

Fractals from Newton's Method

Newton's method for solving the equation $f(z) = 0$ generates successive estimates for the root from the iteration

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

Here we take $f(z) = z^n - 1$, whose roots are at $z = e^{i\theta}$ where $\theta = 2\pi k/n$, where k takes integer values from 0 to $n-1$.

It turns out that, for $n > 2$, the boundaries of the regions of the complex plane which converge to each of the roots have a **complicated fractal geometry**.

To see this we generate a function, `newt(n, z)`, which finds which root of $z^n = 1$ one converges too starting from the given value of z , and returns the value of $\theta/(2\pi)$. `newt` uses the *Mathematica* function `FixedPoint[fun, z, k]` (where the third argument is the maximum number of iterations) to determine which root the iterations of the Newton Raphson method converge to. The argument of z then gives θ .

```
In[1]:= Clear["Global`*"]
```

```
In[2]:= newt[n_, z_] := Arg[FixedPoint[# - (#^n - 1) / (n #^(n - 1)) &, N[z], 50]] / (2 Pi);
```

The function $f(z) = z^n - 1$ is represented as a pure function, `#^n - 1` (where `#` is the value of z) which is terminated by a `&`. Hence $z - f(z)/f'(z)$ is represented by the hieroglyphics `# - (#^n - 1)/(n #^(n-1))&`. The `FixedPoint` function will either converge to a fixed point or stop after the specified number of iterations (50 here).

It will be more efficient to compile this code, specifying that n is integer and z is complex:

```
In[3]:= newtC = Compile[{{n, _Integer}, {z, _Complex}},  
  Arg[FixedPoint[# - (#^n - 1) / (n #^(n - 1)) &, N[z], 50]] / (2 Pi)];
```

Note, as discussed before, that the arguments of `newtC` appear as the first argument of `Compile` on the right hand side, where we have the option, used here, of specifying whether they are real, integer, or complex (the default is real).

For example, for $n = 3$, starting from 2, $-2 + 2I$, and $-2 - 2I$, `newt` converges to the three cube roots of -1 (which have argument 0, $2\pi/3$, and $-2\pi/3$, so the output from `newt` is 0, 1/3, and -1/3 since I divide by 2π).

```
In[4]:= {newtC[3, 2], newtC[3, -2 + 2 I], newtC[3, -2 - 2 I]}
```

```
Out[4]= {0., 0.333333, -0.333333}
```

In previous versions of *Mathematica* I constructed a routine `newtonplot` which produces a density plot of the values produced by `newtC`, such that each color corresponds to a particular root. The color is specified by the option `ColorFunction -> (Hue[#]&)` of the `DensityPlot` command. The argument of the function `Hue[h]` has to lie between 0 and 1. (If an argument outside this range is given, it is shifted by an appropriate integer to put it into this range, e.g. $-0.3 \rightarrow 0.7$.) As h ranges between 0 and 1 the corresponding color goes through red, yellow, green, cyan, blue, magenta, and back to red again. The output from `newt` is therefore an appropriate argument for `Hue`. Note that `Hue` is indicated as a "pure function", i.e. its argument is specified by `#` and the end of the function is indicated by `&`. The option `ColorFunctionScaling -> False` prevents *Mathematica* from trying to scale the range of values fed to the `ColorFunction` to lie between 0 and 1, which leads to less satisfactory colors. (It's better to do the scaling oneself, which I have done by dividing the argument of the root by 2π .)

```
newtonplot[n_, npoint_, xmin_, xmax_, ymin_, ymax_] := DensityPlot[  
  newtC[n, x + I y],  
  {x, xmin, xmax}, {y, ymin, ymax}, Mesh -> False, PlotPoints -> npoint,  
  Frame -> False, ColorFunctionScaling -> False, ColorFunction -> (Hue[#] &)]
```

However **DensityPlot** is impossibly slow with version 6.

I therefore, instead, explicitly create a list of values to be plotted, and then plot them using **ListDensityPlot**:

```
In[5]:= array[n_, npoint_, xmin_, xmax_, ymin_, ymax_] := Flatten[Table[{x, y, newtC[n, x + I y]},
  {x, xmin, xmax, (xmax - xmin) / npoint}, {y, ymin, ymax, (ymax - ymin) / npoint}], 1];
```

