SIMULAÇÃO NUMÉRICA DE SISTEMAS DINÂMICOS

Guilherme de Azevedo Silveira
Instituto de Matemática e Estatística - Universidade de São Paulo
R. do Matão, 1010 - Cidade Universitária - CEP 05508-090 - São Paulo
gas@linux.ime.usp.br
Eduardo Colli[1]
Instituto de Matemática e Estatística - Universidade de São Paulo
R. do Matão, 1010 - Cidade Universitária - CEP 05508-090 - São Paulo
colli@ime.usp.br

Resumo

Este artigo descreve o processo de idealização e realização de um software para uso de alunos na área de matemática aplicada com ênfase em sistemas iterados. Também será tratada a idéia de ajudar pesquisas que estudam bacias de atração. O mais importante é que o estudante de matemática, professor ou pesquisador não deve ser obrigado a aprender detalhes de uma linguagem de programação para executar simulações numéricas clássicas podendo, então, focar em suas habilidades matemáticas. Palavras-chave: sistemas dinâmicos, java e matemática, simulação numérica.

Resumo

This paper describes the process of brainstorming, developing and using an open source software in order to help mathematics students dealing with iterated systems. It also aims at helping researchs which are based on watching those 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 make some numeric simulations, so he can focus on his mathematical analysis and skills. Keywords: dynamical systems, java and mathematics, numerical simulations.

[1]Parcialmente financiado pela FAPESP.

1 Introdução

À medida que crescemos e aprendemos matemática na escola ficamos cada vez mais encantados com a mágica de analisar equações complexas e de vivenciar as mais belas propriedades numéricas. Tudo isso muda uma vez que nos tornamos intelectualmente mais adultos e entramos na faculdade, onde encontramos algumas dificuldades.

A maior parte dos alunos não lida tão bem com programação, ferramenta que já se provou extramente útil em diversas áreas da matemática aplicada. Mais grave ainda, isso não acontece somente com alunos mas também com diversos professores e pesquisadores que têm o mesmo receio de programar.

A sala de aula seria um ambiente mais descontraído e vivo se os alunos fossem capazes de focar em suas técnicas matemáticas em vez de ter que aprender os detalhes de shift de bits em C ou qualquer outra linguagem de programação: não importa se na pesquisa ou na aula, muitas vezes o foco principal é na parte matemática mas os alunos e professores precisam gastar muito tempo extra para aprender uma linguagem que talvez não seja o foco do mesmo naquele momento.

Dessas complicações nasce a necessidade de uma ferramenta que torne opcional o conhecimento aprofundado de alguma linguagem de programação, sendo que ela ainda seria capaz de simular alguns sistemas durante a aula, deixando a mesma mais interessante e os exemplos mais práticos, possivelmente diminuindo o número de alunos desinteressados.

O Iterador (Pulga) [1, 2] é um projeto desenvolvido em Java com o foco de tornar possível tudo que foi mencionado até esse instante.

2 Simulação de sistemas dinâmicos

2.1 O problema

Um exemplo prático de uso desse software escrito em Java é a simulação numérica de sistemas dinâmicos contínuos ou discretos através de resultados gráficos.

Um estudante que deseja ver o que acontece para determinada condição inicial do sistema com determinados parâmetros pode fazê-lo facilmente sem se preocupar com detalhes de como um gráfico deve ser mostrado em uma

tela, como por exemplo técnicas de double buffering etc.

Outro exemplo é o professor enriquecer uma aula mostrando para alunos não somente na lousa os resultados númericos de uma iteração mas também graficamente a órbita percorrida, facilitando aos alunos "digerir" tudo aquilo que fora explicado na aula.

2.2 A solução

Com a simples entrada da fórmula de iteração e a escolha do espaço a ser plotado na tela podemos simular a iteração de um sistema.

Por exemplo, começamos com o atrator de Hénon [3] (com $a = 1.4$ e $b = 0.3$) e configurando x1 para **1 − 1.4 * x1 * x1 − 0.3 * x2** e x2 para **x1**. Escolhendo o espaço **-1.5 < x1 < 1.5** e **-1.5 < x2 < 1.5** temos o resultado da Figura 1.

