Plotting **B**asins of **U**nivariate **R**ational **F**unctions with **J**ulia

(User manual of the package PBURF.jl)

Luis Javier Hernández Paricio
University of La Rioja

October 29, 2019

Abstract

The goal of this manual is to present a new package "Plotting Basins of Univariate Rational Functions" (PBURF.jl) written in *Julia v1.1.x* 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.

The main advantage of the algorithm used by this package is that avoids the problem of overflows caused by denominators close to zero and the problem of indetermination which appears when simultaneously the numerator and denominator are equal to zero. This is solved by working with homogeneous coordinates and the iteration of a homogeneous pair on the augmented complex projective line (Riemann sphere plus an additional superzero point). Another good point of this package is that Julia language provides a system having simultaneously productivity and high performance.

This can be applied to any numerical method which constructs a rational map to solve an univariate polynomial equation and verifying that the roots of the equation are fixed points of the associated rational map.

1 Introduction

In this brief introduction some interesting properties of the Julia language and nice properties of the mathematical models used for develop algorithms are presented. In section 2 some instructions about the installation of the package are given. In section 3 we analyze the connection between the mathematical model and the induced logic implementation in Julia. Section 5 is devoted to explain some properties and characteristics of the main function of the package. In the last section, the program is applied to the study of the neighborhood of a critical value of the parameter of a uniparametric family of polynomials when the Newton method is used.

1.1 The advantages of Julia Language

The Julia programming language is announced to have the following nice properties: It has automatic translation of formulas into efficient executable code. It allows programmers to write clear, high-level, generic and abstract code that closely resembles mathematical formulas, as they have grown accustomed to in dynamic systems, yet produces fast, low-level machine code that has traditionally only been generated by static languages.

An essential part of the design philosophy of Julia is that all basic functionality must be possible to implement in Julia: integer arithmetic, for loops, recursion, floating-point operations, etc.

Julia's ability to combine these levels of performance and productivity in a single language stems from the choice of a number of features that work well with each other:

1. An expressive type system, allowing optional type annotations;
2. Multiple dispatch using these types to select implementations;

3. Metaprogramming for code generation;
4. A dataflow type inference algorithm allowing types of most expressions to be inferred;
5. Aggressive code specialization against run-time types;
6. Just-In-Time (JIT) compilation using the LLVM compiler framework; and
7. Julia's carefully written libraries that leverage the language design.

Although a sophisticated type system is made available to the programmer, it is never required to specify types, nor type annotations are necessary for performance. Type information flows naturally through the program due to dataflow type inference. For a more complete information we refer the reader to [2].

1.2 Mathematematical model for iteration without indefinite and indeterminate problems for rational functions

In the this manual, we present a collection of algorithms and implementations 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 permits us to work at the point at infinity;

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

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

(v) the mathematical model denominated complex projective line is improved with the addition of a new element: *the super-zero point*. This permits us to solve indetermination problems which appear when simultaneously the values of the numerator and denominator of a rational function are equal to zero. In these cases, the super-zero is taken as the image of the indetermination point.

2 Instalation of the package PBURF and usage of the main function:

`plottingBasinsUnivariateRationalFunctions`

2.1 Instalation of the package PBURF03

This software can be installed by giving the following command in the julia command line (or in some [IJulia] notebook):

```
Ijulia> Pkg.clone("https://github.com/luisjavierhernandez/PBURF.jl")
```

In this way, all dependencies will be satisfied automatically.

The code will be upgraded every time the `Pkg.update()` command is used.

Alternatively, you can manually copy the whole directory structure to your julia package directory (use `Pkg.dir()` to locate it), and then run `Pkg.update()` to download the dependencies.

You will need to have the Python [Matplotlib] library installed on your machine in order to use PyPlot. You can either do inline plotting with [IJulia] , which doesn't require a GUI backend, or use the Qt, wx, or GTK+ backends of Matplotlib.

2.1.1 Dependencies

The software PBURF03 requires the previous packages: `Polynomials.jl` (Keno Fischer and 12 Contributors), `Pyplot.jl` (Steven G. Johnson and 18 Contributors) and `Colors.jl` (29 Contributors).

These packages can be used typing:

[1]:
```
using Polynomials
using PyPlot
using Colors
```

2.2 Usage

To load the code just type:

```
Ijulia> using PBURF
```

The main function of package is

```
plottingBasinsUnivariateRationalFunctions(coefficientlistnum,coefficientlistden,...)
```

This function takes two lists of complex numbers that correspond to the coefficients of the numerator and denominator of a rational function and returns the list of fixed points of the function, a rectangular plot of the basins of this fixed points and a color palette with colors which correspond to the different fixed points.

The program `plottingBasinsUnivariateRationalFunctions` uses the following arguments:

`coefficientlistnum` : It is the list of the complex coefficients of the numerator of a univariate rational fucntion.

`coefficientlistden` : Is is the list of the complex coefficients of the denominator of a univariate rational function.

This function has the following optional arguments:

`expresolution` : It is a nonnegative integer. The function gives a plot such that the sides are divided into $2^{\texttt{expresolution}}$ subintervals. It default value is `expresolution`=8.

`iterationmax` : It is a nonnegative integer. This is the maximun of possible iterations of the rational function permitted when one runs this function. It default value is `iterationmax`=25.

`iterprecision` : It is a nonnegative integer. The stopping criterium is that chordal distance between two the last consecutive iterates is less than $1/10^{\texttt{iterationmax}}$. It default value is `iterationmax`=3.

`aproxprecision` : It is a nonnegative integer. When the chordal distance between the last iterate and a point of the list of fixed points is less than $1/10^{\texttt{aproxprecision}}$ the function assigns the starting point to the basin of the fixed point. It default value is `aproxprecision`=3.

This function also uses the following keyword arguments:

`colorstrategy`: It is an AbstractString. It possible values are `"positionplusiteration"`, `"iteration"`, and `"positionfixedpoints"`. The default value is `"colorstrategy="positionplusiteration"`. The program provides three different strategies for assigning colors to the different points of a plot.

`model` : It is an AbstractString. The default value is `model="localrectangle"`. Future development of this program will use other models like the Riemann sphere, a pair of disks, etc.

`rectanglesides` : It is a 4-dimensional tuple of real numbers (a,b,c,d) which corresponds to rectangle obtined as the product of the intervarls [a,b] and [c,d]. The defaut value is `rectanglesides=(-1.5,1.5,-1.5,1.5)`.

3 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.

3.1 Discrete metric semi-flows and basins

Let (X, d) be a metric space with metric d.

Given a continuous map $h\colon X \to X$, the pair (X, h) is called a *discrete semi-flow* and the triple (X, d, h) is said to be a metric discrete semi-flow.

Given an integer $n \geq 0$, h^n denotes the n-th composition $h \circ \cdots \circ h$ and $h^0 = id_X$.

Let $X = (X, h)$ be a discrete semi-flow:

A point $x \in X$ is said to be a *fixed point* if, for all $n \in \mathbb{N}$, $h^n(x) = x$, x is said to be a *periodic point* if there exists $n \in \mathbb{N}$, $n \neq 0$, such that $h^n(x) = x$ and x is said to be a *p-cyclic point* if $h^p(x) = x$ and $\{x, h(x), \cdots, h^p(x)\}$ has p elements.

Definition 1 *Given a metric discrete semi-flow $X = (X, d, h)$, the end space of X is defined as the quotient set*

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

(for $x, y \in X$, $(h^n(x)) \sim (h^n(y))$ if and only if $(d(h^n(x), h^n(y))) \xrightarrow[n\to+\infty]{} 0$.) An element $a = [(h^n(x))] \in \Pi(X)$ is called an end point of the metric discrete semi-flow X.

Note that we can define a natural map

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

by $\omega(x) = [(h^n(x))] = [(x, h(x), h^2(x), \dots)]$.