This will show us the regions in the complex plane which converge to each of the roots. We need to remove one set of brackets with **Flatten** (or **[[1]]**). If we don't do that

```
In[10]:= array2[n_, npoint_, xmin_, xmax_, ymin_, ymax_] := Table[{x, y, newtC[n, x + I y]},
  {x, xmin, xmax, (xmax - xmin) / npoint}, {y, ymin, ymax, (ymax - ymin) / npoint}];
```

```
In[12]:= array2[3, 4, 0.1, 1, 0.1, 1]
```

```
Out[12]= {{{0.1, 0.1, -0.333333}, {0.1, 0.325, -0.333333}, {0.1, 0.55, 0.}, {0.1, 0.775, 0.333333},
  {0.1, 1., 0.333333}, {0.325, 0.1, 0.}, {0.325, 0.325, -0.333333},
  {0.325, 0.55, -0.333333}, {0.325, 0.775, 0.333333}, {0.325, 1., 0.333333}},
  {{0.55, 0.1, 0.}, {0.55, 0.325, 0.}, {0.55, 0.55, 0.}, {0.55, 0.775, -0.333333},
  {0.55, 1., -0.333333}}, {{0.775, 0.1, 0.}, {0.775, 0.325, 0.},
  {0.775, 0.55, 0.}, {0.775, 0.775, 0.}, {0.775, 1., 0.333333}},
  {{1., 0.1, 0.}, {1., 0.325, 0.}, {1., 0.55, 0.}, {1., 0.775, 0.}, {1., 1., 0.}}}
```

We indeed have an unwanted extra set of brackets. However, using our function **array** we have a list of elements, each of which is a list of three quantities, x, y, and $\theta / 2\pi$, as desired:

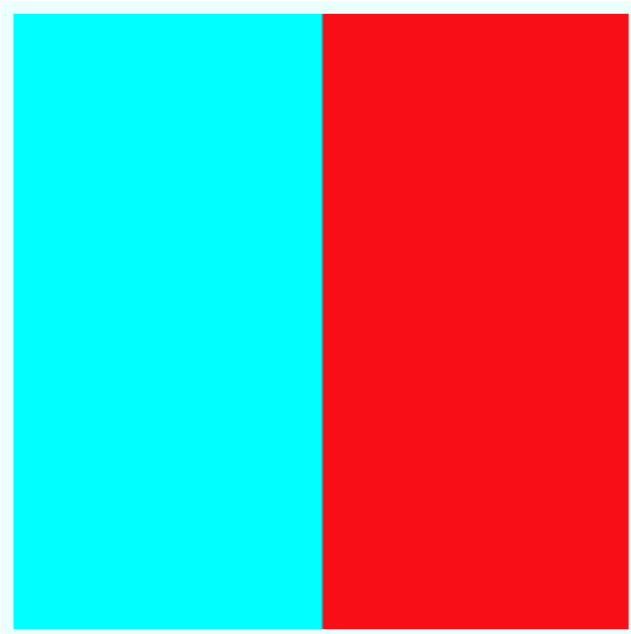
```
In[9]:= array[3, 4, 0.1, 1, 0.1, 1]
```

```
Out[9]= {{0.1, 0.1, -0.333333}, {0.1, 0.325, -0.333333}, {0.1, 0.55, 0.}, {0.1, 0.775, 0.333333},
  {0.1, 1., 0.333333}, {0.325, 0.1, 0.}, {0.325, 0.325, -0.333333}, {0.325, 0.55, -0.333333},
  {0.325, 0.775, 0.333333}, {0.325, 1., 0.333333}, {0.55, 0.1, 0.}, {0.55, 0.325, 0.},
  {0.55, 0.55, 0.}, {0.55, 0.775, -0.333333}, {0.55, 1., -0.333333}, {0.775, 0.1, 0.},
  {0.775, 0.325, 0.}, {0.775, 0.55, 0.}, {0.775, 0.775, 0.}, {0.775, 1., 0.333333},
  {1., 0.1, 0.}, {1., 0.325, 0.}, {1., 0.55, 0.}, {1., 0.775, 0.}, {1., 1., 0.}}
```

If we take $n=2$ the situation is simple (and boring):

```
In[6]:= ListDensityPlot[array[2, 200, -2.001, 2, -2, 2],
  ColorFunction -> Hue, ColorFunctionScaling -> False, Frame -> False]
```

```
Out[6]=
```

