CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
CAMPUS TIMÓTEO

José Geraldo Duarte Junior

LISTA 2 - AED II
PSEUDO GERADORES DE NÚMEROS ALEATÓRIOS

Timóteo

2022

José Geraldo Duarte Junior

LISTA 2 - AED II
PSEUDO GERADORES DE NÚMEROS ALEATÓRIOS

Atividade apresentado ao professor Gustavo Martins, da disciplina de Algoritmo e Estrutura de Dados 2 do curso de Engenharia de Computação do Campus Timóteo do Centro Federal de Educação Tecnológica de Minas Gerais para aprovação na disciplina ofertada pelo mesmo.

Orientador: Gustavo Martins

Timóteo

2022

Lista de ilustrações

Sumário

1 Introdução

“Não existe resposta errada:
ela apenas não responde a essa pergunta”.
Confia

Um gerador de número pseudo-aleatório é um algoritmo derivado de uma função matemática que gera uma seqüência de números, os quais são aparentemente independentes um dos outros. O resultado da maioria dos geradores de números aleatórios não é verdadeiramente aleatório, eles somente possuem algumas das propriedades dos números aleatórios.

Alguns exemplos de números aleatórios são: tempo de resposta de requisições de leitura de um disco rígido e tubos de descarga de gás. Número pseudo-aleatórios são uma parte crítica da computação moderna, da criptografia até o método de Monte Carlo passando por sistemas de simulação. Uma cuidadosa análise matemática é necessária para assegurar que a geração dos números seja suficientemente "aleatória"(WIKIPéDIA, 2022).

No desenvolver deste trabalho serão apresentados processos e resultados de testes de um dos mais conhecidos geradores de números aleatórios, que é baseado no chamado método das congruências lineares, apresentando dados que comprovam a pseudo aleatoriedade deste método e aplicação do mesmo em um algoritmo de preenchimento de vetores em Java.

2 Desenvolvimento

2.1 Implementação do Gerador Congruencial Linear

O código a seguir após implementado consegue receber um vetor por parâmetro e preenche-lo com valores gerados congruencialmente. por parâmetro também são informados os seguintes dados:

m: modulo;

a: multiplicador;

b: incremento;

noOfRandomNums: quantidade de valores aleatórios.

```
private void MetodoCongruencialLinear(int base, int m, int a, int b, int[] randomNums, int noOfRandomNums) {

    // primeira casa do vetor recebe a semente
    randomNums[0] = base;

    for (int i = 1; i < noOfRandomNums; i++) {

        randomNums[i] = ((randomNums[i - 1] * a) + b) % m;
    }
}
```

Figura 1 – Gerador Congruencial linear em Java (??, 2021)

Dado um número inicial conhecido como semente, o próximo número da sequência é dado pela seguinte equação:

$$randomNums[0] = ((base * a) + b)\ \%\ m \tag{2.1}$$

Os valores sucessores são determinados pela seguinte equação:

$$randomNums[i] = ((randomNums[i-1] * a) + b)\ \%\ m \tag{2.2}$$

O operador % indica resto da divisão. Os números assim gerados estão entre 0 e m-1 (MIRANDA, 2014).

Para a amostragem deste trabalho foram utilizados os seguintes valores: **m:** 100000;

a: 157;

b: 1;

noOfRandomNums: 5000.

Com tais valores o código implementado gera um vetor de n posições preenchido com os valores resultantes do gerador.

2.2 Estudo do Período

Durante os teste do gerador é possível notar uma periodicidade nos valores gerados, onde de acordo com os valores pre definidos no mesmo é gerado um numero limitado de valores que se repetem após determinado período.

Para os valores informados na seção 2.1, foi possível obter os seguintes dados para este gerador:

```
Periodo = 3999
5% = 199 Valores
10% = 399 Valores
25% = 999 Valores
50% = 1999 Valores
75% = 2999 Valores
```

Figura 2 – Analise de período do gerador

Utilizando estes valores é possível plotar gráficos que ilustrem um padrão no processo de gerar números, comprovando que estes valores não são aleatórios. A seguir serão apresentados 6 diferentes gráficos, que apresentam diferentes porcentagens do período do gerador implementado, quanto mais valores do período são utilizados para gerar coordenadas X e Y, mais visível o padrão se torna mais visível.

Para tal experimentação, os valores do período foram armazenados em um vetor, onde os valores armazenados neste vetor que possuem índice par representarão as coordenadas X, e os de índices impares as coordenadas Y.