The map ω allows us to decompose any metric discrete semi-flow in the way shown below:

Definition 2 *Let X be a metric discrete semi-flow. The subset denoted by*

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

is called the basin of the end point a.

There is an induced partition of X given by

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

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

3.2 The augmented sphere and the augmented projective line (Normalized homogeneous coordinates and sphere bijections)

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 $\mathbb{C}^2$: $(z,t) \sim (z',t')$ 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 $[z\colon t]$ and the quotient set is denoted by $\mathbf{P}^{1+}(\mathbb{C})$ and it is called the ***augmented complex projective line***. Since $\mathbb{C}^2 \setminus \{(0,0)\}$ is a union of equivalence classes one has a canonical isomorphism:

$$\mathbf{P}^{1+}(\mathbb{C}) \cong \mathbf{P}^1(\mathbb{C}) \sqcup \{[0\colon 0]\}$$

where $\mathbf{P}^1(\mathbb{C})$ is the usual ***complex projective line***. We reduce the notation using

$$\bar{0} = [0\colon 0]$$

Given a point $[z\colon t] \in \mathbf{P}^1(\mathbb{C})$, 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:

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

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

From a computational point of view, If we are working in a computational environment with a prefixed precision 10^{-c}, $c \in \mathbb{N}$ and we have a point $[z\colon t] \in \mathbf{P}^1(\mathbb{C})$ whose homogenous coordinates (z,t) verifies that $|z| < 10^{-c}$ and $|t| < 10^{-c}$, then the computation systems takes (z,t) as $(0,0)$. The usual homogenous coordinates $[z : t]$ satisfy that z or t is not equal to zero, but in the case above the computational system takes $[z : t]$ as $[0 : 0]$. Therefore, under our point of view it is better to add a new super-zero point $\bar{0} = [0 : 0]$ in the mathematical model. To avoid with this delicate situation, our conclusion is that:

- It is better to work with the space $\mathbf{P}^{1+}(\mathbb{C}) = \mathbf{P}^1(\mathbb{C}) \cup \{[0 : 0]\}$.

The homogeneous coordinates have the property that a tuple and its normalization represents the same point. To avoid working with very small and very large tuples, we can normalize homogeneous coordinates with the function `homogeneousNormalization` implemented in Julia as follows

$$\texttt{homogeneousNormalization}\colon \mathbb{C}^2 \times \mathbb{N} \to (D^2 \times \{1\}) \cup (\{1\} \times D^2) \cup \{(0,0)\}$$

given by

$$\texttt{homogeneousNormalization}((z,t),c) = \begin{cases} (0,0) & \text{if } |z| < 10^{-c} \text{ and } |t| < 10^{-c}, \\ (z/t, 1) & \text{if } |t| \geq |z|, \\ (1, t/z) & \text{if } |t| < |z|, \end{cases}$$

A simple implementation in Julia can be given as follows:

[2]:
```
function homogeneousNormalization(
        twotuple::Tuple{Complex{Float64},Complex{Float64}},
        aproxprecision::Int64=15)
    tt1=complex(twotuple[1])
    tt2=complex(twotuple[2])
    if (abs(tt1)<1.0/10^aproxprecision && abs(tt2)<1.0/10^aproxprecision)
        hpoint=(complex(0.0),complex(0.0))
    else
        if abs(tt1)<= abs(tt2)
        hpoint=(tt1/tt2,complex(1.0))
        else
        hpoint=(complex(1.0),tt2/tt1)
        end
    end
    return hpoint
end
```

[2]: homogeneousNormalization (generic function with 2 methods)

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

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

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:

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

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.

Since we are working with the space $\mathbf{P}^{1+}(\mathbb{C})$ it is convenient to consider the ***augmented 2-sphere***

$$S^{2+} = S^2 \sqcup \{(0,0,0)\}$$

and the extended bijection $(\tilde{\theta}^{-1}\theta)^+([z,t]))\colon \mathbf{P}^{1+}(\mathbb{C}) \to S^{2+}$:

$$(\tilde{\theta}^{-1}\theta)^+([z,t])) = \begin{cases} (0,0,0) & \text{if } \bar{t}t + z\bar{z} = 0, \\ \left(\frac{\bar{z}t+z\bar{t}}{\bar{t}t+z\bar{z}}, \frac{i(\bar{z}t-z\bar{t})}{\bar{t}t+z\bar{z}}, \frac{-\bar{t}t+z\bar{z}}{\bar{t}t+z\bar{z}}\right) & \bar{t}t + z\bar{z} \neq 0. \end{cases}$$

Then we consider the map $\texttt{sphereBijection}\colon \mathbb{C}^2 \times \mathbb{N} \to S^{2+}$ given by

$$\texttt{sphereBijection}((z,t),c) = \begin{cases} (0,0,0) & \text{if } \bar{t}t + z\bar{z} < 10^{-c} \\ \left(\frac{\bar{z}t+z\bar{t}}{\bar{t}t+z\bar{z}}, \frac{i(\bar{z}t-z\bar{t})}{\bar{t}t+z\bar{z}}, \frac{-\bar{t}t+z\bar{z}}{\bar{t}t+z\bar{z}}\right) & \bar{t}t + z\bar{z} \geq 10^{-c}. \end{cases}$$

An implementation in Julia can be given as follows:

[4]:
```
function sphereBijection(twotuple::Tuple{Complex{Float64},Complex{Float64}},
                         aproxprecision::Int64=8)
    z=twotuple[1]
    t=twotuple[2]
    if (abs(z)<1.0/10^aproxprecision && abs(t)<1.0/10^aproxprecision)
    point=[0.0,0.0,0.0]
    else
    point=[real((conj(z)*t + conj(t)*z)/(conj(t)*t + conj(z)*z)),
           real((1im*(conj(z)*t - conj(t)*z))/(conj(t)*t + conj(z)*z)),
           real((-conj(t)*t + conj(z)*z)/(conj(t)*t + conj(z)*z))]
    end
    return point
end
```

[4]: sphereBijection (generic function with 2 methods)

[5]:
```
sphereBijection((1.0+0.0*im, 0.0+1.0*im),9)
```

[5]:
```
3-element Array{Float64,1}:
  0.0
 -1.0
  0.0
```

```
In [2]: sphereBijection((1.0+0.0*im, 0.0+1.0*im),9)

Out[2]: 3-element Array{Float64,1}:
          0.0
         -1.0
          0.0
```

Remark 1 *We notice that the homogeneous coordinates presented in this subsection allow us to represent the point at infinity and the indetermination-image point, and the use of normalized coordinates will avoid overflow and underflow errors and indeterminations in our computer programs.*

3.3 Metrics on $S^{2+} \cong \mathbf{P}^{1+}(\mathbb{C})$

We have a natural metric on $S^{2+} = S^2 \cup \{(0,0,0)\}$: since S^{2+} is a subspace of $\mathbb{R}^3$, the usual Euclidean metric of $\mathbb{R}^3$ induces a Euclidean ***chordal metric*** d on S^{2+}.

