Departamento de Ciência da Computação, IME/USP

MAC323 Algoritmos e estruturas de dados II

BCC Primeiro Semestre de 2015

Primeiro Exercício-Programa (EP1)[1] Entregue até 21/4/2015, pelo Paca

O Método de Newton em $\mathbb{C}$

0. Introdução

O objetivo desse EP é implementar o método de Newton para encontrar raízes complexas de equações e produzir figuras que ilustram o processo de convergência do método.

Já conhecemos o método de Newton para equações reais $f(x) = 0$. Começamos com um valor arbitrário $x_0 \in \mathbb{R}$ e iteramos

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})} \tag{1}$$

para todo $n \geq 1$. Este processo define uma seqüência de reais $x_0, x_1, x_2, \ldots$ que converge (em geral) para uma raiz de $f(x) = 0$. O fato crucial para este EP é que *o processo acima também funciona em* $\mathbb{C}$.

Exemplo. Suponha que queremos encontrar todas as raízes da equação

$$z^4 - 1 = 0. \tag{2}$$

(Naturalmente, sabemos que as raízes são ± 1 e $\pm i$, mas (2) é apenas um *exemplo.*) Suponha que definimos a seqüência $z_0, z_1, z_2, \ldots$, pondo

$$z_n = z_{n-1} - \frac{f(z_{n-1})}{f'(z_{n-1})} \tag{3}$$

para todo $n \geq 1$, com $z_0 = a$, b, c, d, e e, onde

$$a = 2 + i \qquad b = 1 + 3i \qquad c = -1 - i/2 \tag{4}$$

e

$$d = -2 + i \qquad e = -1 - 3i. \tag{5}$$

Note que obtemos assim 5 seqüências. Estas seqüências convergem para as raízes ± 1 e $\pm i$. Aliás, qual seqüência converge para qual raiz? (Você já estudou este exemplo no *Creative exercise* 3.2.15 de IntroCS (`http://introcs.cs.princeton.edu/java/32class/`).)

[1] Versão de 30 de março de 2015 (10:13am) [versão preliminar]

1. Seu programa

Seu programa deverá ser composto de várias classes. As classes descritas abaixo devem ser implementadas como especificadas. Classes adicionais que você ache úteis podem também ser implementadas.

1.1. **Manipulação de números complexos.** Seu sistema deve conter uma classe para manipular números complexos. Veja `Complex.java` em `http://introcs.cs.princeton.edu/java/32class/Complex.java.html`. Veja também os *Creative exercises* 3.2.44–46.

1.2. **Funções holomorfas.** Funções complexas que são diferenciáveis em alguma vizinhança de todo ponto de seu domínio são chamadas *funções holomorfas*. Em seu sistema, funções holomorfas devem ser implementadas de forma a implementar a interface `HolomorphicFunction`:

```
public interface HolomorphicFunction {
    public Complex eval(Complex z);
    public Complex diff(Complex z);
}
```

Seja `f` uma `HolomorphicFunction` e `z` um ponto no domínio de `f`. Então `f.eval(z)` deve ser o valor de `f` calculada no ponto `z` e `f.diff(z)` deve ser o valor da derivada de `f` calculada no ponto `z`.

1.3. **Método de Newton.** Você deve implementar a classe `Newton` que contém o método estático

```
public static Complex findRoot(HolomorphicFunction f, Complex z0, Counter N)
```

Este método recebe `f` e um valor inicial `z0` para o método de Newton. Ele também recebe um `Counter N` (veja Seção 1.2 (*Data Abstraction*) em Algs4th: `http://algs4.cs.princeton.edu/12oop/`). Este método deve devolver a (ou, mais precisamente, uma aproximação da) raiz de `f` para a qual o método de Newton converge quando começamos com `z0`. O valor final de `Counter N` dever ser o número de iterações executadas pelo método de Newton.

1.4. **Bacias de Newton.** Seja $f(z)$ uma função holomorfa. Você já viu que o método de Newton aplicado a $f(z)$ particiona o plano complexo em regiões, chamadas *bacias de Newton*. Você já viu um exemplo no *Creative exercise* 3.2.15 de IntroCS.

Seu sistema deve conter a classe `NewtonBasins`, que produz figuras como aquela no enunciado do *Creative exercise* 3.2.15. Sua classe `NewtonBasins` deve conter um método estático com a seguinte assinatura:

```
public static void draw(HolomorphicFunction f, int maxI,
                        double x, double y, double xsize, double ysize,
                        int M, int N)
```

Os parâmetros desse métodos são como segue. O usuário fornecerá uma `HolomorphicFunction f`. Os parâmetros `x`, `y`, `xsize`, `ysize` definem o ponto $x + yi$ em que deve ser centrada a figura a ser gerada e definem as dimensões da região do plano que deve aparecer na figura. A figura deve corresponder ao retângulo com canto inferior $(x - xsize/2) + (y - ysize/2)i$ e canto superior $(x + xsize/2) + (y + ysize/2)i$. A figura a ser gerada deve ter $M \times N$ pixels. Finalmente, o parâmetro `maxI`

deve ser usado da seguinte forma. A intensidade da cor associada a um ponto z deve corresponder ao número de iterações executadas ao se lançar o método de Newton a partir de z. Suponha que z converge para uma raiz r de f e que os pontos iniciais que convergem para r estão sendo coloridos de, digamos, vermelho. Os z para os quais o número de iterações é pequeno devem ser coloridos com um tom de vermelho escuro. Os z para os quais o número de iterações é grande devem ser coloridos com um tom de vermelho claro. Você deve dar intensidade máxima de vermelho a z se o número de iterações for maior ou igual a `maxI`.

