Mandelbrot in *Mathematica*

Guide to plotting the most famous instance of the Mandelbrot Set in Mathematica

Novak Petrovic
npetrovic@gmail.com

0 Version history

20 August 2005: Original release (done in *Mathematica* 5.0.0.0).

1 Introduction

In this article we present and comment on the algorithm required for plotting the most famous picture of the Mandelbrot Set. The main inspiration and information was obtained from [1].

2 Mathematical formulation

The Mandelbrot Set is the set obtained from the quadratic recurrence equation: [1]

$$(1)\quad z_{n+1} = z_n^2 + C$$

with $z_0 = C$. Those points C (in the complex plane) for which the value of z_n does not tend to infinity are in the set. Those points C for which the value of z_n tends to infinity are not in the Mandelbrot Set. By graphically representing (in a premeditated way) the Mandelbrot-qualified set of points C, we obtain our fractal image. While sufficient for our present purposes, this definition and explanation are by no means exhaustively rigorous.

In practical terms, the Mandelbrot Set works as follows. We take a point in the complex plane and call it C. Say C = -1.25 + I*0.75. If we wanted to represent this point graphically, its real part would be plotted on the x axis, while its imaginary part would be plotted on the y axis. We then put this starting point through our recurrence equation (1), noting that by definition we have $z_0 = C$. So, our first task is to obtain z_1, which is straight forward:

$$(2)\quad z_1 = z_0^2 + C = C^2 + C = C\,(C+1)$$

Similarly, we have

$$(3)\quad z_2 = z_1^2 + C = [C\,(C+1)]^2 + C$$

and so on. After a number if iterations, N, z_{n+1} can either converge to a constant value, or diverge to infinity. If z_{n+1} diverges, then C is not in the Mandelbrot Set; if it converges, it is. The information that we gain from the whole process (for a particular starting point C) is not the stable value of z_{n+1}, but rather it is the number of iterations (N) after which the stable value was reached. We note this value of N, and repeat the whole process with another starting complex number C, until we have covered the entire complex plane (or the most important region thereof).

In practical terms (especially for visualisation purposes, which is our main goal), we do not attempt to show divergence of z_{n+1} rigorously. Rather, we ask if $|z_{n+1}| > r_{\max}$. If it is, then we are satisfied that z_{n+1} has converged. An alternative test would be to see if some predetermined value of N has been reached without z_{n+1} converging. If it has been reached, then we can conclude that z_{n+1} has converged.

At the end of the whole process we end up with a list of ordered pairs {C, N} -- our Mandelbrot Set elements C, and the number of iterations after which convergence was converged for those particular elements. If we plot the real part of C on the x axis (of the rectangular coordinate system), the imaginary value on the y axis, N on the z axis, and if we colour all C's that have the same N with the same colour (according to some logical colouring scheme that also reflects the relative value of N for each C) then we can visualise the most famous Mandelbrot fractal set.

In the next section the above procedure will be programmed in *Mathematica* in a very crude way and very explicitly, in order to illustrate the above practical explanation.

3 Prototype code

```
Off[General::spell1];
```

As we cannot test every single point in the complex plane for whether it belongs to the Mandelbrot Set, we need to define a smaller portion of interets. We choose this portion to be a rectangle defined by its lower left and upper right corner points. We call the value of C in the lower left corner "clowerleft", and the value of C in the upper right corner "cupperright". I 'accidentally' happen to know in what region of the complex plane to find the Mandelbrot Set, so we will use that region. Also, we need to define our limit value that will be used in the convergence test ($|z_{n+1}| > r_{\max}$), as well as the maximum number of iterations allowed ("Nmax", for the alternative test):

```
clowerleft = -2.0 - I 2.0;
cupperright = 2.0 + I 2.0;
cstep = 0.05;
rmax = 2.0;
Nmax = 50;
```

"cstep" above determins the number of discrete C points that we shall examine for membership in the Mandelbrot Set (as, again, we do not want to get bogged down too much in calculations at this stage). The following function will be used to calculate the value of z_{n+1}:

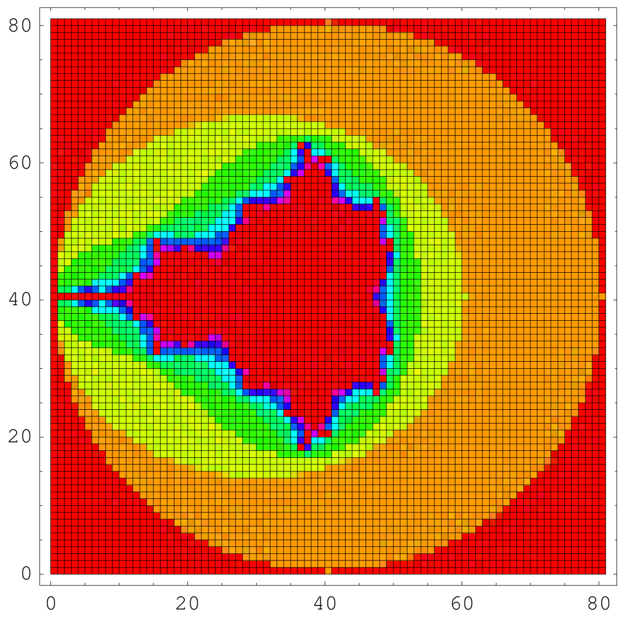
```
znplus1[zn_, c_] := zn^2 + c;
```

In "results" we will store values for plotting, in the format required by *Mathematica*'s ListDensityPlot function, which we will use for visualisation. Due to the peculiar way in which ListDensityPlot processes information, we also need to have an auxiliary list to store intermediate results, "auxresults".

```
results = {};

(* We take the next consecutive value of y,
 from the range determined by the imaginary parts of two corner points. *)

For[cypart = Im[clowerleft], cypart ≤ Im[cupperright], cypart += cstep,
 (* In "auxresults" we keep all output (N) values for the one above y point,
  thus making it a row of N values for all x' s in the range. *)

 auxresults = {};

 (* Then we step through all x values in the region of interest. *)

 For[cxpart = Re[clowerleft], cxpart ≤ Re[cupperright], cxpart += cstep,

  (* Since now we have our starting point defined (we have its x and y value),
   we can set the counter of iterations to zero
     (for testing that particular point). We denote this
     starting point as c0. Note that z0=c0 by definition. *)

  counter = 0;
  c0 = cxpart + I * cypart;
  zvalue = c0;

  (* We use "flag" to break out of the recurrence loop below. *)

  flag = False;
  While[! flag,

   (* We first test if either of the two conditions for the c0 are satisfied. *)

   If[Abs[zvalue] > rmax || counter > Nmax,

     (* If they were, we raise the flag,
      and add the value to the row of x values. *)

     flag = True;
     AppendTo[auxresults, counter],

     (* Otherwise,
      we get our next z_{n+1} value, and increase the iteration counter. *)

     zvalue = znplus1[zvalue, c0];
     counter ++;
    ];
  ];
 ];

 (* Once we have gone through all x values in the range,
  we add that row of x values to our results,
  and take on the next y value (for which there will be another row). *)

 AppendTo[results, auxresults];
];
```

The coloring scheme that we have used is based on colour hues. The result that we obtained shows us that our algorithm is essentially right, but that we need to include more points from the region. However, in order to do that, we need to write more optimised code. This is done in the following section.

```
ListDensityPlot[results, ColorFunction → Hue];
```



4 Optimised code

The optimised code presented here does the same as the code presented in Section 3, but in a much more effective and efficient manner. It is primarily based on the code on the "Fractal.m" package, used in [1].

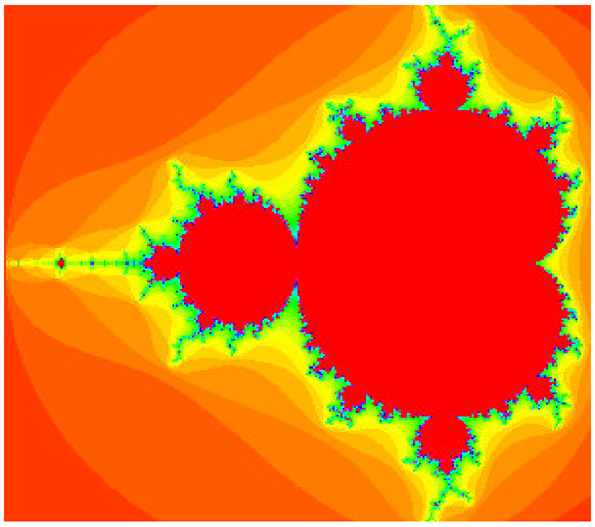
Function "mandelbrotEquation" gives us the number of iterations after which convergence of the recurrence equation is achieved. We asked for this function to be compiled for faster execution.

```
mandelbrotEquation = Compile[{{C, _Complex}, {Nmax, _Integer}, {rmax, _Real}},
   Length[FixedPointList[#^2 + C &, C, Nmax, SameTest → (Abs[#2] > rmax &)]]];
```

