

SME0121 – Processos Estocásticos

Trabalho Prático

ALUNO : Ubiratan Soares (563242)

Data : 24/06/2011

Introdução

Conforme proposta do docente, este trabalho busca aplicar na prática os conceitos vistos ao longo do curso de Processos Estocásticos. O desenvolvimento foi realizado com a linguagem Java, usando a ferramenta de desenvolvimento Eclipse. O suporte às classes que implementam as distribuições de probabilidade foi obtido através do framework *Apache Commons Math*, uma vez que tais distribuições não possuem implementação nativa nessa linguagem. Maiores detalhes podem ser obtidos em :

<http://commons.apache.org/math/>

Nesse trabalho, foi requisitado que fosse simulado um sistema composto por componente sujeitos à falhas que opera ao longo de um ano. Os componentes tem tempo de vida que segue uma distribuição exponencial.

Assim, inicialmente foi implementada a classe **ComponentSimulation.java**, cujo código segue a seguir. Essa classe executa uma simulação única nas condições propostas, ou seja, com componentes falhando em média a cada $1/\lambda = 60$ horas, em um período de simulação de 8760 horas(1 ano).

```
// ComponentSimulation.java
package org.usp.stochastic;

import org.apache.commons.math.distribution.*;
import java.util.*;

public class ComponentSimulation{

    private ArrayList<Double> simulatedFailures = new ArrayList<Double>();
    private double mean = 60;
    private double hours = 8760.0;
    private double accumulated = 0.0;
```

```

ComponentSimulation() throws Exception{

    ExponentialDistributionImpl exp = new ExponentialDistributionImpl(mean);

    while(accumulated < hours){

        try{
            double rexp = exp.sample();
            simulatedFailures.add(rexp);
            accumulated += rexp;
        }
        catch(Exception e){
            System.out.println("ERROR : CANT GENERATE DISTRIBUTION!");
        }

    } //end while

} //end Constructor

public ArrayList<Double> getSimulation(){

    return this.simulatedFailures;

}

} // end ComponentSimulation

```

De posse dessa classe foram implementadas outras duas classes, que tomam a classe acima como base : **ComponentSimulationTest.java** e **ComponentSimulationTestBattery.java**.

A primeira executa uma simulação única e faz todos os cálculos associados, ou seja, automaticamente gera os resultados para os exercícios 2 e 3 desse trabalho. A segunda executa uma bateria de um número arbitrário de testes, a ser definido pelo usuário.

A intenção dessa implementação adicional é verificar de forma estatística o comportamento médio do sistema em questão, ainda que a mesma não seja utilizada nos exercícios aqui resolvidos. Uma execução da classe **ComponentSimulationTestBattery** é colocada na sequência para efeitos de constatação de que, por exemplo, os componentes não falharão mais do que 200 vezes em um ano (estimado empiricamente nos meus testes), por exemplo. Essa classe pode ser estendida para obter resultados mais confiáveis para os problemas propostos nesse trabalho.

```

// ComponentSimulationTestBattery.java
package org.usp.stochastic;

import java.util.*;

public class ComponentSimulationTestBattery {

    public static void main(String[] args) throws Exception{

        ArrayList<Double> meanResults = new ArrayList<Double>();

        int runs = 100;

        for(int i = 0; i < 200; i++) meanResults.add(0.0);
    }
}

