The connected locus for complex cubic iteration

A preprint version of a “Mathematical graphics” column from Mathematica in Education and Research.

Mark McClure

Department of Mathematics
University of North Carolina at Asheville
Asheville, NC 28804

mcmcclure@unca.edu

Abstract

Iteration of quadratic polynomials in complex dynamics leads to the Mandelbrot set, one of the most beautiful and famous images in modern mathematics. There is a generalization of this object to higher degree polynomials. In particular, the connected locus for cubic iteration is a four dimensional set analogous to the Mandelbrot set for quadratic iteration.

Note: To reduce the size of the file, the graphics in this file have all been converted to bitmap form. Of course, they may be regenerated within *Mathematica*.

■ **Initialization**

1. Introduction

This is the first column in a regular series exploring intriguing mathematical graphics. One of the most well known images in all of mathematics is the Mandelbrot set, which arises in the context of the complex dynamics of quadratic polynomials. In this issue's column, we'll explore the cubic connected locus, a four dimensional analog of the Mandelbrot set which arises in the complex dynamics of cubic polynomials.

The major focus of this column will be to understand the mathematics behind the graphics. For example, the Mandelbrot set and the cubic connected locus may technically be defined in just a few lines. However, these definitions make little sense outside of the context of complex dynamics. Thus,we will attempt to describe just enough dynamics to put the material in context.

This column describes the *Mathematica* code to generate images of the cubic connected locus. This code has also been encapsulated in a package (tested with *Mathematica* 4.2 and later) which calls a Java implementation to run quite a bit faster. Note that the package should be contained in a directory specified by *Mathematica*'s **`$Path`** variable and the Java class files should be contained in a directory specified by **`JavaClassPath[]`**. One possibility is to simply leave them all in the SupplementaryFiles directory, which should come with this notebook. The initialization cell at the beginning of this notebook contains a line to add that directory to the appropriate paths.

2. Complex dynamics and Julia sets

In complex dynamics, we study the iteration of a function $f : \mathbb{C} \to \mathbb{C}$. That is, given f and an initial input z_0, we generate a sequence $\{z_0, z_1, z_2, \ldots\}$, where $z_n = f(z_{n-1})$. Given z_0, this sequence is called the *orbit* of z_0 under iteration of f. For example, here are the first few iterates of the point $z_0 = 1/2$ under the action of $f(z) = z^2$.

```
f[z_] := z^2;
NestList[f, 1/2., 4]
```

```
{0.5, 0.25, 0.0625, 0.00390625, 0.0000152588}
```

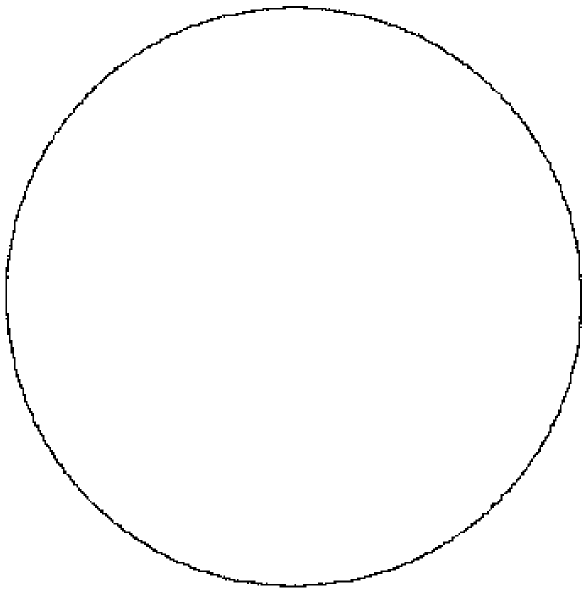
Note that the orbit tends to 0. For this function, it is not difficult to see that the orbit of z_0 will tend to 0 if $|z_0| < 1$, while the orbit of z_0 will diverge to ∞ if $|z_0| > 1$.

In order to understand the global behavior of the dynamics of a function, we divide the complex plane into two regions. The *Fatou set F* may be defined to be the largest open set on which the set of iterates of f forms a normal family. Intuitively, this may be thought of as the largest open set on which the dynamics of f are relatively tame in the sense that points close to one another have similar long term behavior. The *Julia set J* is defined to be the complement of the Fatou set and the dynamics of f are quite chaotic on J.