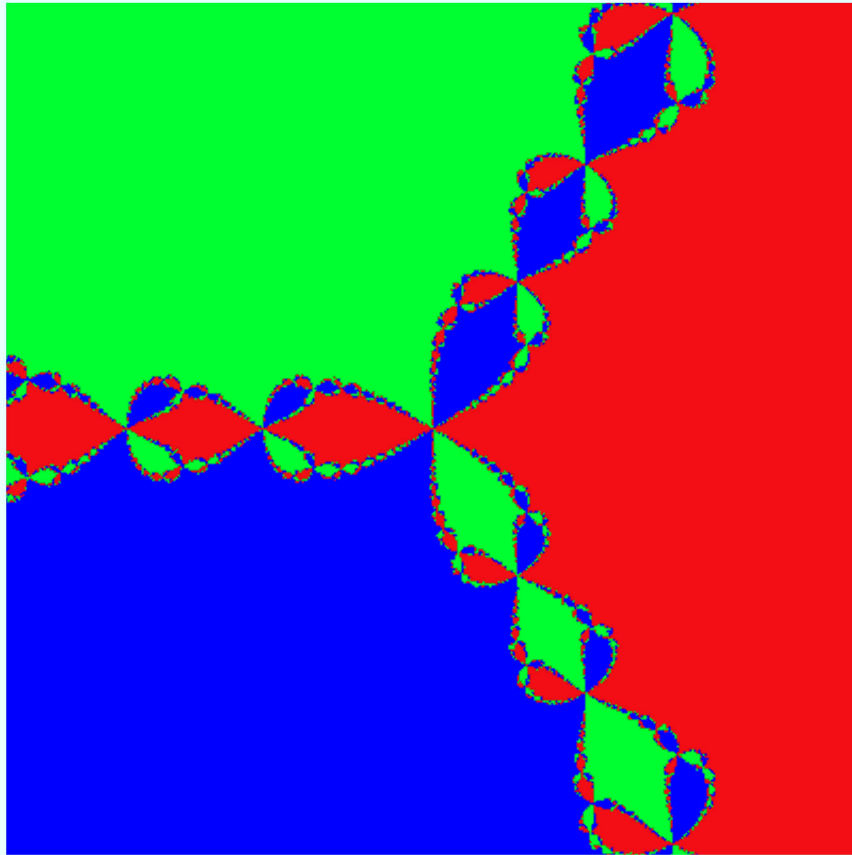


If $x (= \operatorname{Re}(z)) > 0$, we converge to the root at $x = 1$, while if $x < 0$, we converge to the root at $x = -1$. This is not very surprising.

What is surprising, however, is that this simple picture does *not* occur for $n > 2$. To get an idea of what happens we plot the situation for $n = 3$ (this really needs to be seen in color).

```
In[7]:= ListDensityPlot[array[3, 400, -2.001, 2, -2, 2],
  ColorFunction -> Hue, ColorFunctionScaling -> False, Frame -> False]
```

Out[7]=

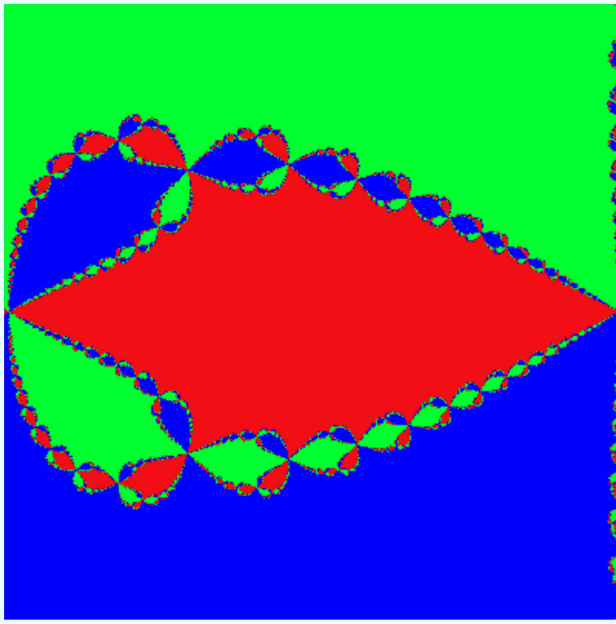


All points in the red region flow to the root at $z = 1$, in the green region to the root at $z = e^{2\pi i/3} = (-1 + \sqrt{3}i)/2$, and in the blue region to the root at $z = e^{-2\pi i/3} = (-1 - \sqrt{3}i)/2$. Naturally the immediate surroundings of each root flow to that root. However, the *boundary* between the different regions is extraordinarily complicated. Look closely and you will see the same structure repeated in different locations and different scales. A structure in which the same pattern repeats down to smaller and smaller scales *ad infinitum* is called a **fractal**. The reason that the fractal structure occurs will be discussed in class.

Next we take $n=3$ and blow up the area to the left of the origin, to better see the fractal structure.

```
In[13]:= ListDensityPlot[array[3, 400, -0.8001, 0, -0.4001, 0.4],  
ColorFunction -> Hue, ColorFunctionScaling -> False, Frame -> False]
```

Out[13]=



Finally we plot the rich structure for $n = 12$:

```
In[14]:= ListDensityPlot[array[12, 400, -2.001, 2, -2.003, 2],  
ColorFunction -> Hue, ColorFunctionScaling -> False, Frame -> False]
```

CompiledFunction::cfn :

Numerical error encountered at instruction 47; proceeding with uncompiled evaluation. >>

Out[14]=