```

```

    for(int i = 0; i < runs; i++){

        ComponentSimulation c = new ComponentSimulation();

        ArrayList<Double> results = c.getSimulation();

        for(int j = 0; j < results.size(); j++){
            meanResults.set(j, meanResults.get(j) + results.get(j));
        }

    }

    for(int i = 0; i < 200; i++) meanResults.set(i, meanResults.get(i) / runs);

    System.out.println("[BATTERY TEST RUN]\n");

    for(int i = 1; i < meanResults.size() + 1; i++){

        if(i % 8 == 0)        System.out.printf("[%d] %.2f \n", i, meanResults.get(i - 1));

        else System.out.printf("[%d] %.2f \t", i, meanResults.get(i - 1));

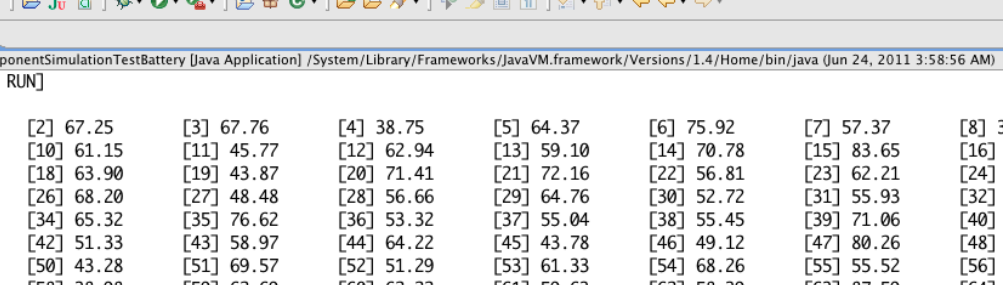
    }

}

//end Main

//end ComponentSimulationTestBattery

