Construção de geradores independentes de números aleatórios para diferentes distribuições probabilísticas

Geraldo F.D. Zafalon[1], **Aleardo Manacero Jr.**[2]

[1]Bach. em Ciência da Computação – Universidade Estadual Paulista - UNESP
15.054-000 – São José do Rio Preto – SP – Brasil

[2]DCCE/Ibilce – Universidade Estadual Paulista - UNESP

zafalon@comput.ibilce.unesp.br, aleardo@ibilce.unesp.br

Abstract. *The determination of systems behaviour, especially the non-deterministic ones, is making a heavy use of simulators. In order to model non-deterministic systems it is necessary the use of probability distribution functions that truly represent the behaviour of each event. This can be performed through the use of different random number generators. Here it is introduced a set of random number generators, developed independently of standard libraries. The results achieved enable the development of efficient independent generators that could be used on discrete event simulators.*

Resumo. *O uso de simuladores para a determinação do comportamento de sistemas tem crescido notadamente, principalmente em sistemas com eventos não determinísticos. A modelagem desses sistemas necessita de funções de distribuição de probabilidades que representem mais fielmente o comportamento de cada evento. Isso é feito por sorteios de números aleatórios segundo diferentes funções de distribuição. Apresenta-se aqui o desenvolvimento de um conjunto de geradores de números aleatórios com funcionamento independente de bibliotecas padrão. Os resultados obtidos permitem construir bons geradores independentes, para uso em simuladores de sistemas discretos.*

1. Introdução

O uso intenso de computadores em todas as áreas de conhecimento é percebido também na simulação de sistemas discretos ou contínuos, como reatores nucleares, corpos celestes, populações e predição de desempenho. Para essa área a computação possui impacto ainda maior, uma vez que sistemas complexos possuem uma quantidade de parâmetros e requisitos que tornam impossível sua simulação sem o auxílio de computadores.

Uma característica comum a grande parte dos sistemas contínuos ou discretos é que os eventos simulados são, em geral, indeterminísticos, ou seja, não podem ser precisamente determinados a partir do estado atual do sistema. Comportamentos indeterminísticos seguem, em geral, padrões que obedecem distribuições de probabilidade bem definidas. Assim, sua simulação correta precisa fazer uso de geradores de números aleatórios que correspondam adequadamente ao perfil de tais distribuições.

Nesse trabalho se examina a construção de um gerador de números pseudoaleatórios, que atenda o perfil de distribuição uniforme. Esse gerador é utilizado como base na construção de geradores independentes para outras funções de distribuição. Nas

próximas seções são apresentadas uma descrição dos modelos de geração computacional de números aleatórios, a especificação dos geradores construídos nesse trabalho e os resultados e conclusões obtidos nos testes de tais geradores.

2. Geradores de números aleatórios

O processo de geração computacional de números aleatórios deve atender vários requisitos para ser eficiente, tais como velocidade e conformidade. Esses requisitos, infelizmente, são contraditórios pois geradores muito rápidos não podem fazer operações mais complexas, que permitiriam melhor conformidade [O. Goldreich and Micali 1986].

Isso leva à investigação de quais geradores possuem um bom comportamento e como podem ser implementados. L'Ecuyer [L'Ecuyer 2001], por exemplo, apresenta um estudo detalhado sobre vários geradores disponíveis em aplicativos bastante populares, tais como Microsoft Excel e Visual Basic. De forma geral, sua conclusão é de que tais geradores não são muito precisos. Outras ferramentas disponibilizam geradores mais eficientes e confiáveis, como Arena[1] e AutoMod[2], que são pacotes de *softwares* de simulação.

Uma grande parte dos geradores aleatórios se baseia no modelo congruencial linear, em que a função de geração usa a multiplicação de parâmetros normalizados para obter os números desejados. Exemplos de Geradores Lineares Congruenciais (GLC) incluem o ANSIC, utilizado na linguagem de programação C, o MINSTD (*Minimal Standard*), que apresenta uma ótima cobertura do período de geração [P.A. Lewis and Miller 1969], e o SIMSCRIPT. A descrição desses e outros geradores pode ser encontrada em [pLab 2005].