For example, our observations above suggest that the Fatou set for $f(z) = z^2$ should consist of two disjoint parts, the interior and the exterior of the unit circle. The dynamics right on the unit circle are much more complicated, however. If $|z_0| = 1$, then we may find points as close as we like to z_0 which tend to zero under iteration of f, and we may also find points as close as we like to z_0 that tend to ∞ under iteration of f. Thus the unit circle is the Julia set for this function.

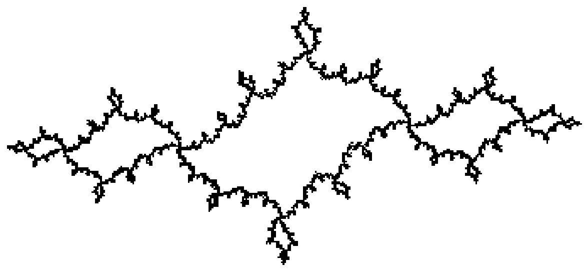
Julia sets are not are main focus here, but they may be generated using the **`Julia`** command defined in the **`CubicIteration`** package. This command uses an inverse iteration algorithm and is fully described in [1], which also contains a more complete description of Julia sets. Here's the Julia set for $f(z) = z^2$.

```
Needs["CubicIteration`"];
Julia[z^2, z];
```



And here is a more complicated Julia set.

```
Julia[z^2 - (1 + i)/4 z - 1 , z];
```



3. Quadratic iteration and the Mandelbrot set

The Mandelbrot set arises out of the desire to classify all possible types of dynamics arising out of the family of quadratic functions. There are two main observations which simplify this objective; (1) the dynamics of any quadratic is similar (conjugate, to be more precise) to some quadratic of the form $f_c(z) = z^2 + c$ and (2) the dynamics of any polynomial are dominated by the orbits of the critical points. Let's take a careful look at both of these points.

■ Dynamical conjugacy

The general quadratic function has the form $g(z) = \alpha\, z^2 + \beta\, z + \gamma$. Thus we could parameterize the space of all complex quadratics using three complex parameters α, β, and γ. It turns out that we can reduce this to just one parameter using the notion of dynamical conjugacy.

Two complex function $f(z)$ and $g(z)$ are said to be *affinely conjugate* if there is an affine function $\phi(z) = m\,z + d$ so that $f(\phi(z)) = \phi(g(z))$.

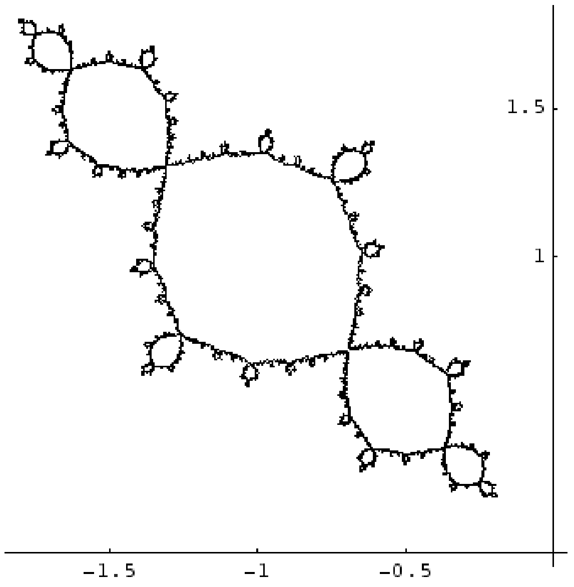
Note that, since ϕ is invertible, this could be written $g = \phi^{-1} \circ f \circ \phi$. Thus $g^2 = \phi^{-1} \circ f \circ \phi \circ \phi^{-1} \circ f \circ \phi = \phi^{-1} \circ f^2 \circ \phi$. More generally, $g^n = \phi^{-1} \circ f^n \circ \phi$ so the two functions have identical dynamics.