```



```
<terminated> ComponentSimulationTestBattery [Java Application] /System/Library/Frameworks/JavaVM.framework/Versions/1.4/Home/bin/java (Jun 24, 2011 3:58:56 AM)
[BATTERY TEST RUN]

[1] 17.29      [2] 67.25      [3] 67.76      [4] 38.75      [5] 64.37      [6] 75.92      [7] 57.37      [8] 39.22
[9] 69.07      [10] 61.15     [11] 45.77      [12] 62.94     [13] 59.10     [14] 70.78     [15] 83.65     [16] 61.96
[17] 68.75     [18] 63.90     [19] 43.87      [20] 71.41     [21] 72.16     [22] 56.81     [23] 62.21     [24] 56.36
[25] 49.53     [26] 68.20     [27] 48.48     [28] 56.66     [29] 64.76     [30] 52.72     [31] 55.93     [32] 80.69
[33] 61.92     [34] 65.32     [35] 76.62     [36] 53.32     [37] 55.04     [38] 55.45     [39] 71.06     [40] 50.38
[41] 65.09     [42] 51.33     [43] 58.97     [44] 64.22     [45] 43.78     [46] 49.12     [47] 80.26     [48] 56.26
[49] 48.20     [50] 43.28     [51] 69.57     [52] 51.29     [53] 61.33     [54] 68.26     [55] 55.52     [56] 53.95
[57] 56.32     [58] 38.98     [59] 62.69     [60] 62.22     [61] 59.63     [62] 58.39     [63] 87.59     [64] 47.71
[65] 73.83     [66] 57.30     [67] 61.58     [68] 56.91     [69] 55.70     [70] 38.18     [71] 61.82     [72] 49.29
[73] 58.63     [74] 49.54     [75] 71.48     [76] 78.82     [77] 44.74     [78] 44.91     [79] 61.76     [80] 54.42
[81] 78.35     [82] 44.20     [83] 68.50     [84] 47.00     [85] 48.94     [86] 52.45     [87] 49.88     [88] 70.38
[89] 45.57     [90] 38.00     [91] 64.67     [92] 59.62     [93] 68.53     [94] 51.68     [95] 95.65     [96] 40.98
[97] 48.37     [98] 51.52     [99] 51.70     [100] 57.46     [101] 52.84     [102] 54.85     [103] 77.08     [104] 64.76
[105] 48.76     [106] 60.07     [107] 57.29     [108] 82.59     [109] 64.99     [110] 60.36     [111] 62.61     [112] 35.81
[113] 51.26     [114] 54.69     [115] 58.18     [116] 47.62     [117] 76.63     [118] 43.08     [119] 50.85     [120] 55.61
[121] 61.76     [122] 50.84     [123] 45.26     [124] 54.05     [125] 60.24     [126] 47.16     [127] 58.96     [128] 89.39
[129] 53.64     [130] 74.87     [131] 45.24     [132] 58.49     [133] 30.59     [134] 51.63     [135] 48.58     [136] 34.85
[137] 72.91     [138] 40.40     [139] 42.07     [140] 25.82     [141] 53.96     [142] 55.57     [143] 39.38     [144] 46.82
[145] 15.82     [146] 37.10     [147] 43.09     [148] 25.56     [149] 56.80     [150] 27.62     [151] 50.83     [152] 48.96
[153] 43.13     [154] 31.28     [155] 19.20     [156] 41.20     [157] 17.90     [158] 19.94     [159] 8.21     [160] 12.46
[161] 8.96      [162] 18.08     [163] 4.09      [164] 12.95     [165] 4.28      [166] 8.69      [167] 7.98      [168] 11.20
[169] 0.61      [170] 0.20      [171] 0.60      [172] 0.48      [173] 0.50      [174] 0.00      [175] 0.00      [176] 0.00
[177] 0.00      [178] 0.00      [179] 0.00      [180] 0.00      [181] 0.00      [182] 0.00      [183] 0.00      [184] 0.00
[185] 0.00      [186] 0.00      [187] 0.00      [188] 0.00      [189] 0.00      [190] 0.00      [191] 0.00      [192] 0.00
[193] 0.00      [194] 0.00      [195] 0.00      [196] 0.00      [197] 0.00      [198] 0.00      [199] 0.00      [200] 0.00
```

```
// ComponentSimulationTest.java
```

```
package org.usp.stochastic;
```

```
import java.util.*;
```

```
public class ComponentSimulationTest {
```

```
    public static void main(String[] args) throws Exception{
```

```
        ComponentSimulation c = new ComponentSimulation();
```

```
        ArrayList<Double> simulation = c.getSimulation();
```

```
        System.out.println("[START] \n\nUNIQUE SIMULATION FOR EACH COMPONENT FAILURE\n");
```

```
        for(int i = 1; i < simulation.size() + 1; i++){
```

```
            if(i % 8 == 0)        System.out.printf("[%d] %.2f \n", i, simulation.get(i - 1));
```

```
            else System.out.printf("[%d] %.2f \t", i, simulation.get(i - 1));
```

```
        }
```

```
        System.out.println("\n\n[END]");
```

```
        int k = 10;
```

```
        double k10time = 0.0;
```

```
        for(int i = 0; i < k; i++) k10time += simulation.get(i);
```

```
        System.out.println("\n -> TIME UNTIL 10th COMPONENT FAIL = " + k10time);
```

```
        ArrayList<Double> costs = new ArrayList<Double>();
```

```
        double r = 0.05;
```

```
        double beta = 10;
```

```
        double lambda = (double) 1/60;
```

```
        double alpha = 1 / (1 + r);
```

```
        for(int i = 0; i < simulation.size(); i++){
```

```
            double cost = Math.exp(-alpha * simulation.get(i)) * beta;
```

```
            costs.add(cost);
```

```
        }
```

```
        double totalCost = 0.0;
```

```
        System.out.println("\n -> MAINTENANCE COST UNTIL 10th COMPONENT FAIL = " + costs.get(10));
```

```
        for(int i = 0; i < costs.size(); i++) {totalCost += costs.get(i);}
```

```
        System.out.println("\n -> TOTAL SYSTEM MAINTENANCE COST = " + totalCost);
```

```
        double estimatedParameter = totalCost / simulation.size();
```

```
        System.out.println("\n -> ESTIMATED PARAMETER FOR COST DISTRIBUTION = " +  
estimatedParameter);
```

```
        System.out.println("\n -> ESTIMATED LAMBDA FOR COST DISTRIBUTION = " +  
1/estimatedParameter);
```

```

    double teorethicalExpectedValueCost = beta * lambda / alpha;

    System.out.println("\n -> TEORETHICAL EXPECTED VALUE FOR COST DISTRIBUTION = " +
    teorethicalExpectedValueCost);

    double praticalExpectedValueCost = totalCost / costs.size();

    System.out.println("\n -> EMPIRIC EXPECTED VALUE FOR COST DISTRIBUTION = " +
    praticalExpectedValueCost);

    }//end Main
}//end ComponentSimulationTest