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:

$$d_1([z,t],[z',t']) = d((\tilde{\theta}^{-1}\theta)^+([z,t]),(\tilde{\theta}^{-1}\theta)^+([z',t']))$$

Then we consider the map $\texttt{chordalMetric} \colon \mathbb{C}^2 \times \mathbb{C}^2 \times \mathbb{N} \to S^{2+}$ given by

$$\texttt{chordalMetric}((z,t),(z',t'),c) = d(\texttt{sphereBijection}((z,t),c),\texttt{sphereBijection}((z',t'),c))$$

An implementation in Julia can be given as follows:

[6]:
```
function chordalMetric(twotuple::Tuple{Complex{Float64},Complex{Float64}},
                       twotuple1::Tuple{Complex{Float64},Complex{Float64}},
                       aproxprecision::Int64=8)
    norma=vector->sqrt(vector[1]^2+vector[2]^2+vector[3]^2)
    return norma((sphereBijection(twotuple,aproxprecision)-
    sphereBijection(twotuple1,aproxprecision)))
end
```

[6]:
```
chordalMetric (generic function with 2 methods)
```

[7]:
```
chordalMetric((1.0+0.0*im, 0.0+1.0*im),(0.0+0.0*im, 1.0+0.0*im),9)
```

[7]:
```
1.4142135623730951
```

3.4 Complex rational maps

Denote by $\mathbb{C}[z]$ the ring of polynomials with complex coefficients and by $\mathbb{C}(z)$ the field of complex rational functions. Any element $f \in \mathbb{C}(z)$ can be uniquely represented by $f(z) = \frac{F(z)}{G(z)}$ where G is a non zero polynomial and with F and G polynomials of lowest degree and G chosen to be monic. The degree d of f is given by $d = \max\{\text{degree}(F), \text{degree}(G)\}$ where the degree of the zero polynomials is taken to be $-\infty$. For a complex rational map $f(z) = \frac{F(z)}{G(z)}$, we can define $A, B \in K[z,t]$ by

$$A(z,t) = t^d F(z/t), B(z,t) = t^d G(z/t).$$

Taking account that $\sum_0^{+\infty} \mathbb{C} \cong \mathbb{C}[z]$, $(a_0, a_1, \cdots, a_n) \to a_0 + a_1 z + \cdots + a_n z^n$, as complex vectorial spaces, we consider the map:

$$\texttt{bivariatepolyfunction} \colon (\sum_0^{+\infty} \mathbb{C}) \times \mathbb{C} \times \mathbb{C} \underline{\times} \mathbb{N} \to \mathbb{C},$$

where

$$(\sum_0^{+\infty} \mathbb{C}) \times \mathbb{C} \times \mathbb{C} \underline{\times} \mathbb{N} = \{((a_0, a_1, \cdots, a_n), u, v, d) | n \leq d\},$$

it is defined by

$$\texttt{bivariatepolyfunction}((a_0, a_1, \cdots, a_n), u, v, d) = a_0 v^d + a_1 u v^{d-1} + \cdots + a_n v^{n-d}.$$

Definition 3 *Given a bivariate polinomial $F \in \mathbb{C}[z,t]$ it is said to be a* homogeneous bivariate polinomial *if $F = 0$ or for every $\lambda \in \mathbb{C} \setminus \{0\}$, there is $k \in \mathbb{N}$ such that for every $z, t \in \mathbb{C}$ we have that $F(\lambda z, \lambda t) = \lambda^k F(z,t)$. Denote by $\mathbb{C}_h[z,t]$ the subset of homogeneous polynomials. Given $F, G \in \mathbb{C}_h[z,t]$, the pair (F,G) is said to be a* homogeneous pair *if F, G are homogeneous bivarite polynomials satisfying that when F and G are non-zero polynomials, F, G have the same homogeneous degree $d \geq 0$. Denote by*

$$\mathbb{C}_h[z,t] \underline{\times} \mathbb{C}_h[z,t] = \{(F,G) \in C_h[z,t] \times \mathbb{C}_h[z,t] | (F,G) \text{ is a homogenous pair } \}$$

Given a homogeneous pair (F,G):

- *If F and G are non-zero polynomials, the* homogeneous degree *of* (F,G) *is given by* d,
- *If* $F=0, G\neq 0$ *and the degree of* G *is* d^G, *then we take* d^G *as* homogeneous degree *of* (F,G),
- *If* $F\neq 0, G=0$ *and the degree of* F *is* d^F, *then we take* d^F *as* homogeneous degree *of* (F,G) ,
- *If* $F=0, G=0$, *then we take* $-\infty$ *as* homogeneous degree *of* (F,G).

Notice that a bivaritate homogeneous polynominal $F\in\mathbb{C}_h[z,t]$ induces a homogeneous map $F\colon \mathbb{C}\times\mathbb{C}\to\mathbb{C}$ and we have canonical inclusions $\mathbb{C}_h[z,t]\to(\mathbb{C}^{\mathbb{C}\times\mathbb{C}})_h$, $\mathbb{C}_h[z,t]\underline{\times}\mathbb{C}_h[z,t]\to(\mathbb{C}^{\mathbb{C}\times\mathbb{C}})_h\underline{\times}(\mathbb{C}^{\mathbb{C}\times\mathbb{C}})_h$, where $(\mathbb{C}^{\mathbb{C}\times\mathbb{C}})_h$ denotes the set of homogeneous maps.

We also consider the map:

$$\texttt{pairoffunctions}\colon (\sum_0^{+\infty}\mathbb{C})\times(\sum_0^{+\infty}\mathbb{C})\to(\mathbb{C}^{\mathbb{C}\times\mathbb{C}})_h\underline{\times}(\mathbb{C}^{\mathbb{C}\times\mathbb{C}})_h,$$

$\texttt{pairoffunctions}((a_0,a_1,\cdots,a_n),(b_0,b_1,\cdots,b_m))(u,v) = (a_0v^d+a_1uv^{d-1}+\cdots+a_nv^{n-d}, b_0v^d+b_1uv^{d-1}+\cdots+b_mv^{m-d})$, where $d=\max\{n,m\}$.

These functions can be implemented as follows:

[8]:
```
function bivariatepolyfunction(coefficientlist::Array{Complex{Float64},1},
     u::Complex{Float64}, t::Complex{Float64},d::Int64)
    ff=coefficientlist[1]*t^d
    for i in 2:length(coefficientlist)
        ff=ff+coefficientlist[i]*u^(i-1)*t^(d-i+1)
        +i
    end
    return ff
end


function bivariatepolyfunction(coefficientlist::Array{T,1},
    u::T, t::T,d::Int64) where {T<:Number}
    ff=coefficientlist[1]*t^d
    for i in 2:length(coefficientlist)
        ff=ff+coefficientlist[i]*u^(i-1)*t^(d-i+1)
        +i
    end
    return complex(ff)
end


function bivariatepolyfunction(coefficientlist::Array{T,1},
         u::Complex{Float64},
        t::Complex{Float64},d::Int64) where {T<:Number}
    coefficientlistcomplex=complex(coefficientlist)
    ff=coefficientlistcomplex[1]*t^d
    for i in 2:length(coefficientlist)
        ff=ff+coefficientlist[i]*u^(i-1)*t^(d-i+1)
    +i
    end
    return ff
end



function pairoffunctions(coefficientlistnum::Array{T,1},
    coefficientlistden::Array{T,1}) where {T<:Number}
    ln=length(coefficientlistnum)-1
    ld=length(coefficientlistden)-1
    d=max(ln,ld)
    fff(u::Complex{Float64}, t::Complex{Float64})=
    bivariatepolyfunction(coefficientlistnum,u,t,d)
    ggg(u::Complex{Float64}, t::Complex{Float64})=
    bivariatepolyfunction(coefficientlistden,u,t,d)
    return fff, ggg
end
```

[8]:
```
pairoffunctions (generic function with 1 method)
```

[9]:
```
coefficientlistnumcompila=complex([1.,0.,0.,2.])
coefficientlistdencompila=complex([0.,0.,3.,0.])
```

[9]:
```
4-element Array{Complex{Float64},1}:
 0.0 + 0.0im
 0.0 + 0.0im
 3.0 + 0.0im
 0.0 + 0.0im
```

[10]:
```
bivarej=bivariatepolyfunction(coefficientlistnumcompila,
    1.0+im* 2.3, 2.0+im*0.9,8)
```

[10]:
```
718.0657168699995 - 1143.102377im
```

[11]:
```
hpaircompila=pairoffunctions(coefficientlistnumcompila,
    coefficientlistdencompila)
```

[11]:
```
(getfield(Main,
Symbol("#fff#5")){Array{Complex{Float64},1},Int64}(Complex{Float64}[1.0+0.0im,
0.0+0.0im, 0.0+0.0im, 2.0+0.0im], 3), getfield(Main,
Symbol("#ggg#6")){Array{Complex{Float64},1},Int64}(Complex{Float64}[0.0+0.0im,
0.0+0.0im, 3.0+0.0im, 0.0+0.0im], 3))
```