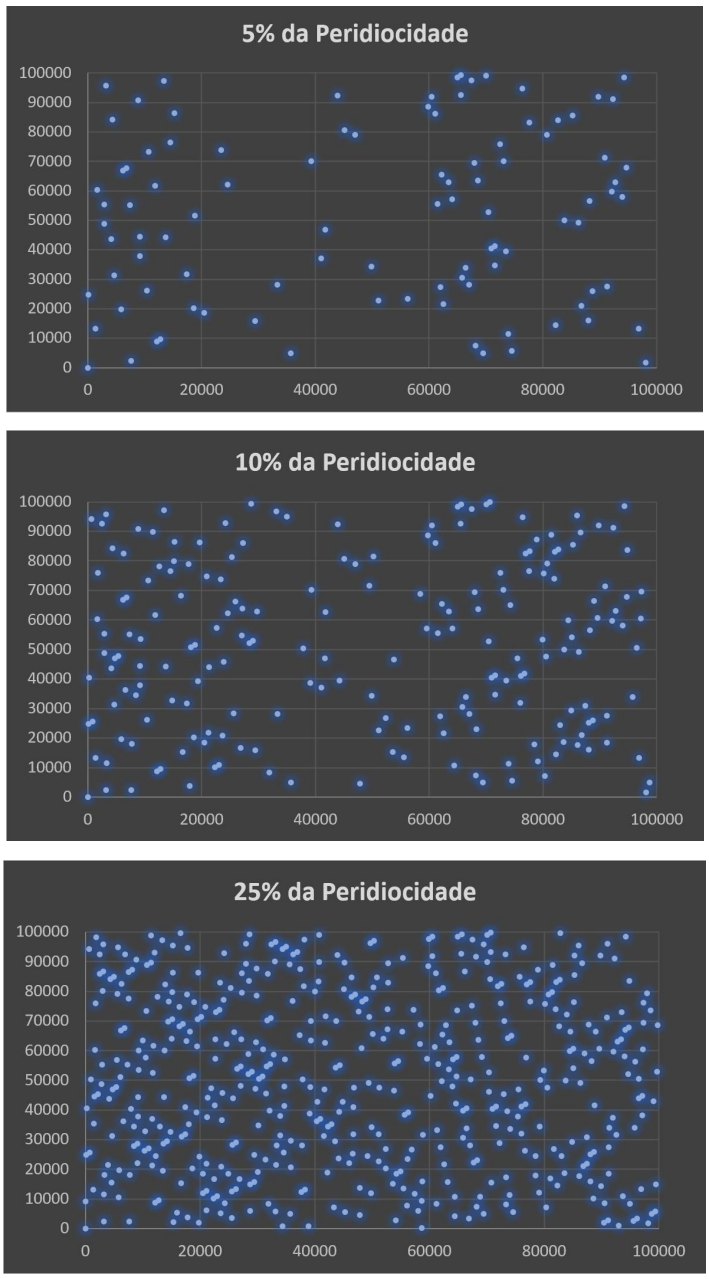

Figura 3 – Gráficos Períodos do Gerador (5% a 25%)