2. Exemplos

2.1. **Raízes quartas da unidade.** Para reproduzir o *Creative exercise* 3.2.15 de IntroCS, você pode usar as seguintes classes.

```
public class FourthRoots {
    public static void main(String[] args) {
        int maxI = Integer.parseInt(args[0]);
        double xc = Double.parseDouble(args[1]);
        double yc = Double.parseDouble(args[2]);
        double xsize = Double.parseDouble(args[3]);
        double ysize = Double.parseDouble(args[4]);
        int M = Integer.parseInt(args[5]);
        int N = Integer.parseInt(args[6]);

        Complex[] r = new Complex[4];
        r[0] = new Complex( 1,  0); r[1] = new Complex(-1,  0);
        r[2] = new Complex( 0,  1); r[3] = new Complex( 0, -1);
        HolomorphicFunction f = new Poly(r);

        NewtonBasins.draw(f, maxI, xc, yc, xsize, ysize, M, N);
    }
}
```

A classe acima usa a seguinte classe, para a criação de polinômios com raízes dadas.

```
public class Poly implements HolomorphicFunction {
    private int d;
    private Complex[] r;

    public Poly(Complex[] r) {
        this.d = r.length;
        this.r = r;
    }

    public Complex eval(Complex x) {
        Complex p = new Complex(1.0, 0.0);
        for (int i = 0; i < d; i++)
```

```
            p = p.times(x.minus(r[i]));
        return p;
    }

    public Complex diff(Complex x) {
        Complex s = new Complex(0.0, 0.0);
        for (int i = 0; i < d; i++) {
            Complex p = new Complex(1.0, 0.0);
            for (int j = 0; j < d; j++)
                if (j != i)
                    p = p.times(x.minus(r[j]));
            s = s.plus(p);
        }
        return s;
    }
```

2.2. **Um exemplo com uma equação transcendental.** A equação aqui é $f(z) = \sin(2\pi z) - c$, onde $c \in \mathbb{C}$ é uma constante.

```
public class Sinfn implements HolomorphicFunction {
    private Complex c;

    public Sinfn(Complex c) { this.c = c; }

    // returns f(x) = sin(2\pi x) - c
    public Complex eval(Complex x) {
        return x.times(2*3.1415926535897932385).sin().minus(c);
    }

    // returns f'(x) = 2\pi cos(2\pi x)
    public Complex diff(Complex x) {
        return x.times(2*3.1415926535897932385).cos().times(2*3.1415926535897932385);
    }
```

Para desenhar as bacias de Newton, usamos a seguinte classe:

```
public class Sin {
    public static void main(String[] args) {
        int maxI = Integer.parseInt(args[0]);
        double xc = Double.parseDouble(args[1]);
        double yc = Double.parseDouble(args[2]);
        double xsize = Double.parseDouble(args[3]);
        double ysize = Double.parseDouble(args[4]);
        int M = Integer.parseInt(args[5]);
        int N = Integer.parseInt(args[6]);

        double a = StdIn.readDouble();
```

```
        double b = StdIn.readDouble();
        HolomorphicFunction f = new Sinfn(new Complex(a, b));

        NewtonBasins.draw(f, maxI, xc, yc, xsize, ysize, M, N);
    }
}
```

Você pode ver duas figuras geradas para a equação deste exemplo com $c = 2$ em `http://www.ime.usp.br/~yoshi/2015i/mac323/html/docs/EPs/EP1/`.

3. Bônus

Uma vez que seu sistema estiver funcionando, você pode considerar implementar versões com mais funcionalidades. No momento, este enunciado contém apenas uma sugestão; é possível que outras sugestões sejam adicionadas no futuro. Essas funcionalidades valerão nota extra de EP.

3.1. ***Zooming.*** Após gerar algumas imagens, você achará interessante poder fazer *zooming* em regiões mais interessantes da figura. O usuário deverá poder indicar a região na qual fazer o *zoom* usando o mouse: por exemplo, o usuário poderia clicar em dois pontos da figura e isso poderia ser interpretado como os cantos de um retângulo que deve compor a nova imagem a ser gerada (você terá de decidir quantos pixels comporão essa figura). Se você quiser fazer algo mais sofisticado, você poderia implementar algo parecido com como se tira *screenshots* (veja, por exemplo, `http://www.take-a-screenshot.org`). Para implementar esta interação com o usuário através do mouse, você deve usar as classes `StdDraw` ou `Draw` de Sedgewick e Wayne.

Para implementar *zooming*, você achará interessante implementar uma versão orientada a objetos de `NewtonBasins`. De fato, você achará interessante ter objetos que são compostos de uma figura e uma associação de raízes e cores; sem esta informação adicional de raízes encontradas e cores usadas, ao se fazer o *zooming* você poderá gerar imagens com cores diferentes, que seria indesejável.

4. Entrega

Você deve entregar seu EP no Paca. Você deve entregar as classes que compõe seu sistema e deve também entregar classes adicionais para gerar exemplos interessantes. Por exemplo, você poderia escrever uma classe semelhante à classe `FourthRoots`, mas para outros polinômios. Você poderia adicionar um novo construtor à classe `Poly`, para construir polinômios a partir de seus coeficientes. Você poderia também considerar outras equações transcendentais. Disponibilize via `http` várias figuras geradas pelo seu sistema, com detalhes sobre como seus programas foram executados para se obter aquelas saídas (essas saídas devem ser reprodutíveis).

Você deve entregar documentação suficiente para que os monitores não tenham dúvida de como funciona seu sistema.