Function "showMandelbrotSet" plots the results of "mandelbrotEquation", for all points in the rectangle defined by "zmin" and "zmax", with user-specified "Nmax" and "rmax", as per their previous definitions. These two points (note that now they are given as a list) are the lower left and upper right corners of the rectangle. We have also fine-tuned the look of our output slightly. It is important to note here that we are using the continuous DensityPlot function (as we have defined our fundamental recurrence equation as a function), rather than the discrete-valued ListDensityPlot function.

```
showMandelbrotSet[{zmin_, zmax_}, Nmax_, rmax_] := Module[{x, y},
    DensityPlot[
      mandelbrotEquation[x + I y, Nmax, rmax],
      Evaluate[{x, Re[zmin], Re[zmax]}],
      Evaluate[{y, Im[zmin], Im[zmax]}],
      Mesh → False,
      AspectRatio → Automatic,
      Frame → False,
      ColorFunction → Hue,
      PlotPoints → 300
     ];
   ] /; Im[zmin] ≠ 0 || Im[zmax] ≠ 0;
```

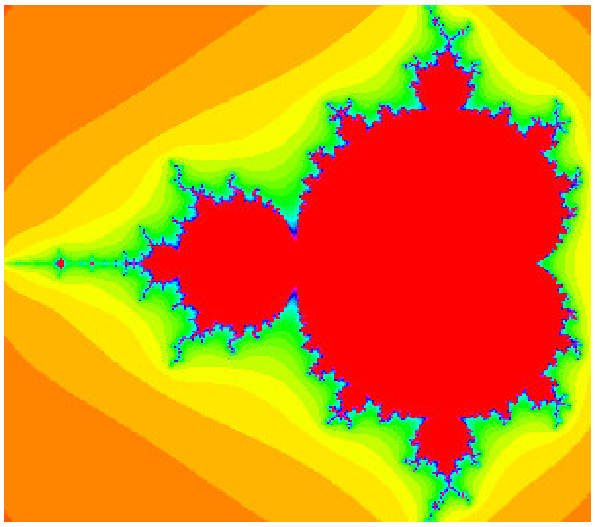
```
showMandelbrotSet[{-2 - 1.1 I, 0.5 + 1.1 I}, 50, 2.0];
```



The above picture is what we were after: a plot of the most famous instance of the Mandelbrot Set in *Mathematica*.

By changing the following values: maximum number of allowed iterations (currently N = 50), limit value for the divergence test (currently r_{max} = 2.0), the number of plot points (currently PlotPoints = 300), and the type of colouring function, we can change the way in which the above representation of the Mandelbrot Set looks like. The output produced below is one such example.

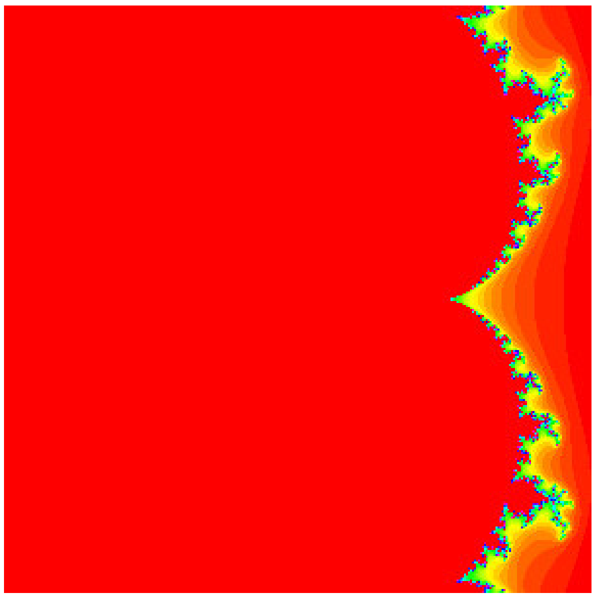
```
showMandelbrotSet[{-2 - 1.1 I, 0.5 + 1.1 I}, 33, 3.3];
```



