.

[12] J. L. Varona, Graphic and numerical comparison between iterative methods, *The Mathematical Intelligencer* **24** (2002), n.º 1, 37–46.

[13] S. Wolfram, *The Mathematica Book*, 4.ª ed., Wolfram Media/Cambridge University Press, Champaign, 1999.

Juan L. Varona,
Departamento de Matemáticas y Computación,
Universidad de La Rioja,
Calle Luis de Ulloa s/n, 26004 Logroño.
e-mail: jvarona@dmc.unirioja.es
web: http://www.unirioja.es/dptos/dmc/jvarona/hola.html