

Interacting with `FractalStream`

M. Noonan
Cornell University

November 17, 2009

`FractalStream` is a program designed to allow exploration, experimentation, and research on a wide variety of dynamical systems.

1 The Scripting Language

Dynamical systems in `FractalStream` are described by a high-level scripting language, which is then compiled into fast, native machine code.

1.1 Arithmetic

`FractalStream` is specially tailored towards complex dynamics, though it can be used for real dynamics as well. By default, all variables are complex numbers.

Commands have the form of simple sentences, and all end with a period (“.”). There is no sensitivity to case — “z” and “Z” denote the same expression. Here are some simple commands and operators which may be used:

- “`set var to expr.`”
Sets the value of the variable *var* to the arithmetic expression *expr*.
- “`default var to expr.`”
Allows the variable *var* to be edited in the Parameters tab of the Settings window. Useful if there is a value which you would like to make user-editable. The variable *var* will start with the value *expr*.

- Comments may be placed inside parentheses: “(all of this is a comment)”.
- Most arithmetic operations are supported for both real and complex variables. If x and y are variables or expressions, then you can also form
 - Constants: “17”, “-1.2”, “i”, “17 - 1.2i”.
 - Complex arithmetic: “x+y”, “x-y”, “x*y”, “x/y”.
 - Exponentiation “x^n”, where n is a positive constant integer.
 - Functions of a complex variable: “exp(x)”, “cos(x)”, “sin(x)”, “tan(x)”, “cosh(x)”, “sinh(x)”, “tanh(x)”.
 - Functions for manipulating complex numbers: “re(x)”, “im(x)”, “bar(x)”. The function “arg(x)” computes the inverse tangent of $im(x)/re(x)$.
 - You can use both “log(x)” and “sqrt(x)”, which have fixed branch cuts for complex numbers.
 - For real variables, you may also use the inverse trig functions “arccos(x)”, “arcsin(x)”, and “arctan(x)”. Real remainders can also be taken using the command “reduce var mod expr..”
- You can execute conditional code using “if cond then command.” or “if cond then command1. else command2.” where *cond* is a conditional statement and *command* is any command. Conditional statements can be of the forms
 - “ $expr1 = expr2$ ”. Evaluates to true if the distance between $expr1$ and $expr2$ is less than `minRadius`. The parameter `minRadius` may be manipulated in the Settings window of the viewer.
 - “ $expr1 < expr2$ ”, “ $expr1 \leq expr2$ ”, “ $expr1 > expr2$ ”, “ $expr1 \geq expr2$ ”. Perform the given comparison. If either $expr1$ or $expr2$ is complex, the comparison is by norm. Thus, “ $1 + i > 1.5$ ” evaluates to true. Remember that variables default to complex, so even if you know that a variable r is supposed to have imaginary part zero, you should use a comparison like “ $re(r) < -1$ ” rather than “ $r < -1$ ”. Since `FractalStream` assumes that r is complex, the latter expression will always evaluate to false.

- “*expr escapes*”. Evaluates to true if the norm of *expr* is greater than `maxRadius`. The parameter `maxRadius` may be manipulated in the Settings window of the viewer. If “*x escapes*” is true, we can loosely interpret this as saying *x* is at infinity.
 - “*expr vanishes*”. This is shorthand for “*expr = 0*”, so “*expr vanishes*” is true when the norm of *expr* is less than `minRadius`.
 - “*expr stops*”. This is a special condition which can only be used inside a loop. It is true when the value of *expr* has not changed since the last time through the loop.
 - Conditionals can be connected using the boolean connectives “or”, “and”, and “xor”.
- Commands may be combined into a single unit by using the “`block`” command. A block is terminated with “`end.`” For example, the code

```
if x < 1 then block
    set a to 2.
    set b to 3.
end.
```

will set `a` to 2 and `b` to 3 when `x` has magnitude less than 1. You can also use “`:`” as a synonym for “`block`”, so the following code is equivalent to the last example:

```
if x < 1 then:
    set a to 2.
    set b to 3.
end.
```

