Plotting basins of end points of rational maps with Sage

Luis Javier Hernández Paricio, Miguel Marañón Grandes, María Teresa Rivas Rodríguez [*]

Department of Mathematics and Computer Science, University of La Rioja, Building VIVES, Luis de Ulloa St., 26004 Logroño, Spain.

E-mail: {`luis-javier.hernandez, miguel.maranon, maria-teresa.rivas`}`@unirioja.es`

2000 Mathematics Subject Classification. **37-04**. 14E05, 00A66.
Keywords. Discrete semi-flow, Sage, rational map, complex dynamical system.

Abstract

The aim of this work is to present a new program written in *Sage* which allows us to visualize the attraction basins associated to the end points of a discrete semi-flow induced by a rational function on the Riemann sphere by using its geometry and complex structure. One interesting novelty brought by the developed program is that it is able to plot fractals not only in a determined rectangular area of the complex plane, but also on the whole surface of the Riemann sphere. Another advantage of the program is that it permits us to visualize not only the attraction basins of fixed points, but also the basins of end points associated with periodic points. In addition, some applications of the described algorithms to Numerical Analysis and Fractal Geometry are suggested.

Introduction

Visualization and graphic techniques are powerful tools that can be used to reach different proposals. One of its possible applications is to promote advances in some mathematical subjects: Numerical Analysis, Dynamical Systems, Fractal Geometry, et cetera. Some researchers have claimed the design of visualization tools that can be used either to validate the theoretical results obtained or to perform some graphic experiments that can help guessing new conjectures and theorems.

For example, in Numerical Analysis some authors [5, 7, 8, 9] have developed graphic algorithms to study the attraction basin of a root when a numerical method is employed. Although some of these algorithms have been implemented in commercial software (*Mathematica*, *Matlab*, *Maple*), it would be desirable to have implementations in open source software (*Sage*) that provide free access to mathematicians and other interested researchers.

In this context, our work has a twofold objective: firstly, it aims to improve some aspects of existing programs for the visualization of attraction basins and Julia sets associated to rational functions on the Riemann sphere and, secondly, we want to implement new programs in *Sage*; this will allow to dispose many computational resources provided by *Sage* environment as well as to have free access to them.

For an interesting example on how visualization tools can be used in Numerical Analysis, we refer the reader to J. L. Varona [9]. In this paper, several iterative methods are compared graphically by representing the basins of attraction associated to the roots of a given complex polynomial which have been generated by each method. In his work, Varona also develops a program written in

[*]The authors acknowledge the financial help given by project API12/10.

Mathematica that is able to plot fractals on a rectangular piece of the plain of complex numbers $\mathbb{C}$ chosen by the user, employing one of the three strategies which will be shown in paragraph 2.6.1. However, it cannot manage possible overflows and underflows (actually, it just turns off error messages) and the complex function has to be defined inside the program code (it is not an input parameter).

In the literature, there are many other similar computational programs devoted to plot attraction basins. For instance, O. Lewis [5] uses *Matlab* and C++ in order to generate Newton basins for polynomials and speed up computation, respectively; nevertheless, the fractals can only be plotted on a rectangle in $\mathbb{C}$ and the program provides just one color assignment strategy (according to the fixed point to which each point converges).

M. McClure [7] develops a program written in *Mathematica* that, like the algorithms of Varona and Lewis, is able to draw fractals induced by Newton's method only on a rectangle, but it also assigns colors to the points in $\mathbb{C}$ taking into account both the fixed point to which their trajectories converge and the speed of convergence; moreover, it highlights all the fixed points of the induced rational functions (next versions of our program will do that) and seeks that the user has total control upon the palette employed to color each complex point (it provides just one color map, though). However, McClure's program cannot detect basins of end points associated to n-cycles.

W. T. Shaw [8] presents a software developed in *Mathematica* that gives the possibility of drawing the attraction basin of a unique end point. It provides the same color strategy as McClure's algorithm and several color maps. In addition, it can build "fractal planets" by mapping the plane $\mathbb{C}$ onto the unit 2-sphere S^2, but truncating the value of the polar coordinates at the North and South pole to avoid kernel complaints about singular behavior. The advantage provided by Shaw's algorithm is that it is quite quick.

Finally, we remark that programs like Fractal Domains, which is described in [2], permit to plot Julia and Mandelbrot sets on local rectangles in a very fast way, but no more mathematical information such as multiplicators, coordinates of fixed points, etc. can be obtained.

In the present article, we present a collection of algorithms based on the canonical bijection of the complex projective line and $\mathbb{C} \cup \{\infty\}$ which give us the following advantages:

(i) the use of homogeneous coordinates which permits us to work at the point at infinity $\infty \in \mathbb{C} \cup \{\infty\}$,

(ii) the representation of a rational function by a pair of homogeneous polynomials of two variables and with the same degree that allows us to compute the numerical value of the function at any pole point and at the point at infinity,

(iii) the calculus of multiplicators which enables the usual classification of fixed points: super-attracting, attracting, indifferent or repelling,

(iv) the use of normalized homogeneous coordinates that avoids overflow and underflow errors in our algorithms.

Other of our subroutines are based on the stereographic bijection from the unit 2-sphere to $\mathbb{C} \cup \{\infty\}$, and it permits us:

(v) to compute the distance from an ordinary point to the point at infinity using the chordal metric on the 2-sphere,

(vi) to plot 3D-spherical global versions of the attraction basins,

(vii) to draw global attraction basins using two discs that correspond to the south and north hemispheres of the 2-sphere.

In the present paper, a new notion of end point is given, which provides us the possibility of studying attraction basins not only of fixed points, but also of periodic points. For a given positive integer n, our algorithms allow us:

(viii) to plot the attraction basins of end points associated to n-periodic points.

Although the authors in [6] have developed an implementation of some of these algorithms in *Mathematica*, the possibility of having open access to these programs for workers on Numerical Analysis or Fractal Geometry has been the main motivation for developing this new version implemented in *Sage*.

We note that some of these new facts increase the computational cost and, for this very reason, the authors are studying the possibility of using Cython to improve the execution speed of some algorithms. On the other hand, the authors are also working on raising the quality of the obtained graphics with *Sage*.

The present article is divided into four parts. In section 1, a mathematical theoretical basis for our program is given. Section 2 describes the tasks and source codes of our algorithms. Section 3 includes a brief user manual that explains how to use the developed software properly. In the end, section 4 shows some applications of our algorithms to certain mathematical fields and some future implementations are suggested.

1 Mathematical framework and theoretical justification of the algorithms

In order to create a theoretical basis to hold and justify the correct construction of our algorithms for the representation of basins of end points corresponding to rational maps, we shall use the mathematical techniques described below in this section. This study will be developed within the theoretical framework of complex dynamics on the Riemann sphere.

1.1 Discrete semi-flows

Firstly, we recall some basic notions about discrete semi-flows.

Definition 1.1. A ***discrete semi-flow*** on a (topological space) set X is a (continuous) map $\varphi\colon \mathbb{N} \times X \to X$ such that:

(i) $\varphi(0, p) = p$, $\forall p \in X$.

(ii) $\varphi(n, \varphi(m, p)) = \varphi(n + m, p)$, $\forall p \in X$, $\forall n, m \in \mathbb{N}$.

We shall use $n \cdot x = \varphi(n, x)$ for short.

Given a discrete semi-flow $\varphi\colon \mathbb{N}\times X \to X$, $n_0 \in \mathbb{N}$, $x_0 \in X$, we have the induced maps $\varphi^{n_0}\colon X \to X$, $\varphi^{n_0}(x) = \varphi(n_0, x)$, and the ***trajectory*** of x_0 given by $\varphi_{x_0}\colon \mathbb{N} \to X$, $\varphi_{x_0}(n) = \varphi(n, x_0)$. Note that a discrete semi-flow $\varphi\colon \mathbb{N}\times X \to X$ induces a map $\varphi^1\colon X \to X$ and, conversely, a (continuous) map $f\colon X \to X$ induces a discrete semi-flow $\varphi\colon \mathbb{N}\times X \to X$ given by $\varphi(n,x) = f^n(x)$, where f^n denotes the function composition $f \circ \underbrace{\dots}_{n \text{ times}} \circ f$ and $f^0 = \mathrm{Id}_X$. The discrete semi-flow induced by f will be denoted by (X, f) or X for short.

Definition 1.2. Let X be a discrete semi-flow and x be a point of X.

(i) x is said to be a ***fixed point*** if, for all $n \in \mathbb{N}$, $n \cdot x = x$.

(ii) x is said to be a ***periodic point*** if there exists $n \in \mathbb{N}$, $n \neq 0$, such that $n \cdot x = x$.

The subset of fixed points of a discrete semi-flow will be denoted by $\mathrm{Fix}(X)$ and the subset of periodic points by $P(X)$.

1.2 End points associated to a discrete semi-flow

Let (X, d) be a metric space with metric d. Given a discrete semi-flow induced by a continuous map $f\colon X \to X$, the triple (X, d, f) will be called ***metric discrete semi-flow***.

Next we introduce a notion of end point based on the existence of the metric d; for other notions of end point of a dynamical system, we refer the reader to [4].