We note that each pair $(F,G) \in \mathbb{C}_h[z,t] \underline{\times} \mathbb{C}_h[z,t]$ we have an induced map $[F\colon G]\colon \mathbf{P}^{1+}(\mathbb{C}) \to \mathbf{P}^{1+}(\mathbb{C})$, given by $[F\colon G]([z\colon t]) = [F(z,t)\colon G(z,t)]$

When the homogenous coordinates $(F(z,t), G(z,t))$ are computed, we can find that some overflow or underflow problems when one iterates this process. This difficulty can be avoid taking normalized coordinates to the image point. Therefore we consider the map

$$\texttt{rationalFunction}\colon ((\mathbb{C}^{\mathbf{C}\times\mathbf{C}})_h \underline{\times} (\mathbb{C}^{\mathbf{C}\times\mathbf{C}})_h) \times \mathbb{C} \times \mathbb{C} \times \mathbb{N} \to (D^2 \times \{1\}) \cup (\{1\} \times D^2) \cup \{(0,0)\}$$

$$\texttt{rationalFunction}(F,G)(z,t,c) = \texttt{homogeneousNormalization}((F(z,t),(F,G)(z,t)),c)$$

We have implemented this map as follows:

[12]:
```
function rationalFunction(hpair::Tuple{Function,Function},
    twotuple::Tuple{Complex{Float64},Complex{Float64}},
    aproxprecision::Int64=8)
    c=twotuple[1]
    d=twotuple[2]
    F=hpair[1]
    G=hpair[2]
    cnew=F(c,d)
    dnew=G(c,d)
    hresult=homogeneousNormalization((cnew,dnew),aproxprecision)
    return  hresult
end
```

[12]:
```
rationalFunction (generic function with 2 methods)
```

[13]:
```
rationalFunction(hpaircompila,(1.0+0.0*im, 0.0+1.0*im),9)
```

[13]:
```
(-0.3333333333333333 - 0.6666666666666666im, 1.0 + 0.0im)
```

Note that $\mathbb{C}_h[z,t] \underline{\times} \mathbb{C}_h[z,t]$ is a graduate module over the graduate ring $\mathbb{C}_h[z,t]$. We can consider the following relations on

$$C_h[z,t] \underline{\times} \mathbb{C}_h[z,t] \setminus \{(0,0)\}, \quad \mathbb{C}_h[z,t] \underline{\times} \mathbb{C}_h[z,t].$$

Given two homogeneous pairs $(F,G), (F_1,G_1)$, one has that $(F,G) \sim' (F_1,G_1)$ if there is $H \in \mathbb{C}_h[z,t] \setminus \{0\}$ satisfying that $(F_1,G_1) = (FH,GH)$. Now consider the equivalence relation $\sim$ generated by $\sim'$. Denote by $\mathbf{P}^1(\mathbb{C}_h[z,t])$ and by $\mathbf{P}^{1+}(\mathbb{C}_h[z,t])$ the corresponding quotients. The equivalence class of a homogenous pair is denoted by $[F\colon G]$.

Note that $\mathbf{P}^{1+}(\mathbb{C}_h[z,t]) = \mathbf{P}^1(\mathbb{C}_h[z,t]) \sqcup \{[0\colon 0]\}$. We also have that $\mathbf{P}^1(\mathbb{C}_h[z,t])$ is the set of analytic self-maps of $\mathbf{P}^1(\mathbb{C})$. We also remark that $\mathbf{P}^{1+}(\mathbb{C}_h[z,t])$ has and additional element $[0\colon 0]$ which corresponds with the constant map associated to $\bar{0} \in \mathbf{P}^{1+}(\mathbb{C})$.

4 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 $F(z,t)$, $G(z,t)$ of the same degree (see subsection 3.4). We shall show in the next lines the algorithms which have been developed to study the basins induced by f.

4.1 Calculation of the fixed points of f

Given $f = [F\colon G]\colon \mathbf{P}^{1+}(\mathbb{C}) \to \mathbf{P}^{1+}(\mathbb{C})$. Then $[z_0, t_0]$ is a fixed point if and only if there is $\lambda \in \mathbb{C} \setminus \{0\}$ such that $F(z_0,t_0) = \lambda z_0$ and $G(z_0,t_0) = \lambda t_0$. In this subsection, we assume that $f = [F : G]$ is represented by a irreducible parir (F,G).

We have the following cases:

- If degree$(F,G) \leq 0$, then $F = a, G = b$ are constant polynomials. In this case the unique fixed point is $[a\colon b]$.
- If degree$(F,G) \geq 1$ and $tF(z,t) = zG(z,t)$, then $[F : G] = [z : t]$. This case corresponds to the identity and all the points of $\mathbf{P}^{1+}(\mathbb{C})$ are fixed points.
- If degree $(F,G) \geq 1$ and $tF(z,t) - zG(z,t)$ is a non identically zero polinomial. If either $F = 0$ or $G = 0$ we have in one of the anterior cases. Therefore we can suppose that degree(F) =degree(G)=degree$(F,G) \geq 1$.

If $F(z_0,t_0) = \lambda z_0$ and $G(z_0,t_0) = \lambda t_0$, then $t_0F(z_0,t_0) - z_0G(z_0,t_0) = 0$.

- If $t_0 = 0$, $F(0,0) = 0$ and $G(0,0) = 0$, then $[0 : 0]$ is a fixed point of f.
- If $t_0 = 0$, $F(1,0) \neq 0$ and $G(1,0) = 0$, then $[1 : 0]$ is a fixed point of f.
- If $t_0 \neq 0$, $t_0F(z_0,t_0) - z_0G(z_0,t_0) = 0$, and $G(z_0,t_0) = 0$ then $[z_0 : t_0]$ is an indetermination point of (F,G). This contradicts the fact that the pair (F,G) is irreducible.
- If $t_0 \neq 0$, $t_0F(z_0,t_0) - z_0G(z_0,t_0) = 0$, and $G(z_0,t_0) \neq 0$ then $[z_0 : t_0]$ is a fixed point of f.

A point $p \in \mathbf{P}^{1+}(\mathbb{C})$ is a fixed point module a given tolerance $c \in \mathbb{N}$, if $d(p, f(p)) < 10^{-c}$. Denote by $F(f,c)$ the set of fixed points of $f \neq \mathrm{id}$ (with a given complete order if it is finite and with some possible multiple elements).

Given a set X, we denote:

$$\mathrm{Finite1Arrays}(X) = \bigsqcup_{k=0}^{+\infty} (X)^k.$$

$$\mathrm{Finite2Arrays}(X) = \bigsqcup_{p=0,q=0}^{+\infty} ((X)^p)^q.$$

Then we consider the map

$$\texttt{fixedPointsofaIrreduciblePair}\colon (\mathbf{P}^{1+}(\mathbb{C}) \setminus \{\mathrm{id}\}) \times \mathbb{N} \to \mathrm{Finite1Arrays}(\mathbf{P}^{1+}(\mathbb{C}))$$
$$\texttt{fixedPointsofaIrreduciblePair}(f,c) = F(f,c)$$

Taking into account all the cases above we have the following implementation in Julia:

[14]:
```
function fixedPointsofaIrreduciblePair(
        coefficientlistnum::Union{Vector{Float64},Vector{Complex{Float64}}},
        coefficientlistden::Union{Vector{Float64},Vector{Complex{Float64}}},
        aproxprecision::Int64=8)
    coefficientlistnumcomplex=complex(coefficientlistnum)
    coefficientlistdencomplex=complex(coefficientlistden)
```

```
    if length(coefficientlistnum)!=length(coefficientlistden)
    println("The length of the first list
            is different of the length of the second")
    elseif  length(coefficientlistnum)==1 && length(coefficientlistden)==1
    finallistoffixedpoint=homogeneousNormalization((coefficientlistnumcomplex[1],
    coefficientlistdencomplex[1]),
    aproxprecision)
    return [finallistoffixedpoint]
    elseif abs(coefficientlistnum[1])==0.0 &&
        coefficientlistden[length(coefficientlistden)]==0.0 &&
        sum(i->abs(coefficientlistnum[i]-coefficientlistden[i-1]),
        2:length(coefficientlistden))==0.0
    println("All the points are fixed points")
    else
    ffgg=pairoffunctions(coefficientlistnumcomplex, coefficientlistdencomplex)
    Polynumerator=Poly(coefficientlistnumcomplex)
    Polydenominator=Poly(coefficientlistdencomplex)
    Polyx=Poly([0.0+ 0.0* im,1.0+0.0*im])
    lookingzeros= Polynumerator- Polydenominator * Polyx
    fix=roots(lookingzeros)
    le=length(fix)
        fixed=[(complex(fix[i]),1.0+0.0*im) for i in 1:le]
        if abs(ffgg[1](complex(0.0),complex(0.0)))==0.0 &&
            abs(ffgg[2](complex(0.0),complex(0.0)))==0.0
       newfixed=[(complex(0.0),complex(0.0))]
        else
       newfixed=[]
        end
        if abs(ffgg[1](complex(1.0),complex(0.0)))!=0.0 &&
            abs(ffgg[2](complex(1.0),complex(0.0)))==0.0
       morenewfixed=[(complex(1.0),complex(0.0))]
        else
       morenewfixed=[]
        end
        return newfixed, morenewfixed, fixed
    end
end
```

[14]:
```
fixedPointsofaIrreduciblePair (generic function with 2 methods)
```

[15]:
```
fixedPointsofaIrreduciblePairex=
    fixedPointsofaIrreduciblePair(coefficientlistnumcompila,
    coefficientlistdencompila,9)[3]
```

[15]:
```
3-element Array{Tuple{Complex{Float64},Complex{Float64}},1}:
 (-0.5000000000000001 + 0.8660254037844439im, 1.0 + 0.0im)
 (-0.5000000000000001 - 0.8660254037844439im, 1.0 + 0.0im)
 (1.0 + 0.0im, 1.0 + 0.0im)
```

In this implementation we have used the package `Polynomials` developed by Keno Fischer from a previous version of Jameson Nash. Polynomials has the function `roots` that constructs the "companion matrix" of a polynomial p and computes the eigenvalues of $C(p)$ which are the roots of the polynomial p.

The "companion matrix" of the monic polynomial

$$p(t) = c_0 + c_1 t + \cdots + c_{n-1} t^{n-1} + t^n \,,$$

is the square matrix defined as

$$C(p) = \begin{bmatrix} 0 & 0 & \dots & 0 & -c_0 \\ 1 & 0 & \dots & 0 & -c_1 \\ 0 & 1 & \dots & 0 & -c_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & -c_{n-1} \end{bmatrix}.$$

4.2 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

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

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 $k < l$.

We consider the following algorithm checking the logical condition LC

LC: While the chordal distance from $f^k(x)$ to $f^{k+1}(x)$ greater than 10^{-c} and $k < l$, then a) is applied; otherwise, b) is obtained.

a) a new iteration is done and the logical condition LC is verified again for the new point $f(f^k(x)$ and the new integer $k+1$.

b) the output $[f^{k+1}(x), k]$ is taken.

$\texttt{newstep}\colon \mathbf{P}^1(\mathbb{C}_h[z,t]) \times \mathbb{N} \times \mathbb{N} \times \times \mathbb{C} \to (D^2 \times \{1\}) \cup (\{1\} \times D^2) \cup \{(0,0)\} \times \mathbb{N}$

$$\texttt{newstep}(f,l,c,x) = \begin{cases} (f^{l+1}(x), l) & \text{if } d(f^k(x), f^{k+1}(x) > 10^{-c}, 0 \le k < l, \text{ or } k = l, \\ (f^{k+1}(x), k) & k = \min\{s, d(f^s(x), f^{s+1}(x) < 10^{-c} \text{ and } s < l\}. \end{cases}$$

The following implementation, `newstep`, was developed in *Julia*. The source code of the function is:

[16]:
```
function newstep(hpair::Tuple{Function,Function},
                 iter::Int64,
                 iterprecision::Int64,
                 hpoint::Tuple{Complex{Float64},Complex{Float64}})
    point = hpoint
    tol=1.0/10^(iterprecision)
    number = 0
    imagepoint = rationalFunction(hpair,point,iterprecision)
    while
    (chordalMetric(point, imagepoint,iterprecision) > tol)&& (number < iter)
    point = imagepoint
    imagepoint = rationalFunction(hpair,point,iterprecision)
    number=number+1
    end
    return [imagepoint,number]
end
```

[16]:
```
newstep (generic function with 1 method)
```

[17]:
```
newstep(hpaircompila, 11, 3, (1.0+0.0*im, 0.0+1.0*im))
```

[17]:
```
2-element Array{Any,1}:
  (-0.4999999962890296 - 0.8660253983385868im, 1.0 + 0.0im)
 4
```

$\texttt{rectangle}\colon (\mathbb{R} \times \mathbb{R}) \times (\mathbb{R} \times \mathbb{R}) \times \mathbb{N} \to \text{Finite2Arrays}(\mathbf{P}^1(\mathbb{C}))$

$$\texttt{rectangle}((a,b),(c,d),k) = \left([(a + \frac{i}{2^k}) + (c + \frac{j}{2^k})im : 1] \right)$$

where im is the complex such that $im^2 = -1$ and i, j are non negative integers such that $\frac{i}{2^k} < |b-a|$ and $\frac{j}{2^k} < |d-c|$

When we are working with a 2-array of points and `newstep` is applied we will obtain a 2-array in the output.

[18]:
```
function rectangle(xinterval::Tuple{Float64,Float64}=(-1.5,1.5),
                   yinterval::Tuple{Float64,Float64}=(-1.5,1.5),
                   expprecision=10)
    tol=1.0/(2^expprecision)
    a=xinterval[1]
    b=xinterval[2]
    c=yinterval[1]
    d=yinterval[2]
    red=[(complex(r,i),complex(1.0)) for i=d:-tol:c, r=a:tol:b]
    return red
end


function newstep(hpair::Tuple{Function,Function}, iter::Int64,
        iterprecision::Int64,
        hrectangle::Array{Tuple{Complex{Float64},Complex{Float64}},2})
    size1=size(hrectangle)
    result=[newstep(hpair, iter, iterprecision, hrectangle[i,j])
```

```
        for i=1:size1[1], j=1:size1[2]]
    return result
end
```

[18]: newstep (generic function with 2 methods)

[19]:
```
#Example
rectanglecompila=rectangle((0.0,1.0),(0.0,1.0),1)
newstep(hpaircompila, 11, 3, rectanglecompila)
```

[19]:
```
3×3 Array{Array{Any,1},2}:
 [(-0.5+0.866025im, 1.0+0.0im), 4]  ...  [(1.0+0.0im, 1.0+2.55681e-8im), 6]
 [(1.0+0.0im, -0.5-0.866025im), 8]       [(1.0+9.96594e-11im, 1.0+0.0im), 4]
 [(1.0+0.0im, 0.0+0.0im), 1]             [(1.0+0.0im, 1.0+0.0im), 0]
```

4.3 Computing position-interation arrays of a list of points with respect to a list of fixed points

The subroutine `positionuptotolerance` 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, up to a given tolerance, to any fixed point of the list, it returns 0.