```

Uma execução dessa classe é colocada abaixo e é utilizada para responder as questões do trabalho :

```

Eclipse File Edit Run Source Refactor Navigate Search Project Window Help
Java - Trabalho Estocásticos/src/org/usp/stochastic/ComponentSimulationTest.java - Eclipse SDK -

[START]

UNIQUE SIMULATION FOR EACH COMPONENT FAILURE

[1] 181.20 [2] 223.69 [3] 15.97 [4] 49.25 [5] 83.98 [6] 6.96 [7] 24.15 [8] 35.55
[9] 66.80 [10] 67.72 [11] 24.74 [12] 7.71 [13] 141.55 [14] 427.71 [15] 171.98 [16] 7.02
[17] 31.59 [18] 2.31 [19] 38.48 [20] 75.02 [21] 10.66 [22] 23.56 [23] 65.70 [24] 23.18
[25] 0.21 [26] 13.82 [27] 68.98 [28] 115.81 [29] 14.38 [30] 42.89 [31] 4.46 [32] 63.68
[33] 11.23 [34] 39.72 [35] 50.26 [36] 21.09 [37] 37.80 [38] 15.03 [39] 28.78 [40] 20.99
[41] 163.00 [42] 0.07 [43] 78.22 [44] 13.16 [45] 3.11 [46] 84.68 [47] 1.09 [48] 37.59
[49] 13.81 [50] 42.99 [51] 13.95 [52] 54.89 [53] 15.22 [54] 140.99 [55] 31.19 [56] 13.66
[57] 43.85 [58] 154.11 [59] 109.26 [60] 16.47 [61] 7.15 [62] 78.54 [63] 97.39 [64] 30.55
[65] 63.22 [66] 131.81 [67] 14.01 [68] 37.88 [69] 26.08 [70] 40.46 [71] 82.65 [72] 78.73
[73] 62.44 [74] 47.87 [75] 64.47 [76] 97.83 [77] 2.48 [78] 66.20 [79] 31.58 [80] 146.45
[81] 37.55 [82] 19.42 [83] 42.02 [84] 31.88 [85] 5.83 [86] 101.70 [87] 144.14 [88] 30.72
[89] 19.06 [90] 285.60 [91] 31.59 [92] 43.35 [93] 135.44 [94] 70.87 [95] 33.39 [96] 36.86
[97] 6.72 [98] 32.27 [99] 56.27 [100] 44.82 [101] 119.67 [102] 134.77 [103] 60.28 [104] 16.82
[105] 32.60 [106] 24.11 [107] 29.70 [108] 34.48 [109] 19.67 [110] 181.83 [111] 91.70 [112] 21.90
[113] 81.49 [114] 70.39 [115] 2.83 [116] 75.81 [117] 45.12 [118] 56.65 [119] 41.12 [120] 63.41
[121] 37.87 [122] 11.95 [123] 13.67 [124] 89.46 [125] 57.01 [126] 83.88 [127] 4.10 [128] 20.98
[129] 40.47 [130] 48.93 [131] 16.37 [132] 81.37 [133] 48.62 [134] 11.58 [135] 107.30 [136] 144.75
[137] 14.64 [138] 119.36 [139] 12.62 [140] 7.68 [141] 3.92 [142] 13.31 [143] 165.27 [144] 1.94
[145] 11.38 [146] 25.89 [147] 27.30 [148] 48.31 [149] 52.94 [150] 99.87 [151] 43.00 [152] 150.63
[153] 14.88 [154] 74.75 [155] 51.79

[END]

-> TIME UNTIL 10th COMPONENT FAIL = 755.278060225791
-> MAINTENANCE COST UNTIL 10th COMPONENT FAIL = 5.859007258983542E-10
-> TOTAL SYSTEM MAINTENANCE COST = 26.5794669624752
-> ESTIMATED PARAMETER FOR COST DISTRIBUTION = 0.17148043201596902
-> ESTIMATED LAMBDA FOR COST DISTRIBUTION = 5.831569166485862
-> TEORETHICAL EXPECTED VALUE FOR COST DISTRIBUTION = 0.175
-> EMPIRIC EXPECTED VALUE FOR COST DISTRIBUTION = 0.17148043201596902

```

Exercício 01

Conforme pode ser conferido na simulação acima retratada, 155 componentes apresentaram falhas durante um ano de operação do sistema. Em particular, o tempo em horas até a falha do décimo componente é de **67.72 horas**.

