An open source program for studying iterated dynamical systems

Guilherme de Azevedo Silveira, Eduardo Colli

Instituto de Matemática e Estatística – Universidade de São Paulo (USP)
R. do Matão, 1010 – Cidade Universitária – CEP 05508-090 – São Paulo – SP – Brazil

gas@linux.ime.usp.br, colli@ime.usp.br

Abstract. *This paper describes the process of brainstorming, developing and using an open source software in order to help mathematics students dealing with iterated systems which often appears while they go through their academic life. The software also aims at helping researchs which are based on watching dynamical system's iterations and attractors basins, giving the feeling of what is going on. The main idea is that the math student, teacher or researcher should not need to learn advanced topics of a programming language in order to simulate what he needs, so he can focus on his mathematical analysis and skills.*

1. The need for a tool

As we grow up learning math at school we face the joy of dealing with loads of numbers and their pretty properties but as soon as we start to study the real thing in a university we face our first problem.

Although we love to write complex formulas most of us have not that much fun dealing with computer programming, which has proved to be useful and sometimes even essential for some researchs. And that does not only happens with students as there are some teachers and researchers who have exactly the same fear of facing some code.

The classes could be funier and more dynamic, the students would be able to analyse intricate dynamical systems and researchers would not have to worry to learn the newest trend on computer programming in order to get what they need: whether it is for researching, teaching, simulating or studying, there is a need for an open source tool that should be capable of taking the 'computer code' out of their way.

2. Open source community

In the last few years the open source community has proven to help developing systems which can be used by everybody. Most of all, it allows students who would not be able to achieve a bigger goal and compete with other non-free softwares without the help of others, to join their strength in an effort to accomplish this big project.

With all this information in mind, there are a few languages which are capable of handling such needs in a simple way, but Java is a special one which gives platform and operational system independence, appart from having big open source communities as java.net and arca.ime.usp.br.

3. Simulating dynamical systems

3.1. The problem

A student who is trying to see what happens with that starting point in a dynamical system would first have to learn how to code his equations in a specific programming language, how to deal with on-screen graphics, perhaps double buffering for better performance and other problems that were tackled with this software.

Another example is that the teacher has just showed the students what is an attractor, how it works, and simulated its iteration values in the blackboard. It might be crystal clear for the teacher but not that clear for the students, so one might simulate its values and draw it on the computer screen.

This will help them understanding what is going on as it is easier to understand what they see rather than an abstract formula.

3.2. The solution

The software called Iterador makes it easier by allowing the user to simply provide the equations and set his scales in order to see what happens after **n** iterations.

For instance, we can simulate Hénon's attractor (with a = 1.4 and b = 0.3) by setting x1 to **1 – 1.4 * x1 * x1 – 0.3 * x2** and x2 to **x1**, meaning that after each iteration, the actual value of x1 and x2 is used to compute their new values. By configuring the scale to **-1.5 < x1 < 1.5** and **-1.5 < x2 < 1.5** we will be able to see the famous attractor as shown in Figure 1.

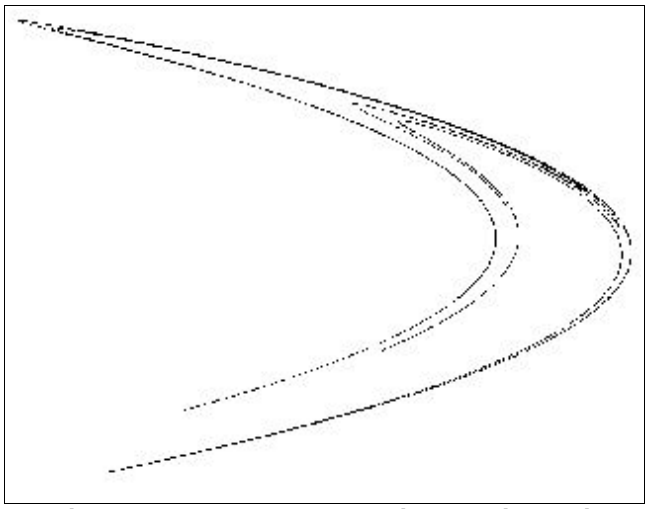

Figure 1. Henon's attractor (a = 1.4, b = 0.3)

Therefore, the only knowledge the user needs is how the Hénon map works and enter its data.

3.3. Behind the scene

As the user sets his equations, the software compiles it using Janino, a Java real time compiler, and when the first asks for redrawing, the program will iterate over the points using the compiled data and compute its new values: this is completely transparent for the user.

Simple equations can be solved in two different ways: either by parsing its data and implementing a parser or by compiling it into Java code and running it. Due to the nature of the program, which runs those equations more than one thousand times, it is much better to make use of Java's Just In Time Compiler options to make it faster.