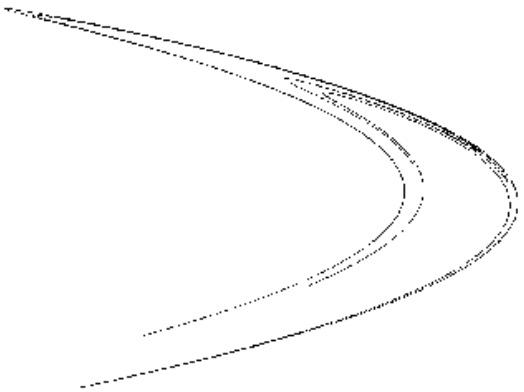

Figura 1: Atrator de Hénon ($a = 1.4$, $b = 0.3$)

Portanto o único conhecimento que se faz necessário é o de saber a equação do mapa. Uma vez que o programa compila todas as equações em código Java, ele permite que o usuário avançado aprenda a linguagem e evolua para sistemas mais complexos.

Resumindo, existem dois caminhos: usar os recursos básicos simulando iterações sem precisar conhecer uma linguagem ou se aprofundar e obter resultados avançados.

3 Diagramas de bifurcação

É possível simular um diagrama de bifurcação utilizando uma funcionalidade bem simples do Iterador: basta configurar o tamanho do intervalo após o qual a condição inicial e os parâmetros devem ser alterados.

Por exemplo, supondo a existência de um parâmetro a e x sendo o espaço de configuração: após o número desejado de iterações para um determinado valor de a, este último deve ser incrementado e x deve voltar ao valor inicial para ir à próxima coluna e continuar com o diagrama, sem nenhuma complexidade extra:

$a = a + 0.1;$
$x = x_{inicial};$

Um exemplo prático é a família unidimensional cuja seqüência de iterados apresenta comportamento semelhante ao da seqüência de intervalos entre bolhas [4], quando uma mangueira de ar é injetada na base de um líqüido viscoso. O parâmetro "horizontal"(fi) corresponde à mudança da vazão do ar. Mudar os valores de l corresponde a mudar o comprimento da mangueira que injeta o ar (por incrível que pareça esse parâmetro afeta bastante o resultado).

No cálculo de x_{k+1} em função de x_k aparece uma função cúbica que depende deste último. É necessário calcular a maior raiz desta função, portanto foi criada uma *expressão intermediária* (t_0) que serve para escolher uma condição inicial para o método de Newton, e a outra ("raiz") acha a raíz propriamente dita iterando algumas vezes esse método.

A seguir, o exemplo do método newton feito de maneira programática na Figura 2 encontra-se o resultado desse diagrama.

raiz = t_0;
for (int i = 0; i < 20; i++)
    raiz = (2*m*raiz*raiz*raiz - x1)/(3*m*raiz*raiz – l);

4 Plugins e flexibilidade

A arquitetura foi montada com a idéia de utilizar a tecnologia Java chamada Reflection, que permite a adição de diversas extensões ao programa.

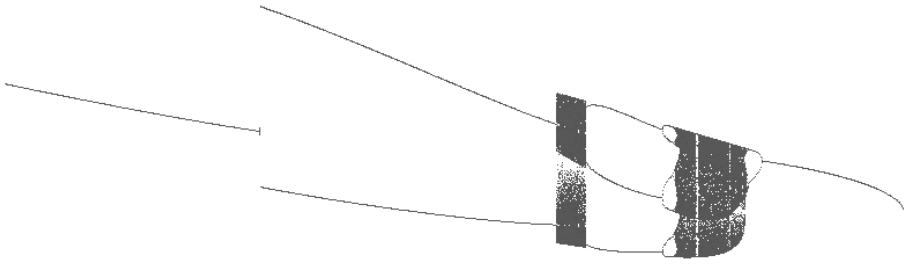

Figura 2: Diagrama de bifurcação do sistema das bolhas

Por exemplo, um professor pode criar um plugin onde é possível configurar padrões de exportação e disponibilizar tais plugins para os alunos que, por sua vez, ficam com duas opções: escrever o código java eles mesmos ou utilizar algo que o professor criou.