Note that “`block ... end.`” is treated as a single command, so to use in conjunction with “`if ... else ...`” you should write

```
if x < 1 then:
    set a to 2.
    set b to 3.
end.
else:
```

```
    set a to -2.
    set b to 8.
end.
```

1.2 Loops

Since `FractalStream` is designed for studying the properties of a map under iteration, most scripts will be centered around a loop of some sort. Loops in `FractalStream` can be made with the following commands:

- “do *command1*. *command2*. ... *commandN*. until *cond*.”
This creates a simple loop which executes the commands until the condition *cond* is met or the loop reaches `maxIter` iterations. The parameter `maxIter` can be adjusted in the Settings window of the viewer. Note that these loops are guaranteed to exit after some number of iterations, so a script cannot get stuck in an infinite loop.

`FractalStream` scripts maintain a hidden internal variable `iters`. Whenever a do loop exits, `iters` is updated to contain the number of successful iterations performed by the loop before exiting.

- “iterate *expr* on *var* until *cond*.”
The “iterate” command is a shorthand for the loop

```
do
    set var to expr.
until cond.
```

If you omit “on *var*”, the variable is taken to be `z`. Thus, the statement “iterate $z^2 + c$ until `z` escapes.” is equivalent to

```
do
    set z to  $z^2 + c$ .
until z escapes.
```

- “repeat *n* times *command*.”
Simply repeats *command* a fixed number of times. *n* must be a constant

positive integer. By taking *command* to be “`block ... end.`” or “`: ... end.`” you can repeat a block of commands some number of times.

2 The Structure of a Script

Scripts describe one of two different objects: either a dynamical system, or a parameter plane for a 1D (complex) or 2D (real) dynamical system. The archetypical example of a dynamical system is the Julia set for the complex map $f_C(z) = z^2 + C$, while the parameter planes describe objects such as the Mandelbrot set.

We begin by treating dynamical systems.

2.1 Programming a Dynamical System

When `FractalStream` is drawing a dynamical system, the viewer will run your script once for each point which needs to be drawn. To communicate the location of the current point in dynamical space to your script, `FractalStream` uses the complex variable `z`.

Let us consider the following simple script:

```
iterate z^2 - 1 until z escapes.
```

This script will run once for each point in the viewer, taking the corresponding `z` coordinate and iterating it under the map $f(z) = z^2 - 1$ until the iterates tend towards infinity.

2.2 The Coloring Method

Once your script has run for a given value of `z`, it will report several things back to `FractalStream`:

1. The hidden variable `iters`, which will hold the number of iterations through the last, outermost loop of the script.
2. The hidden variable `flag`, which will be described later.
3. The final value of `z`. This can be changed to any other variable using the command “`report var.`”. Note that this command does not stop

the script; it just changes which variable gets reported at the end. See the sections on Probes and on Autocoloring for places where this value is used.

This data is used to color the corresponding point in the viewer. Each color scheme in `FractalStream` is a gradient. The viewer will use `iters` as an index into this gradient in order to choose which color to use. Thus, points which take a slightly different number of iterations through the final loop of your script will get colored with slightly different colors.

If `iters = maxIters`, then the final loop of your script must have exited because it took too many iterations, not because its exit condition was met. In this case, `FractalStream` will color the corresponding point black.

Consider what these behaviors mean for a simple one-line script like “iterate $z^2 - 1$ until z escapes.”.

- If a point fails to escape within `maxIters` iterations, it will be colored black. In particular, points which are attracted to a periodic orbit will be colored black. These points give an approximation to the filled Julia set.
- If a point does escape to “infinity” before `maxIters` iterations, then it will be given a color. This color will vary based on how fast the point escaped, yielding an approximation of the Green’s function for the exterior of the Julia set.

You cannot set the value of `iters` directly, but you can affect it by using the command “`succeed.`” which sets `iters` to 0 and “`fail.`” which sets `iters` to `maxIters`.

2.3 Using Flags

Often, you will want to color a point differently depending on some condition. As a simple example, let us write a script which draws the unit disc.