One might argue that the software will not be able to handle more complex algorithms, i.e. Recursion, but it is completely supported through Java code as more advanced users can write the following factorial method using recursion:

```
public class SlowFactorial {
        public static long factorial(long i) {
                if(i==0) return 1;
                else return  i * factorial(i-1);
        }
}
```

Summing up, one can either make use of a simple graphical interface and write his simple equation in order to implement a solution to a problem or write his own customized code and use it when the software does not cope with one's needs. Moreover, due to its internal architecture, it allows the user to use any Java library already available, like Jakarta's Math library which contains a mathematics and statistics classes and methods.

4. Other features and bifurcations diagrams

The software can also run any piece of code during its iterations. As for example, one might change the color after each five hundred iterations, or set other values for the current position in order to simulate a change of initial conditions by writing something as follows:

```
if(iteration % 500 == 0) color.change(255,0,0);
else color.change(0,255,0);
```

Using this resource it is possible to watch a bifurcation diagram simulation by simply providing the intervals and after a fixed number of iterations changing the initial condition and the current parameters.

For example, lets say one has an **a** parameter and **x** being the configuration space: after the number of desired iterations, the user can change the current value of **a** and reset **x**, in order to go to the next column and going further in the drawing process, without the need of any complex Java code:

```
a = a + 0.1;
x = initial_x;
```

5. Basin of attraction

5.1. The problem

A few questions arrive from our system: how does a set of points reacts after iterations? To which attractor does an orbit go to? Was it a strange attractor or a periodic orbit? How many attractors do exist? How would one deal with infinite as an attractor?

Even if it is a free tool that is able to iterate over a function and display its data during its progress, it will be not enough to answer this kind of questions.

5.2. The solution

The software has a plugin architecture which allows other developers to implement solutions to questions like the ones mentioned before. For example, there is already a Basin of Attraction plugin which is able to answer those questions.

The user is actually able to use this plugin first by configuring his method as he would do it earlier without this new feature. At the following step, he can start the Average Simulator which picks random points on the screen and uses it to iterate, calculating the average of a pair of functions during the iteration process.

The pair of functions can be changed and adapted, so there is nothing completely automatic in the entire process which the user is not able to change. If he desires to use the default functionality, there is no need for more complex configurations.

By picking two different average functions, many pairs corresponding to random starting conditions are displayed in a canvas where the scale can be easily changed (during the process!) in order to display which area of visualization is required. The user can now select which area represents the averages of orbits going to an attractor, so he identifies it with a circle by using the mouse: the attractor is then recognized by the Basin of Attractor plugin and marked with a unique color.

The circle sets a criterium: if a starting condition leads to an average pair inside the marked circle then the computer concludes that the orbit has been captured by that attractor and assigns the chosen color to this starting condition. A particular larger circle that the user also draws and contains all others, stablishes a criterium to the “attractor at infinity”.

5.3. Behind the scene

Most of the coding done here is responsible for dealing with the user selection of areas of each attractor in the Average Simulator view. After that, it runs through all points in the screen, goes through their iterations and checks for each one of these points the attractor which is its final destination using the user-defined circle criterium.

Based on this information, and after a while, the program is capable of drawing beautiful fractals.

5.4. Practical example

The program was tested with Hénon by changing its parameters to a = 1.2, b = 0.2 and moving two steps at a time to simulate two different attractors.

By using the following two average functions, the software will build a map which will help the student to locate groups of points, the initial conditions, whose orbit will go through the same attractor.

$$Average_1 = \sum (|x_1| * |x_2|)$$

$$Average_2 = \sum (x_1 * x_1 + x_2 * x_2)$$

Figure 2 shows the result of the average picker for this system using two diferent average functions and the users selection for the circle criteria.

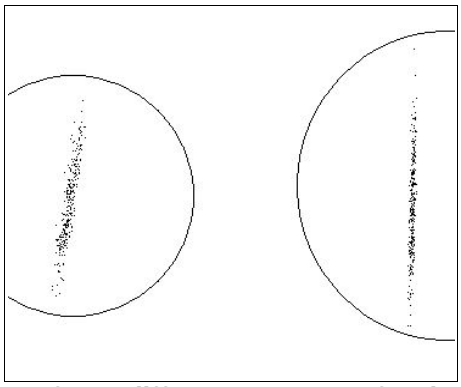

Figure 2. Average pairs of two different attractors for the twice iterated Henon map with a = 1.2 and b = 0.2.

Both attractors can be seen in figure 3 as some initial conditions will lead to an orbit in the left attractor and other conditions will lead to the other attractor.