Definition 1.3. Given a metric discrete semi-flow $X = (X, d, f)$, the ***end point space*** of X is defined as the quotient set

$$\Pi(X) = \frac{\{(f^n(x))_{n\in\mathbb{N}} \mid x \in X\}}{\sim},$$

where, given $x, y \in X$, $(f^n(x)) \sim (f^n(y))$ if and only if

$$(d(f^n(x), f^n(y))) \xrightarrow[n\to\infty]{} 0.$$

An element $a = [(f^n(x))] \in \Pi(X)$ is called an ***end point*** of the metric discrete semi-flow.

Note that, if $y \in \mathrm{Fix}(X)$, we can interpret that y is an end point of the form $y = [(y)] = [(y, y, \dots)] \in \Pi(X)$. We can define the natural map

$$\omega\colon X \to \Pi(X)$$

given by $\omega(x) = [(f^n(x))] = [(x, f(x), f^2(x), \dots)]$. The map ω allows to decompose any metric discrete semi-flow in the way shown below.

Definition 1.4. Let X be a metric discrete semi-flow. The subset denoted by

$$X_a = \omega^{-1}(a), \quad a \in \Pi(X)$$

is called the ***basin of the end point*** a.

There is an induced partition of X given by

$$X = \bigsqcup_{a\in\Pi(X,d)} X_a,$$

which will be called ω-***decomposition*** of the metric discrete semi-flow X.

1.3 Smooth and complex structures on $\mathbb{C} \cup \{\infty\}$

Let $S^2 = \{(r_1, r_2, r_3) \in \mathbb{R}^3 \mid r_1^2 + r_2^2 + r_3^2 = 1\}$ be the unit 2-sphere and let $N = (0, 0, 1)$ be the north pole. Consider the stereographic atlas $\{\hat{x}, \hat{y}\}$ for S^2, where $\hat{x}\colon S^2 \setminus \{N\} \to \mathbb{R}^2$ and $\hat{y}\colon S^2 \setminus \{-N\} \to \mathbb{R}^2$ are both charts given by

$$\hat{x}(r_1, r_2, r_3) = \left(\frac{r_1}{1 - r_3}, \frac{r_2}{1 - r_3}\right),$$

$$\hat{y}(r_1, r_2, r_3) = \left(\frac{r_1}{1 + r_3}, \frac{r_2}{1 + r_3}\right).$$

The stereographic atlas gives a 2-dimensional smooth structure to S^2.

We can consider in a natural way a bijection $\tilde{\theta}\colon S^2 \to \mathbb{C} \cup \{\infty\}$ given as follows:

$$\tilde{\theta}(r_1, r_2, r_3) = \begin{cases} \frac{r_1}{1-r_3} + i\frac{r_2}{1-r_3}, & \text{if } r_3 < 1, \\ \infty, & \text{if } r_3 = 1. \end{cases}$$

In this way, we can also regard $\mathbb{C} \cup \{\infty\}$ as a 2-dimensional smooth manifold by using the bijection $\tilde{\theta}$.

Take the following equivalence relation on $\mathbb{C}^2 \setminus \{(0, 0)\}$: $(z, t) \sim (z', t')$ if there exists a $\lambda \in \mathbb{C} \setminus \{0\}$ such that $(z, t) = (\lambda z', \lambda t')$. The equivalence class of (z, t) is denoted by $[z, t]$ and the quotient set is denoted by $\mathbf{P}^1(\mathbb{C})$ and it is called the ***complex projective line***.

Let x and y be functions from $\mathbf{P}^1(\mathbb{C})$ to $\mathbb{C}$ with domains $\operatorname{Dom} x = \{[z, t] \in \mathbf{P}^1(\mathbb{C}) \mid t \neq 0\}$ and $\operatorname{Dom} y = \{[z, t] \in \mathbf{P}^1(\mathbb{C}) \mid z \neq 0\}$ given by $x([z, t]) = z/t$ and $y([z, t]) = t/z$. Then, the atlas $\{x, y\}$ provides $\mathbf{P}^1(\mathbb{C})$ with a 1-dimensional complex structure. Given a point $[z, t] \in \mathbf{P}^1(\mathbb{C})$, the coordinates (z, t) are called the homogeneous coordinates of the point and t/z (or z/t where appropriate) are the absolute coordinates of that point. In our study, we often use ***normalized homogeneous coordinates*** for any point in $\mathbf{P}^1(\mathbb{C})$, which are given as follows:

$$[z, t] = \begin{cases} [z/t, 1] & \text{if } |t| \geq |z|, \\ [1, t/z] & \text{if } |t| < |z|, \end{cases}$$

where $|t|$ and $|z|$ represent the absolute value (or modulus) of the complex numbers t and z, respectively.

We also have the induced bijection $\theta\colon \mathbf{P}^1(\mathbb{C}) \to \mathbb{C} \cup \{\infty\}$ given by

$$\theta([z, t]) = \begin{cases} z/t, & \text{if } t \neq 0, \\ \infty, & \text{if } t = 0. \end{cases}$$

All the bijections above induce a new bijection $\theta^{-1}\tilde{\theta}\colon S^2 \to \mathbf{P}^1(\mathbb{C})$, which can be defined as follows:

$$\theta^{-1}\tilde{\theta}(r_1, r_2, r_3) = [r_1 + ir_2, 1 - r_3].$$

The inverse map of this bijection $\tilde{\theta}^{-1}\theta\colon \mathbf{P}^1(\mathbb{C}) \to S^2$ is given by the following formula:

$$\tilde{\theta}^{-1}\theta([z,t]) = \left(\frac{\bar{z}t + z\bar{t}}{\bar{t}t + z\bar{z}}, \frac{i(\bar{z}t - z\bar{t})}{\bar{t}t + z\bar{z}}, \frac{-\bar{t}t + z\bar{z}}{\bar{t}t + z\bar{z}} \right).$$

We recall that a surface with a 1-dimensional complex structure is said to be a ***Riemann surface*** and a Riemann surface of genus 0 is said to be a ***Riemann sphere***. Using the bijections defined above, we have that S^2 and $\mathbb{C} \cup \{\infty\}$ are Riemann spheres.

Remark 1.5. We notice that the homogeneous coordinates presented in this subsection allow us to represent the point at infinity, and the use of normalized coordinates will avoid overflow and underflow errors in our computer programs.

1.4 Complex rational maps

Consider a rational function $h\colon \mathbb{C} \to \mathbb{C}$ of the form $h(u) = a\dfrac{F(u)}{G(u)}$, where $u, a \in \mathbb{C}$, $a \neq 0$, $F(u) = (u - z_1)\cdots(u - z_p)$ and $G(u) = (u - l_1)\cdots(u - l_q)$. Suppose that $\{z_1, \ldots, z_p\} \cap \{l_1, \ldots, l_q\} = \varnothing$. Then, the function h induces an extension map $h^+\colon \mathbb{C} \cup \{\infty\} \to \mathbb{C} \cup \{\infty\}$, where $h^+(l_i) = \infty$ and $h^+(\infty)$ is given as follows:

$$h^+(\infty) = \begin{cases} \infty, & \text{if } q < p, \\ 0, & \text{if } q > p, \\ a, & \text{if } q = p. \end{cases}$$

Observe that the bijection

$$\theta\colon \mathbf{P}^1(\mathbb{C}) \to \mathbb{C} \cup \{\infty\}$$

induces the map $h^1\colon \mathbf{P}^1(\mathbb{C}) \to \mathbf{P}^1(\mathbb{C})$ defined by $h^1 = \theta^{-1}h^+\theta$, which is expressed in homogeneous coordinates as follows:

$$h^1([z,t]) = \begin{cases} [a(z - tz_1)\cdots(z - tz_p), t^{p-q}(z - tl_1)\cdots(z - tl_q)], & \text{if } p \geq q, \\ [at^{q-p}(z - tz_1)\cdots(z - tz_p), (z - tl_1)\cdots(z - tl_q)], & \text{if } p \leq q. \end{cases}$$

In this context, it is important to notice that, in the case that we take $u = z/t$ and $n = \max\{p, q\}$ in the expression

$$\frac{a(u^p + a_1u^{p-1} + \cdots + a_p)}{(u^q + b_1u^{q-1} + \cdots + b_q)},$$

we have

$$\frac{a(z^pt^{n-p} + a_1z^{p-1}t^{n-p+1} + \cdots + a_pt^n)}{z^qt^{n-q} + b_1z^{q-1}t^{n-q+1} + \cdots + b_qt^n}.$$

Now, take the following homogeneous polynomials of degree n in the variables z, t:

$$F_1(z,t) = a(z^pt^{n-p} + a_1z^{p-1}t^{n-p+1} + \cdots + a_pt^n),$$

$$G_1(z,t) = z^qt^{n-q} + b_1z^{q-1}t^{n-q+1} + \cdots + b_qt^n;$$

by using these polynomials, one has that

$$h^1([z,t]) = [F_1(z,t), G_1(z,t)].$$

Conversely, if $A(z,t)$ and $B(z,t)$ are homogeneous polynomials of degree n, then $A(\lambda z, \lambda t) = \lambda^n A(z,t)$ and $B(\lambda z, \lambda t) = \lambda^n B(z,t)$. This implies that the pair of homogeneous polynomials $A(z,t)$ and $B(z,t)$ induces the map $f\colon \mathbf{P}^1(\mathbb{C}) \to \mathbf{P}^1(\mathbb{C})$ defined as follows:

$$f([z,t]) = [A(z,t), B(z,t)].$$

The associated rational map is given by $F(z) = A(z,1)$ and $G(z) = B(z,1)$.