We can achieve this by using the hidden variable `flag`. This variable is string-valued, and is initially set to `Default Exit Condition`. The hidden variable `flag` is changed by placing square brackets around a name, such as “[`Disc`].” In this case, `flag` is set to `Disc`. After your script has operated on a point, the `flag` variable is reported back to `FractalStream`, where it is used to select a color scheme for the corresponding point.

So to draw a disc, we could use the script “`if z < 1 then [Disc].`”. If you compile this script, you should see the plane colored in one color, with the unit disc colored in a second. Going to the Settings window under Colors, the pull-down button will now list both `Default Exit Condition` and `Disc`. Here, you can edit the corresponding color schemes.

We can use this idea to draw a checkerboard around our Julia set. The outgoing lines of this checkerboard give an approximation of the external rays (gradient lines of the Green’s function) for the exterior of the Julia set. The script

```
iterate z^2 - 1 until z escapes.  
if im(z) > 0 then [Checkedred].
```

will draw a Julia set as before, but with checkers on the exterior. Notice that the shades of the checkers vary — this is because the hidden variable `iters` is changing depending on how long it takes `z` to escape. The flag selects the color gradient, while `iters` is used to choose an offset into the gradient.

You can also provide hints to `FractalStream` about what color to use for each flag. To do this, use the command “`[flagname | color]`”. For example,

```
iterate z^2 - 1 until z escapes.  
if im(z) > 0 then [Checkedred | red].
```

will draw the checkedred Julia set as before, and will initially set the color of the checkers to red.

2.4 Drawing Parameter Space

The other common type of script describes a parameter space for some family of dynamical systems. `FractalStream` reserves the variable `c` to represent a parameter. If a script uses the variable `c`, then `FractalStream` will create a picture of the parameter plane (the `c` plane) rather than the dynamical plane (the `z` plane).

Drawing a parameter plane works much like drawing a dynamical plane. The main differences are:

1. Points in the viewer correspond to different values of `c`, so `c` is initialized

to the current point.

2. `z` is always initialized to 0.

Coloring works exactly the same. Thus, the script “`iterate z^2 + c until z escapes.`” will draw a parameter space picture for the family $f_C(z) = z^2 + C$. A point C is given a color based on how fast 0 escapes under f_C or is colored black if 0 does not escape, resulting in the Mandelbrot set.

Note that this worked because $z = 0$ is a critical point for each f_C . For the family $g_C(z) = z^2 + z + C$ you would want a script like

```
set z to -1/2.  
iterate z^2 + z + c until z escapes.
```

Of course, in general the critical point of f_C is a function of C . For a family like $h_C(z) = z^2 + Cz + 1$ you should write

```
set z to -c/2.  
iterate z^2 + c*z + 1 until z escapes.
```

2.5 Moving from Parameter Space to Dynamical Space

When viewing a parameter plane in `FractalStream`, you can investigate the dynamics at a particular point by switching to the `Dynamics` tool and clicking on the parameter value of interest¹. The program will now draw the dynamical plane corresponding to your chosen `c` value.

This works fine for scripts such as “`iterate z^2 + c until z escapes.`” since the same script makes sense for both parameter space and dynamical space. But for the other two examples in the previous section, there is a problem — in the first line, we overwrite whatever `z` value is set by `FractalStream` when drawing dynamical space.

To resolve this problem there are two special commands:

1. `par command`.
Only execute `command` when drawing a parameter plane.

¹The same result is obtained by right-clicking when in the `Zoom` tool.

2. `dyn` command.

Only execute *command* when drawing a dynamical plane.

Using these commands, we can correct the last example to

```
par set z to -c/2.  
iterate z^2 + c*z + 1 until z escapes.
```

The use of `par` ensures that the value of `z` is only modified when drawing parameter space. When drawing dynamical space this line is ignored, leading to the correct picture.

Of course, the *command* option for `par` or `dyn` can be “`block ... end.`” or “`: ... end.`” to execute a whole block of commands only in the given plane.

2.6 Real Dynamics

Although `FractalStream` was designed for complex dynamics, it also supports real dynamics. A script will use real dynamics if it starts with the line `{real}`

All variables are then assumed to be real. For real dynamics, `z` and `c` are no longer special variables. Instead, the real variables `x`, `y` are used for dynamics and `a`, `b` are used for parameters.