We have the map:

$$\texttt{positionuptotolerance}\colon \text{Finite1Arrays}(\mathbf{P}^{1+}(\mathbb{C}) \times \mathbb{N} \times \mathbf{P}^{1+}(\mathbb{C}) \to \mathbb{N}$$

$$\texttt{positionuptotolerance}((p_1, \cdots, p_k), c, p) = \begin{cases} \min\{r | d(p_r, p) < 10^{-c}\}, & \text{if } \{r | d(p_r, p) < 10^{-c}\} \neq \emptyset, \\ 0, \text{ otherwise.} \end{cases}$$

An implementation of this subroutine is shown in the next lines:

[20]:
```
function positionuptotolerance(fixedPointList::Array{Tuple{Complex{Float64},
                            Complex{Float64}},1},
                            aproxprecision::Int64,
                            twotuple::Tuple{Complex{Float64},Complex{Float64}})
    pos=0
    it=1
    le = length(fixedPointList)
    tol=1/10^(aproxprecision)
    while (it < le+1)
        if (chordalMetric(twotuple, fixedPointList[it],aproxprecision) < tol)
            pos = it
        end
    it=it+1
    end
    return convert(Int64,pos)
end
```

[20]: positionuptotolerance (generic function with 1 method)

[21]:
```
positionuptotolerance(fixedPointsofaIrreduciblePairex,3,(-0.5 - 0.28867513459481275im,1.0 + 0.0im) )
```

[21]: 0

For a given point the following subroutine obtains a pair of non negative integers (position, iteration), where position determines the fixed point and iteration is the number of iterations which are necessary to approaches to a fixed point up to a given tolerance.

$$\texttt{position\_iteration\_upto\_tolerances}\colon \mathbf{P}^1(\mathbb{C}_h[z,t]) \times \text{Finite1Arrays}(\mathbf{P}^{1+}(\mathbb{C})) \times \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbf{P}^{1+}(\mathbb{C}) \to \mathbb{N} \times \mathbb{N}$$

$$\texttt{position\_iteration\_upto\_tolerances}(f, (p_1, \cdots, p_k), l, c_1, c_2, p) =$$
$$(\texttt{positionuptotolerance}((p_1, \cdots, p_k), c_2, \texttt{newstep}(f, l, c_1, p)[1]), \texttt{newstep}(f, l, c_1, p)[2])$$

This functions has been implemented as follows:

[22]:
```
function position_iteration_upto_tolerances(hpair::Tuple{Function,Function},
        fixedPointList::Array{Tuple{Complex{Float64},Complex{Float64}},1},
        iter::Int64, iterprecision::Int64,
        aproxprecision::Int64,
```

```
        twotuple::Tuple{Complex{Float64},Complex{Float64}})
    endpoint_iterations=newstep(hpair, iter, iterprecision, twotuple)
    endpoint=endpoint_iterations[1]
    iterations=convert(Int64,endpoint_iterations[2])
    pos=positionuptotolerance(fixedPointList, aproxprecision, endpoint)
    return (pos,   iterations)
end
```

[22]:
```
position_iteration_upto_tolerances (generic function with 1 method)
```

[23]:
```
position_iteration_upto_tolerances(hpaircompila,fixedPointsofaIrreduciblePairex, 11, 3, 2, (1.0+0.0*im, 0.0+1.0*im))
```

[23]:
```
(2, 4)
```

For a given rectangle and resolution, we have a 2-dimensional grid whose vertexes are uniformly distributed. Computing the pair (position, iteration) for each vertex of the grid, one has a 2-dimensional array of 2-tuples of non negative integers

Then, we have the map:

`position_iteration_upto_tolerances` from
$\mathbf{P}^1(\mathbb{C}_h[z,t]) \times \text{Finite1Arrays}(\mathbf{P}^{1+}(\mathbb{C})) \times \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \text{Finite2Arrays}(\mathbf{P}^{1+}(\mathbb{C}))$
to
$\text{Finite2Arrays}(\mathbb{N} \times \mathbb{N})$
$\texttt{position\_iteration\_upto\_tolerances}(f,(p_1,\cdots,p_k),l,c_1,c_2,(p_{ij})) =$
$(\texttt{position\_iteration\_upto\_tolerances}(f,(p_1,\cdots,p_k),l,c_1,c_2,p_{ij}))$

[24]:
```
function position_iteration_upto_tolerances(hpair::Tuple{Function,Function},
        fixedPointList::Array{Tuple{Complex{Float64},Complex{Float64}},1},
        iter::Int64, iterprecision::Int64, aproxprecision::Int64,
        hrectangle::Array{Tuple{Complex{Float64},Complex{Float64}},2})
    size1=size(hrectangle)
    result=[position_iteration_upto_tolerances(hpair,fixedPointList, iter,
        iterprecision, aproxprecision,
        hrectangle[i,j]) for i=1:size1[1], j=1:size1[2]]
    return result
end
```

[24]:
```
position_iteration_upto_tolerances (generic function with 2 methods)
```

[25]:
```
position_iteration_upto_tolerances(hpaircompila,fixedPointsofaIrreduciblePairex,11, 3, 2, rectanglecompila)
```

[25]:
```
3×3 Array{Tuple{Int64,Int64},2}:
 (1, 4)  (2, 6)  (3, 6)
 (1, 8)  (1, 7)  (3, 4)
 (0, 1)  (3, 5)  (3, 0)
```

For a given rational map (that is, a pair of homogenous polynomias) the following functions compute the ordinary fixed points of the map, it checks if the super-zero and the infinity point are fixed points and it completes the list of ordinary fixed points. Then given an array of points in a rectangle construct the matrix of (position, tolerance) non negative integer of the 2-array of points with respect the complete lists of fixed points.

`fixedPointListex_matrixpositioniterations_RationalFucntion` from
$\mathbf{P}^1(\mathbb{C}_h[z,t]) \times \times \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{R} \times \mathbb{R} \times \mathbb{R} \times \mathbb{N}$
to
$\text{Finite1Arrays}(\mathbf{P}^{1+}(\mathbb{C}) \times \text{Finite2Arrays}(\mathbb{N} \times \mathbb{N})$
$\texttt{fixedPointListex\_matrixpositioniterations\_RationalFucntion}(f,(p_1,\cdots,p_k),r,l,c_1,c_2,(a,b,c,d)) =$
$(\texttt{fixedPointsofaIrreduciblePair}(f,c_2),$
$\texttt{position\_iteration\_upto\_tolerances}(f,\texttt{fixedPointsofaIrreduciblePair}(f,c_2),l,c_1,c_2,\texttt{rectangle}((a,b),(c,d),r)))$

[26]:
```
function fixedPointListex_matrixpositioninterations_RationalFunction(
        coefficientlistnum::Array{T,1},
        coefficientlistden::Array{T,1},
        preexpresolution::Int=8,
        preiteration_max::Int=25,
        preiterprecision::Int64=3,
        preaproxprecision::Int64=3,
        premodeldomain::AbstractString="localrectangle",
        prerectanglesidesdomain::Tuple{Float64,Float64,Float64,Float64}=
        (-1.5,1.5,-1.5,1.5)) where {T<:Number}
```

```
    xinterv=(prerectanglesidesdomain[1],prerectanglesidesdomain[2])
    yinterv=(prerectanglesidesdomain[3],prerectanglesidesdomain[4])
    rect=rectangle(xinterv,  yinterv, preexpresolution)
        hpair=pairoffunctions(coefficientlistnum, coefficientlistden)
        coefficientlistnumcomplex=complex(coefficientlistnum)
        coefficientlistdencomplex=complex(coefficientlistden)
    fixedPointListexcomplete=
        fixedPointsofaIrreduciblePair(coefficientlistnumcomplex,
        coefficientlistdencomplex,
        preaproxprecision)
    fixedPointListexcompletea=fixedPointListexcomplete[1]
    if length(fixedPointListexcomplete[2])==0
    fixedPointListexcompleteb=fixedPointListexcompletea
    else
    fixedPointListexcompleteb=union(fixedPointListexcompletea,
        [(complex(1.0),complex(0.0))])
    end
    fixedPointListexample=
    union(fixedPointListexcompleteb,fixedPointListexcomplete[3])
    positer=position_iteration_upto_tolerances(hpair,fixedPointListexample,
    preiteration_max, preiterprecision,
    preaproxprecision, rect)
    return   fixedPointListexample, positer, fixedPointListexcomplete
end
```