The next lemma discuss how to find all the fixed points of any rational map which is represented by a pair of coprime homogeneous polynomials, even if it is composed with itself a certain number of times.

Lemma 1.6. Let f be a rational map represented by a pair of coprime homogeneous polynomials $A(z,t)$, $B(z,t)$ of degree n. Then:

(i) the set $\{[z_1,t_1], \ldots, [z_{n+1}, t_{n+1}]\}$ of roots of $A(z,t)t - B(z,t)z$ is the set of fixed points of f,

(ii) f^r is a rational map of degree n^r which has $n^r + 1$ fixed points (taking into account its multiplicity).

Remark 1.7. The representation of a rational function with a pair of homogeneous polynomials of two variables with the same degree combined with normalized homogeneous coordinates permits to work with poles and the point at infinity, as well as to avoid overflows and underflows in our algorithms.

1.5 Metrics on $S^2 \cong \mathbb{C} \cup \{\infty\} \cong \mathbf{P}^1(\mathbb{C})$

We have two natural metrics on S^2: since S^2 is a subspace of $\mathbb{R}^3$, the usual Euclidean metric of $\mathbb{R}^3$ induces a Euclidean metric d^E on S^2; on the other hand, we have as well that S^2 inheres a Riemannian metric d^R from the canonical Riemannian structure of $S^2 \subset \mathbb{R}^3$. The connection between Riemannian metric d^R and Euclidean metric d^E on S^2 is given by the expression:

$$d^E((r_1,r_2,r_3),(r_1',r_2',r_3')) = 2\sin\left(\frac{d^R((r_1,r_2,r_3),(r_1',r_2',r_3'))}{2}\right).$$

Using the bijection $\tilde{\theta}^{-1}\theta\colon \mathbf{P}^1(\mathbb{C}) \to S^2$, we can translate the metric structures from S^2 to $\mathbf{P}^1(\mathbb{C})$ with the following formulas:

$$\begin{aligned} d_1^E([z,t],[z',t']) &= d^E(\tilde{\theta}^{-1}\theta([z,t]), \tilde{\theta}^{-1}\theta([z',t'])), \\ d_1^R([z,t],[z',t']) &= d^R(\tilde{\theta}^{-1}\theta([z,t]), \tilde{\theta}^{-1}\theta([z',t'])). \end{aligned}$$

An explicit formula for the ***chordal metric*** d_1^E is given by:

$$d_1^E([z,t],[z',t']) =$$

$$\left(\left(\tfrac{\bar{z}t+z\bar{t}}{\bar{t}t+z\bar{z}} - \tfrac{\bar{z}'t'+z\bar{t}'}{\bar{t}'t'+z'\bar{z}'}\right)^2 + \left(\tfrac{i(\bar{z}t-z\bar{t})}{\bar{t}t+z\bar{z}} - \tfrac{i(\bar{z}'t'-z'\bar{t}')}{\bar{t}'t'+z'\bar{z}'}\right)^2 + \left(\tfrac{-\bar{t}t+z\bar{z}}{\bar{t}t+z\bar{z}} - \tfrac{-\bar{t}'t'+z'\bar{z}'}{\bar{t}'t'+z'\bar{z}'}\right)^2\right)^{\frac{1}{2}}.$$

1.6 Tangent map of a rational map

Given an analytic map $f\colon \mathbf{P}^1(\mathbb{C}) \to \mathbf{P}^1(\mathbb{C})$ and a point $p = [z, t] \in \mathbf{P}^1(\mathbb{C})$, there is an induced map between the tangent spaces

$$T_p f\colon T_p(\mathbf{P}^1(\mathbb{C})) \to T_{f(p)}(\mathbf{P}^1(\mathbb{C})).$$

Taking the bases $\frac{\partial}{\partial x}$ if $|t| \geq |z|$ and $\frac{\partial}{\partial y}$ if $|t| < |z|$ of the complex tangent space and writing $f(p) = [z', t']$, we have four cases when giving the 1×1 Jacobian matrix of $T_p f$:

$$\begin{aligned}
J_p^{x,x} &= \big((xfx^{-1})'(z/t)\big), \quad \text{if } |t| \geq |z| \text{ and } |t'| \geq |z'|,\\
J_p^{y,x} &= \big((yfx^{-1})'(z/t)\big), \quad \text{if } |t| \geq |z| \text{ and } |t'| < |z'|,\\
J_p^{x,y} &= \big((xfy^{-1})'(t/z)\big), \quad \text{if } |t| < |z| \text{ and } |t'| \geq |z'|,\\
J_p^{y,y} &= \big((yfy^{-1})'(t/z)\big), \quad \text{if } |t| < |z| \text{ and } |t'| < |z'|.
\end{aligned}$$

Notice that if f is a rational map induced by polynomials $A(z, t), B(z, t)$, then the coordinate representations of f with respect to the corresponding pairs of charts are given by

$$\begin{aligned}
xfx^{-1}(z) &= A(z, 1)/B(z, 1),\\
yfx^{-1}(z) &= B(z, 1)/A(z, 1),\\
xfy^{-1}(t) &= A(1, t)/B(1, t),\\
yfy^{-1}(t) &= B(1, t)/A(1, t),
\end{aligned}$$

so that we can consider its corresponding derivatives:

$$\frac{d(A(z,1)/B(z,1))}{dz},$$

$$\frac{d(B(z,1)/A(z,1))}{dz},$$

$$\frac{d(A(1,t)/B(1,t))}{dt},$$

$$\frac{d(B(1,t)/A(1,t))}{dt}.$$

Hence, the norm of the tangent function at p, taking into account the metrics of $T_p(\mathbf{P}^1(\mathbb{C}))$ and $T_{f(p)}(\mathbf{P}^1(\mathbb{C}))$, is given by the formula:

$$|J_p(f)| = \begin{cases}
\frac{1+z\bar{z}}{1+z'\bar{z}'}\operatorname{Abs}((xfx^{-1})'(z)), & \text{if } t = 1, t' = 1,\\
\frac{1+z\bar{z}}{1+t'\bar{t}'}\operatorname{Abs}((yfx^{-1})'(z)), & \text{if } t = 1, z' = 1,\\
\frac{1+t\bar{t}}{1+z'\bar{z}'}\operatorname{Abs}((xfy^{-1})'(t)), & \text{if } z = 1, t' = 1,\\
\frac{1+t\bar{t}}{1+t'\bar{t}'}\operatorname{Abs}((yfy^{-1})'(t)), & \text{if } z = 1, z' = 1.
\end{cases}$$

We remark that, in the case of a fixed point $p = f(p) = [z,t]$ and using normalized homogeneous coordinates, we only have two cases for the 1×1 Jacobian matrix:

$$\begin{aligned} J_p^{x,x} &= \big((xfx^{-1})'(z)\big) \quad \text{if } t = 1, \\ J_p^{y,y} &= \big((yfy^{-1})'(t)\big) \quad \text{if } z = 1. \end{aligned}$$

We can use the norm of the tangent map to give the following definition:

Definition 1.8. Let $f\colon \mathbf{P}^1(\mathbb{C}) \to \mathbf{P}^1(\mathbb{C})$ be an analytic function and $p \in \mathbf{P}^1(\mathbb{C})$ a fixed point. Then, p is said to be a ***super-attracting***, ***attracting***, ***indifferent*** or ***repelling*** fixed point if the norm (absolute value) of the tangent map at that point is zero, lower than 1, equal to 1 or greater than 1, respectively.

Then, in order to know if a fixed point is super-attracting, attracting, indifferent or repelling, it suffices to check if

$$|J_p(f)| = \begin{cases} \mathrm{Abs}((xfx^{-1})'(z)), & \text{if } t = 1, t' = 1, \\ \mathrm{Abs}((yfy^{-1})'(t)), & \text{if } z = 1, z' = 1, \end{cases}$$

is zero, lower than, equal to or greater than 1.

Knowing if a fixed point is super-attracting, attracting indifferent or repelling will be helpful later. In general, the attraction basins of super-attracting and attracting fixed points are “easily visible”; however, for repelling fixed points it may be necessary to apply some zooms on suitable local rectangles in order to see their attraction basins.

1.7 Basins of end points induced by a rational function on $\mathbb{C} \cup \{\infty\}$

Let $z \in \mathbb{C}$ and consider a function $h\colon \mathbb{C} \to \mathbb{C}$ of the form $h(z) = a\,\dfrac{P(z)}{Q(z)}$, where $a \in \mathbb{C}$, $a \neq 0$ and $P(z), Q(z)$ is a pair of irreducible polynomials of degree p, q, respectively. We have seen in subsection 1.4 that h induces a new map $f = h^+$ on $\mathbb{C} \cup \{\infty\}$, which gives to $\mathbb{C} \cup \{\infty\}$ the structure of a discrete semi-flow by the formula $n \cdot p = f^n(p)$. We also have the canonical map

$$\omega\colon \mathbb{C} \cup \{\infty\} \to \Pi(\mathbb{C} \cup \{\infty\})$$

given by $\omega(p) = [(p, f(p), f^2(p), \dots)]$.