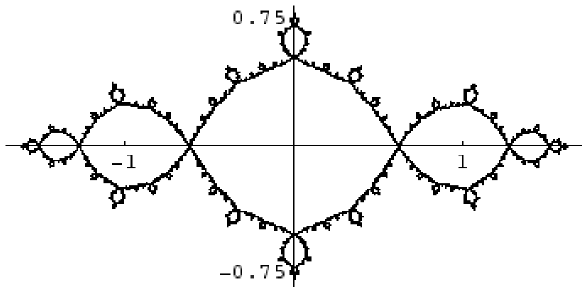
It turns out that any function of the form $g(z) = \alpha\, z^2 + \beta\, z + \gamma$ is conjugated by $\phi(z) = \alpha\, z + \beta/2$ to $f_c(z) = z^2 + c$, where $c = (2\,\beta - \beta^2 + 4\,\alpha\,\gamma)/4$. We can use *Mathematica* to check this.

```
g[z_] := α z^2 + β z + γ;
f[z_] := z^2 + 1/4 (2 β - β^2 + 4 α γ);
ϕ[z_] := α z + β / 2;
Expand[f[ϕ[z]]] == Expand[ϕ[g[z]]]

True
```

For example, $g(z) = (1 + i)\, z^2 + 4\, z + 1/(1 + i)$ is conjugate to $f_{-1}(z) = z^2 - 1$, since $(2 \cdot 4 - 4^2 + 4\,(1 + i)/(1 + i))/4 = -1$. To illustrate the significance of this, notice the geometric similarity between the Julia sets of these two functions.

```
Show[GraphicsArray[
   {{Julia[(1 + ⅈ) z² + 4 z + 1 / (1 + ⅈ), z,
      DisplayFunction → Identity, Axes → True,
      Ticks → {{-1.5, -1, -.5}, {1.5, 1, 1.5}}]},
    {Julia[z² - 1, z,
      DisplayFunction → Identity, Axes → True,
      Ticks → {{-1, 1}, {-.75, .75}}]}}]];
```





■ Critical points and the definition of the Mandelbrot set

It turns out that the dynamical behavior of a polynomial is dominated by the behavior of it's critical points, i.e. complex numbers z such that $f'(z) = 0$. The orbit of such a point is called a *critical orbit*. The influence of the critical orbits is stated by the following theorem, the parts of which are essentially theorems III.4.1 and III.4.2 in [2].

Theorem: Let $f : \mathbb{C} \to \mathbb{C}$ be a polynomial.
(a) The Julia set of f is connected if and only if all critical orbits are bounded.
(b) If all critical orbits diverge to ∞, then the Julia set of f is totally disconnected.

Of course any function of the form $f_c(z) = z^2 + c$ has precisely one critical point, namely $z = 0$. Thus there are only two possibilities.

(1) The orbit of 0 under f_c is bounded, in which case the Julia set of f_c is connected or
(2) The orbit of 0 under f_c diverges to ∞, in which case the Julia set of f_c is totally disconnected.

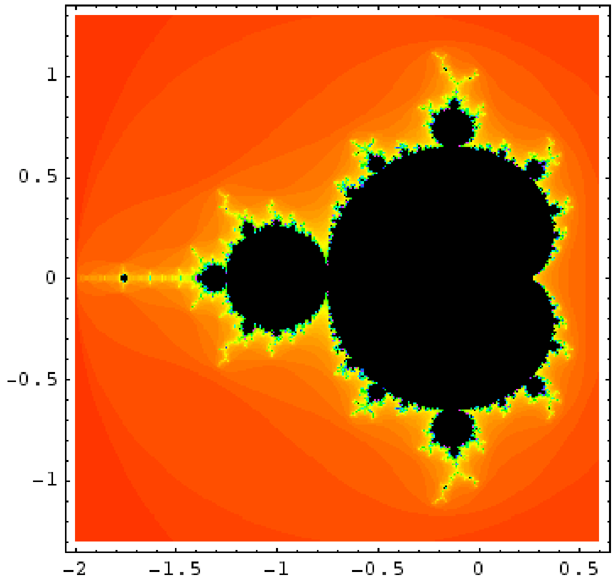
Furthermore, it may be proved that if $|f_c^n(0)| > 2$ for some n, then the critical orbit escapes to ∞. (See, for example, [2] Theorem VIII.1.2.) We call the first n so that $|f_c^n(0)| > 2$, the *escape time* of the critical orbit of f_c.