[26]:
```
fixedPointListex_matrixpositioninterations_RationalFunction (generic function
with 7 methods)
```

[27]:
```
fmf=fixedPointListex_matrixpositioninterations_RationalFunction(
    coefficientlistnumcompila,
    coefficientlistdencompila,3)
```

[27]:
```
(Tuple{Complex{Float64},Complex{Float64}}[(0.0+0.0im, 0.0+0.0im), (1.0+0.0im,
0.0+0.0im), (-0.5+0.866025im, 1.0+0.0im), (-0.5-0.866025im, 1.0+0.0im),
(1.0+0.0im, 1.0+0.0im)], Tuple{Int64,Int64}[(3, 4) (3, 4) ... (5, 7) (5, 7); (3,
4) (3, 4) ... (5, 7) (5, 6); ... ; (4, 4) (4, 4) ... (5, 7) (5, 6); (4, 4) (4, 4) ...
(5, 7) (5, 7)], (Tuple{Complex{Float64},Complex{Float64}}[(0.0+0.0im,
0.0+0.0im)], Tuple{Complex{Float64},Complex{Float64}}[(1.0+0.0im, 0.0+0.0im)],
Tuple{Complex{Float64},Complex{Float64}}[(-0.5+0.866025im, 1.0+0.0im),
(-0.5-0.866025im, 1.0+0.0im), (1.0+0.0im, 1.0+0.0im)]))
```

4.4 Plotting position-interation arrays

The function `plot_matrixpositer_RationalFunction` transforms a 2-dimensional array of pairs of nonnegative integers into a plot that Ijulia visuliazes in the working notebook. This function has a keyword argument: `thecolorstrategy` that has three possible values: `"positionfixedpoints"`, `"iteration"`, `"positionplusiteration"`.

Taking `thecolorstrategy="positionfixedpoints"`, the function uses the first element of the pair (position, iteration) to assign a color of each value of position, the value iteration is ignored. When `thecolorstrategy="iteration"`, the function uses only the second element of the pair (position, iteration) to assign a color. Finally taking `thecolorstrategy="positionplusiteration"` for each value of position a color is chosen and the different values of iteration change the "intensity" of this color.

In the code of `plot_matrixpositer_RationalFunction` the function `distinguishable_colors` of the package `"Colors"` generates n maximally distinguishable colors in LCHab space. This last function can take a seed of prefixed colors. The black color is reserved for the value position=0. In the final plot obtained by the main function this means that a point will have black color either it is not in the basin of a fixed point (for instance, it can be situated in the basin of an attraction 2-cycle) or the length iteration sequence of the point has reached the maximum number of permitted iterations and the last iterate is not close to any fixed point upto a prefixed tolerance. Distinguishability is maximized with respect to the CIEDE2000 color difference formula. We refer the reader to the information of the package `Colors` and the paper [12]. The grey color is also reserved to the basin of the superzero point. This means that in the last step of the iteration sequence of a starting point an indetermination has been obtained (some numerator and denominator are simultaneously equal to zero). The yelow color is also reserved for the case that the infinity point is a fixed points, otherwise we only work with two reserved colors (black and grey).

The image displayed in the screen if obtained with the function `PyPlot.imshow` of the package `PyPlot`. Note that a colormap is obtained by the function `PyPlot.ColorMap` of the package `PyPlot` using the palette of colors constructed by `distinguishable_colors`.

[28]:
```
function plot_matrixpositer_RationalFunction(
    fixedpointsmatrixpositer,iteration_max::Int=25,
    thecolorstrategy::AbstractString="positionplusiteration",
    themodel::AbstractString="localrectangle",
    therectanglesides::Tuple{Float64,Float64,Float64,Float64}=
    (-1.5,1.5,-1.5,1.5))
    numberoffixedPointListex=length(fixedpointsmatrixpositer[1])
    positer=fixedpointsmatrixpositer[2]
    ab=size(positer)
        if thecolorstrategy=="positionplusiteration"
        numberofcolors=numberoffixedPointListex+1
        integermatrix=
        [(positer[i,j][1]+(1.0-(positer[i,j][2])/iteration_max))+1.0  for i=1:ab[1], j=1:ab[2]];
        integermatrix[1,1]=1;integermatrix[ab[1],ab[2]]=numberofcolors-2
        elseif thecolorstrategy=="iteration"
        numberofcolors=iteration_max+1
        integermatrix=[positer[i,j][2]+0.25 for i=1:size(positer)[1],
            j=1:size(positer)[2]];integermatrix[1,1]=iteration_max
        else thecolorstrategy=="positionfixedpoints"
        numberofcolors=numberoffixedPointListex+1
        integermatrix=[positer[i,j][1]+0.25 for i=1:size(positer)[1],
            j=1:size(positer)[2]]; integermatrix[1,1]=0;integermatrix[ab[1],
            ab[2]]=numberoffixedPointListex
        end
    isinfinity=length(fixedpointsmatrixpositer[3][2])
    if  isinfinity==0
        seed1=[RGB(0.0,0.0,0.0),RGB(0.5,0.5,0.5)]
    else
        seed1=[RGB(0.0,0.0,0.0),RGB(0.5,0.5,0.5),RGB(1.0,1.0,0.0)]
    end
    seed=union(seed1,[RGB(1.0,0.0,0.0),RGB(0.0,1.0,0.0),RGB(0.0,0.0,1.0)])
    gseed=distinguishable_colors(numberofcolors, seed)
    fpcm=PyPlot.ColorMap(gseed)
    img=PyPlot.imshow(integermatrix, cmap=fpcm, extent=[therectanglesides[1],
        therectanglesides[2],
        therectanglesides[3],therectanglesides[4]])
    return  fixedpointsmatrixpositer[3],gseed, img
end
```

[28]:
```
plot_matrixpositer_RationalFunction (generic function with 5 methods)
```

5 The main function for plotting basins of a univariate rational map

In this section we describe the main function of the package PBURF.

`plottingBasinsUnivariateRationalFunctions`

This function uses all the subroutines defined in section 3. At the present version, given the vertexes of grid contained in a given rectangle, an 2-array of pairs of nonnegative numbers is constructed. Given a vertex of the grid, an iteration sequence is constructed where the stopping criterium is stablished by when the chordal distance between the last two iterations is less that a prefixed tolerance or when the maximum number of permitted iterations has been reached. This also gives the second integer of the pair an integer between 0 and the iter=the maximum number of iterations. Since one has constructed the list of fixed points of the rational function, each fixed point has a position in this ordered list. The first integer of the pair is either 0 is the last iterate is not close to any fixed point of the list or it is the position of the the first fixed point which is closer, upto a prefixed tolerance, to the last iterate point of the iteration sequence.