Next, we shall study a particular example of discrete semi-flow induced by a rational function. Consider $h(z) = P(z)/Q(z)$, where $P(z) = 1 + 4z^5$ and $Q(z) = 5z^4$. In this case, the induced map $f = h^+$ has six fixed points:

$$p_0 = \infty, \quad p_1 = -0.809017 - 0.587785i, \quad p_2 = -0.809017 + 0.587785i,$$

$$p_3 = 0.309017 - 0.951057i, \quad p_4 = 0.309017 + 0.951057i, \quad p_5 = 1.$$

Therefore, the space X is divided into seven regions:

$$X = (X \setminus D) \sqcup D_\infty \sqcup D_{p_1} \sqcup D_{p_2} \sqcup D_{p_3} \sqcup D_{p_4} \sqcup D_{p_5},$$

where $D = D_\infty \cup D_{p_1} \cup D_{p_2} \cup D_{p_3} \cup D_{p_4} \cup D_{p_5}$. We can associate each one with a different color, as we see in Table 1. It is shown, moreover, which kind of fixed point (super-attracting, attracting, indifferent, or repelling) corresponds to every attraction basin.

Region	Color	Type of associated fixed point
$X \setminus D$	0	
$D_\infty = \omega^{-1}\omega(p_0)$	1	Repelling
$D_{p_1} = \omega^{-1}\omega(p_1)$	2	Super-attracting
$D_{p_2} = \omega^{-1}\omega(p_2)$	3	Super-attracting
$D_{p_3} = \omega^{-1}\omega(p_3)$	4	Super-attracting
$D_{p_4} = \omega^{-1}\omega(p_4)$	5	Super-attracting
$D_{p_5} = \omega^{-1}\omega(p_5)$	6	Super-attracting

Table 1. Relationship between regions, colors and fixed points.

In region $X \setminus D$, we can find points whose attraction basin corresponds to an end point which is not associated to any fixed point (for example, end points associated to a 2-cycle) or even points such that, after doing a prefixed limited number of iterations, belong to a sequence that did not converge to any fixed point yet (modulo a determined precision fixed beforehand). The rest of colors correspond to points which belong to the attraction basin of an end point (and they are associated to a certain fixed point).

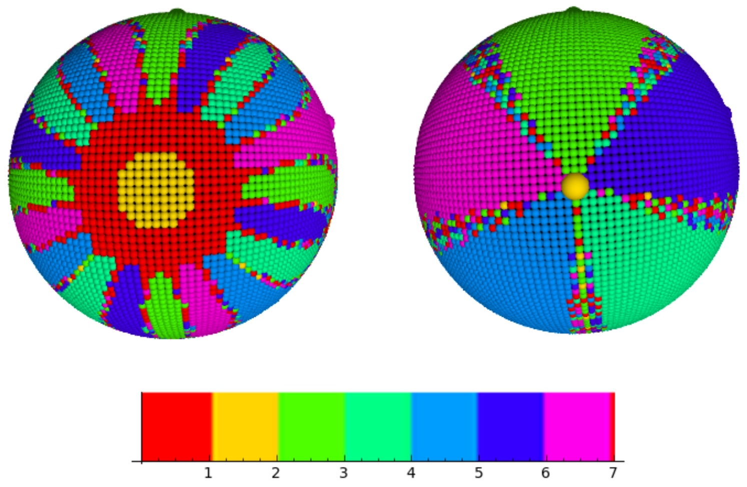

Figure 1. 3D fractal on sphere S^2 plotted with *Sage*. The image on the left represents those regions which are near the origin of coordinates (south pole of the sphere) and the image on the right shows regions which are close to the point at infinity (north pole of the sphere).

Figure 1 illustrates the particular example that we have considered in this subsection. It is clear that the basins of the points $\{p_1, p_2, p_3, p_4, p_5\}$ corresponds to colors $\{2, 3, 4, 5, 6\}$. The properties of the points of regions of color 0 (red) and 1 (yellow) is a little more complicated. Note that the points colored in yellow belong to the basin of a repelling fixed point. In this case, since we have chosen neither a high precision nor a high maximum number of iterations, some of these yellow points are not actually in the basin of the repelling fixed point ∞. The red points correspond to basins of end points induced by cycles and points that, with the specified maximum number of iterations and the given precision, are not close to a fixed point yet.

2 Description of the employed algorithms

Along previous subsections, we have introduced some mathematical techniques and developed basic theoretical aspects necessary to build computer programs with the ability of representing attraction basins of end points associated to a determined rational function. As usual in this work, a rational function f on $\mathbb{C} \cup \{\infty\}$ will be represented by a pair of homogeneous polynomials $A(z,t)$, $B(z,t)$ of the same degree (see subsection 1.4). We shall show in the next lines the algorithms which have been developed to study the basins induced by f.

2.1 Calculation of the fixed points of f

By Lemma 1.6, the set of roots

$$\{[z_1,t_1],\dots,[z_{n+1},t_{n+1}]\}$$

of $A(z,t)t - B(z,t)z = 0$ coincides with the set of fixed points of f.

- If $t = 0$ is a root of $B(1,t)$ and $z_1,\dots,z_n$ are the roots of $A(z,1) - B(z,1)z = 0$, then the set of fixed points of f is $\{[1,0],[z_1,1],\dots,[z_n,1]\}$.
- If $t = 0$ is not a root of $B(1,t)$ and $z_1,\dots,z_n,z_{n+1}$ are the roots of $A(z,1) - B(z,1)z = 0$, $\{[z_1,1],\dots,[z_n,1],[z_{n+1},1]\}$ is the set of fixed points of f.

A function called `fixedPointsZeros(A,B)` has been built in *Sage*. This function returns a list containing all the fixed points of a given rational function f induced by A, B.

The following is an example of use: let

```
P.<z,t> = PolynomialRing(CC,2)
A=t**4+3*z**4
B=4*t*z**3
```

If we take as input

```
fixedPointsZeros(A,B),
```

the obtained output is

```
[(1, 0), (-1.000, 1), (1.000, 1), (-1.000*I, 1), (1.000*I, 1)]
```

Note that every point in the output is given in normalized homogeneous coordinates. In this case, the first point represents ∞.

The developed algorithm is:

```
def fixedPointsZeros (U, V) :
    U = U + 0*I
    V = V + 0*I
    L = CC[z]
    if (expand(U(t=1) - V(t=1)*z)!=0):
        con = V(z=1, t=0)
        solaux = (L(U(t=1) - V(t=1)*z)).roots()
        sol1 = [homogeneousNormalization((t[0], 1))
                for t in solaux]
        if (con == 0): sol1.insert(0, (1, 0))
    else:
        sol1 = []
        print "There was a problem when solving equation
                f(x)=x: cannot solve equation 0=0."
    return sol1
```

Observe that the subroutine `homogeneousNormalization` is used within the algorithm. Its implementation is:

```
def homogeneousNormalization(twotuple):
    if abs(twotuple[0]) < abs(twotuple[1]):
        return (twotuple[0] / twotuple[1], 1)
    else:
        return (1, twotuple[1] / twotuple[0])
```

This subroutine allows to obtain the normalized homogeneous coordinates of any point of $\mathbb{C}\cup\{\infty\}$.

2.2 Distance between two points

The chordal distance between any two points in $\mathbf{P}^1(\mathbb{C})$ can be obtained by using the bijection from $\mathbf{P}^1(\mathbb{C})$ to S^2 which appeared in subsection 1.5 and the Euclidean metric on S^2. To that end, the following functions were developed in *Sage* (an example of use is given):

```
                    sphereBijection((1,0))
                           (0,0,1)
                 chordalMetric((1,0),(1,1))
                      1.41421356237310
```

Function `chordalMetric` uses the Euclidean metric d^E to calculate distances between pairs of points in S^2, instead of the Riemannian metric. This is because its computational cost would be greater and it would be more inefficient if it employed the metric d^R, since in that case it would use inverse trigonometric functions to do the appropriate calculations. In fact, `chordalMetric` makes use of the map d_1^E defined in subsection 1.5.

An implementation in *Sage* of the functions that we have just presented is shown below:

```
def sphereBijection(twotuple):
    z = twotuple[0]
    t = twotuple[1]
    return ((conjugate(z)*t + conjugate(t)*z) /
            (conjugate(t)*t + conjugate(z)*z),
        (I*(conjugate(z)*t - conjugate(t)*z)) /
            (conjugate(t)*t + conjugate(z)*z),
        (-conjugate(t)*t + conjugate(z)*z) /
            (conjugate(t)*t + conjugate(z)*z))

def chordalMetric(twotuple, twotuple1):
    t1 = sphereBijection(twotuple)
    t2 = sphereBijection(twotuple1)
    m1 = Matrix([[t1[0], t1[1], t1[2]]]);
    m2 = Matrix([[t2[0], t2[1], t2[2]]]);
    return n(norm(m1-m2))
```

2.3 Iteration of the rational map f

With a view to find an end point associated to a point $x \in \mathbf{P}^1(\mathbb{C})$, the rational map f must be iterated to obtain a finite sequence

$$(x, f(x), f^2(x), f^3(x), \ldots, f^{k-1}(x), f^k(x)).$$

In this context, remind that a maximum number of iterations l must be considered and a certain precision c must be prefixed to determine when to stop the iterative process while programming the function which returns such sequence. That is why we shall always work with sequences in which $k < l$.

After each iteration, there will be two possible cases:

1) If the chordal distance from $f^{k-1}(x)$ to $f^k(x)$ is lower than 10^{-c}, then take as output the list $[f^k(x), k]$; otherwise, case 2) is applied.