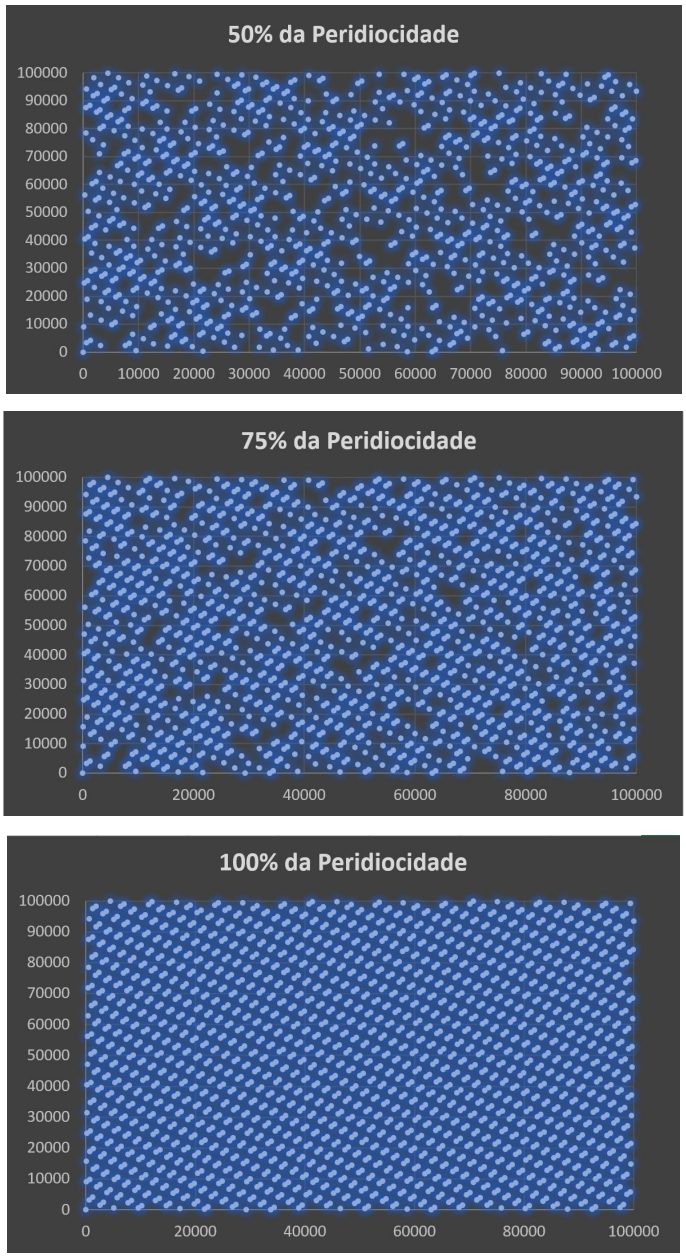

Figura 4 – Gráficos Períodos do Gerador - (50% a 100%)

2.3 Math.random

Também é possível gerar números pseudoaleatórios usando o método estático random() da classe Math. Este método retorna um valor double no intervalo de 0.0 até 1.0, sem incluir este último valor (0.0 <= x < 1.0). Caso se necessite de outros intervalos de valores, será preciso efetuar operações aritméticas com os valores gerados, como por exemplo uma multiplicação(REIS, 2018).

Um dado importante sobre o método Math.random é que ele, internamente, utiliza a própria classe java.util.Random para gerar os números aleatórios. Para gerar números inteiros com esse método é necessário realizar cast para int do valor gerado(REIS, 2018).

Para exemplificar a capacidade de gerar valores da biblioteca nativa do Java, foram gerados 4000 valores aleatórios, que foram posicionados em pares ordenados utilizando a mesma regra implementada nos testes do gerador congruencial.

Com estes teste foi possível ver que a amplitude entre a semente e o fim do periodo do gerador desta biblioteca é muito grande, onde nos teste não foi possivel ter uma percepção visual da pseudoaletoriedade.

Os resultados foram plotados no gráfico a seguir:

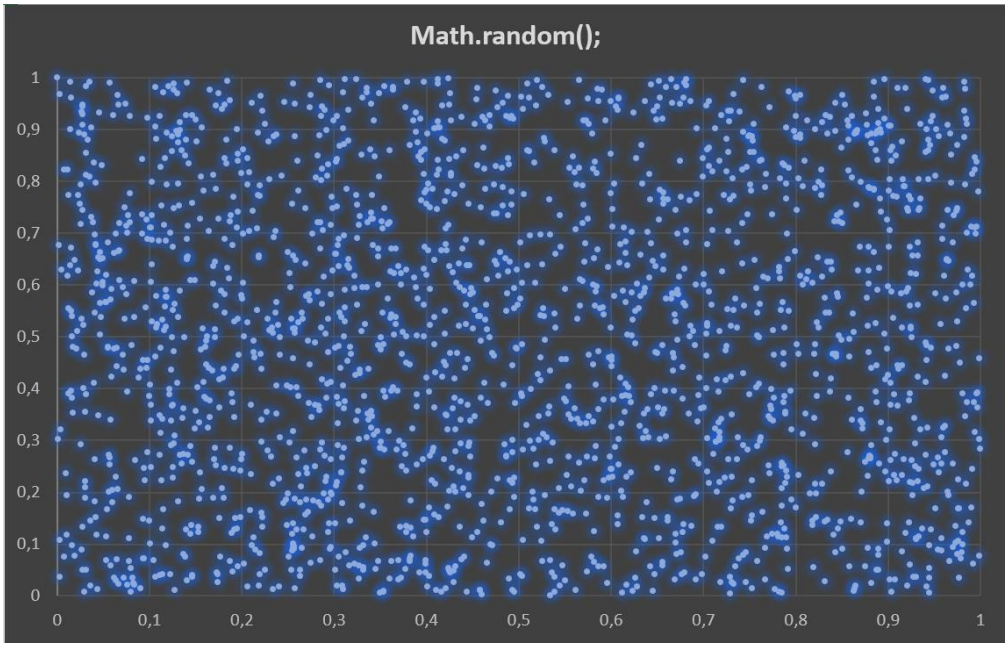

Figura 5 – Periodo de 2000 valores gerados na biblioteca Math.random()