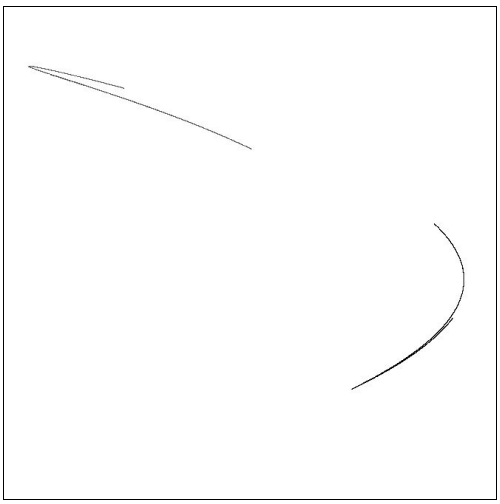

Figure 3. Two different attractors

5.5. Inciting the students curiosity

Now Figure 4 shows this system's basins of attraction: points which go to the infinity attractor are painted white, while the light and dark grey mean the first and second attractors.

But the user might easily find out that there are still some black points in this figure and ask what they mean. Those points (initial conditions) did not go to the infinity attractor nor to any of the two attractors seen before.

The student is then incited and ready to look out what happens with those initial conditions painted black and see for himself that there are two other small attractors.

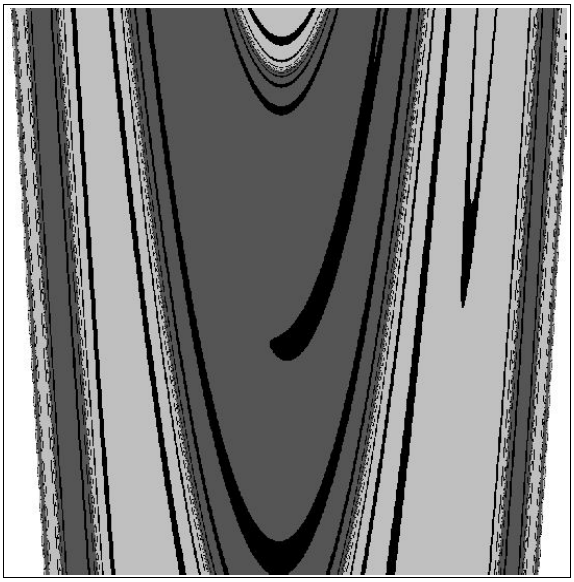

Figure 4. Basin of attraction

6. Integrating differential equations

Whenever a student has the job to integrate a differential equation he might simply use a software that will do its job and show its result, but that is not exactly something good for the student to learn what is going on.

It could be much more valuable for him to implement his preferred iteration method as to solve the equations by writing only his solution's kernel. After that it is possible to watch his method actually solving the problem, not only its result.

It is possible to create intermediate expressions which run for each iteration and using this technology one is capable of executing highly complex iterations, like the Runge-Kutta method for integrating differential equations using a step of a fixed size.

But if the result is not satisfactory or trustworthy, one might code a variation of the Runge-Kutta using different steps by creating those intermediate expressions which regulate the value of the step size.

Another example is to find a function root value (by Newton's Method, for example) or estimate a numerical value of an integral (by Simpson's Method, for example). The user has the option to make it by writing a simple piece of Java code in the graphical context of the program, without the need to learn how to compile it. More advanced users can always write full Java functions as shown in the SlowFactorial class earlier.

7. Plugins and flexibility

The plugin architecture can be used to create a graphical interface to common used programs, like the color option mentioned before.

As another, let's say one wants to export all generated points to a text file in order to analyse its generated data in another software. There is already such a plugin and its code is quite simple, it simply opens a file and prints each point to the file stream.

Using such information, a teacher might create a plugin for its most complex function and give it to the students so they just configure it in a graphical manner, instead of learning Java and coding it. If the emphasis is in the theory of dynamical systems, there are many plugins that should be created, containing algorithms usually applied to this kind of systems, such as the estimation of Liapunov exponents and fractal dimension of attractors and boundaries of their basins of attraction.

The program may also be used in order to do something that was not conceived by the authors, and the user still does not get into programming. For example, in a Mandelbrot-like set, where you explore the entire parameter's space and checks what happens with an initial fixed condition, one can actually paint the points one color if the initial condition goes to infinite or another color otherwise.

8. Conclusion

There is a real need in universities for tools which make it easier for both students while learning and teachers while teaching and this software is able to help both of them in many Mathematical fields and applications and aims at allowing but not oblying them to learn a programming language to solve their problems.

Hopefully, as time goes by, most teachers will use tools like this one to make it easier for students to learn. If our teachers did not think this way before us, we would still be writing on rocks.

9. References

Hénon, M. (1976) “A two-dimensional map with a strange attractor”, Comm. Math. Phys. 50, p. 69-77.

Suganuma, T., Ogasawara, T., Takeuchi, M., Yasue, T., Kawahito, M., Ishizaki, K., Komatsu, H. and Nakatani, T. (2000) “Overview of the IBM Java Just-In-Time Compiler”, IBM Systems Journal Volume 39, Number 1, http://www.research.ibm.com/journal/sj/391/suganuma.html

Jakarta Math Library, http://jakarta.apache.org/commons/math/

Unkrig, A., Janino, http://www.janino.net/

Silveira G, Iterador, http://www.linux.ime.usp.br/~gas/iniciacao/iterador.html