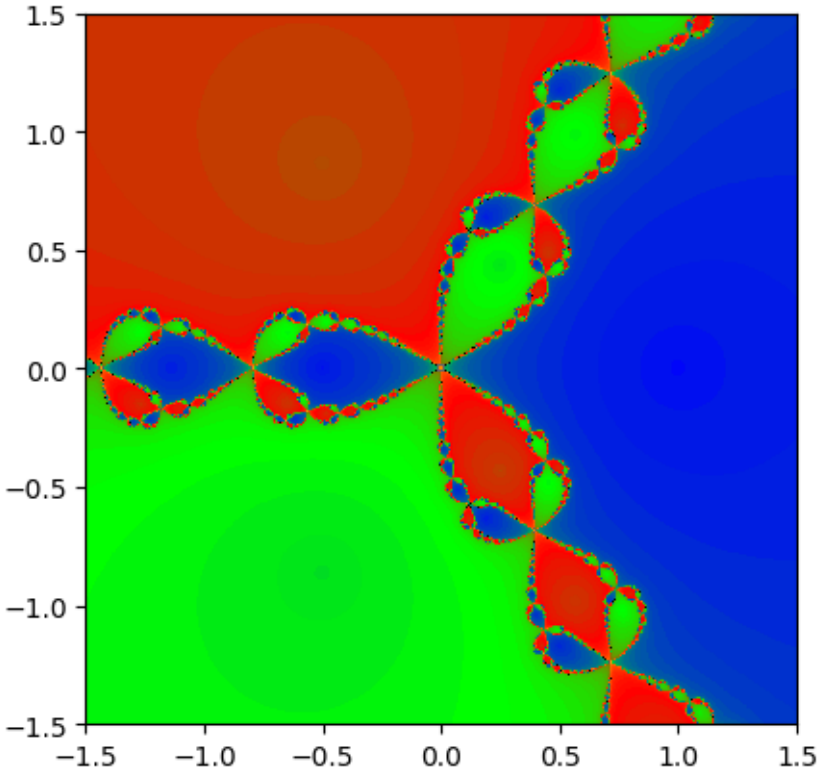
Now this main function apply the function above `plot_matrixpositer_RationalFunction` to transform this 2-array of pairs of non negative integer into a rectangular plot. The program assigns a color to this starting point using three possible strategies (`"positionfixedpoints"`, `"iteration"`, `"positionplusiteration"`) which use the first number of the pair, the second or a rational number constructed with the the two integers, respectively.

[29]:
```
function plottingBasinsUnivariateRationalFunctions(
    coefficientlistnum::Array{T,1},
    coefficientlistden::Array{T,1},
    expresolution::Int=8,
    iterationmax::Int=25,
    iterprecision::Int64=3,
    aproxprecision::Int64=3;
    colorstrategy::AbstractString="positionplusiteration",
    model::AbstractString="localrectangle",
    rectanglesides::Tuple{Float64,Float64,Float64,Float64}=
```

```
        (-1.5,1.5,-1.5,1.5)) where {T<:Number}
    maxdegree=max(length(coefficientlistnum),length(coefficientlistden))
    samedegreecoefficientlistnum=zeros(Complex{Float64}, maxdegree)
    for i in 1:length(coefficientlistnum)
        samedegreecoefficientlistnum[i]=coefficientlistnum[i]
    end
    samedegreecoefficientlistden=zeros(Complex{Float64}, maxdegree)
    for i in 1:length(coefficientlistden)
        samedegreecoefficientlistden[i]=coefficientlistden[i]
    end
    preexpresolution=expresolution
    preiterationmax=iterationmax
    preiterprecision=iterprecision
    preaproxprecision=aproxprecision
    premodeldomain=model
    prerectanglesidesdomain=rectanglesides
    fixedpointsmatrixpairpositerex=
    fixedPointListex_matrixpositioninterations_RationalFunction(
    samedegreecoefficientlistnum,
    samedegreecoefficientlistden, preexpresolution,preiterationmax,
    preiterprecision,preaproxprecision,
     premodeldomain,  prerectanglesidesdomain)
    fixedpointsmatrixpairpositer3seedimg=
    plot_matrixpositer_RationalFunction(fixedpointsmatrixpairpositerex,
                                        iterationmax,
                                        colorstrategy,
                                        model,
                                        rectanglesides)
    return  fixedpointsmatrixpairpositer3seedimg
end
```

[29]: plottingBasinsUnivariateRationalFunctions (generic function with 5 methods)

[30]:
```
result=plottingBasinsUnivariateRationalFunctions(coefficientlistnumcompila,
    coefficientlistdencompila)
```



[30]:
```
((Tuple{Complex{Float64},Complex{Float64}}[(0.0+0.0im, 0.0+0.0im)],
Tuple{Complex{Float64},Complex{Float64}}[(1.0+0.0im, 0.0+0.0im)],
Tuple{Complex{Float64},Complex{Float64}}[(-0.5+0.866025im, 1.0+0.0im),
```

```
(-0.5-0.866025im, 1.0+0.0im), (1.0+0.0im, 1.0+0.0im)]),
RGB{Float64}[RGB{Float64}(0.0,0.0,0.0), RGB{Float64}(0.5,0.5,0.5),
RGB{Float64}(1.0,1.0,0.0), RGB{Float64}(1.0,0.0,0.0), RGB{Float64}(0.0,1.0,0.0),
RGB{Float64}(0.0,0.0,1.0)], PyObject <matplotlib.image.AxesImage object at
0x13b7bc978>)
```

The output of the main function gives a plot with the basins of the fixed points (when the keyword argument strategy is different to the value `iteration`). Moreover, it contains more useful information: The list of fixed points is divided into three lists, the fix list is empty if the rational function has degree equal to zero, otherwise it contains the super-zero fixed point. The second list is empty if the infinity point is not a fixed point, otherwise this list contains the infinity point. The third list contains the rest of fixed points of the rational function. There is an important exception, the identity rational map $[z : t]$ which has the property that every point of the augmented projective line is a fixed point. The output also contains the color palette associated to the list of fixed points (this can be obtained if you type in Ijulia, `ans[2]`, afther run an instance of the main function.

In the example above, we have three reserved colors: black (no convergency), grey (convergency to the super-zero) and yellow (convergence to the infinity point). The red, green and blue colors are used for the rest of fixed points.

It is important to observe that in many cases the measure of a basin of a repelling or indifferent fixed point is zero and the corresponding basin is not visible in the obtained plot.

References

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

[2] Jeff Bezanson, Alan Edelman,Stefan Karpinski and Viral B. Shah, Julia: A Fresh Approach to Numerical Computing, CoRR, abs/1411.1607,(2014), http://arxiv.org/abs/1411.1607, http://dblp.uni-trier.de/rec/bib/journals/corr/BezansonEKS14.

[3] J. M. Gutiérrez, L. J. Hernández-Paricio, M. Marañón-Grandes and M. T. Rivas-Rodríguez, Influence of the multiplicity of the roots on the basins of attraction of Newton's method, *Numer. Algorithms*, vol. 66 (3) (2014), 431–455.

[4] L. J. Hernández, Bivariate Newton-Raphson method and toroidal attraction basins, DOI: 10.1007/s11075-015-9996-3t, 2015.

[5] L. J. Hernández, M. Marañón and M. T. Rivas, Plotting basins of end points of rational maps with Sage, *Tbilisi Math. J.*, vol. 5 (2) (2012), 71–99 .

[6] Ihaka, R. (2003). Colour for Presentation Graphics. In K Hornik, F Leisch, A Zeileis (eds.), Proceedings of the 3rd International Workshop on Distributed Statistical Computing, Vienna, Austria. ISSN 1609-395X

[7] Zeileis, A., Hornik, K., and Murrell, P. (2009). Escaping RGBland: Selecting colors for statistical graphics. Computational Statistics and Data Analysis, 53(9), 3259?3270. doi:10.1016/j.csda.2008.11.033

[8] Schanda, J., ed. Colorimetry: Understanding the CIE system. Wiley-Interscience, 2007.

[9] Sharma, G., Wu, W., and Dalal, E. N. (2005). The CIEDE2000 color?difference formula: Implementation notes, supplementary test data, and mathematical observations. Color Research & Application, 30(1), 21?30. doi:10.1002/col

[10] Ihaka, R., Murrel, P., Hornik, K., Fisher, J. C., and Zeileis, A. (2013). colorspace: Color Space Manipulation. R package version 1.2-1.

[11] Lindbloom, B. (2013). Useful Color Equations

[12] Wijffelaars, M., Vliegen, R., van Wijk, J., van der Linden, E-J. (2008). Generating Color Palettes using Intuitive Parameters Georg A. Klein Industrial Color Physics. Springer Series in Optical Sciences, 2010. ISSN 0342-4111, ISBN 978-1-4419-1197-1.