Outros trabalhos buscam um melhor desempenho dos geradores. Uma alternativa envolve o uso de funções não multiplicativas [Wong 1990], em que se procura uma maior eficiência com geradores que executam apenas operações de adição e comparação, em vez de realizar operações de multiplicação.

3. Construção de geradores aleatórios

O processo de construção de geradores aleatórios deve obedecer restrições de velocidade e cobertura do intervalo de geração. Além disso, para seu uso em simuladores é preciso que seja capaz de gerar números que obedeçam diferentes funções de distribuiçao de probabilidade (fdp), tais como uniforme, normal e exponencial [Papoulis and Pillai 2002]. Nesses casos é preciso ainda que esses geradores sejam independentes, para que não ocorram interferências entre os números gerados em cada distribuição. Esse último problema leva ao gerador da distribuição uniforme, que é a base dos demais geradores.

3.1. Definição de um gerador uniforme congruencial linear (GCL)

Nesse trabalho adotou-se um gerador congruencial linear como base do gerador uniforme. Geradores desse tipo são bastante utilizados, uma vez que a adoção de parâmetros adequados permite a construção de geradores que atendam eficientemente as restrições de velocidade e cobertura do intervalo. Um gerador para distribuição uniforme, segundo as propriedades dos geradores congruenciais lineares, usa a seguinte regra de recursão:

$$Z_n = (a.Z_{n-1}) mod(m)$$

[1] `http://www.arenasimulation.com`
[2] `http://www.autosim.com/index.asp`

Neste caso, *a* é o multiplicador do gerador e *m* determina o tamanho do intervalo de geração, ou o seu ciclo. É importante que a escolha dos valores de *a* e *m* seja bem feita, pois eles é que ditam a qualidade de um determinado gerador [Wong 1990]. O gerador implementado utiliza os parâmetros $a = 16807$ e $m = (2^{31} - 1)$, que são bastante conhecidos e proporcionam uma boa cobertura sobre o intervalo de geração [Knuth 1981].

3.2. Definição de geradores para funções de distribuição variadas

A partir do gerador para a distribuição uniforme é possível definir geradores para outras funções de distribuição, como a normal e a exponencial, através da aplicação da função inversa [Knuth 1981, Papoulis and Pillai 2002]. Dentre as várias funções de distribuição existentes, adotaram-se nesse trabalho as funções normal, exponencial, gama e pareto. As funções normal, exponencial e gama foram escolhidas devido a sua usabilidade, pois elas conseguem modelar, com boa fidelidade, eventos como o número de acessos, ao longo do dia, em um servidor de e-mails. Já a distribuição pareto foi escolhida por se caracterizar como uma distribuição de cauda longa, isto é, uma distribuição que permite a ocorrência de eventos muito distantes dos valores médios da função, típicas na modelagem de cargas de trabalho, tanto em processadores como em serviços *Web*.

4. Testes e Resultados

A validação dos geradores de números aleatórios descritos na seção anterior ocorreu através de uma série de testes, envolvendo tanto exames sobre a completude de cobertura do intervalo de geração, quanto testes analíticos sobre a qualidade dos números gerados, em especial através do teste do *chi-quadrado*. Esses testes são descritos a seguir.

4.1. Teste do gerador uniforme

Para o gerador uniforme verificou-se, para diferentes intervalos de geração, o número médio de vezes que cada valor foi gerado. Verificou-se também a dispersão desses valores pelo intervalo de geração. Comparou-se esses valores com os obtidos com a função *rand()* da linguagem *C*. Os resultados podem ser vistos na tabela 1. Em todos os casos foram gerados números para intervalos (períodos) distintos de geração, sendo que para cada intervalo o número de valores gerados foi igual a 1000 vezes o tamanho do período.

Na tabela cada linha apresenta o total de ocorrências dentro do intervalo de geração em que o número de vezes que o valor foi gerado ficou distante do valor médio em mais do que *k%*, sendo *k* o valor da coluna *Desvio*. Por exemplo, tem-se que o número de valores entre 1 e 10.000, cuja ocorrência ficou fora do intervalo de 4% da média (960 a 1040 ocorrências) foi de 2004 para o gerador implementado e 2033 para a função *rand()*.