O programa também pode ser usado de certas maneiras que não foram pensadas anteriormente pelos autores, ainda assim sem ter de programar. Por exemplo, é possível desenhar um conjunto de Mandelbrot, onde um espaço de parâmetros é explorado para uma condição inicial fixa e pintado de acordo com sua órbita. O truque é "fingir" que os parâmetros são variáveis e plotar o conjunto de Mandelbrot como se fosse uma bacia de atração.

Um plugin simples que facilita o uso do programa é o de condição inicial, que permite selecionar diversos pontos que serão utilizados para analisar suas órbitas em uma única imagem. Na Figura 3 podemos ver o resultado desse plugin para diversas condições iniciais em um mapa citado por Ragazzo-Zanata:

5 Bacia de atração

5.1 O problema

Diversas questões podem surgir sobre um sistema: como uma condição inicial reage após n iterações? Para qual atrator a órbita de uma determinada condição inicial converge? Será que é para uma órbita periódica? Quantos atratores existem? Como lidar com o infinito? Um simples processo de iteração não resolveria tais perguntas.

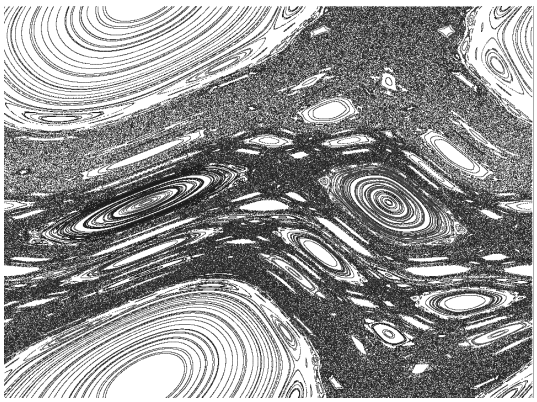

Figura 3: Ragazzo-Zanata

5.2 A solução

Com o plugin de bacia de atração, o usuário começa configurando o sistema como de costume. Em um segundo passo é iniciado o Average Picker, que escolhe aleatoriamente condições iniciais contidas no retângulo definido previamente e itera por eles. Usando um par de funções customizáveis, ele calcula médias dos pontos que compõem a órbita a partir da condição inicial.

Os resultados dessas médias são mostrados em um canvas que representa o espaço das médias, que também é configurável (inclusive durante o processo). A partir desse momento o usuário pode selecionar, através de polígonos, áreas que chamamos de *nuvens* e identificam diferentes atratores marcados com cores únicas.

Se a iteração de um ponto inicial tem como resultado uma média dentro de uma *nuvem*, o programa conclui que tal órbita foi capturada pelo atrator e pinta esse ponto com a cor pré-determinada. Um polígono que contém todos os outros pode ser desenhado e define um "atrator no infinito".

5.3 O desafio

O plugin demora muito para mostrar o resultado final devido ao número de iterações que são feitas para analisar a convergência da órbita de um ponto. Sendo assim foi utilizado um algoritmo iterativo capaz de mostrar rapidamente resultados parciais da análise para que o usuário tenha a opção de abortar o processo.

5.4 Exemplo prático

O programa foi testado com o mapa de Hénon, parâmetros $a = 1.2$, $b = 0.2$, e executando duas iterações por vez para que atratores de período par se dividissem em dois atratores com metade do período (aumentando assim o número de atratores). Configurando duas funções de média, o sistema desenha a bacia de atração, ajudando ao aluno identificar quais condições iniciais levam à convergência da órbita para quais atratores.