2) If $k < l$, a new iteration is done and case 1) is applied again; otherwise (if $k = l$), then the output $[f^l(x), l]$ is taken.

The following implementation, `newstep`, was developed in *Sage* according to what was intended (an example of use is given):

```
newstep(A,B,4,25,(0,1))
        [(1,1),2]
```

The source code of the function is:

```
def newstep(U, V, iter, precision, pointinternumber):
    point = pointinternumber
    number = 0
    imagepoint = rationalFunction(U, V, point)
    while (chordalMetric(point, imagepoint) > 10.**(-precision))
```

```
            and (number < iter):
        point = imagepoint
        imagepoint = rationalFunction(U, V, point)
        number = number + 1
    return [imagepoint, number]
```

Observe that `newstep` uses the subroutine `rationalFunction` in order to calculate after each iteration the image by the given rational map of the corresponding point and to normalize its coordinates. The source code of the subroutine is shown below:

```
def rationalFunction(U, V, twotuple):
    m = twotuple[0]
    n = twotuple[1]
    return homogeneousNormalization((U(m, n), V(m, n)))
```

2.4 Determination of the fixed point to which an iteration sequence converges and number of iterations until convergence

Consider the ordered sequence of fixed points $\{x_1, x_2, \ldots, x_{n+1}\}$ associated to a rational map. In the same way, given a point $x \in \mathbb{C} \cup \{\infty\}$, consider the iteration sequence

$$(x, f(x), f^2(x), f^3(x), \ldots, f^{k-1}(x), f^k(x)).$$

If there exists an $i \in \{1, \ldots, n+1\}$ such that the chordal distance from $f^k(x)$ to the fixed point x_i is lower than 10^{-c}, then the function `positionIterationNumber` described below must return $[i, k]$. Otherwise, $k = l$ and the output must be $[0, l]$, where l is the maximum number of iterations which was prefixed beforehand. Examples of use:

```
positionIterationNumber(A,B,fixedPointsZeros(A,B),25,4,(-0.1-0.1*i,1))
                                [0,25]

positionIterationNumber(A,B,fixedPointsZeros(A,B),25,4,(-0.1-0.09*i,1))
                                [5,23]
```

The input parameter `fixedpointlist` is assigned to the list of fixed points associated to the given rational map, which will have been previously created within the function `fixedPointsZeros`.

The implementation of `positionIterationNumber` is:

```
def positionIterationNumber(U, V, fixedpointlist, iter,
        precision, twotuple):
    result = newstep(U, V, iter, precision, twotuple)
    if (result[1] != iter):
        return [position(fixedpointlist, precision, result[0]),
                result[1]]
    else:
        return [0, result[1]]
```

The subroutine `position`, which appears in the subprogram above, returns the exact position within the list `fixedpointlist` where the fixed point to which the iteration sequence converges is found; in case that such sequence does not converge to any fixed point, it returns 0. An implementation of this subroutine is shown in the next lines:

```
def position(fixedPointList, precision, twotuple):
    pos = -1
    iter = 0
    le = len(fixedPointList)
    while (iter < le) and (pos == -1):
        if (chordalMetric(twotuple, fixedPointList[iter]) <
                10.**(-precision)):
            pos = iter
        else:
            iter = iter + 1
    else:
        return pos + 1
```

2.5 Derivative of a rational function at a fixed point

In order to know if a fixed point is super-attracting, attracting, indifferent or repelling (see subsection 1.6), the derivative of the corresponding rational function can be calculated by using the following algorithm (an example of use is given):

```
fixedPointsTangentMapNorm(A,B,(1,0))
     ((1,0),1.33333333333333)
```

Suppose that $A(z,t)$, $B(z,t)$ are homogeneous polynomials and that $[z,t]$ is a fixed point represented in normalized homogeneous coordinates. Then, the subprogram `fixedPointsTangentMapNorm` returns a list containing two elements: the considered fixed point $[z,t]$ and the absolute value of the derivative of the rational function at that point.

The implementation of the described algorithm, developed in *Sage*, is given by:

```
def fixedPointsTangentMapNorm(A, B, twotuple):
    a, b = var('a, b')
    nor = homogeneousNormalization(twotuple)
    if (nor[1] == 1):
        return (nor, abs(derivative(A(a, 1)/B(a, 1),a)(a = nor[0])))
    else:
        return (nor, abs(derivative(B(1, b)/A(1, b),b)(b = nor[1])))
```

2.6 Fractal plotting

Next we shall show the algorithms and subroutines which have been developed with the aim of representing basins of end points associated to a rational function on $\mathbb{C} \cup \{\infty\}$ and even on S^2, as well as the strategies employed to build those subprograms which have been followed in order to plot the corresponding fractals.

2.6.1 Strategies

We shall plot a fractal by using the pair given by the fixed point to which the iteration sequence converges (maybe such point does not exist) and the number of iterations until convergence, together with one of the following strategies:

1) Fixed point to which the iteration sequence converges: A color is assigned to each point x in $\mathbb{C}\cup\{\infty\}$ according to the fixed point to which the trajectory $(f^k(x))_{k\in\mathbb{N}}$ converges. That point is drawn with another different color if the trajectory did not converge yet after a determined number of iterations. In this way, attraction basins can be distinguished by their colors. The algorithm which sets the color of each point in $\mathbb{C}\cup\{\infty\}$ is:

```
def onlyPosition(U, V, fixedpointlist, iter, precision,
        twotuple):
    return (positionIterationNumber(U, V, fixedpointlist,
            iter, precision, twotuple)[0])
```

2) Number of iterations until convergence: Instead of assigning a color to each point taking into account the reached fixed point, that color is assigned in accordance with the number of needed iterations until convergence, given a prefixed precision. Eye-catching drawings may be generated with this strategy, too. The subroutine which allows to find the number of iterations associated to a given point is:

```
def onlyConvergence(U, V, fixedpointlist, iter, precision,
        twotuple):
    return (positionIterationNumber(U, V, fixedpointlist,
            iter, precision, twotuple)[1])
```

3) Combination of the both previous strategies: In this case, a color is assigned to each attraction basin, but making it lighter or darker depending on the number of needed iterations until convergence. An implemented subprogram in *Sage* which satisfies this strategy is shown below:

```
def positionPlusConvergence(U, V, fixedpointlist, iter,
        precision, twotuple):
    pair = positionIterationNumber(U, V, fixedpointlist, iter,
            precision, twotuple)
    if(pair[0] == 0):
        away = 0
    else:
        away = pair[1]
    return n(pair[0] + away / iter * 3/4)
```

Our program uses specific color palettes to plot fractals, based on the color maps of *Sage*. If we are working with strategy 1) and the homogeneous polynomials which induce the rational function are of degree n, then a palette with $n+2$ colors will be used: the first color of the palette will be associated to points belonging to a basin of an end point which corresponds to no fixed points (for example, end points associated to 2-cycle points) or points whose induced trajectories did not converge to any fixed point yet, after a prefixed finite number of iterations; the other colors are related to points which are in the attraction basin of an end point associated to a fixed point. If strategy 2) is considered, a palette with $l+1$ colors will be used (where l is the number of predefined maximum iterations), being the first color reserved for those points whose corresponding trajectory does not converge to any fixed point (because of any of the reasons explained above in this paragraph) and being the rest of colors associated to each one of the possible numbers of iterations $k \in \{0, 1, \ldots, l-1\}$. On the other hand, if strategy 3) is chosen, then a graduated palette constructed from a specified color map of *Sage* will be given. Figure 2 shows some examples of such color palettes.

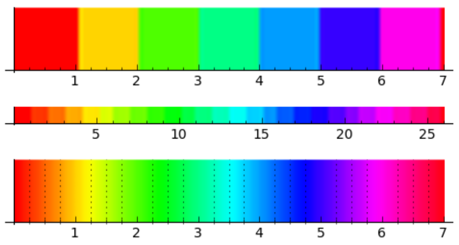

FIGURE 2. Different color palettes associated to the same color map of *Sage* ('hsv', in this case). The images correspond respectively to the strategies 1), 2) and 3) shown above.

2.6.2 Algorithms

The subprograms developed in *Sage* responsible for plotting fractals corresponding to basins of end points associated to rational functions are the following: `fractalPlotInsideOutside`, `fractalPlot`, `spherePlot` and `cubicSpherePlot`:

- Function `fractalPlotInsideOutside` returns two disks: one of them represents the intersection between the attraction basins and the unit disk, and the other shows by means of the inversion method the intersection of that basins with the complementary of the unit disk on $\mathbb{C} \cup \{\infty\}$ (see Figure 3).

- With function `fractalPlot`, a colored fractal in a rectangular region is obtained (see Figure 4).

- A 3D fractal in the unit sphere is obtained with `spherePlot`, showing all the fixed points in a bigger size than the others. An example of what we can get with this function was shown in Figure 1, and another one can be found in Figure 5.