Sabemos que esse componente tem distribuição $x \sim \text{Exp}(\lambda)$. Dessa forma, para esse componente temos $1/\lambda = 67.72$ ou seja $\lambda = 0.1476$.

Além disso, da distribuição exponencial sabemos que :

- $E(x) = 1/\lambda = 67.72$
- $\sigma(x) = \sqrt{1/\lambda^2} = 67.72$

Finalmente, para estimar o tempo de vida com probabilidade de 95%, tomamos a inversa da função de distribuição acumulada da exponencial, fazendo :

$$F^{-1}(p, \lambda) = -\frac{\ln(1-p)}{\lambda} = -\frac{\ln(0.05)}{0.1476} = -20.29$$

Exercício 02

Para essa questão, tem-se que o tempo de substituição do k-ésimo componente é dado por:

$$c_k = \beta e^{-\alpha T_k}$$

Sendo cada T_k positivo, o custo do k-ésimo componente é uma função exponencial negativa, ou seja, apresenta um valor cada vez menor conforme k aumenta, e no infinito esse custo tende a zero. Para o décimo componente podemos esperar um custo bem baixo, e conforme a execução da simulação

```
-> TIME UNTIL 10th COMPONENT FAIL = 755.278060225791  
-> MAINTENANCE COST UNTIL 10th COMPONENT FAIL = 5.859007258983542E-10
```

Pelos resultados acima, vemos que decorreram mais de 755 horas até que o décimo componente fosse substituído, ou seja, apresentasse falha. O custo dessa substituição é próximo de zero, conforme o esperado da natureza da fórmula do custo. A implementação também obteve o custo total de manutenção do sistema para essa simulação :

```
-> TOTAL SYSTEM MAINTENANCE COST = 26.5794669624752
```

Estamos interessados em estimar a distribuição do custo total de manutenção do sistema, que como calculado acima é dado por

$$C = \sum_{k=0}^n c_k$$

Assim, olhamos para o custo médio dos componentes para obter um estimador do parâmetro da distribuição exponencial que rege o comportamento de **C**, ou seja, fazemos $\bar{C} = 1/\hat{\lambda} = \mathbf{0.1714}$

-> ESTIMATED PARAMETER FOR COST DISTRIBUTION = 0.17148043201596902
-> ESTIMATED LAMBDA FOR COST DISTRIBUTION = 5.831569166485862

Logo, o parâmetro estimado é $\hat{\lambda} = \mathbf{5.8315}$.

Adicionalmente, sabemos que a distribuição exponencial apresenta o fenômeno de falta de memória. Dessa maneira, tantos os custos individuais como o custo total tem mesmo parâmetro para a distribuição exponencial, ou seja $c_k \sim \exp(\hat{\lambda})$ e $C \sim \exp(\hat{\lambda})$ de maneira que temos :

- $E(c_k) = E(C) = \mathbf{0.1714}$
- $\sigma(c_k) = \sigma(C) = \sqrt{1/\lambda^2} = \mathbf{0.1714}$
- FDA : $F(x) = 1 - e^{-\hat{\lambda}x}$

Exercício 03

Nesse exercício, comparamos os resultados práticos com as expectativas teóricas. Em particular, estamos interessados no valor esperado, de maneira que o resultado teórico diz que :

$$E(C) = \beta \frac{\lambda}{\alpha}, \text{ com } \alpha = 1/1 + r, r \text{ e } \beta \text{ dados}$$

De fato, a simulação mostra a proximidade dos resultados :

-> THEORETICAL EXPECTED VALUE FOR COST DISTRIBUTION = 0.175
-> EMPIRIC EXPECTED VALUE FOR COST DISTRIBUTION = 0.17148043201596902

Por questões de limitação desse editor de texto (MS Word), a demonstração desses resultados se encontra em anexo, escrita à mão.