$Average_1 = \sum (|x_1| * |x_2|)$
*(código java: Math.abs(x1) * Math.abs(x2))*

$Average_2 = \sum (x_1 * x_1 + x_2 * x_2)$
*(código java: x1 * x1 + x2 * x2)*

A Figura 4 mostra o resultado das médias e o desenho de Hénon nesse caso.

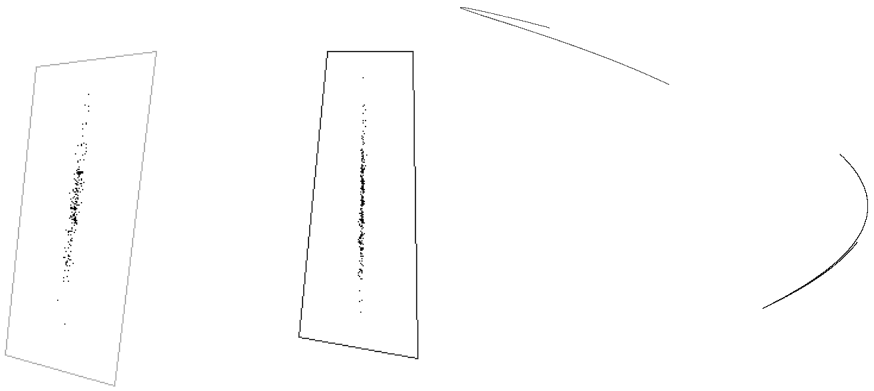

Figura 4: Médias de dois atratores e os atratores em seu espaço

Já a bacia de Hénon utilizando para $a = 1.2$ e $b = 0.2$ e definidas três cores: cinza escuro para uma nuvem, cinza claro para a outra, branco para o infinito e preto para algo desconhecido resulta na imagem Figura 5.

Portanto o próprio programa acaba incentivando o aluno a descobrir o que são esses pontos pretos, para onde vão as órbitas dos mesmos e o que eles representam.

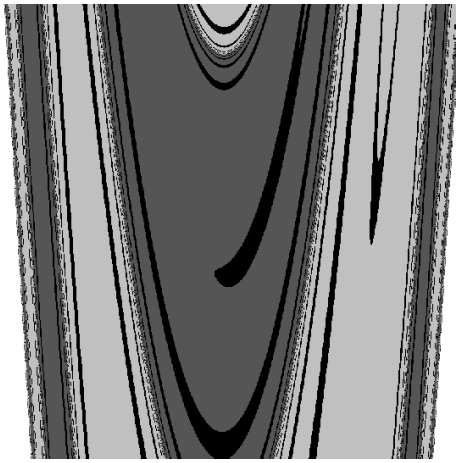

Figura 5: Bacia de Hénon ($a = 1.2$, $b = 0.2$)

5.5 Conjunto de Mandelbrot

Apesar de não ser uma das idéias principais desse programa, podemos utilizá-lo para gerar um conjunto de Mandelbrot utilizando o plugin da bacia de atração. A Figura 6 mostra o resultado da bacia de atração adaptada para tal necessidade.

Todo plugin implementa um método chamado getIterationCode que retorna qualquer código Java que será executado entre iterações. Sendo assim fica fácil criar um plugin capaz de parar as iterações quando x1 e x2 se tornarem infinito:

```
public String getIterationCode() {
return "if(x1==Double.POSITIVE_INFINITY x2==Double.POSITIVE_INFINITY)
return false;";
}
```

5.6 Tecnologias e padrões utilizados

Diversas tecnologias e padrões são utilizados para garantir a flexibilidade necessária para esse software.

A primeira idéia foi a de utilizar Java devido à natureza do programa: existe uma necessidade de executar um pedaço de código milhões de vezes

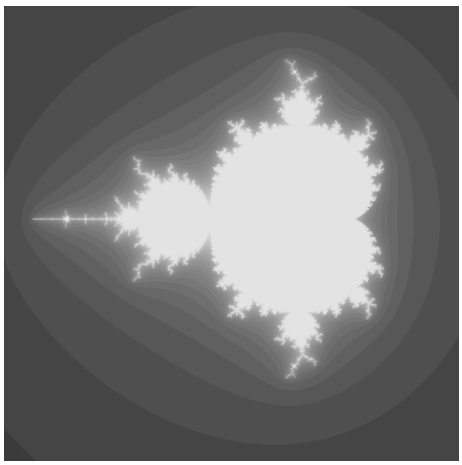

Figura 6: Conjunto de mandelbrot

e o just-in-time compiler [5] das máquinas virtuais (VM) que suportam a linguagem e tecnologia Java otimizam esse código.

