INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Modelação e Simulação de Sistemas Naturais

TPC1

Daniel Santos 32078

08/10/17

Índice

Exercicios

- **Raiz cúbica**

Método iterativo para determinar a raíz cúbica de um número (positivo ou negativo):

Para o cálculo da raiz cúbica é utilizado o método de Newton-Raphson que consiste na seguinte equação:

$$X_{(n+1)} = X_n - \frac{f(x_n)}{f'(x_n)}$$

sendo $f(x_n)$ a equação que pretendemos calcular e $f'(x_n)$ a derivada desta finalmente $x(0)$ é uma aproximação inicial.

```java
public class CubicSquareRoot {
    double x0;
    double number;
    int max = 8;
    boolean negative = false;

    //initializes variables
    public CubicSquareRoot(double number) {
        if(number < 0) {
            negative = true;
        }
        this.number = Math.abs(number);
        x0 = interval(this.number);
    }

    //Calculates approximation value
    public double interval(double number) {
        int val = (int) Math.pow(number, 0.333);
        return (double)((val + val + 1) / 2);
    }

    //Calculates function
    public double fn(double x) {
        return Math.pow(x, 3)-number;
```

```java
    //Calculates derivative function
    public double derfn(double x) {
        return 3 * Math.pow(x, 2);
    }

    //Calculates next iteration of the Newton - Raphson method
    public double fnnext(double x) {
        return x - fn(x) / derfn(x);
    }

    //Calculates and prints cubic root result
    public void calcRoot() {
        for(int i = 0; i < 8; i++) {
            x0 = fnnext(x0);
        }
        if(!negative) {
            System.out.println("A raiz cúbica de " + number + " é: " + x0);
        }else {
            x0 = -1*x0;
            this.number = -1* this.number;
            System.out.println("A raiz cúbica de " + number + " é: " + x0);
        }
    }
}
```

- **Solitário Búlgaro N=15**

Determinar quantas iterações são necessárias para o solitário Búlgaro convergir para o ponto fixo para o caso N = 15 e com estado inicial(5,5,5)

```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;

public class BulgarianSolitaire {
    List<Integer> cards ;
    ArrayList<Object> ss = new ArrayList<>();
    List<Integer>  triangularsolution = new ArrayList<>();
    int iterations = 0;



    //Constructor
    public BulgarianSolitaire(List<Integer> a) {
        cards = new ArrayList<Integer>(a);
        Collections.sort(cards);
        triangularSolution();
        cycle();
    }
```

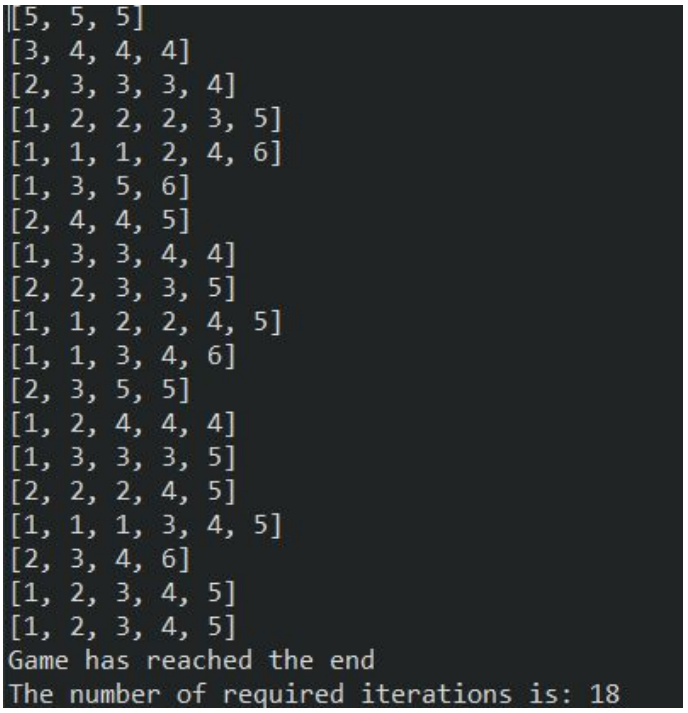
```java
//Solution if(N= total number of cards)
//the input is a triangular number
public List<Integer> triangularSolution() {
    triangularsolution.add(1);
    triangularsolution.add(2);
    triangularsolution.add(3);
    triangularsolution.add(4);
    triangularsolution.add(5);
    triangularsolution.add(6);
    triangularsolution.add(7);
    triangularsolution.add(8);
    triangularsolution.add(9);

    return triangularsolution;
}

//Creates a deep copy of a recieved list of integers
public List<Integer> deepCopy(List<Integer> aux) {
    List<Integer>newList = new ArrayList<Integer>();
    for(Integer p : aux) {
        newList.add(p);
    }
    return newList;
}
```

```java
//Verifies if the game has reached the ending conditions
public boolean gameOver(List<Integer> current) {

    for(int i = 0; i < ss.size();i++) {
        if(Arrays.asList(ss.get(i)).contains(triangularsolution) ||
                Arrays.asList(ss.get(i)).contains(current)) {
            System.out.println("Game has reached the end");
            return true;
        }
    }
    ss.add(deepCopy(cards));
    return false;
}
```

```java
//Cycles the game till end condition is reached
public void cycle() {
    int counter;

    while(!gameOver(cards)) {


        for(counter = 0; counter < cards.size(); counter++) {

            int value = cards.get(counter);

            cards.set(counter, value-1);
        }

        iterations++;
        removeZeros(cards);
        cards.add(counter);
        Collections.sort(cards);

        counter = 0;


        System.out.println(cards);
    }
    System.out.println("The number of required iterations is: " + iterations);
}
```