2.7 Pixel Size

In certain situations it is useful to know the current size of a pixel in the viewer. This can be accessed using the special variable `pixel`.

2.8 Autocoloring

If autocoloring is enabled then `FractalStream` uses the reported value (usually `z`) to decide on which color to use when coloring a point. The idea is that autocoloring should be used when `z` is being attracted to a set of fixed points. Then the reported value of `z` can be used to determine which fixed point `z` was attracted to. The autocoloring routine then sets up a color scheme for each fixed point. Many of the scripts in the library under Newton’s Method are well-suited for autocoloring.

3 Interaction with Viewer

A `FractalStream` script does not have to be a static entity — there are several ways in which a script may interact with the viewer.

3.1 Editable Parameters

The simplest way is through editable parameters. As mentioned in the first section, the command “`default var to expr.`” creates a variable *var* which then appears in the Parameters tab of the Settings menu.

3.2 Probes

Probes are a way for scripts to create small custom tools which can report back a single piece of data to the viewer. As a simple example, let us modify the script which drew the unit disc so that we can ask it what the magnitude of a given point is. To do this, we will use the `probe` command:

```
if z < 1 then [Disc].
probe real "Magnitude":
  set r to sqrt(z*bar(z)).
  report r.
end.
```

After compiling this script, you will find a tool named `Magnitude` has been added to the tools. Selecting this tool will open a window. Clicking in the view will show you the magnitude of the corresponding point.

In general, a probe operates by displaying the reported value when a point is clicked. The format is controlled by writing “`probe format ...`” where *format* is one of `complex`, `real`, `integer`, or `rational`. If *format* is omitted, the probe is assumed to be complex. For `rational` probes, the numerator and denominator are taken to be the real and imaginary parts of the returned value.

For an interesting example of a probe, let us consider the following script:

```
probe integer "Cycle Length":
  repeat 500 times set z to z^2 + c.
  set count to 0.
```

```

set savedZ to z.
do
  set count to count + 1.
  set z to z^2 + c.
until z = savedZ.
report count.
end.
iterate z^2 + c until z escapes.

```

This script simply draws a Mandelbrot set, but also makes a tool which computes the length of the attracting cycle for Julia sets. Try using the `Cycle Length` tool and clicking in various bulbs of the Mandelbrot.

3.3 Special Tools

For very complex or specialized scripts, it is possible to create and load additional tools which extend `FractalStream`'s capabilities. These additional tools should be placed inside the `FractalStream` application bundle, inside `/Contents/PlugIns/Special Tools/`. Once a tool is in place, a script can request to load it through the command “`load toolname.`” This will attempt to load the tool “`/Contents/PlugIns/Special Tools/toolname.fstool`”.

3.4 Data Sources

The most complex script interaction is iteration over a data source. A data source is an external program, module, or tool which provides a list of values to your script. You can think of a data source as a function Δ which takes a complex number and returns a list of complex numbers.

The motivating example is the `SaddleDrop` script. In `SaddleDrop`, a special tool (also called `SaddleDrop`) performs computations to determine the location of several saddles in the dynamical plane for each value of `c`. The script then uses these values to color parameter space intelligently.

Data sources are accessed using the command “`using each var in datasource at expr command.`” where `var` is a variable name, `datasource` is the name of the data source, `expr` is the point for which you are requesting data, and `command` is the command to execute.

This command obtains a list $L = \Delta(expr)$ of complex numbers from the data source. *var* is set to the first element of L and *command* is executed. Then *var* is set to the second element of L , *command* is executed again, and so forth. Once all of L has been processed, the data source sets the hidden variables `iters` and `flags` and decides on a final value to assign to *var*.

This mechanism allows for arbitrary data to be piped into the scripts, while probes can be used to get data out. `SaddleDrop` uses both of these tricks — the location of the saddles is managed by a special tool which also provides the saddle locations as a data source, while the script helps the special tool locate saddles by implementing one step of Newton's method in a probe.