We now define the *Mandelbrot set*, M, to be the set of all complex numbers c such that the Julia set of f_c is connected. Thus M naturally decomposes the complex plane into two parts corresponding to the two possible types of dynamical behavior for quadratic functions; $c \in M$ if the Julia set of f_c is connected and $c \notin M$ if the Julia set of f_c is totally disconnected. Furthermore, the role of the critical orbit provides an algorithm to generate an image of the Mandelbrot set. We first write a function to define a test to check if the complex number is in M or outside of M.

```
bail = 100;
MandelbrotFunction = Compile[{{c, _Complex}},
   Length[NestWhileList[#^2 + c &, 0,
     Abs[#] ≤ 2 &, 1, bail]]];
```

Note that the **NestWhileList** command iterates $z^2 + c$ (represented as a pure function) up to 100 times or until the value exceeds 2 in absolute value. The **Length** command then returns how many iterations took place. We may now generate the Mandelbrot set by simply making a **DensityPlot** of the **MandelbrotFunction**.

```
DensityPlot[MandelbrotFunction[x + y ⅈ], {x, -2, 0.6},
  {y, -1.3, 1.3}, Mesh → False, AspectRatio → Automatic, PlotPoints → 300,
  ColorFunction → (If[# == 1, RGBColor[0, 0, 0], Hue[0.9 #]] &)];
```

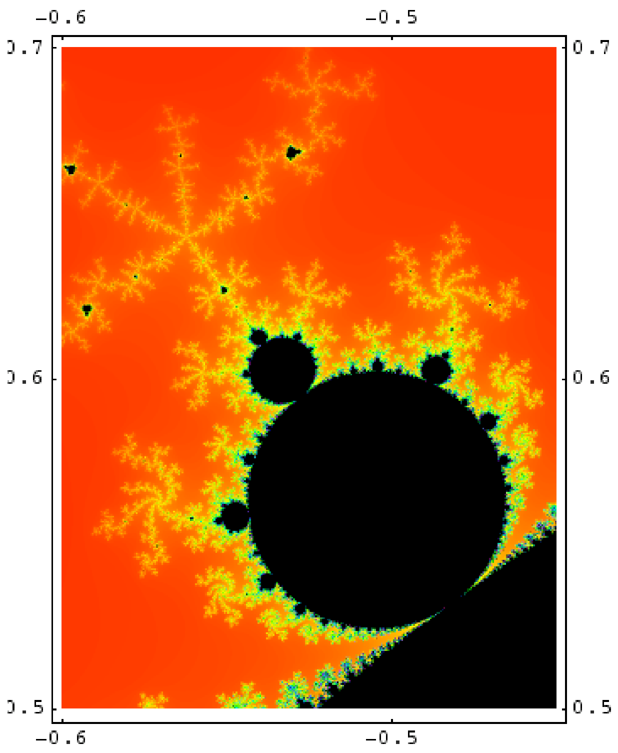


The above code is a minor variation of that found in [3]. This technique has been encapsulated in the package function **ShowMandelbrotSet**. While algorithm is the same, the package version calls a java implementation which runs much faster. This allows us increase the iteration and resolution considerably to zoom in on the set. The basic prototype of **ShowMandelbrotSet** is as follows.

```
ShowMandelbrotSet[cMin, cMax, bail, res];
```

The complex numbers **cMin** and **cMax** determine the lower left and upper right corners of the rectangular region in the complex plane to draw. The integer **bail** determines a maximum number of iterations and the integer **res** determines the resolution of the image.

```
ShowMandelbrotSet[-.6 + .5 ⅈ, -.45 + .7 ⅈ, 500, 600,
  FrameTicks → {{-.6, -.5}, {.5, .6, .7}}];
```



4. Cubic iteration

We now turn to the question of cubic iteration. It turns out that the general cubic is affinely conjugate to one of the form $f_{a,b}(z) = z^3 - 3\,a^2\,z + b$. The coefficients are chosen so that the critical points of $f_{a,b}$ are $\pm\,a$. We can see that there are two main difficulties in this situation: (1) The set of all cubics is naturally described using two complex parameters, so we may think of it as four dimensional. (2) A cubic may have two distinct critical points versus a single critical point for quadratics.

The first problem is dealt with simply enough; we will only generate two dimensional slices of the parameter space. For example, we might hold a constant and allow b to vary within a picture. A collection of such pictures will give us some understanding of the overall structure.

To deal with the second point it is natural to decompose the overall four dimensional parameter space $\mathbb{C}^2$ into the following four parts.

C : The set of all a and b values so that both critical orbits of $f_{a,b}$ are bounded. This is called the *cubic connected locus*.

C_+ : The set of all a and b values so that the orbit of a is bounded while the orbit of $-a$ is unbounded.

C_- : The set of all a and b values so that the orbit of $-a$ is bounded while the orbit of a is unbounded.

D : The set of all a and b values so that both critical orbits of $f_{a,b}$ diverge to ∞.