5 More general code

"showMandelbrotSet" function, as defined above, only works (as can be seen from its last line) when Im[zmin] != 0 OR Im[zmax] != 0. In other words, only one of the two corners can lie on the x (real) axis. In case that Im[zim] = 0 AND Im[zmax] = 0 (both corners are real values), then we have the following definition of "showMandelbrotSet" (which is coexistent with the previous definition):

```
showMandelbrotSet[{zmin_, zmax_}, Nmax_, rmax_] :=
 showMandelbrotSet[{zmin (I + 1), zmax (I + 1)}, Nmax, rmax] /; Im[zmin] == Im[zmax] == 0

showMandelbrotSet[{-0.5, 0.5}, 50, 2.0];
```



Note that the supplied real values are still converted to complex ones before running the routine.

6 More general Mandelbrot Set equations

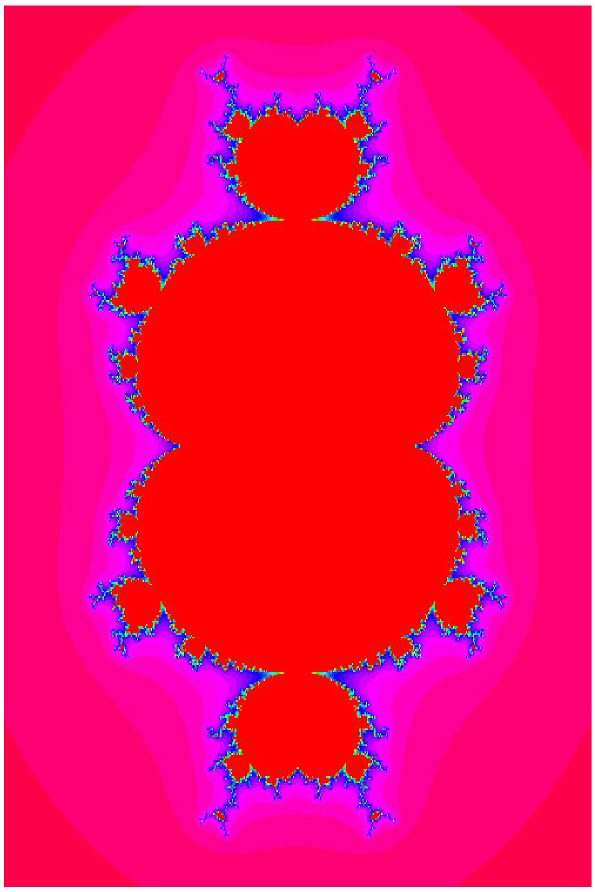
Finally, here is the code for slightly more general Mandelbrot Sets, in which raising of z_n to the power of two in the recurrence equation is replaced by raising to any arbitrary power (denoted by "n"). Note that we have incorporated the "mandelbrotEquation" function into the "showMandelbrotSet" function, for convenience, and that we have three definitions of the functions, depending on the characteristics of zmin, zmax, and n.

```
showMandelbrotSet[pow_Integer?Positive,
   {zmin_, zmax_}, Nmax_, rmax_] := Module[{x, y},
    DensityPlot[
      -Length[
        FixedPointList[#^pow + (x + I y) &, x + I y, Nmax, SameTest → (Abs[#2] > rmax &)]],
      {x, Re[zmin], Re[zmax]},
      {y, Im[zmin], Im[zmax]},
      PlotPoints → 500,
      Mesh → False,
      Frame → False,
      AspectRatio → Automatic,
      ColorFunction → Hue];
   ] /; Im[zmin] ≠ 0 || Im[zmax] ≠ 0;
```