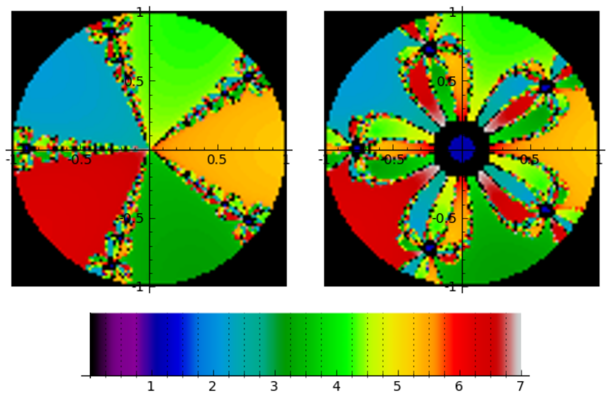

FIGURE 3. Fractal plotted by the algorithm `fractalPlotInsideOutside`, obtained applying strategy 3).

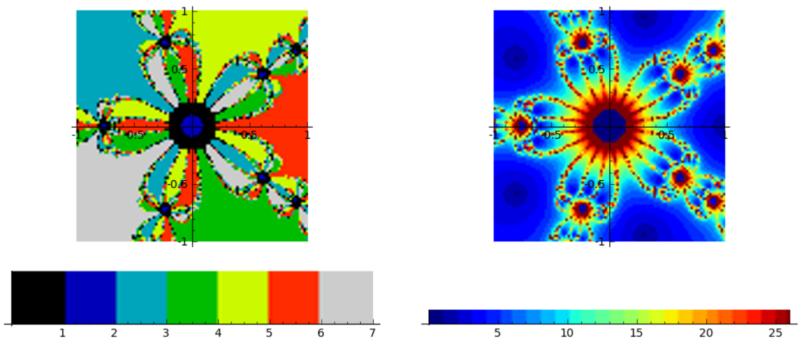

FIGURE 4. Fractals plotted by the algorithm `fractalPlot`. The image on the left corresponds to strategy 1) and the image on the right was drawn applying strategy 2).

- The program `cubicSpherePlot` returns the same as `spherePlot`, but the sphere obtained with the former function is a bit different from the one returned by the latter, since its points are distributed all over its surface in a different way (by projecting the boundary of a subdivided cube onto the unit sphere). A comparison between both functions is established in Figure 6.

In all cases above, a list containing the fixed points of the rational map, the absolute values of the derivative of the rational function at that fixed points (which allows to know if every fixed point is super-attracting, attracting, repelling or indifferent) and a color palette associated to the attraction basins are returned as well.

`fractalPlotInsideOutside`, `spherePlot` and `cubicSpherePlot` always have the numerator and denominator of a rational function as input parameters, whereas that `fractalPlot` have in

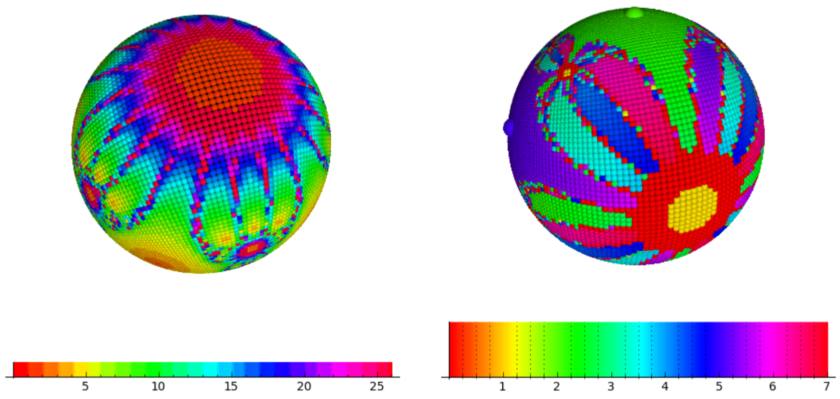

FIGURE 5. Fractals plotted by the algorithm `spherePlot` by considering strategies 2) and 3), respectively. Both fractals were obtained from the same rational function.

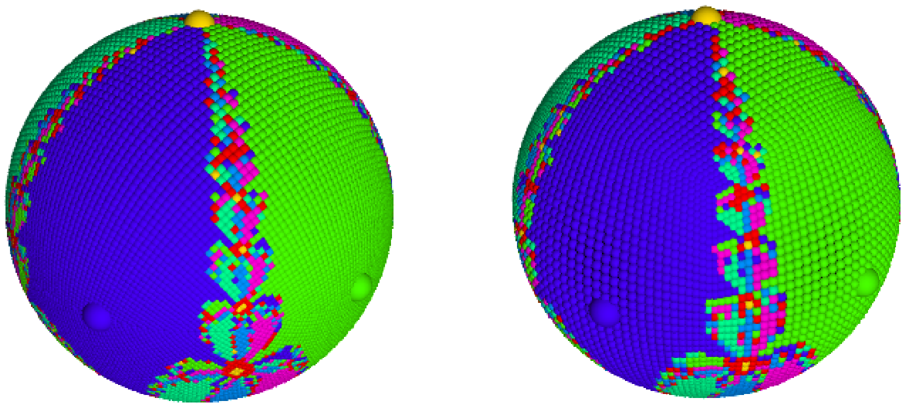

FIGURE 6. Comparison between output plots obtained with `spherePlot` (on the top) and `cubicSpherePlot` (on the bottom). The algorithm `cubicSpherePlot` projects points belonging to the faces of a cube onto the unit sphere.

addition as optional input parameters the points which delimit the rectangular area where the fractal will be plotted. Furthermore, all the plotting algorithms described also have several other optional input parameters: precision, maximum number of iterations, plot points, coloring strategy, color map (except `spherePlot` and `cubicSpherePlot`) and number of compositions of the given rational function f with itself.

This latter parameter allows to work with polynomials associated to f^2, f^3,... As an example of what it is useful for, note that the attraction basins of the fixed points of $f^2 = f \circ f$ correspond, together, to basins of fixed points and 2-cycle points of f; that is, if two fixed points of f^2 form a 2-cycle, its associated attraction basin is the union of the basins of these two fixed points. This fact can be generalized to any number of iterations.

The subprogram responsible for composing n times a given rational function f/g with itself and obtaining a homogeneous rational map from it is:

`composeHomogenize(f,g,n)`.

An implementation in *Sage* of this subprogram is given:

```
def composeHomogenize(f, g, n):
    x = var('x')
    if(g.degree() == 0):
        comp = compose(f, n)
        homo = homogenize(comp, g)
    else:
        h = f/g;
        comp = compose(h, n)
        comp1 = (comp+0*I).simplify_rational('simple')
        S = P.fraction_field()
        comp2 = S(sage_eval(repr(comp1),locals=locals()))
        num = comp2.numerator(); den = comp2.denominator()
        homo = homogenize(num, den)
    A = homo[0]; B = homo[1]
    return [A, B]
```

Observe that subroutines `compose(h,num)` and `homogenize(f,g)` are used in the subprogram above. The first of these subroutines composes a rational function h with itself a number of times indicated by the input parameter *num* (option `simplify_rational('simple')` is passed to preserve the composed rational function in the form of a quotient of polynomials) whereas the second one homogenizes a given rational function f/g. Their implementations developed in *Sage* are the following:

```
def compose(h, num):
    H = h
    if (num < 1):
        print("Rational function must be composed once at least.")
    for cont in range(num-1): H = h.subs(x = H)
    return H

def homogenize(f, g):
    z = var('z')
    P.<x,t> = PolynomialRing(CC,2)
    fdeg = f.degree(); gdeg = g.degree()
    deg = max(fdeg, gdeg)
    f = f.homogenize('t'); g = g.homogenize('t')
    if (fdeg > gdeg): g = g*t**(fdeg - gdeg)
    else: f = f*t**(gdeg - fdeg)
    F = f.subs(x = z); G = g.subs(x = z)
    return [F, G]
```