Num exame dessa tabela percebe-se que a convergência de valores é maior no gerador implementado do que no gerador *rand()*. Este resultado reforça a adoção de um gerador diferenciado para a construção dos demais geradores. Num teste mais preciso, o desvio padrão obtido para o gerador implementado foi de 30,875, contra 32,123 para a função *rand()*. Isso corrobora a impressão, criada pelo exame visual da tabela 1, de que o gerador implementado funciona melhor que o da função *rand()*.

4.2. Resultados para os demais geradores

Após a determinação de que o GLC implementado apresentou boa qualidade, passou-se ao exame dos demais geradores (normal, exponencial, gama e pareto). Os próximos resultados ilustram parte dos testes realizados, uma vez que esses envolveram uma variedade

Tabela 1. Número de ocorrências com desvios em relação à média

Desvio (%)	Gerador implementado Intervalo de Geração			Gerador da linguagem C Intervalo de Geração		
	1 a 100	1 a 10.000	1 a 100.000	1 a 100	1 a 10.000	1 a 100.000
± 2%	58	5118	48419	52	5204	51718
±4%	23	2004	16682	17	2033	20119
±6%	3	495	3977	7	576	5543
±8%	0	94	608	1	105	1103
±10%	0	60	64	0	15	163
±12%	0	0	3	0	0	19
±14%	0	0	0	0	0	1
±16%	0	0	0	0	0	1
±18%	0	0	0	0	0	0
±20%	0	0	0	0	0	0

de testes que não caberiam neste texto. Em geral pode-se afirmar que não houve variações na qualidade do gerador implementado, independente do tipo de distribuição. As curvas apresentadas a seguir foram construídas para 10 milhões de números gerados, a partir da distribuição Uniforme, aplicando-se a transformação específica de cada distribuição.

- O gráfico da figura 1 apresenta o resultado obtido com a geração da uma curva segundo a função de distribuição de probabilidade Normal. Gerou-se essa curva com os parâmetros de média (μ) igual a 50 e desvio padrão (σ) igual 8.

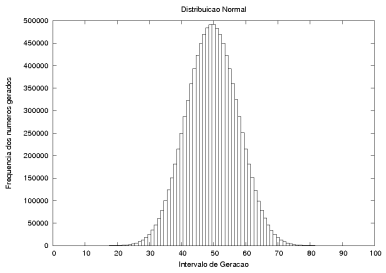

Figura 1. Curva Normal para $\mu = 50$ e $\sigma = 8$, respectivamente

- O gráfico da figura 2 mostra o resultado obtido para a função de distribuição Exponencial. Para essa curva utilizou-se $\mu = 20$.

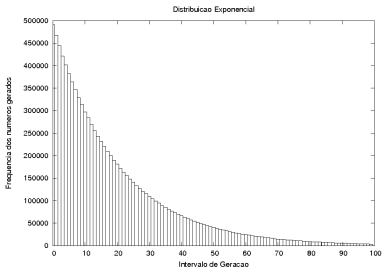

Figura 2. Curva Exponencial para $\mu = 20$

- A figura 3 apresenta uma curva obtida para a função de distribuição de probabilidade Gama. Nesse caso usou-se $\mu = 20$.

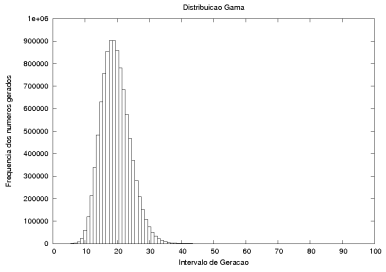

Figura 3. Curva Gama para $\mu = 20$

- Por fim, na figura 4 apresenta-se o gráfico para a função Pareto. Os dois parâmetros que determinam o formato da função Pareto, $\alpha = 20$ e $\beta = 20$.

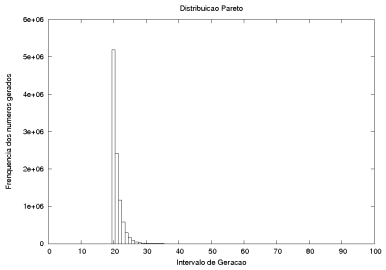

Figura 4. Curva Pareto para $\alpha = 20$ e $\beta = 20$

4.3. Teste do Chi-quadrado

Ao se implementar geradores de números aleatórios é preciso validar sua precisão através de testes estatísticos, como o do chi-quadrado. Este teste permite comparar os valores obtidos através do gerador com os que seriam determinados pela curva real da função.