Já os arquivos são guardados em formato xml através da Xstream, capaz de transformar objetos em xml através de um mapeamento de classes para tags e, em conjunto com Reflection, é possível armazenar nesses arquivos informações sobre a própria estrutura das classes que eram utilizadas como plugins ativos no momento de salvar o trabalho.

Ainda baseado em Reflection, mas usando Beanshell [6], foi criada uma estrutura extremamente dinâmica de controle de lógicas simples para os menus do sistema e com o uso da primeira tornou-se possível gerar diversas janelas através de arquivos de configuração simples uma vez que a maior parte das tecnologias disponíveis para criar janelas (como thinlet por exemplo) foram feitas para grandes sistemas comerciais, que não é exatamente o tipo de software de que estamos tratando.

Janino [7] foi utilizado para compilar as expressões escritas pelo usuário, dando suporte a diversas estruturas da linguagem Java em sua versão 1.4. Classes são criadas em tempo de execução e embutidas no sistema através de classloaders também gerados dinâmicamente pelo Janino que então passam a ser otimizados pela VM como explicado anteriormente, tirando proveito de todas as otimizações possíveis em tempo de execução: a grande vantagem de não compilar nosso código estaticamente e usar uma linguagem dinâmica.

Outra idéia interessante de grande valia para o usuário final é a do Big

Fat Jar, que permite criar um único arquivo .jar, que é o único arquivo necessário para a execução do programa, fazendo com que o usuário leigo na computação não sofra com as complicações típicas de um processo de instalação mais complexo.

Por fim, diversos design patterns [8] e boas práticas foram aplicados ao sistema, desde o Singleton para controle de instâncias únicas, Service Locator e Command para controle das lógicas de negócio, preferência a composição em vez de herança, Builder para criação de determinados objetos em diversos passos como no caso da criação de uma nuvem no plugin de bacia de atração e no momento de gerar a classe dinâmica.

6 Conclusão

Existe uma necessidade em universidades para ferramentas que permitam aos alunos focar no lado matemático da matéria sem obrigar, mas possibilitando aprender programação para resolver seus problemas.

Existe sempre a esperança de que professores se adaptem a esse mundo novo e utilizem tais ferramentas não só para manter os alunos acordados durante a aula mas também para incentivar o aprendizado e facilitar a compreensão de determinados assuntos.

Referências

[1] Silveira, G. & Colli, E., An open source program for studying iterated dynamical systems, *Anais do VI Workshop de Software Livre*, 108-190 (2005).

[2] Silveira, G., Iterador, http://www.linux.ime.usp.br/~gas/iterador

[3] Hénon, M., A two-dimensional map with a strange attractor, *Comm. Math. Phys.* **50**, 69-77 (1976).

[4] Colli, E., Piassi, V., Tufaile A. & Sartorelli, J. C., Bistability in bubble formation, *Physical Review E*, **70** (2004).

[5] Suganuma, T., Ogasawara, T., Takeuchi, M., Yasue, T., Kawahito, M., Ishizaki, K., Komatsu, H. et al, Overview of

the IBM Java Just-In-Time Compiler, *IBM Systems Journal* **39, 1**, http://www.research.ibm.com/journal/sj/391/suganuma (2000).

[6] Niemeyer P., Beanshell, http://www.beanshell.org

[7] Unkrig, A., Janino, http://www.janino.net/

[8] Gamma, E., Helm, R., Johnson, R. & Vlissides J., Design Patterns, Elements of Reusable Object-Oriented Software, Addison-Wesley (1994).