There is one other more subtle complication; determination of the appropriate escape radius. It turns out that if for some n, $\left| f_{a,b}^n(z_0) \right| > \max\left(|b|, \sqrt{9|a|^2+2} \right)$, then the orbit of z_0 will diverge to ∞. (See [4], page 266.) This number $\max\left(|b|, \sqrt{9|a|^2+2} \right)$ is the escape radius for $f_{a,b}$.

We now have a strategy to generate a two dimensional slice of the cubic parameter space.

Fix some value of a and suppose that b is allowed to vary inside a rectangular region in $\mathbb{C}$. This defines a two dimensional family of cubic polynomials given by $f_{a,b} = z^3 - 3a^2 z + b$.

Iterate $f_{a,b}$ starting from both critical points $\pm a$.

If after some pre-specified number of iterations neither critical orbit has escaped, color b black.

If a escapes but -a does not escape, color b according to the escape time of a.

If -a escapes but a does not escape, color b according to the escape time of -a.

If both critical points escape then color the point using some combination of the escape times.

To implement this, we first write a function **CubicEscapeTimes** which computes the escape times of $\pm a$ under iteration of $f_{a,b}$.

```
bail = 100;
CubicEscapeTimes = Compile[{{a, _Complex}, {b, _Complex}},
   n1 = Length[NestWhileList[#^3 - 3 a^2 # + b &, a,
      Abs[#] ≤ Max[Abs[b], Sqrt[Abs[9 a^2] + 2]] &, 1, bail]];
   n2 = Length[NestWhileList[#^3 - 3 a^2 # + b &, -a,
      Abs[#] ≤ Max[Abs[b], Sqrt[Abs[9 a^2] + 2]] &, 1, bail]];
   {n1, n2}];
```

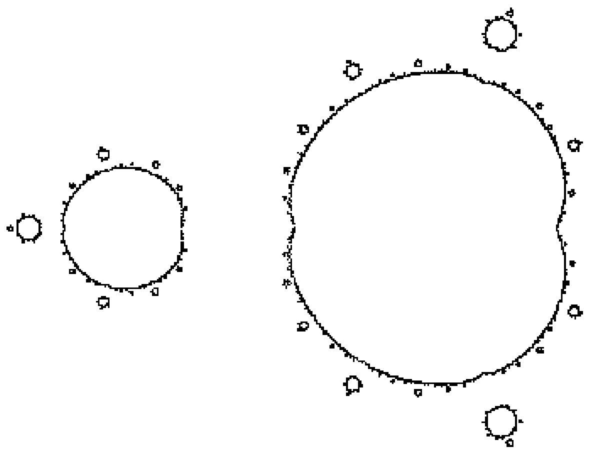
For example, the following computation shows that for the function $f(z) = z^3 - 3(.5)^2 z + .8$, the critical point $-.5$ escapes after just 5 iterations while the critical point .5 does not escape after 100 iterations.

```
CubicEscapeTimes[.5, .8]
```

```
{101., 5.}
```

Note that the structure of the Julia set of this function is not possible for a quadratic Julia set; it consists of infinitely many distinct connected components.

```
Julia[z^3 - 3/4 z + .8 , z];
```

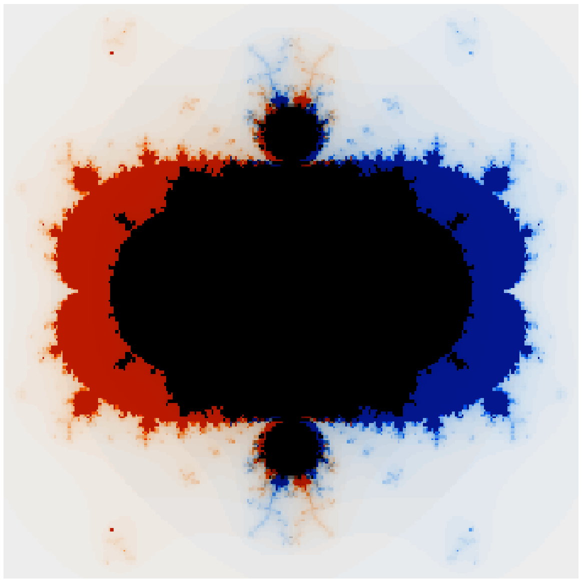


The following function now turns a pair of escape times into an **RGBColor** directive.

```
CubicColor[{l1_, l2_}] := Which[
   (* Both orbits bounded, paint black *)
   l1 ≥ bail && l2 ≥ bail, RGBColor[0, 0, 0],

   (* a escapes, shade red *)
   l1 < bail && l2 ≥ bail, RGBColor[(1 - l1 / bail) .7, 0, 0],

   (* -a escapes, shade blue *)
   l1 ≥ bail && l2 < bail, RGBColor[0, 0, (1 - l2 / bail) / 2.],

   (* both escape, use a combination of colors *)
   l1 < bail && l2 < bail, RGBColor[(1 - l1 / bail) ^3,
    (1. - l1 / (2. bail) - l2 / (2 bail)) ^3, (1. - l2 / bail) ^3]
  ];
```