Apresenta-se aqui apenas o resultado obtido com o gerador para a função uniforme, uma vez que esse gerador é a base dos demais. O cálculo do chi-quadrado para o gerador uniforme implicou em um valor de fidelidade da ordem de 50% (obtido a partir de tabelas de chi-quadrado, como [Papoulis and Pillai 2002]). Esse valor está dentro da faixa de valores aceitáveis, uma vez que toda a teoria de testes indica que valores de chi-quadrado acima de 25% implicam em boa correlação e valores muito elevados implicam na ausência de aleatoriedade.

Para as demais funções o cálculo do chi-quadrado apresenta problemas intrínsecos de metodologia. Ocorre que as funções aqui implementadas são, teoricamente, contínuas. Entretanto os geradores são construídos para gerarem valores discretos. Desse modo, como o cálculo do chi-quadrado envolve a comparação entre valores teóricos e valores experimentais, temos o problema de como definir o ponto de acumulação da curva teórica para a comparação com a experimental.

5. Conclusões e perspectivas futuras

O objetivo desse trabalho foi o desenvolvimento de geradores de números aleatórios com boa qualidade para funções variadas de distribuição de probabilidades. Nesse sentido, os

resultados obtidos com os geradores aqui desenvolvidos se mostram bastante promissores. O gerador uniforme apresentou resultados ligeiramente superiores aos da função *rand()* da linguagem C, muito utilizada na implementação de simuladores. Isso permite seu uso como gerador independente na construção de cada um dos demais geradores.

A precisão obtida com os demais geradores também foi significativa, permitindo que se trabalhe com a adequação de cada gerador a valores distintos de média e desvio padrão. Assim, pode-se concluir que os geradores implementados são suficientemente robustos e podem ser usados na construção de ferramentas mais genéricas de simulação.

Do ponto de vista de velocidade os geradores implementados apresentam resultados relativamente bons. Por exemplo, a geração de um valor uniforme toma cerca de 86ns, contra 34ns da função *rand()*. Esse maior tempo se justifica pela necessidade do gerador implementado em fazer uso de outras funções matemáticas (fmod). Além disso, como o objetivo é a criação de geradores independentes, não se pode usar a mesma função (*rand()*) na construção dos demais geradores. De qualquer modo, como a chamada do gerador em aplicações reais ocorre dentro de funções complexas, essa diferença de tempo deverá ser desprezível.

Na continuidade desse trabalho planeja-se o desenvolvimento de uma biblioteca e de um simulador de filas em Java. A biblioteca de geradores em Java permitiria uma fácil portabilidade dos geradores, bem como a sua adoção em ferramentas com interfaces gráficas mais elaboradas. Já o simulador de filas é útil em várias aplicações. Uma primeira versão apresenta uma interface bastante amigável e didática, permitindo seu uso no ensino de teoria de filas. Outro simulador de sistemas de eventos discretos, também em desenvolvimento, deverá ser aplicado em questões de planejamento de capacidade e suscetibilidade à rajadas de demanda em redes de computadores.

Referências

Knuth, D. (1981). *The art of Computer Programming*, volume 2 of *Seminumerical Algorithms*. Addison-Wesley, 2nd edition.

L'Ecuyer, P. (2001). Software for uniform random number generation: distinguishing the good and the bad. In *Proceedings of the 2001 Winter Simulation Conference*.

O. Goldreich, S. G. and Micali, S. (1986). How to construct random functions. *Journal of the Association for Computing Machinery*, 33(4):792–807.

P.A. Lewis, A. G. and Miller, J. (1969). A pseudo-random number generator for the system/360. *IBM Syst. J.*, 8:136–146.

Papoulis, A. and Pillai, S. (2002). *Probability, Random Variables and Stochastic Processes*. McGraw-Hill, 4th edition.

pLab (2005). Theory and practice of random number generation. In *http://crypto.mat.sbg.ac.at/∼charly/server/node3.html*. acessado em 20/03/2006.

Wong, P.-C. (1990). Random number generation without multiplication. In *9th Annual Intl Phoenix Conf on Computers and Communications*, pages 217–221.