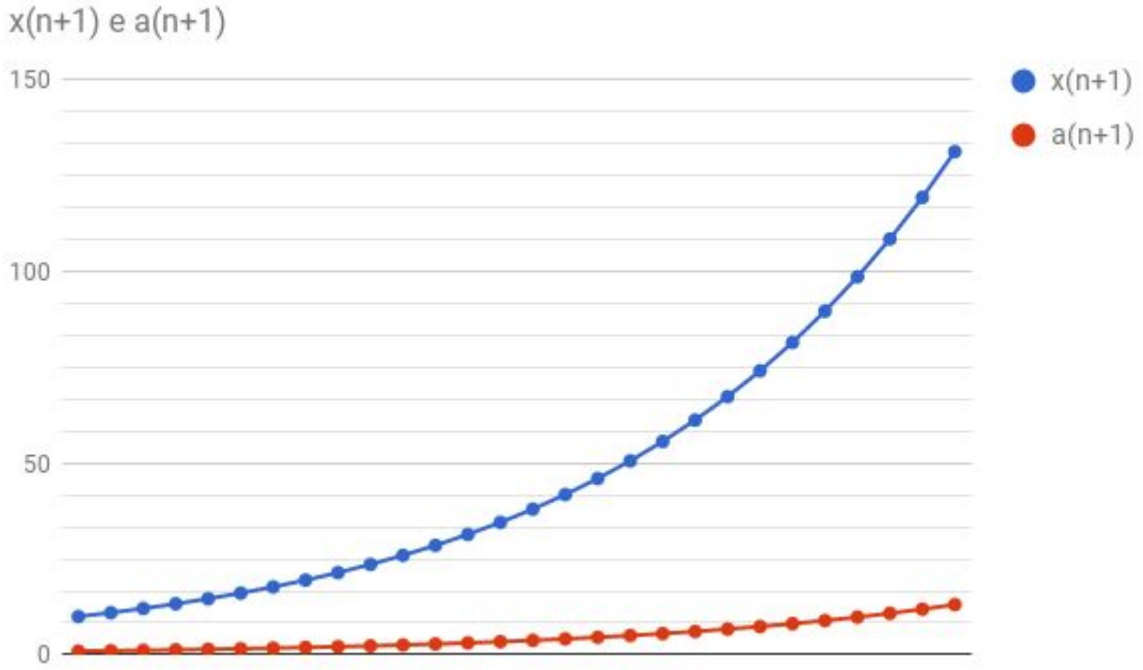
```java
    public void removeZeros(List<Integer> current) {

        while(current.get(0) == 0) {
            current.remove(0);
        }
    }


    public static void main(String[] args) {
        ArrayList<Integer> a = new ArrayList<>();
        a.add(5);
        a.add(5);
        a.add(5);

        new BulgarianSolitaire(a);
    }
```

Console output:



- **Sistema dinâmico:** $x(n+1) = x(n) + a(n),\ a(n) = 0.1x(n)$ **com** $x(0) = 10$

Gráfico:



Evolução do Sistema:
Tendo como ponto inicial x(0)=10 o sistema claramente tende para $+\infty$ pois é incrementado por ele mesmo e por a(n).