To conclude, the source codes of the plotting algorithms we have just presented are shown below.

```
from sage.plot.density_plot import DensityPlot
def fractalPlotInsideOutside(M,N,points=100,function=onlyPosition,
        iter=25,prec=3,ncomp=1,reflection=-1,colorfunction='spectral'):
    ch=composeHomogenize(M,N,ncomp)
    A=ch[0];B=ch[1]
    p=fixedPointsZeros(A,B)
    if len(p)>0:
        grad=len(p)-1
        if(function==onlyConvergence): range=iter+1
        else: range=grad+1.99
        x,y=var('x,y')
        def f(x,y):
            if(x**2+y**2>1): return 0
            else: return function(A,B,p,iter,prec,(x+y*I,1))
        def f1(x,y):
            if(x**2+y**2>1): return 0
            else:
                return function(A,B,p,iter,prec,
                        (1,x+reflection*y*I))
        table=[fixedPointsTangentMapNorm(A,B,t) for t in p]
        if(function!=positionPlusConvergence):
            return density_plot(f,(x,-1,1),(y,-1,1),
                    cmap=colorfunction,plot_points=points),
                density_plot(f1,(x,-1,1),(y,-1,1),
                    cmap=colorfunction,plot_points=points),
                density_plot(floor(x),(x,0,range),(y,0,1),
                    cmap=colorfunction,
                    plot_points=100).show(ticks=[None,[]]),
                table
        else:
            return density_plot(f,(x,-1,1),(y,-1,1),
                    cmap=colorfunction,plot_points=points),
                density_plot(f1,(x,-1,1),(y,-1,1),
                    cmap=colorfunction,plot_points=points),
                density_plot(x,(x,0,range),(y,0,1),
                    cmap=colorfunction,
                    plot_points=100).show(ticks=[None,[]],
                            gridlines=["minor",None]),
                table

def fractalPlot(M,N,xmin,xmax,ymin,ymax,points=100,
        function=onlyPosition,iter=25,prec=3,ncomp=1,
    colorfunction='spectral'):
    ch=composeHomogenize(M,N,ncomp)
    A=ch[0];B=ch[1]
    p=fixedPointsZeros(A,B)
    if len(p)>0:
        grad=len(p)-1
        if(function==onlyConvergence):
            range=iter+1
        else:
            range=grad+1.99
        x,y=var('x,y')
        def f(x,y):
            return function(A,B,p,iter,prec,(x+y*I,1))
        table=[fixedPointsTangentMapNorm(A,B,t) for t in p]
        if(function!=positionPlusConvergence):
            return density_plot(f,(x,xmin,xmax),(y,ymin,ymax),
                    cmap=colorfunction,plot_points=points),
                density_plot(floor(x),(x,0,range),(y,0,1),
                    cmap=colorfunction,
                    plot_points=100).show(ticks=[None,[]]),
                table
        else:
            return density_plot(f,(x,xmin,xmax),(y,ymin,ymax),
                    cmap=colorfunction,plot_points=points),
                density_plot(x,(x,0,range),(y,0,1),
                    cmap=colorfunction,
                    plot_points=100).show(ticks=[None,[]],
                        gridlines=["minor",None]),
                table
```

```
def spherePlot(M,N,function=onlyPosition,rotzoom=((0,0,0),1),points=100,
        ncomp=1,view='tachyon',iter=25,prec=3):
    ch=composeHomogenize(M,N,ncomp)
    A=ch[0];B=ch[1]
    p=fixedPointsZeros(A,B)
    if len(p)>0:
        grad=len(p)-1
        if(function==onlyConvergence): ran=iter+1
        else: ran=grad+2
        x,y,twot=var('x,y,twot')
        def f(x,y,twot):
            if(sphereBijection(twot)[2]>0):
                return function(A,B,p,iter,prec,(1,x-y*I))
            else:
                return function(A,B,p,iter,prec,(x+y*I,1))
        def g(x,y):
            if(x**2+y**2<=1):
                return point3d(sphereBijection((1,x-y*I)),
                        color=hue(f(x,y,(1,x-y*I))/ran))
        def g1(x,y):
            if(x**2+y**2<=1):
                return point3d(sphereBijection((x+y*I,1)),
                        color=hue(f(x,y,(x+y*I,1))/ran))
        table=[fixedPointsTangentMapNorm(A,B,t) for t in p]
        l=[(n(k/(points//2))) for k in range(1,(points//2)+1)]
        for k in range((points//2)):
            l.append(n((-1)*l[k]))
        l.append(0)
        pl=[g(u,v) for u in l for v in l]
        for u in l:
            for v in l:
                pl.append(g1(u,v))
        if(function!=onlyConvergence):
            for ind in range(ran-1):
                pl.append(point3d(sphereBijection(p[ind]),size=15,
                        color=hue((ind+1)/ran)))
        if(function!=positionPlusConvergence):
            return sum(pl,sphere(color='black')).rotateX(rotzoom[0][0])
                    .rotateY(rotzoom[0][1]).rotateZ(rotzoom[0][2])
                    .show(frame=False,viewer=view,aspect_ratio=[1,1,1],
                            zoom=rotzoom[1]),
                density_plot(floor(x),(x,0,ran),(y,0,1),
                    cmap='hsv',plot_points=100).show(ticks=[None,[]]),
                table
        else:
            return sum(pl,sphere(color='black')).rotateX(rotzoom[0][0])
                    .rotateY(rotzoom[0][1]).rotateZ(rotzoom[0][2])
                    .show(frame=False,viewer=view,aspect_ratio=[1,1,1],
                            zoom=rotzoom[1]),
                density_plot(x,(x,0,ran),(y,0,1),
                    cmap='hsv',plot_points=100).show(ticks=[None,[]],
                        gridlines=["minor",None]),
                table

def cubicSpherePlot(M,N,function=onlyPosition,rotzoom=((0,0,0),1),numdiv=40,
        ncomp=1,view='tachyon',iter=25,prec=3):
    ch=composeHomogenize(M,N,ncomp);A=ch[0];B=ch[1];p=fixedPointsZeros(A,B)
    if len(p)>0:
        grad=len(p)-1
        if(function==onlyConvergence): ran=iter+1
        else: ran=grad+2
        cube=[]
        for y1 in range(numdiv):
            for x1 in range(numdiv):
                cube.append((((2*x1/numdiv-1)+(2*(x1+1)/numdiv-1))/2,
                        ((2*y1/numdiv-1)+(2*(y1+1)/numdiv-1))/2,-1))
                cube.append((((2*x1/numdiv-1)+(2*(x1+1)/numdiv-1))/2,
                        ((2*y1/numdiv-1)+(2*(y1+1)/numdiv-1))/2,1))
                cube.append((((2*x1/numdiv-1)+(2*(x1+1)/numdiv-1))/2,
                        -1,((2*y1/numdiv-1)+(2*(y1+1)/numdiv-1))/2))
                cube.append((((2*x1/numdiv-1)+(2*(x1+1)/numdiv-1))/2,
                        1,((2*y1/numdiv-1)+(2*(y1+1)/numdiv-1))/2))
                cube.append((-1,((2*x1/numdiv-1)+(2*(x1+1)/numdiv-1))/2,
                        ((2*y1/numdiv-1)+(2*(y1+1)/numdiv-1))/2))
                cube.append((1,((2*x1/numdiv-1)+(2*(x1+1)/numdiv-1))/2,
                        ((2*y1/numdiv-1)+(2*(y1+1)/numdiv-1))/2))
        def cubeSphere(p):
            root=sqrt(p[0]**2+p[1]**2+p[2]**2)
            return (n(p[0]/root),n(p[1]/root),n(p[2]/root))
        def sphereComplexProjLine(p):
            if(p[2]==1): return (1,0)
            else: return (p[0]/(1-p[2])+I*p[1]/(1-p[2]),1)
        pl=[]
        for k in cube:
            spherePoint=cubeSphere(k)
            pl.append(point3d(spherePoint,color=hue(function(A,B,p,iter,
                    prec,sphereComplexProjLine(spherePoint))/ran)))
        if(function!=onlyConvergence):
            for ind in range(ran-1):
                pl.append(point3d(sphereBijection(p[ind]),size=15,
                        color=hue((ind+1)/ran)))
        table=[fixedPointsTangentMapNorm(A,B,t) for t in p];x,y=var('x,y')
        if(function!=positionPlusConvergence):
            return sum(pl,sphere(color='black')).rotateX(rotzoom[0][0])
                    .rotateY(rotzoom[0][1]).rotateZ(rotzoom[0][2])
                    .show(frame=False,viewer=view,aspect_ratio=[1,1,1],
                            zoom=rotzoom[1]),
                density_plot(floor(x),(x,0,ran),(y,0,1),
                    cmap='hsv',plot_points=100).show(ticks=[None,[]]),table
        else:
            return sum(pl,sphere(color='black')).rotateX(rotzoom[0][0])
                    .rotateY(rotzoom[0][1]).rotateZ(rotzoom[0][2])
                    .show(frame=False,viewer=view,aspect_ratio=[1,1,1],
                            zoom=rotzoom[1]),
                density_plot(x,(x,0,ran),(y,0,1),
                    cmap='hsv',plot_points=100).show(ticks=[None,[]],
                            gridlines=["minor",None]),table
```

3 User manual

The program is really easy to use. In order to plot the fractal associated to a given rational function, it suffices to specify the polynomials which form its numerator and denominator in variable x and execute one of the following subroutines, depending on what kind of fractal we want to draw: either `fractalPlotInsideOutside` or `fractalPlot` or `spherePlot` or `cubicSpherePlot` (see paragraph 2.6.2). For example, the fractal plotted in Figure 3, whose associated rational function is $\frac{4x^5+1}{5x^4}$, was obtained simply by typing the following sequence in *Sage*:

```
P.<x,t> = PolynomialRing(CC,2)
M=4*x**5+1; N=5*x**4
fractalPlotInsideOutside(M,N,100,positionPlusConvergence)
```

Next we show the input parameters of the plotting functions that are supported:

- **fractalPlotInsideOutside**(*M, N, points = 100, function = onlyPosition, ncomp = 1, colorfunction = 'spectral', iter = 25, prec = 3, reflection = -1*)

  - `M,N` are the numerator and the denominator of the given rational function in variable x, respectively.
  - `points` is an integer (default: 100) that represents the number of points to plot in each direction of the grid.
  - `function` indicates the strategy employed in order to plot the fractal: `onlyPosition` (which is set by default), `onlyConvergence` or `positionPlusConvergence`.
  - `ncomp` is an integer (default: 1) which represents the number of times that the rational function has to be composed with itself.
  - `colorfunction` is a colormap of *Sage* that is used to assign a color to each complex point. The colormap set by default is `spectral`.
  - `iter` is an integer (default: 25) that represents the maximum number of iterations of the rational function.
  - `prec` is an integer (default: 3) such that, if the distance between two points is lower than $10^{-\text{prec}}$, then the developed algorithm considers those points as if they were the same.
  - `reflection` is a number either equal to 1 or to -1 that indicates the sign of the reflection of the inversion method.