We now compute a **Table** of escape time pairs, convert them to colors, and display the result using **RasterArray**.

```
a = .5;
escapeTimes = Table[CubicEscapeTimes[a, x + ⅈ y],
   {y, -1.2, 1.2, .01}, {x, -1.2, 1.2, .01}];
colors = Map[CubicColor, escapeTimes, {2}];
Show[Graphics[RasterArray[colors]],
  AspectRatio → Automatic];
```

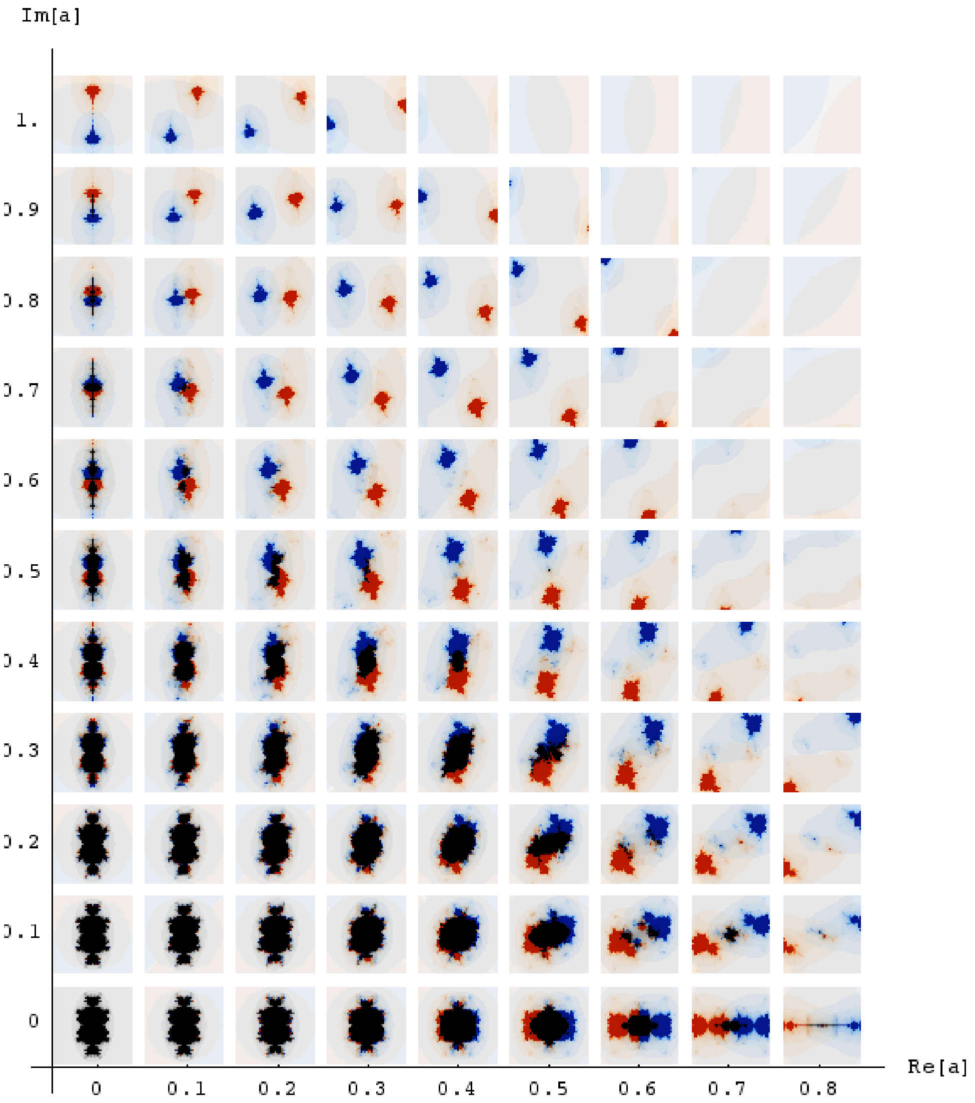


As in the quadratic case, this has all been implemented in Java allowing us to generate images much more quickly. The essential command is

```
ShowCubicLocus[a, bMin, bMax, bail, res];
```