Ciclos Causais:

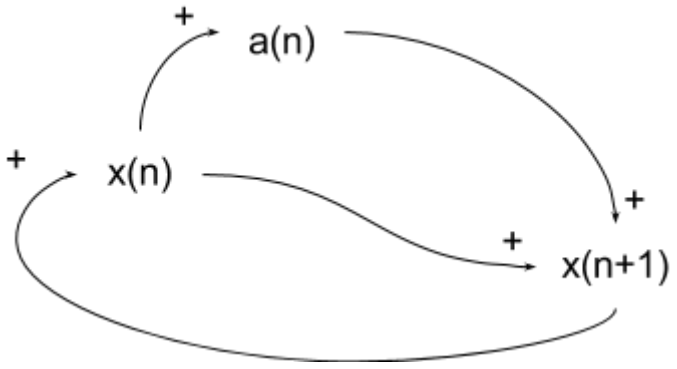
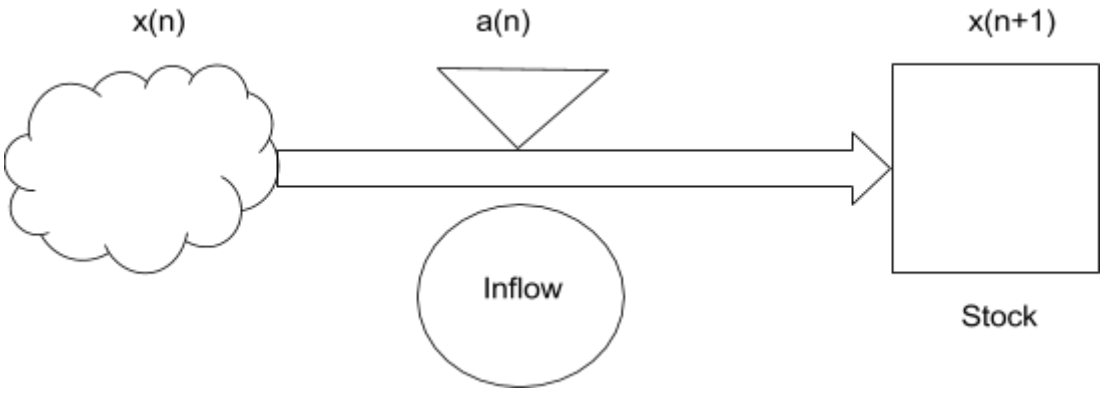
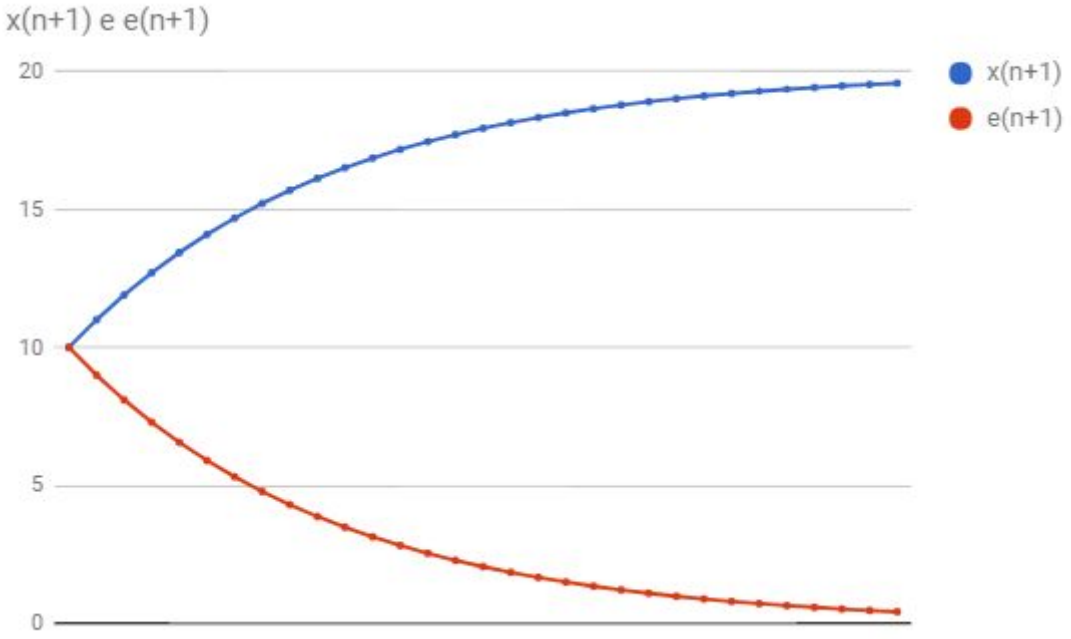


Diagrama de níveis e fluxos:



- **Sistema dinâmico** $x(n+1) = x(n) + ke(n),\ e(n) = T - x(n)$ **com** $x(0) = 10$ **,** $k = 0.1$ **e** $T = 20$

Gráfico:



Evolução do Sistema:

Tendo como ponto inicial x(0)=10 o gráfico possui em k um fator de multiplicação que faz com que o sistema cresça ainda que ao mesmo tempo devido à fórmula de e(n) em que é efectuada uma subtração T-x(n) faz com que o sistema tenha um crescimento que tende claramente para 20.