- **fractalPlot**(*M, N, xmin, xmax, ymin, ymax, points = 100, function = onlyPosition, ncomp = 1, colorfunction = 'spectral', iter = 25, prec = 3*)
  - `M,N` are the numerator and the denominator of the given rational function in variable x, respectively.
  - The tuple given by `xmin,xmax,ymin,ymax` represents the vertices of the rectangle in which the fractal will be plotted.
  - `points` is an integer (default: 100) that represents the number of points to plot in each direction of the grid.
  - `function` indicates the strategy employed in order to plot the fractal: `onlyPosition` (which is set by default), `onlyConvergence` or `positionPlusConvergence`.
  - `ncomp` is an integer (default: 1) which represents the number of times that the rational function has to be composed with itself.
  - `colorfunction` is a colormap of *Sage* that is used to assign a color to each complex point.
  - `iter` is an integer (default: 25) that represents the maximum number of iterations of the rational function.
  - `prec` is an integer (default: 3) such that, if the distance between two points is lower than $10^{-\text{prec}}$, then the developed algorithm considers those points as if they were the same.

- **spherePlot**(*M, N, function = onlyPosition, rotzoom = ((0,0,0),1), points = 100, ncomp = 1, view = 'tachyon', iter = 25, prec = 3*)
  - `M,N` are the numerator and the denominator of the given rational function in variable x, respectively.

- `function` indicates the strategy employed in order to plot the fractal: `onlyPosition` (which is set by default), `onlyConvergence` or `positionPlusConvergence`.
- `rotzoom` is a tuple given by two elements: a 3-tuple $(xrot, yrot, zrot)$ and a positive real number z. Due to this parameter, the algorithm returns the sphere zoomed z times and self-rotated about the x-axis, y-axis and z-axis by the angles $xrot$, $yrot$ and $zrot$, respectively.
- `points` is an integer (default: 100) that represents the number of points to plot in each direction of the grid.
- `ncomp` is an integer (default: 1) which represents the number of times that the rational function has to be composed with itself.
- `view` is a string that indicates the viewer which will be used in order to see the plot. Possible values: 'jmol', 'tachyon' (by default), 'java3d' and 'canvas3d'.
- `iter` is an integer (default: 25) that represents the maximum number of iterations of the rational function.
- `prec` is an integer (default: 3) such that, if the distance between two points is lower than $10^{-\text{prec}}$, then the developed algorithm considers those points as if they were the same.

- **cubicSpherePlot**(*M, N, function = onlyPosition, rotzoom = ((0,0,0),1), numdiv = 40, ncomp = 1, view = 'tachyon', iter = 25, prec = 3*)
  - `M,N` are the numerator and the denominator of the given rational function in variable x, respectively.
  - `function` indicates the strategy employed in order to plot the fractal: `onlyPosition` (which is set by default), `onlyConvergence` or `positionPlusConvergence`.
  - `rotzoom` is a tuple given by two elements: a 3-tuple $(xrot, yrot, zrot)$ and a positive real number z. Due to this parameter, the algorithm returns the sphere zoomed z times and self-rotated about the x-axis, y-axis and z-axis by the angles $xrot$, $yrot$ and $zrot$, respectively.
  - `numdiv` is an integer (default: 40) that indicates the number of subdivisions of the faces of the cube which is projected upon the unit sphere.
  - `ncomp` is an integer (default: 1) which represents the number of times that the rational function has to be composed with itself.
  - `view` is a string that indicates the viewer which will be used in order to see the plot. Possible values: 'jmol', 'tachyon' (by default), 'java3d' and 'canvas3d'.
  - `iter` is an integer (default: 25) that represents the maximum number of iterations of the rational function.
  - `prec` is an integer (default: 3) such that, if the distance between two points is lower than $10^{-\text{prec}}$, then the developed algorithm considers those points as if they were the same.

4 Applications of the algorithms

4.1 Rational functions induced by iterative numerical methods

In the next paragraph, we shall show how to apply the algorithms described above to iterative numerical processes (like Newton or Chebyshev methods) which are used in the search of complex polynomial roots.

Consider the Taylor expansion of a polynomial $f\colon \mathbb{C} \to \mathbb{C}$:

$$f(x) = f(x_n) + f'(x_n)(x - x_n) + \frac{f''(x_n)}{2}(x - x_n)^2 + \cdots$$

If this expansion is truncated at the second (linear) term and evaluated at x_{n+1}, it follows that:

$$f(x_{n+1}) \simeq f(x_n) + f'(x_n)(x_{n+1} - x_n).$$

Moreover, if it is supposed that $f(x_{n+1}) \simeq 0$ (that is, x_{n+1} approaches a root of f), we obtain the expression:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)},$$

which is known as ***Newton-Raphson algorithm***. Taking this into account, the rational function induced by this method is:

$$\mathrm{N}_f(x) = x - \frac{f(x)}{f'(x)} = \frac{xf'(x) - f(x)}{f'(x)}.$$

Therefore, if we start from the polynomial $f(x) = x^5 - 1$, this process induces the rational function

$$\mathrm{N}_f(x) = \frac{xf'(x) - f(x)}{f'(x)} = \frac{4x^5 + 1}{5x^4},$$

which is exactly the function considered in the last section. That means that, in this work, we have actually found the roots of the polynomial $f(x) = x^5 - 1$ by calculating the fixed points of the rational function $\dfrac{4x^5+1}{5x^4}$.

In general, the roots of a complex polynomial are fixed points of the rational function obtained with the most usual iterative numerical methods. As indicated in subsection 1.2, each fixed point of a rational function can be directly regarded as an end point. Thus, every attraction basin of a complex polynomial root is just the attraction basin of its associated end point.

If p is a root of the original complex polynomial, the meaning that a point x is in the attraction basin corresponding to the fixed point p is that, when applying the rational function associated to the considered numerical method to the point x, the trajectory is approaching the root p of the given polynomial. The designed programs are able to calculate graphically those attraction basins and, in addition, to indicate the number of iterations until convergence, which will be useful to know the speed of convergence of a point that belongs to the basin of a determined root.

There are many other iterative methods that, like Newton-Raphson method, induce a rational function which can be used to find the roots of a complex polynomial. Several of them can be found in [9], where a visual comparison of the fractal pictures that appear when the different iterative methods for solving a unique equation are applied is given. For a general study about iteration of complex rational functions, see [1].

4.2 Connections with Fractal Geometry

This work is closely connected to some aspects of Fractal Geometry, like Fatou set and Julia set; we shall remind briefly these both notions in order to relate them to what we are studying.

For a rational function f of degree greater than or equal to 2, the ***Julia set*** $J(f)$ can be described as:

$$J(f) = \overline{\{x \in X \mid x \text{ is a periodic repelling point}\}}.$$

The corresponding ***Fatou set*** is precisely the complementary set of $J(f)$ in X. Intuitively, we can say that a point $x \in X$ belongs to the Fatou set if there exists an open neighborhood U of x such that $\omega(x) = \omega(y)$, $\forall y \in U$ (that is, if the basin of an end point associated to any point which is close to x is the same as the basin of the end point to which x belongs).

As we can read in [3], the Julia set $J(f)$ is an uncountable compact set containing no isolated points and it is the boundary of the basin of attraction of each attractive fixed point of f, including ∞, and $J(f) = J(f^p)$ for each positive integer p. Consequently, in the present case, we could consider that the Julia set is formed by points which are in the boundary of the basins of attractive points (and hence, the rest of points of X are in the Fatou set).

4.3 Future applications

Our algorithms may be employed with the aim of giving a numerical estimate of the areas of the attraction basins associated to each root of a polynomial, as well as the probability of an initial point $x_0 \in \mathbb{C} \cup \{\infty\}$ being in one region or another. This purpose becomes possible by adding the surface areas of all the small spherical quadrilaterals projected from a cube onto the unit sphere (that is the same idea as we used in subroutine `cubicSpherePlot`), grouping them by their colors. All these considerations may be important in the context of some future research directions in the field of Numerical Analysis.

Furthermore, some portions of the algorithms could also be useful to make a detailed study of the Julia sets of the obtained fractals by calculating their fractal dimension and Betti numbers, which would allow us to know the number of connected components, topological holes, etc.

Bibliography

[1] A. F. Beardon, *Iteration of Rational Functions,* Graduate Texts in Mathematics, Springer-Verlag, New York, 1991.

[2] D. C. De Mars, *Fractal Domains: User Manual*, 2004.

[3] K. Falconer, *Fractal Geometry. Mathematical Foundations and Applications,* Wiley, New York, 204, 1990.

[4] J. M. García, L. J. Hernández and M. T. Rivas, *Limit and end functors of dynamical systems via exterior spaces*, Preprint, arXiv:1202.1635v1, 2012.

[5] O. Lewis, *Gereralized Julia sets: An extension of Cayley's problem,* Doctoral thesis, Harvey Mudd College (Claremont, California), Department of Mathematics (May 2005).

[6] M. Marañón, *Semiflujos discretos exteriores en la esfera de Riemann, algoritmos e implementaciones*, Publicaciones de la Universidad de La Rioja, 2012.

[7] M. McClure, *Newton's method for complex polynomials,* Mathematica in Education and Research 11(2) (2006).

[8] W. T. Shaw, *Complex Analysis with Mathematica,* Cambridge University Press, 2006.

[9] J. L. Varona, *Graphic and numerical comparison between iterative methods,* Math. Intelligencer 24(1) (2002), 37–46.