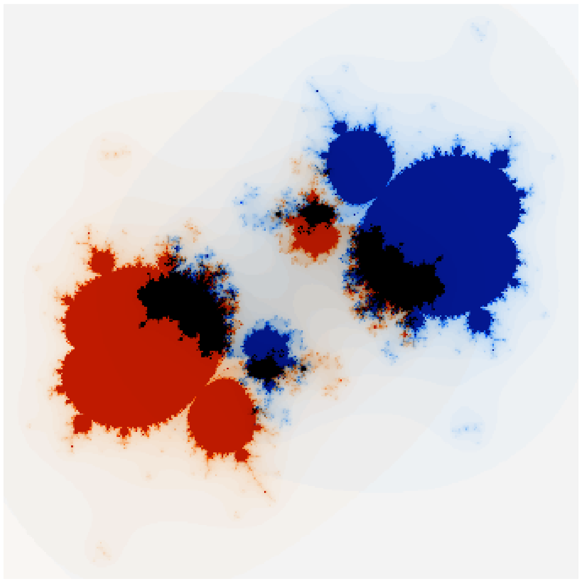
We now use **ShowCubicLocus** to generate an array of images which captures the overall four dimensional structure of the set. Inside each individual image, *a* is fixed and *b* varies. The variable *a* changes as we move from picture to picture as indicated by the axes marks. Symmetry allows us to focus on the first quadrant.

```
slices = Reverse[Table[ShowCubicLocus[x + ⅈ y,
     -1.5 - 1.5 ⅈ, 1.5 + 1.5 ⅈ,
     50, 50, DisplayFunction → Identity],
    {y, 0., 1., .1}, {x, 0., .8, .1}]];
Show[GraphicsArray[slices],
  Axes → True, AxesLabel → {"Re[a]", "Im[a]"},
  RotateLabel → False,
  Ticks → {Table[{i + .5, i / 10.}, {i, 0, 8}],
    Table[{i + .5, i / 10.}, {i, 0, 10}]}];
```



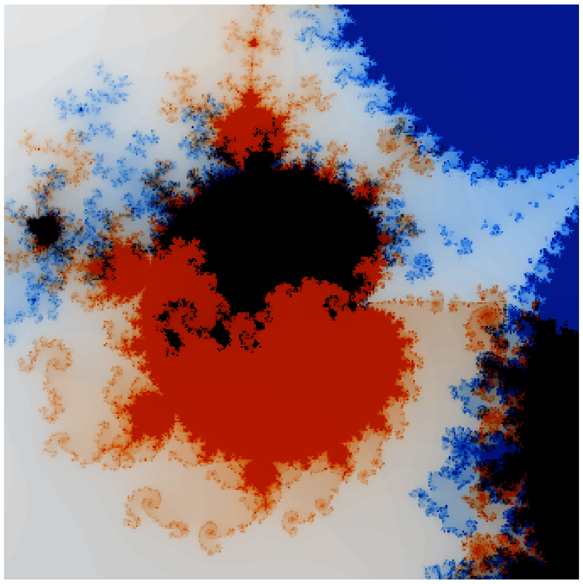
We can take a closer at these images.

```
ShowCubicLocus[.6 + .1 ⅈ, -1.5 - 1.5 ⅈ, 1.5 + 1.5 ⅈ,
  100, 400];
```



We can zoom in on them as well.

```
ShowCubicLocus[.6 + .1 ⅈ, -.1 + .1 ⅈ, .4 + .6 ⅈ,
  100, 400];
```



References

1. M. McClure, Inverse Iteration algorithms for Julia sets, *Mathematica in Education and Research*, 7 #2 (1998) 22-28.

2. Carleson, L. and Gamelin, T.W. *Complex Dynamics*. Springer-Verlag, New York, 1993.

3. Dickau, R. M. Compilation of iterative and list operations in "Tricks of the trade." *The Mathematica Journal,* **7** #1 (1997)14-15.

4. Devaney, R.L. *A First Course in Chaotic Dynamical Systems*. Addison-Wesley, Reading, Mass. 1992.