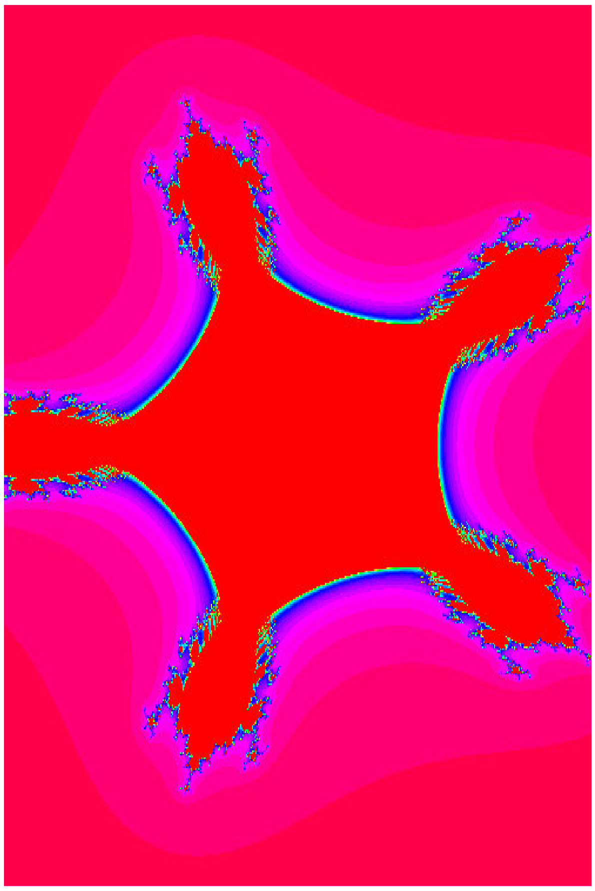
```
showMandelbrotSet[pow_Integer?Negative, {zmin_, zmax_}, Nmax_, rmax_] :=
  Module[{x, y}, DensityPlot[-Length[
        FixedPointList[Conjugate[#] ^ (-pow) + (x + I y) &,
         x + I y, Nmax, SameTest → (Abs[#2] > rmax &)]
       ],
      {x, Re[zmin], Re[zmax]},
      {y, Im[zmin], Im[zmax]},
      PlotPoints → 500,
      Mesh → False,
      Frame → False,
      AspectRatio → Automatic,
      ColorFunction → Hue];
   ] /; Im[zmin] ≠ 0 || Im[zmax] ≠ 0;


showMandelbrotSet[n_Integer, {zmin_, zmax_}, Nmax_, rmax_] :=
  showMandelbrotSet[n, {zmin (I + 1), zmax (I + 1)}, Nmax, rmax] /;
   Im[zmin] == Im[zmax] == 0 && n ≠ 0;
```

Here are three different results in three different cases, for illustration purposes.

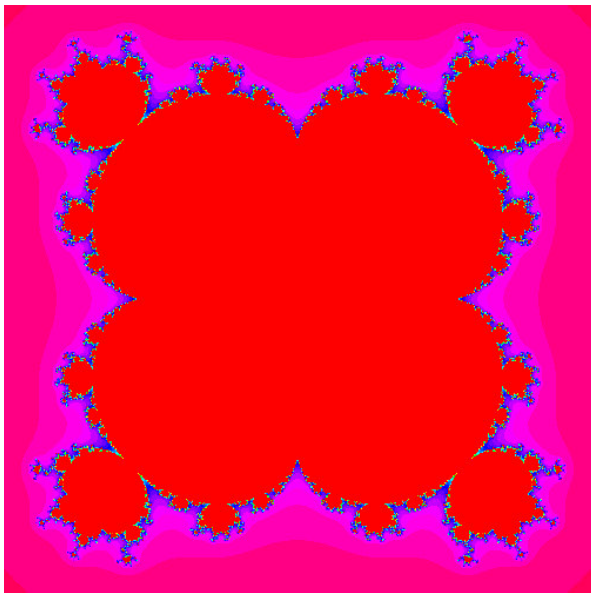
```
showMandelbrotSet[3, {-1.0 - 1.5 I, 1.0 + 1.5 I}, 40, 3.0];
```



```
showMandelbrotSet[-4, {-1.0 - 1.5 I, 1.0 + 1.5 I}, 40, 3.0];
```



```
showMandelbrotSet[5, {-1.0, 1.0}, 33, 3.3];
```



7 Conclusion

In this article we have presented and commented on the algorithm required for plotting the most famous picture of the Mandelbrot Set. We have stressed that the key information required for colouring the complex plane is the number of iterations after which the characteristic equation converges, given a particular starting point in the complex plane. Each starting point in the complex plane is coloured based on the number of iterations.

References

[1] Eric W. Weisstein. "Mandelbrot Set." From *MathWorld*--A Wolfram Web Resource. http://mathworld.wolfram.com/-MandelbrotSet.html