3 Conclusão

“Palavras não bastam, não dá pra entender
E esse medo que cresce não para
É uma história que se complicou
Eu sei bem o porquê ”
Tiê

Logo ao comparar estes geradores percebemos que a biblioteca nativa do java trás benefícios ao ser mais facilmente incluído em um código por ser nativo, mas possui o contra onde para que seu período não seja somente entre 0 e 1, e para burlar isso os seus valores precisam passar por operações matemáticas aplicadas por aquele que esta escrevendo o código.

Já o gerador implementado nos testes anteriores possui também seus prós e contras, este exige sua implementação e até mesmos ajustes de acordo com onde será utilizado e também necessite de um estudo de um modulo, um multiplicador, e um incremento para construir um período que atenda as necessidades, mas este possui o beneficio de sua simplicidade e uma execução sem muito custo computacional.

Também foi possível comparar o tempo de execução entre estes algoritmos, e perceber que o PRNG implementado manualmente supera em a biblioteca do java no quesito velocidade para preencher um vetor com valores gerados por estes métodos, veja um exemplo de resultado na imagem a seguir:

```
        comparaTempo();
    }

    static public void comparaTempo() {
        PRNG teste = new PRNG();
        long tempo = System.nanoTime();
        int t[] = teste.LCM(100000, 5000);
        tempo = System.nanoTime() - tempo;
        System.out.println("Tempo para o PRNG implementado preencher um vetor de 5 mil posicoes: " + tempo + " Nanossegundos");

        tempo = System.nanoTime();
        double t2[] = new double[5000];
        for (int i = 0; i < t2.length; i++) {
            t2[i] = Math.random();
        }
        tempo = System.nanoTime() - tempo;
        System.out.println("Tempo para o preencher um vetor de 5 mil posicoes com Math.random(): " + tempo + " Nanossegundos");

    }

    static public void preencheRand(double v[]) { ... 5 linos ... }
```

praticaaedrand.PraticaAEDrand › comparaTempo

Output - PraticaAEDrand (run)

```
run:
Tempo para o PRNG implementado preencher um vetor de 5 mil posicoes: 116300 Nanossegundos
Tempo para o preencher um vetor de 5 mil posicoes com Math.random(): 242800 Nanossegundos
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figura 6 – Comparação de tempos de execução

Por fim o mais importante a se concluir com este experimento é que não podemos considerar estes geradores como aleatórios, estes possuem sim um padrão de resultados que pode ser ilustrado. Mas o importante é que mesmo sendo pseudo aleatórios estes não deixam de ser uteis e em determinadas implementações cumprem o papel de geradores aleatórios.

Referências

?? *MÉTODO DE CONGRUÊNCIA LINEAR PARA GERAR NÚMEROS PSEUDO-ALEATÓRIOS*. 2021. Urlhttps://acervolima.com/metodo-de-congruencia-linear-para-gerar-numeros-pseudo-aleatorios/. Citado nas páginas 2 e 5.

MIRANDA, P. A. V. *Números pseudo-aleatórios*. 2014. Urlhttp://www.vision.ime.usp.br/ pmiranda/mac110$_1 s14/EPs/ep01/numeros_a leatorios.html. Citadonapágina$

REIS, F. dos. *Como gerar números aleatórios em Java com java.util.Random e Math.random()*. 2018. Urlhttp://www.bosontreinamentos.com.br/java/como-gerar-numeros-aleatorios-em-java-com-java-util-random-e-math-random/. Citado na página 9.

WIKIPéDIA. *Gerador de números pseudoaleatórios — Wikipédia, a enciclopédia livre*. 2022. [Online; accessed 23-agosto-2022]. Disponível em: <https://pt.wikipedia.org/w/index.php?title=Gerador_de_n%C3%BAmeros_pseudoaleat%C3%B3rios&oldid=64257701>. Citado na página 4.